ISSN 1818-1015
eISSN 2313-5417

2021 Volume 28 No 1

MODELING AND ANALYSIS OF INFORMATION SYSTEMS

SCIENTIFIC JOURNAL

Start date of publication — 1999
Published quarterly

FOUNDER
P.G. Demidov Yaroslavl State University

EDITORIAL OFFICE
14 Sovetskaya str., Yaroslavl 150003, Russian Federation

Website: http://mais-journal.ru
E-mail: mais@uniyar.ac.ru
Phone: +7 (4852) 79-77-73

2021 Том 28 № 1

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ

НАУЧНЫЙ ЖУРНАЛ

Издается с 1999 года
Выходит 4 раза в год

УЧРЕДИТЕЛЬ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Ярославский государственный университет им. П. Г. Демидова»

РЕДАКЦИЯ
ул. Советская, 14, Ярославль, 150003, Российская Федерация
Website: http://mais-journal.ru
E-mail: mais@uniyar.ac.ru
Телефон: +7 (4852) 79-77-73

Свидетельство о регистрации СМИ ПИ № ФС 77–66186 от 20.06.2016 выдано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций. Подписной индекс — 31907 в Объединенном каталоге «Пресса России». Технический редактор, компьютерная вёрстка – М. С. Каряева. Корректор английского текста – М. С. Комар. Подписано в печать 21.03.2021. Дата выхода в свет 31.03.2021. Формат 200×265 мм. Объем 120 с. Тираж 40 экз. Свободная цена. Заказ 008/021. Адрес типографии: ул. Советская, 14, оф. 109, Ярославль, 150003, Россия. Адрес издателя: Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, Ярославль, 150003, Россия.
Содержание предназначено для детей старше 12 лет.

Contents

Содержание

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 1, 2021
journal homepage: www.mais-journal.ru

ALGORITHMS

Backtracking Algorithms for Constructing the Hamiltonian Decomposition of a 4-regular Multigraph

A. V. Korostil[1], A. V. Nikolaev[1]

DOI: 10.18255/1818-1015-2021-1-6-21

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya, Yaroslavl 150003, Russia.

MSC2020: 05C85, 52B05
Research article
Full text in Russian

Received February 15, 2021
After revision March 10, 2021
Accepted March 12, 2021

We consider a Hamiltonian decomposition problem of partitioning a regular graph into edge-disjoint Hamiltonian cycles. It is known that verifying vertex non-adjacency in the 1-skeleton of the symmetric and asymmetric traveling salesperson polytopes is an NP-complete problem. On the other hand, a sufficient condition for two vertices to be non-adjacent can be formulated as a combinatorial problem of finding a Hamiltonian decomposition of a 4-regular multigraph. We present two backtracking algorithms for verifying vertex non-adjacency in the 1-skeleton of the traveling salesperson polytope and constructing a Hamiltonian decomposition: an algorithm based on a simple path extension and an algorithm based on the chain edge fixing procedure.

Based on the results of the computational experiments for undirected multigraphs, both backtracking algorithms lost to the known heuristic general variable neighborhood search algorithm. However, for directed multigraphs, the algorithm based on chain fixing of edges showed comparable results with heuristics on instances with existing solutions, and better results on instances of the problem where the Hamiltonian decomposition does not exist.

Keywords: Hamiltonian decomposition, traveling salesperson polytope, 1-skeleton, vertex adjacency, backtracking

INFORMATION ABOUT THE AUTHORS

Alexander V. Korostil	orcid.org/0000-0003-1881-0207. E-mail: av.korostil@gmail.com postgraduate student.
Andrei V. Nikolaev correspondence author	orcid.org/0000-0003-4705-2409. E-mail: andrei.v.nikolaev@gmail.com PhD, associate professor.

Funding: This work was supported by P. G. Demidov Yaroslavl State University Project № VIP-016.

For citation: A. V. Korostil and A. V. Nikolaev, “Backtracking Algorithms for Constructing the Hamiltonian Decomposition of a 4-regular Multigraph”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 6-21, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 1, 2021
сайт журнала: www.mais-journal.ru

ALGORITHMS

Алгоритмы поиска с возвратом для построения гамильтонова разложения 4-регулярного мультиграфа

А. В. Коростиль[1], А. В. Николаев[1]

DOI: 10.18255/1818-1015-2021-1-6-21

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, г. Ярославль, 150003 Россия.

УДК 519.16, 004.021, 514.172.45
Научная статья
Полный текст на русском языке

Получена 15 февраля 2021 г.
После доработки 10 марта 2021 г.
Принята к публикации 12 марта 2021 г.

Рассматривается задача построения гамильтонова разложения регулярного мультиграфа на гамильтоновы циклы без общих рёбер. Известно, что проверка несмежности вершин в полиэдральных графах симметричного и асимметричного многогранников коммивояжёра является NP-полной задачей. С другой стороны, достаточное условие несмежности вершин можно сформулировать в виде комбинаторной задачи построения гамильтонова разложения 4-регулярного мультиграфа. В статье представлены два алгоритма поиска с возвратом для проверки несмежности вершин в полиэдральном графе коммивояжёра и построения гамильтонова разложения 4-регулярного мультиграфа: алгоритм на основе последовательного расширения простого пути и алгоритм на основе процедуры цепного фиксирования рёбер.

По результатам вычислительных экспериментов для неориентированных мультиграфов оба переборных алгоритма проиграли известному эвристическому алгоритму поиска с переменными окрестностями. Однако для ориентированных мультиграфов алгоритм на основе цепного фиксирования рёбер показал сопоставимые результаты с эвристиками на экземплярах задачи, имеющих решение, и лучшие результаты на экземплярах задачи, для которых гамильтонова разложения не существует.

Ключевые слова: Гамильтоново разложение, многогранник коммивояжёра, полиэдральный граф, смежность вершин, поиск с возвратом

ИНФОРМАЦИЯ ОБ АВТОРАХ

Александр Васильевич Коростиль | orcid.org/0000-0003-1881-0207. E-mail: av.korostil@gmail.com
аспирант.

Андрей Валерьевич Николаев
автор для корреспонденции | orcid.org/0000-0003-4705-2409. E-mail: andrei.v.nikolaev@gmail.com
канд. физ.-мат. наук, доцент.

Финансирование: Работа выполнена в рамках инициативной НИР ЯрГУ им. П. Г. Демидова № VIP-016.

Для цитирования: A. V. Korostil and A. V. Nikolaev, “Backtracking Algorithms for Constructing the Hamiltonian Decomposition of a 4-regular Multigraph”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 6-21, 2021.

Введение

Гамильтоновым разложением регулярного мультиграфа называется разбиение множества его рёбер на гамильтоновы циклы. Задача поиска в заданном регулярном графе гамильтоновых циклов без общих рёбер находит применение в комбинаторной оптимизации [1], теории кодирования [2, 3], алгоритмах распределенного интеллектуального анализа данных [4], анализе взаимосвязанных сетей [5] и других областях. См. также теоретические результаты по оценке числа гамильтоновых разложений регулярных графов [6]. В данной работе задача построения гамильтонова разложения возникает в области полиэдральной комбинаторики.

1. Многогранник коммивояжёра

Рассматривается классическая постановка задачи коммивояжёра: задан полный взвешенный граф (или орграф) $K_n = (V, E)$, найти гамильтонов цикл минимального веса. Обозначим через HC_n множество всех гамильтоновых циклов в графе K_n и сопоставим каждому гамильтонову циклу $x \in HC_n$ характеристический вектор $x^v \in \mathbb{R}^E$ по следующему правилу:

$$x_e^v = \begin{cases} 1, & \text{если цикл } x \text{ содержит ребро } e, \\ 0, & \text{в противном случае.} \end{cases}$$

Многогранник

$$\mathrm{STSP}(n) = \mathrm{conv}\{x^v \mid x \in HC_n\}$$

называется *многогранником симметричной задачи коммивояжёра.*

Многогранник асимметричной задачи коммивояжёра $\mathrm{ATSP}(n)$ определяется аналогично как выпуклая оболочка характеристических векторов всех возможных гамильтоновых циклов в полном орграфе K_n.

Подход к решению задачи коммивояжёра с помощью целочисленного линейного программирования был впервые представлен в классической работе Данцига, Фалкерсона и Джонсона для 49 городов США [7]. Лучшие на данный момент точные алгоритмы для решения задачи коммивояжёра основаны на частичном описании фасет многогранника коммивояжёра и методе ветвей и отсечений для целочисленного линейного программирования [8].

Полиэдральным графом многогранника называется граф, вершинами которого являются вершины многогранника, а рёбрами – геометрические рёбра, т.е. одномерные грани. Исследование полиэдральных графов представляет интерес, так как, с одной стороны, некоторые комбинаторные алгоритмы для таких задач как совершенное паросочетание, покрытие множества, независимое множество, ранжирование объектов, задачи с нечёткими мерами и ряд других основаны на отношении смежности вершин в полиэдральном графе и технике локального поиска (когда от текущего решения переход осуществляется к «лучшему» решению среди смежных) [9—13]. С другой стороны, различные характеристики полиэдрального графа задачи, такие как диаметр и кликовое число (число вершин в наибольшей клике), служат оценками сложности для различных моделей вычислений и классов алгоритмов [14—16].

К сожалению, на пути исследования полиэдрального графа многогранника коммивояжёра встаёт классический результат Пападимитриу.

Теорема 1 (Пападимитриу [17])**.** *Задача проверки несмежности вершин в многогранниках* $\mathrm{STSP}(n)$ *симметричной и* $\mathrm{ATSP}(n)$ *асимметричной задач коммивояжёра является NP-полной.*

Отметим, что дополнительная задача проверки смежности вершин в полиэдральном графе многогранника коммивояжёра будет co-NP-полной.

2. Гамильтоново разложение и достаточное условие несмежности вершин

В связи с результатом Пападимитриу о NP-полноте проверки несмежности вершин в полиэдральном графе многогранника коммивояжёра особый интерес представляют различные достаточные условия несмежности. В частности, известны полиномиально разрешимые достаточные условия для пирамидальных циклов [18], пирамидальных циклов с шагами назад [19] и «родословных» [20, 21]. В данной работе рассматривается наиболее общее из известных – достаточное условие Рао.

Пусть $x = (V, E_x)$ и $y = (V, E_y)$ – два гамильтоновых цикла на множестве вершин V. Обозначим через $x \cup y$ 4-регулярный неориентированный (2-регулярный ориентированный) мультиграф $(V, E_x \cup E_y)$, который содержит копию каждого ребра x и y. Отметим, что если два цикла содержат одно и то же ребро e, то в мультиграф $x \cup y$ добавляются обе копии ребра.

Лемма 1 (Рао [22]). *Пусть x, y – два гамильтоновых цикла. Если мультиграф $x \cup y$ содержит два гамильтоновых цикла z и w без общих рёбер, отличных от x и y, то соответствующие вершины x^v и y^v многогранника коммивояжёра* STSP(n) *(или* ATSP(n)*) несмежны.*

С геометрической точки зрения достаточное условие Рао означает, что отрезок, соединяющий вершины x^v и y^v, пересекается с отрезком, соединяющим две другие вершины z^v и w^v многогранника коммивояжёра, следовательно, они не могут быть смежны. Пример выполненного достаточного условия приведён на Рис. 1.

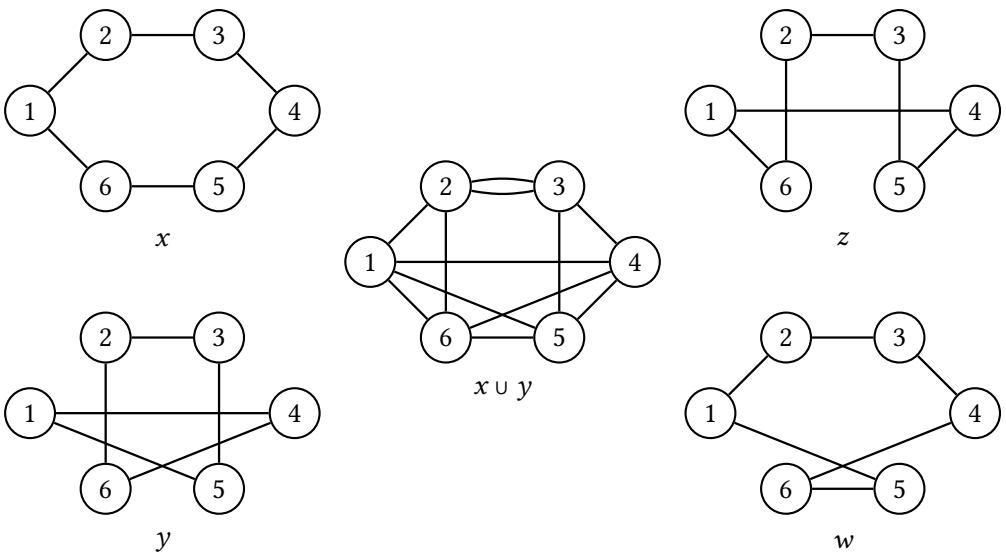

Fig. 1. An example of a satisfied sufficient condition for nonadjacency

Рис. 1. Пример выполненного достаточного условия несмежности

Таким образом, проверка достаточного условия несмежности вершин в полиэдральном графе многогранника коммивояжёра сводится к поиску двух различных гамильтоновых разложений в 4-регулярном мультиграфе $x \cup y$. Сформулируем достаточное условие в виде комбинаторной задачи.

Условие. Заданы два гамильтоновых цикла x и y.

Задача. Найдутся ли в мультиграфе $x \cup y$ два гамильтоновых цикла z и w без общих рёбер, отличные от x и y?

Отметим, что проверка содержит ли произвольный граф гамильтоново разложение является NP-полной задачей уже для 4-регулярных неориентированных мультиграфов и 2-регулярных ориентированных мультиграфов [23].

Ранее задача проверки несмежности вершин в полиэдральном графе коммивояжёра и построения гамильтонова разложения 4-регулярного мультиграфа исследовалась в работах [24, 25], где был предложен ряд эвристических алгоритмов на основе построения покрытия графа циклами без общих вершин: имитация отжига и поиск с переменными окрестностями. Эвристические алгоритмы оказались очень эффективными на экземплярах задачи имеющих решение, особенно на неориентированных графах. Однако на экземплярах задачи не имеющих решения эвристики сталкиваются со значительными затруднениями. В данной работе рассматриваются два точных переборных алгоритма поиска с возвратом построения гамильтонова разложения 4-регулярного мультиграфа.

3. Алгоритм поиск с возвратом на основе построения простого пути

Напомним, что *поиск с возвратом* (*backtracking*) – один из общих методов нахождения решения задачи полным перебором. Процедура заключается в последовательном расширении частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше [26].

В работе [27] был представлен алгоритм поиска с возвратом для задачи построения гамильтонова разложения 4-регулярного мультиграфа на основе построения простого пути. Ниже приводится его модифицированная версия.

Пусть здесь и далее частичное решение состоит из двух компонент z и w. Идея алгоритма заключается в последовательном построении простого пути в компоненте z. При этом рёбра, не попавшие в z, отправляются в компоненту w.

Рассмотрим пример мультиграфа $x \cup y$, приведённый на Рис. 1. Построим для него частичное решение, соответствующее простому пути $2 - 3 - 5 - 6$ (Рис. 2). Здесь сплошными выделены рёбра компоненты z, пунктирными – рёбра компоненты w. Так как степень каждой вершины в мультиграфе $x \cup y$ равна 4, то продолжить простой путь в z из вершины 6 можно не более чем тремя способами: $(6, 1)$, $(6, 2)$ и $(6, 4)$. При этом какое бы ребро мы не выбрали для продолжения пути, так как уже два ребра инцидентных вершине 6 добавлены в z, то два других ребра могут попасть только в компоненту w.

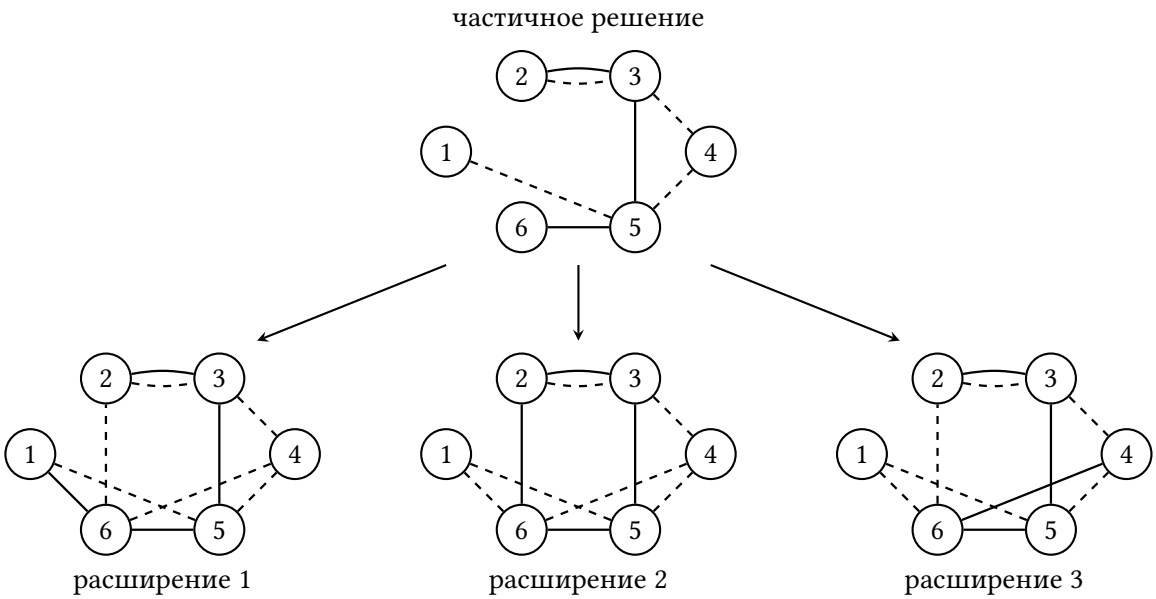

Fig. 2. Backtracking based on simple path expansion

Рис. 2. Поиск с возвратом на основе расширения простого пути

Мы последовательно рассматриваем все три варианта и для каждого проверяем корректность частичного решения:

- расширение 1 недопустимо, так как содержит цикл на вершинах 2, 3, 4, 6 в компоненте w;
- расширение 2 недопустимо, так как содержит цикл на вершинах 2, 3, 5, 6 в компоненте z;
- расширение 3 является корректным.

В общем случае условия корректности частичного решения имеют следующий вид:

- в z содержится простой путь (по построению) или гамильтонов цикл;
- в w содержится лес (граф без циклов) со степенью вершин не более 2 или гамильтонов цикл.

Если вариант расширения частичного решения оказался некорректным, то мы откатываемся на шаг назад и рассматриваем следующий вариант. Если все три варианта расширения недопустимы, то данное частичное решение невозможно расширить и мы также возвращаемся на шаг назад.

Общая схема поиска с возвратом для неориентированных графов приведена в псевдокоде Алгоритма 1.

Алгоритм 1 Поиск с возвратом на основе построения простого пути

```
1:  procedure Backtracking_Simple_Path(z, w, (i, j), x ∪ y)
2:      Добавить ребро (i, j) в z
3:      if Вершине i в z инцидентны 2 ребра then
4:          Добавить свободные ребра инцидентные i в w
5:      end if
6:      if частичное решение z, w не является корректным then
7:          return                                  ▷ Вернуться на шаг назад
8:      end if
9:      if z и w гамильтоновы циклы (отличные от x и y) then
10:         return гамильтоново разложение z и w    ▷ Решение найдено
11:     end if
12:     Отсортировать рёбра (j, k) из вершины j в порядке возрастания свободных степеней k
13:     for каждого свободного ребра (j, k) из вершины j do
14:         z, w ← Backtracking_Simple_Path(z, w, (j, k), x ∪ y)
15:         if гамильтоново разложение найдено then
16:             return гамильтоново разложение z и w
17:         end if
18:     end for
19: end procedure

20: procedure Algorithm_Simple_Path(x ∪ y)
21:     z, w ← ∅
22:     Выбрать начальное ребро (i, j) мультиграфа x ∪ y
23:     z, w ← Backtracking_Simple_Path(z, w, (i, j), x ∪ y)
24:     if z и w найдены then
25:         return Найдено гамильтоново разложение z и w (вершины многогранника несмежны)
26:     end if
27:     return Гамильтоново разложение не найдено (вершины возможно смежны)
28: end procedure
```

Для ориентированных графов алгоритм отличается только тем, что из каждой вершины мультиграфа $x \cup y$ выходит ровно два ребра, следовательно, существует не более двух вариантов расширения простого пути в z. Все остальные шаги полностью аналогичны.

4. Алгоритм поиска с возвратом на основе цепного фиксирования рёбер

Второй алгоритм поиска с возвратом основан на процедуре цепного фиксирования рёбер мультиграфа в компонентах z и w. Алгоритмы для ориентированных и неориентированных графов несколько отличаются, поэтому опишем их по отдельности.

4.1. Ориентированные мультиграфы

Рассмотрим ориентированный 2-регулярный мультиграф $x \cup y$, для которого полустепени захода и исхода каждой вершины равны двум. Выберем некоторое ребро (i, j) и зафиксируем его в компоненте z, тогда второе ребро (i, k), выходящее из i, и второе ребро (h, j), входящее в j, очевидно не могут попасть в z. Мы зафиксируем эти рёбра в w (Рис. 3).

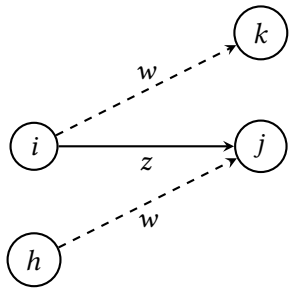

Fig. 3. Fixing the edge (i, j) to z

Рис. 3. Фиксирование ребра (i, j) в z

Основная идея алгоритма заключается в том, что рёбра (i, k) и (h, j), зафиксированные в w, в свою очередь, запускают рекурсивные цепочки фиксирования рёбер в z и т. д.

Рассмотрим в качестве примера ориентированный 2-регулярный мультиграф, приведённый на Рис. 4. Отметим, что мультиграф содержит две копии ребра $(2, 3)$. Одинаковые ребра не могут попасть в один гамильтонов цикл, поэтому зафиксируем по одной копии в z и w. Выберем некоторое ребро, например $(1, 2)$, и зафиксируем его в компоненте z, тогда:

1) ребро $(1, 2)$ зафиксировано в z, следовательно, рёбра $(1, 4)$ и $(6, 2)$ отправляются в w;
2) ребро $(1, 4)$ зафиксировано в w, следовательно, ребро $(3, 4)$ отправляется в z, ребро $(6, 2)$ зафиксировано в w, следовательно, ребро $(6, 1)$ отправляется в z;
3) ребро $(3, 4)$ зафиксировано в z, следовательно, ребро $(3, 5)$ отправляется в w, ребро $(6, 1)$ зафиксировано в z, следовательно, ребро $(5, 1)$ отправляется в w;
4) ребро $(3, 5)$ зафиксировано в w, следовательно, ребро $(4, 5)$ отправляется в z, ребро $(5, 1)$ зафиксировано в w, следовательно, ребро $(5, 6)$ отправляется в z.

На выходе только по одному зафиксированному в z ребру $(1, 2)$ мы получаем однозначно гамильтоновы циклы z и w (Рис. 4). Учитывая, что ребро $(1, 2)$ должно принадлежать хотя бы одному гамильтонову циклу, то данный 2-регулярный ориентированный мультиграф содержит единственное гамильтоново разложение, которое и было найдено.

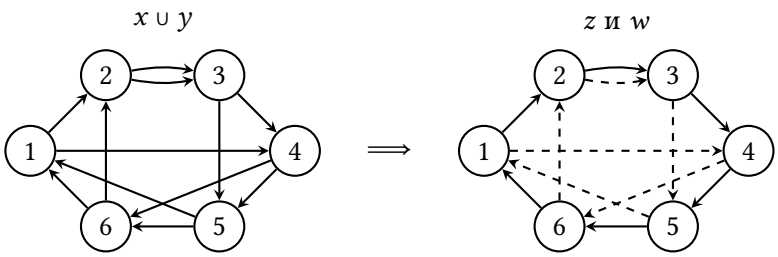

Fig. 4. The result of fixing the edge $(1, 2)$ in z

Рис. 4. Результат фиксирования ребра $(1, 2)$ в z

Общая схема поиска с возвратом для ориентированных мультиграфов приведена в псевдокоде Алгоритма 2.

Алгоритм 2 Поиск с возвратом с цепным фиксированием рёбер для ориентированных графов

```
1:  procedure Chain_Edge_Fixing_Directed((i, j) в z)          ▷ Рекурсивное фиксирование рёбер
2:      Зафиксировать ребро (i, j) в z
3:      if ребро (i, k) не зафиксировано then
4:          Chain_Edge_Fixing_Directed((i, k) в w)
5:      end if
6:      if ребро (h, j) не зафиксировано then
7:          Chain_Edge_Fixing_Directed((h, j) в w)
8:      end if
9:  end procedure

10: procedure Backtracking_Chain_Edge_Fixing_Directed(z, w, (i, j), x ∪ y)
11:     Chain_Edge_Fixing_Directed((i, j) в z)
12:     if z и w – гамильтоновы циклы (отличные от x и y) then
13:         return z и w – решение
14:     end if
15:     if z или w содержит не гамильтонов цикл then          ▷ Частичное решение некорректно
16:         return                                            ▷ Вернуться на шаг назад
17:     end if
18:     Выбрать вершину i′, из которой выходят свободные рёбра
19:     for каждого свободного ребра (i′, j′) из вершины i′ do
20:         z, w ← Backtracking_Chain_Edge_Fixing_Directed(z, w, (i′, j′), x ∪ y)
21:         if гамильтоново разложение найдено then
22:             return гамильтоново разложение z и w
23:         end if
24:     end for
25: end procedure

26: procedure Algorithm_Chain_Edge_Fixing_Directed(x ∪ y)
27:     z, w ← ∅
28:     Найти совпадающие рёбра в x ∪ y и зафиксировать в z и w по одной копии
29:     Выбрать свободное ребро (i, j) мультиграфа x ∪ y
30:     z, w ← Backtracking_Chain_Edge_Fixing_Directed(z, w, (i, j), x ∪ y)
31:     if z и w найдены then
32:         return Найдено гамильтоново разложение z и w (вершины многогранника несмежны)
33:     end if
34:     return Гамильтоново разложение не найдено (вершины возможно смежны)
35: end procedure
```

На шаге предварительной обработки данных (строка 28, Алгоритм 2) мы находим все совпадающие рёбра в мультиграфе $x \cup y$ и фиксируем в z и w по одной копии, так как эти рёбра не могут попасть в один гамильтонов цикл.

Частичное решение считается корректным, если компоненты z и w представляют собой ориентированные ациклические графы с полустепенями захода и исхода не более одного или ориентированные гамильтоновы циклы (контуры). По построению, полуступени вершин в z и w не

могут быть равны двум, так что для проверки корректности достаточно убедиться в отсутствии не гамильтоновых циклов.

В отличие от алгоритма поиска с возвратом на основе построения простого пути (Алгоритм 1) вершину i' для построения расширения частичного решения (строка 18, Алгоритм 2) имеет смысл выбирать так, чтобы цепочки фиксирования запускались с обоих концов ребра (i', j'). Чем больше рёбер будет зафиксировано на одном шаге, тем меньше окажется глубина рекурсии.

Отметим, что мы не рассматриваем различные варианты при выборе и фиксировании первого свободного ребра (i, j) (строка 29, Алгоритм 2), так как это ребро должно попасть в одну из компонент решения. Назовём компонентой z ту, что содержит ребро (i, j).

Следует также отметить, что хотя процедура Chain_Edge_Fixing_Directed (строки 1-9, Алгоритм 2) рекурсивного фиксирования рёбер на каждом шаге вызывает до двух своих подзадач, общая трудоёмкость линейна ($O(V)$), так как каждое ребро может быть зафиксировано не более одного раза, и $|E| = 2|V|$.

4.2. Неориентированные мультиграфы

Псевдокод алгоритма для неориентированных мультиграфов приведён в Алгоритме 3.

Алгоритм 3 Поиск с возвратом с цепным фиксированием рёбер для неориентированных графов

```
1:  procedure Chain_Edge_Fixing_Undirected((i, j) в z)
2:      Зафиксировать ребро (i, j) в z
3:      if в z вершине i инцидентны 2 зафиксированных ребра then
4:          Chain_Edge_Fixing_Undirected((i, k) в w)        ▷ Зафиксировать два других ребра в w
5:          Chain_Edge_Fixing_Undirected((i, h) в w)
6:      end if
7:      if в z вершине j инцидентны 2 зафиксированных ребра then
8:          Chain_Edge_Fixing_Undirected((j, k) в w)        ▷ Зафиксировать два других ребра в w
9:          Chain_Edge_Fixing_Undirected((j, h) в w)
10:     end if
11: end procedure

12: procedure Backtracking_Chain_Edge_Fixing_Undirected(z, w, (i, j), x ∪ y)
13:     Chain_Edge_Fixing_Undirected((i, j) в z)
14:     if z и w – гамильтоновы циклы (отличные от x и y) then
15:         return z и w – решение
16:     end if
17:     if z или w содержит не гамильтонов цикл then          ▷ Частичное решение некорректно
18:         return                                            ▷ Вернуться на шаг назад
19:     end if
20:     Выбрать вершину i с минимальной степенью d свободных рёбер
21:     Отсортировать рёбра (i, j_k) из вершины i в порядке возрастания свободных степеней j_k
22:     for k ← 1 to d do
23:         z, w ←Backtracking_Chain_Edge_Fixing_Undirected(z, w, (i, j_k), x ∪ y)
24:         if гамильтоново разложение найдено then
25:             return гамильтоново разложение z и w
26:         end if
27:     end for
28: end procedure
```

Процедура фиксирования рёбер срабатывает, как только в одном из частичных решений вершине инцидентны 2 ребра, тогда два других ребра отправляется в другое частичное решение (строки 3-10, Алгоритм 3).

Частичное решение считается корректным, если компоненты z и w представляют собой леса со степенью вершин не более двух или гамильтоновы циклы. По построению, мы добавляем в z и w по одному ребру за раз. Причём, как только степень какой-то вершины в одной из компонент становится равной двум, то два оставшихся ребра отправляются в другую компоненту. Таким образом, для проверки корректности частичного решения достаточно убедиться, что z и w не содержат не гамильтоновых циклов.

Отметим, что на каждом шаге для расширения частичного решения мы выбираем вершину i с минимальной степенью по свободным рёбрам с целью уменьшить ветвление рекурсии. Выходящие из i рёбра рассматриваются в порядке возрастания степеней смежных вершин по свободным рёбрам (строки 20-27, Алгоритм 3).

4.3. Смежность вершин и предварительная обработка данных

Отметим, что рассматриваемые в данной статье алгоритмы разработаны непосредственно для задачи построения гамильтонова разложения 4-регулярного мультиграфа $x \cup y$. Алгоритмы обращаются напрямую к циклам x и y только при проверке, что найденное разложение z и w отлично от исходного. Если опустить эту проверку, то мы получим алгоритмы для построения гамильтонова разложения без какой-либо привязки к полиэдральной комбинаторике. В противном случае, если нас интересует именно смежность вершин в полиэдральном графе многогранника коммивояжёра, то рассматриваемые алгоритмы имеет смысл усилить, добавив шаг предварительной обработки данных, на котором будут проверены известные полиномиальные достаточные условия несмежности вершин: пирамидальные циклы [18], пирамидальные циклы с шагами назад [19], «родословные» [20, 21] и т. д.

5. Вычислительные эксперименты

Алгоритмы тестировались на случайных ориентированных и неориентированных гамильтоновых циклах. Для каждого размера графа с помощью алгоритма тасования Фишера-Йетса [28] были сгенерированы 100 пар случайных перестановок с равномерным распределением вероятности.

Для сравнения были выбраны 3 алгоритма:

- поиск с возвратом на основе построения простого пути (BSP);
- поиск с возвратом на основе цепного фиксирования рёбер (BCEF);
- эвристический алгоритм поиска с переменными окрестностями (GVNS) из работы [25], который является модификацией алгоритма имитации отжига [24] и работает на основе построения покрытия циклами без общих вершин через поиск совершенного паросочетания и нескольких операций объединения циклов.

Алгоритмы поиска с возвратом реализованы на Python, для эвристического алгоритма поиска с переменными окрестностями взята готовая реализация на Node.js [25]. Вычислительные эксперименты проведены на машине с Intel(R) Core(TM) i5-4460 с CPU 3.20GHz и 16GB RAM.

Результаты вычислительных экспериментов для неориентированных графов приведены в Таблице 1 и графике на Рис. 5. Результаты для ориентированных графов представлены в Таблице 2 и графиках на Рис. 6 и Рис. 7.

Для каждого набора из 100 тестов приведено среднее время работы алгоритмов в секундах отдельно по задачам имеющим решение и по задачам не имеющим решения. Для двух алгоритмов поиска с возвратом (BSP и BCEF) было установлено ограничение по времени в 2 часа на каждый набор из 100 тестовых задач. Соответственно, в таблицах указано, сколько задач из 100 алгоритм успел решить за 2 часа. Для эвристического алгоритма поиска с переменными окресностями (GVNS)

Table 1. Computational results for 100 random undirected Hamiltonian cycles

Таблица 1. Вычислительные результаты для 100 неориентированных гамильтоновых циклов

	BSP				BCEF				GVNS			
	Есть решение		Нет решения		Есть решение		Нет решения		Есть решение		«Нет решения»	
|V|	N	время (с)	N	время (с)	N	время (с)	N	время (с)	N	время (с)	N	время (с)
32	100	0.030	–	–	100	0.015	–	–	100	0.001	–	–
48	100	0.151	–	–	100	0.032	–	–	100	0.002	–	–
64	100	0.154	–	–	100	0.052	–	–	100	0.003	–	–
96	100	0.908	–	–	100	0.09	–	–	100	0.007	–	–
128	100	3.045	–	–	100	0.157	–	–	100	0.012	–	–
192	100	4.257	–	–	100	0.22	–	–	100	0.023	–	–
256	100	19.625	–	–	100	0.487	–	–	100	0.033	–	–
384	9	80.037	–	–	100	1.278	–	–	100	0.079	–	–
512	0	–	–	–	100	2.447	–	–	100	0.121	–	–
768	0	–	–	–	100	3.591	–	–	100	0.272	–	–
1024	0	–	–	–	100	6.305	–	–	100	0.468	–	–
1536	0	–	–	–	100	8.799	–	–	100	1.011	–	–
2048	0	–	–	–	100	18.499	–	–	100	1.821	–	–
3072	0	–	–	–	100	45.542	–	–	100	3.852	–	–
4096	0	–	–	–	86	82.973	–	–	100	7.768	–	–

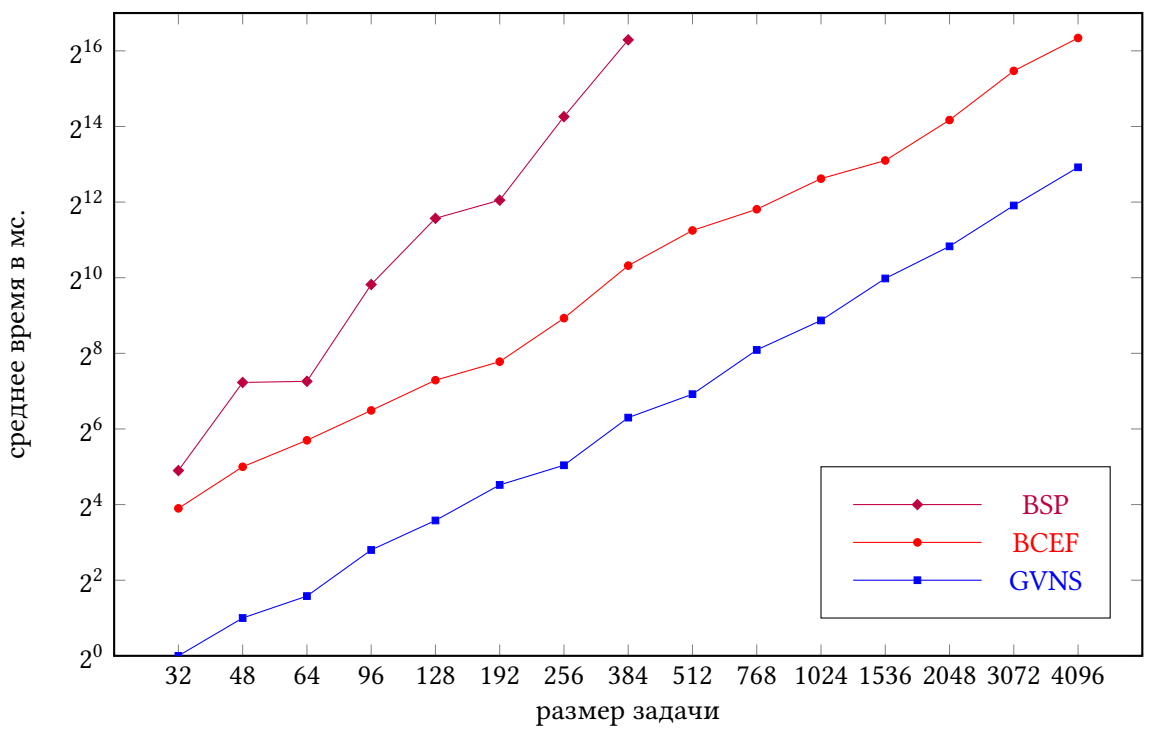

Fig. 5. Computational results for undirected graphs

Рис. 5. Вычислительные результаты для неориентированных графов

Table 2. Computational results for 100 random directed Hamiltonian cycles

Таблица 2. Вычислительные результаты для 100 ориентированных гамильтоновых циклов

	BSP				BCEF				GVNS			
	Есть решение		Нет решения		Есть решение		Нет решения		Есть решение		«Нет решения»	
|V|	N	время (с)	N	время (с)	N	время (с)	N	время (с)	N	время (с)	N	время (с)
32	30	0.069	70	0.592	30	0.006	70	0.011	30	0.001	70	0.248
48	20	1.838	80	17.005	20	0.011	80	0.023	20	0.002	80	0.440
64	8	36.850	17	403.016	20	0.026	80	0.044	20	0.002	80	0.714
96	0	–	0	–	19	0.032	81	0.086	19	0.005	81	1.330
128	–	–	–	–	18	0.055	82	0.141	18	0.009	82	2.312
192	–	–	–	–	21	0.059	79	0.262	21	0.014	79	4.569
256	–	–	–	–	25	0.134	75	0.628	25	0.093	75	9.120
384	–	–	–	–	20	0.083	80	1.646	20	0.104	80	19.215
512	–	–	–	–	22	0.143	78	2.131	22	0.112	78	29.048
768	–	–	–	–	19	0.426	81	2.241	19	0.404	81	55.991
1024	–	–	–	–	17	0.847	83	10.545	17	1.679	83	60.000
1536	–	–	–	–	16	0.369	84	7.844	16	0.943	84	60.000
2048	–	–	–	–	15	1.651	85	16.179	14	1.987	86	60.000
3072	–	–	–	–	21	1.059	79	46.589	21	2.999	79	60.000
4096	–	–	–	–	17	1.551	78	91.963	18	4.846	82	60.000

было установлено ограничение в 1 минуту на каждый тест. Это обусловлено тем, что эвристический алгоритм обладает односторонней ошибкой. Если алгоритм находит решение, то решение существует. Однако, эвристический алгоритм не может гарантировать, что решения задачи не существует, лишь, что решение не найдено за заданное время или число итераций.

Известно, что случайные неориентированные регулярные графы обладают гамильтоновым разложением с очень высокой вероятностью [29]. Действительно, для всех тестовых задач на неориентированных мультиграфах (Таблица 1) существовало гамильтоново разложение на циклы, отличные от исходных, и вершины многогранника коммивояжёра были не смежны. С геометрической точки зрения это означает, что степени вершин в полиэдральном графе значительно меньше общего числа вершин, поэтому две случайные вершины с высокой вероятностью оказываются не смежными.

По результатам тестирования для неориентированных гамильтоновых циклов оба алгоритма поиска с возвратом проиграли эвристическому алгоритму поиска с переменными окрестностями (GVNS). За отведённое время алгоритм на основе построения простого пути (BSP) решил 709 задач, алгоритм на основе цепного фиксирования рёбер (BCEF) – 1486 задач, и поиск с переменными окрестностями (GVNS) решил все 1500 задач из 1500. По времени работы на решённых задачах GVNS оказался в среднем в 13 раз быстрее чем BCEF и в 640 раз быстрее чем BSP. Среди двух алгоритмов поиска с возвратом, алгоритм на основе цепного фиксирования рёбер (BCEF) оказался значительно эффективнее, решив все тестовые задачи до 3072 вершин и показав время работы на решённых задачах в среднем в 48 раз быстрее, чем поиск на основе простого пути (BSP).

Результаты вычислительных экспериментов для ориентированных циклов (Таблица 2, Рис. 6 и 7) оказались принципиально иными. Только примерно 20% случайных тестовых задач имели решение. Отметим, что это не означает, что соответствующие вершины многогранника коммивояжёра смежны, так как задача построения гамильтонова разложения проверяет лишь достаточное условие несмежности вершин в полиэдральном графе.

Худшие результаты из трёх рассмотренных алгоритмов показал поиск с возвратом на основе построения простого пути (BSP). За отведённое на тест время алгоритм решил всего 225 тестовых задач из 1500, не решив ни одной задачи для графов на более чем 64 вершинах. В то время как алгоритм цепного фиксирования рёбер (BCEF) решил 1495 задач, а поиск с переменными

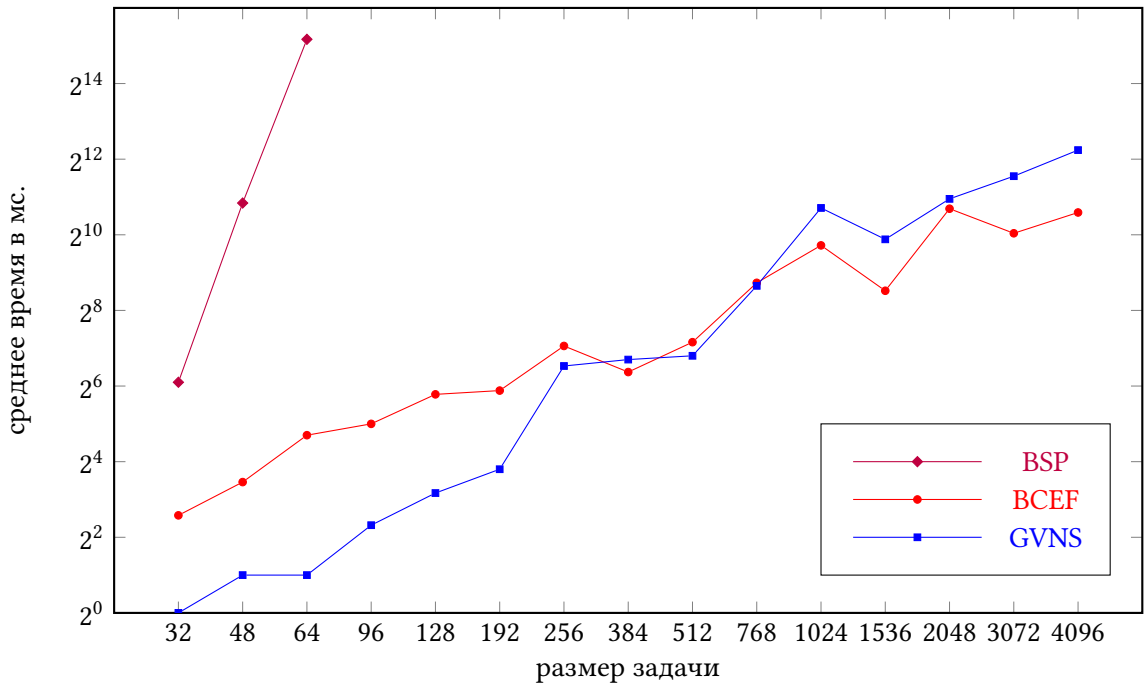

Fig. 6. Computational results for directed graphs with solution

Рис. 6. Вычислительные результаты для ориентированных графов в задачах с решением

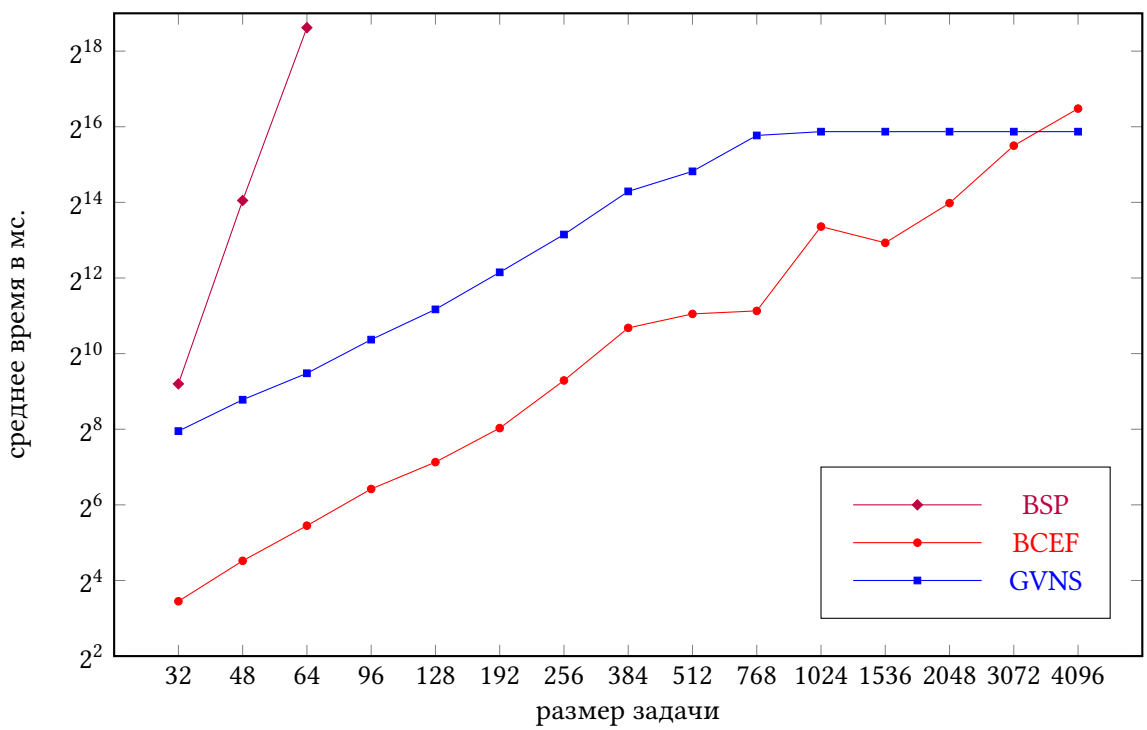

Fig. 7. Computational results for directed graphs without solution

Рис. 7. Вычислительные результаты для ориентированных графов в задачах без решения

окрестностями (GVNS) справился со всеми тестовыми задачами. По времени работы на задачах, имеющих решение, эвристический алгоритм GVNS показал преимущество над BCEF на небольших графах до 512 вершин в среднем в 6 раз. Однако, на графах с более 512 вершинами уже BCEF оказался в среднем в 2 раза быстрее, чем GVNS. В целом, на задачах, для которых решение существует, алгоритмы BCEF и GVNS показали сопоставимые результаты. Некоторый разброс производительности может быть обусловлен разными реализациями алгоритмов. Тем не менее, отметим, что при увеличении размера графа преимущество BCEF над GVNS нарастало. С другой стороны, эвристический алгоритм поиска с переменными окрестностями (GVNS) столкнулся со значительными затруднениями на задачах не имеющих решений. Алгоритм не может определить этот сценарий и выходит лишь при достижении ограничения на время работы или число итераций. На подобных задачах алгоритм на основе цепного фиксирования рёбер (BCEF) оказался в среднем в 16 раз быстрее, чем GVNS. Отчасти это означает, что порог итераций для эвристического алгоритма можно было бы понизить. Однако в таком случае возникла бы опасность потерять существующие решения.

Заключение

В работе были рассмотрены два алгоритма поиска с возвратом для задачи построения гамильтонова разложения 4-регулярного мультиграфа. По результатам вычислительных экспериментов на случайных ориентированных и неориентированных мультиграфах алгоритм поиска с возвратом на основе цепного фиксирования рёбер оказался значительно эффективнее, чем поиск с возвратом на основе построения простого пути. Кроме того, на ориентированных мультиграфах алгоритм цепного фиксирования рёбер показал сопоставимые результаты с ранее известным эвристическим алгоритмом поиска с переменными окрестностями на тестовых задачах, имеющих решение, и значительно превзошёл эвристику на задачах, для которых решения не существует.

Рассматриваемые алгоритмы поиска с возвратом были разработаны для задачи проверки несмежности вершин в полиэдральном графе многогранника коммивояжёра. Однако они также могут быть применены непосредственно к задаче построения гамильтонова разложения регулярного мультиграфа и множеству её приложений.

References

[1] J. Krarup, “The peripatetic salesman and some related unsolved problems”, in *Combinatorial Programming: Methods and Applications*, vol. 19, Springer Netherlands, 1995, pp. 173–178. DOI: 10.1007/978-94-011-7557-9_8.

[2] M. M. Bae and B. Bose, “Edge disjoint Hamiltonian cycles in k-ary n-cubes and hypercubes”, *IEEE Transactions on Computers*, vol. 52, no. 10, pp. 1271–1284, 2003. DOI: 10.1109/TC.2003.1234525.

[3] R. F. Bailey, “Error-correcting codes from permutation groups”, *Discrete Mathematics*, vol. 309, no. 13, pp. 4253–4265, 2009. DOI: 10.1016/j.disc.2008.12.027.

[4] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu, “Tools for Privacy Preserving Distributed Data Mining”, *SIGKDD Explor. Newsl.*, vol. 4, no. 2, pp. 28–34, 2002. DOI: 10.1145/772862.772867.

[5] R. W. Hung, “Embedding two edge-disjoint Hamiltonian cycles into locally twisted cubes”, *Theoretical Computer Science*, vol. 412, no. 35, pp. 4747–4753, 2011. DOI: 10.1016/j.tcs.2011.05.004.

[6] R. Glebov, Z. Luria, and B. Sudakov, “The number of Hamiltonian decompositions of regular graphs”, *Israel Journal of Mathematics*, vol. 222, no. 1, pp. 91–108, 2017. DOI: 10.1007/s11856-017-1583-y.

[7] G. Dantzig, R. Fulkerson, and S. Johnson, “Solution of a Large-Scale Traveling-Salesman Problem”, *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954. DOI: 10.1287/opre.2.4.393.

[8] D. L. Applegate, R. E. Bixby, V. Chvatál, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006, ISBN: 9780691129938.

[9] N. E. Aguilera, R. D. Katz, and P. B. Tolomei, "Vertex adjacencies in the set covering polyhedron", *Discrete Applied Mathematics*, vol. 218, pp. 40–56, 2017. DOI: 10.1016/j.dam.2016.10.024.

[10] M. L. Balinski, "Signature Methods for the Assignment Problem", *Operations Research*, vol. 33, no. 3, pp. 527–536, 1985. DOI: 10.1287/opre.33.3.527.

[11] C. R. Chegireddy and H. W. Hamacher, "Algorithms for finding K-best perfect matchings", *Discrete Applied Mathematics*, vol. 18, no. 2, pp. 155–165, 1987. DOI: 10.1016/0166-218X(87)90017-5.

[12] E. F. Combarro and P. Miranda, "Adjacency on the order polytope with applications to the theory of fuzzy measures", *Fuzzy Sets and Systems*, vol. 161, no. 5, pp. 619–641, 2010. DOI: 10.1016/j.fss.2009.05.004.

[13] H. N. Gabow, "Two Algorithms for Generating Weighted Spanning Trees in Order", *SIAM Journal on Computing*, vol. 6, no. 1, pp. 139–150, 1977. DOI: 10.1137/0206011.

[14] V. A. Bondarenko, "Nonpolynomial lower bounds for the complexity of the traveling salesman problem in a class of algorithms", *Automation and Remote Control*, vol. 44, no. 9, pp. 1137–1142, 1983.

[15] V. Bondarenko and A. Nikolaev, "On Graphs of the Cone Decompositions for the Min-Cut and Max-Cut Problems", *International Journal of Mathematics and Mathematical Sciences*, vol. 2016, p. 7 863 650, 2016. DOI: 10.1155/2016/7863650.

[16] M. Grötschel and M. Padberg, "Polyhedral theory", in *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley, Chichester, 1985, pp. 251–305.

[17] C. H. Papadimitriou, "The adjacency relation on the traveling salesman polytope is NP-Complete", *Mathematical Programming*, vol. 14, no. 1, pp. 312–324, 1978. DOI: 10.1007/BF01588973.

[18] V. A. Bondarenko and A. V. Nikolaev, "On the Skeleton of the Polytope of Pyramidal Tours", *Journal of Applied and Industrial Mathematics*, vol. 12, no. 1, pp. 9–18, 2018. DOI: 10.1134/S1990478918010027.

[19] A. Nikolaev, "On vertex adjacencies in the polytope of pyramidal tours with step-backs", in *Mathematical Optimization Theory and Operations Research. MOTOR 2019*, ser. LNCS, vol. 11548, Springer, 2019, pp. 247–263. DOI: 10.1007/978-3-030-22629-9_18.

[20] T. S. Arthanari, "On pedigree polytopes and Hamiltonian cycles", *Discrete Mathematics*, vol. 306, no. 14, pp. 1474–1492, 2006. DOI: 10.1016/j.disc.2005.11.030.

[21] T. S. Arthanari, "Study of the pedigree polytope and a sufficiency condition for nonadjacency in the tour polytope", *Discrete Optimization*, vol. 10, no. 3, pp. 224–232, 2013. DOI: 10.1016/j.disopt.2013.07.001.

[22] M. R. Rao, "Adjacency of the Traveling Salesman Tours and 0 – 1 Vertices", *SIAM Journal on Applied Mathematics*, vol. 30, no. 2, pp. 191–198, 1976. DOI: 10.1137/0130021.

[23] B. Péroche, "NP-completeness of some problems of partitioning and covering in graphs", *Discrete Applied Mathematics*, vol. 8, no. 2, pp. 195–208, 1984. DOI: 10.1016/0166-218X(84)90101-X.

[24] A. Kozlova and A. Nikolaev, "Simulated annealing approach to verify vertex adjacencies in the traveling salesperson polytope", in *Mathematical Optimization Theory and Operations Research. MOTOR 2019*, ser. LNCS, vol. 11548, Springer, 2019, pp. 374–389. DOI: 10.1007/978-3-030-22629-9_26.

[25] A. Nikolaev and A. Kozlova, "Hamiltonian decomposition and verifying vertex adjacency in 1-skeleton of the traveling salesperson polytope by variable neighborhood search", *Journal of Combinatorial Optimization*, 2020. DOI: 10.1007/s10878-020-00652-7.

[26] S. S. Skiena, *The Algorithm Design Manual*. Springer, 2008. DOI: 10.1007/978-1-84800-070-4.

[27] A. V. Korostil and A. V. Nikolaev, "Algoritm poiska s vozvratom dlya postroeniya gamil'tonova razlozheniya 4-regulyarnogo mul'tigrafa (Backtracking algorithm to construct the Hamiltonian decomposition of a 4-regular multigraph)", in *Zametki po informatike i matematike (Notes on Computer Science and Mathematics)*, vol. 12, P.G. Demidov Yaroslavl State University, 2020, pp. 91–97.

[28] D. E. Knuth, *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. USA: Addison-Wesley Longman Publishing Co., Inc., 1997, ISBN: 0201896842. DOI: 10.5555/270146.

[29] J. H. Kim and N. C. Wormald, "Random Matchings Which Induce Hamilton Cycles and Hamiltonian Decompositions of Random Regular Graphs", *Journal of Combinatorial Theory, Series B*, vol. 81, no. 1, pp. 20–44, 2001. DOI: 10.1006/jctb.2000.1991.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 1, 2021
journal homepage: www.mais-journal.ru

ALGORITHMS

NP-completeness of the Minimum Spanning Tree Problem of a Multiple Graph of Multiplicity $k \geqslant 3$

A. V. Smirnov[1]

DOI: 10.18255/1818-1015-2021-1-22-37

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya, Yaroslavl 150003, Russia.

MSC2020: 05C05, 05C65
Research article
Full text in Russian

Received March 1, 2021
After revision March 10, 2021
Accepted March 12, 2021

In this paper, we study undirected multiple graphs of any natural multiplicity $k > 1$. There are edges of three types: ordinary edges, multiple edges and multi-edges. Each edge of the last two types is a union of k linked edges, which connect 2 or $(k+1)$ vertices correspondingly. The linked edges should be used simultaneously. If a vertex is incident to a multiple edge, it can be also incident to other multiple edges and it can be the common end of k linked edges of some multi-edge. If a vertex is the common end of some multi-edge, it cannot be the common end of another multi-edge.
A multiple tree is a connected multiple graph with no cycles. Unlike ordinary trees, the number of edges in a multiple tree is not fixed. The problem of finding the spanning tree can be set for a multiple graph. Complete spanning trees form a special class of spanning trees of a multiple graph. Their peculiarity is that a multiple path joining any two selected vertices exists in the tree if and only if such a path exists in the initial graph. If the multiple graph is weighted, the minimum spanning tree problem and the minimum complete spanning tree problem can be set. Also we can formulate the problems of recognition of the spanning tree and complete spanning tree of the limited weight. The main result of this article is the proof of NP-completeness of such recognition problems for arbitrary multiple graphs as well as for divisible multiple graphs in the case when multiplicity $k \geqslant 3$. The corresponding optimization problems are NP-hard.

Keywords: multiple graph, multiple tree, divisible graph, spanning tree, complete spanning tree, minimum spanning tree, NP-completeness

INFORMATION ABOUT THE AUTHORS

Alexander Valeryevich Smirnov
correspondence author

orcid.org/0000-0002-0980-2507. E-mail: alexander_sm@mail.ru
PhD, Associate Professor, Department of Theoretical Computer Science.

Funding: This work was supported by P. G. Demidov Yaroslavl State University Project № VIP-016.

For citation: A. V. Smirnov, "NP-completeness of the Minimum Spanning Tree Problem of a Multiple Graph of Multiplicity $k \geqslant 3$", *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 22-37, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 1, 2021
сайт журнала: www.mais-journal.ru

NP-полнота задачи о минимальном остовном дереве в кратном графе кратности $k \geqslant 3$

А. В. Смирнов[1]

DOI: 10.18255/1818-1015-2021-1-22-37

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, г. Ярославль, 150003 Россия.

УДК 519.17, 519.161
Научная статья
Полный текст на русском языке

Получена 1 марта 2021 г.
После доработки 10 марта 2021 г.
Принята к публикации 12 марта 2021 г.

В статье рассматриваются неориентированные кратные графы произвольной натуральной кратности $k > 1$. Кратный граф содержит ребра трех типов: обычные, кратные и мультиребра. Ребра последних двух типов представляют собой объединение k связанных ребер, которые соединяют 2 или $(k + 1)$ вершину соответственно. Связанные ребра могут использоваться только согласованно. Если вершина инцидентна кратному ребру, то она может быть инцидентна другим кратным ребрам, а также она может быть общим концом k связанных ребер мультиребра. Если вершина является общим концом мультиребра, то она не может быть общим концом никакого другого мультиребра.

Кратное дерево определяется как связный кратный граф без циклов. В отличие от обычных деревьев количество ребер в кратных деревьях не фиксировано. Для кратного графа можно поставить задачу поиска его остовного дерева. Среди всех остовных деревьев в кратном графе выделяется особый класс – полные остовные деревья. Кратный путь между любой парой вершин в таком дереве существует тогда и только тогда, когда кратный путь между ними существует в исходном графе. Если кратный граф является взвешенным, то для него можно поставить задачу о минимальном остовном дереве, а также о минимальном полном остовном дереве. Кроме того, можно сформулировать задачи распознавания остовного и полного остовного дерева ограниченного веса. Основным результатом данной статьи является доказательство того, что указанные задачи распознавания являются NP-полными как в случае произвольных, так и в случае делимых кратных графов, если кратность $k \geqslant 3$. Соответствующие оптимизационные задачи являются NP-трудными.

Ключевые слова: кратный граф, кратное дерево, делимый граф, остовное дерево, полное остовное дерево, минимальное остовное дерево, NP-полнота

ИНФОРМАЦИЯ ОБ АВТОРАХ

Александр Валерьевич Смирнов
автор для корреспонденции

orcid.org/0000-0002-0980-2507. E-mail: alexander_sm@mail.ru
канд. физ.-мат. наук, доцент, кафедра теоретической информатики.

Финансирование: Работа выполнена в рамках инициативной НИР ЯрГУ им. П. Г. Демидова № VIP-016.

Для цитирования: A. V. Smirnov, “NP-completeness of the Minimum Spanning Tree Problem of a Multiple Graph of Multiplicity $k \geqslant 3$”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 22-37, 2021.

Введение

В данной статье мы рассмотрим задачу о *минимальном остовном дереве в кратном графе* в различных вариантах ее постановки. Определение кратного графа кратности $k > 1$ и делимого кратного графа было сформулировано в статье [1]. Там же была рассмотрена задача о кратчайшем кратном пути между двумя вершинами. Кратные графы содержат три типа ребер (обычные, кратные и мультиребра) и являются обобщением обычных графов – по сути, обычный граф имеет кратность $k = 1$. Понятия остовного и полного остовного дерева в кратном графе были введены в статье [2]. Там же приведена постановка задачи о минимальном остовном дереве в кратном графе и о минимальном полном остовном дереве, предложен эвристический алгоритм решения второй задачи, который обобщает известный алгоритм Краскала (см. [3]).

Среди других известных обобщений графов наиболее близкими нам концепциями являются мультиграфы, гиперграфы (см., например, [4, 5]), а также метаграфы (см. [6, 7]). Действительно, как и в мультиграфах, в кратных графах допускается наличие нескольких ребер между парой вершин (набор таких ребер мы будем в дальнейшем называть *кратным ребром*), однако в случае кратного графа количество таких ребер должно быть строго равным k. В кратных графах присутствуют *мультиребра*, соединяющие между собой $(k + 1)$ вершину. Но в отличие от гиперребер гиперграфа, мультиребро представляется в виде k связанных ребер, имеющих один общий конец, причем все эти k ребер должны использоваться согласованно. По сути, понятие мультиребра близко понятию ребра между вершиной и метавершиной в метаграфе. При этом в метаграфе, напомним, метапуть между двумя метавершинами фактически моделирует причинно-следственные связи в некоторой предметной области. Однако в кратном графе используется принципиально иной подход к определению пути: *кратный путь* должен состоять ровно из k обычных путей, проходящих по обычным ребрам, а также по связанным ребрам кратных и мультиребер; при этом пути должны быть согласованы (одинаковы) на кратных и мультиребрах. Поэтому кратный граф нельзя считать частным случаем метаграфа.

Отметим также, что частным случаем кратного графа является кратная сеть (см. [8, 9]). Задача о наибольшем потоке в кратной сети обобщает классическую задачу (см. [10]) и имеет ряд приложений в сфере экономики, управления, финансов. В частности, кратные сети и потоки используются для поиска решения NP-полной задачи целочисленного сбалансирования трех- и четырехмерной матрицы (см., например, [11, 12]).

В данной статье мы рассмотрим сложностной аспект задачи о минимальном остовном дереве в кратном графе и докажем, что при кратности графа $k \geqslant 3$ соответствующая задача распознавания является NP-полной, а исходная оптимизационная задача – NP-трудная. Этот результат будет справедлив как для полных, так и для произвольных остовных деревьев.

1. Кратные графы и деревья. Необходимые определения

Напомним несколько определений, связанных с кратными графами и деревьями, которые ранее были сформулированы в статьях [1, 2].

Определение 1. Кратный граф G *произвольной натуральной кратности* $k > 1$ *– это граф, вершины которого могут соединяться ребрами одного из 3 видов:*

1. Обычное ребро e^o*; множество обычных ребер обозначим через* E^o.
2. Кратное ребро e^k *между двумя вершинами, которое состоит из* k *одинаковых связанных ребер; связанные ребра кратного ребра могут использоваться только согласованно; множество кратных ребер обозначим через* E^k.
3. Связанное ребро e *между двумя вершинами, имеющее один общий конец с другим* $(k - 1)$ *ребром (у любых двух из* k *связанных ребер только один конец является общим); множество связанных*

общей вершиной ребер будем называть мультиребром e^m*; связанные ребра мультиребра могут использоваться только согласованно; множество мультиребер обозначим через* E^m.

Если вершина инцидентна какому-либо кратному ребру, то она может быть инцидентна другим кратным ребрам, а также она может быть общим концом какого-либо мультиребра.

Если вершина является общим концом какого-либо мультиребра, то она не может быть общим концом никакого другого мультиребра.

Если вершина является отдельным концом мультиребра или инцидентна обычному ребру, то она не может быть общим концом мультиребра и не может быть инцидентна кратному ребру.

Множества вершин и ребер графа G обозначим через V и E соответственно. Заметим, что $E = E^o \cup E^k \cup E^m$.

В данной статье рассматриваются только неориентированные кратные графы.

Рис. 1 и 2 иллюстрируют определение 1. В левой части рис. 1 кратное ребро представлено в виде объединения k одинаковых ребер между двумя вершинами, что показано штрихами. Равенство (или согласованность) связанных ребер предполагает, что все характеристики этих ребер (например, вес) одинаковы, и эти ребра могут использоваться только одновременно. Так, если осуществляется проход в определенном направлении по одному из связанных ребер, то одновременно с этим все остальные ребра проходятся в том же самом направлении. Кратное ребро может включаться в какие-либо новые структуры только целиком. В дальнейшем мы будем обозначать кратные ребра жирными линиями, как в правой части рис. 1.

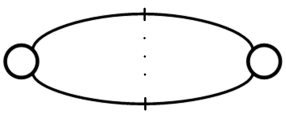

Fig. 1. Multiple edge

Рис. 1. Кратное ребро

В левой части рис. 2 мультиребро $\{x_0, \{x_1, \ldots, x_k\}\}$ представлено в виде объединения k одинаковых ребер, связывающих общую вершину x_0 с k разными вершинами $x_1, \ldots, x_k$. Как и на рис. 1, равенство ребер показано штрихами. Согласованность связанных ребер имеет тот же смысл, что и для кратных ребер. В дальнейшем мультиребра мы будем изображать при помощи расщепляющихся на k частей линий, как в правой части рис. 2.

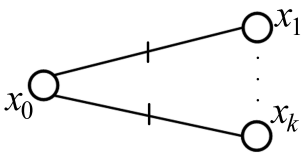

Fig. 2. Multi-edge

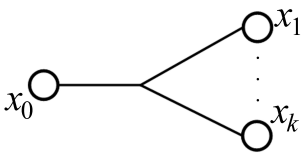

Рис. 2. Мультиребро

Определение 2. Обычной вершиной *назовем вершину, которая инцидентна обычному ребру или является отдельным концом мультиребра.*

Кратной вершиной *назовем вершину, которая инцидентна кратному ребру или является общим концом мультиребра.*

Из определения 1 следует, что множества обычных и кратных вершин не пересекаются. При этом кратная вершина может быть соединена с обычными только посредством мультиребра.

Определение 3. Делимым кратным графом *назовем такой граф, в котором между двумя концами одного мультиребра не существует пути, проходящего только по обычным ребрам.*

При удалении всех мультиребер делимый граф распадется на n компонент связности (связность здесь понимается в том же смысле, что и для обычных графов), каждая из которых содержит только кратные ребра либо только обычные ребра. При этом связанные ребра каждого мультиребра можно пронумеровать от 1 до k таким образом, что каждой компоненте связности, содержащей только обычные ребра, будут инцидентны связанные ребра мультиребер с одинаковыми номерами.

Определение 4. Частью G_i ($i \in \overline{1,k}$) *делимого графа* $G(V,E)$ *назовем подграф, содержащий связанные ребра с номером i всех кратных и мультиребер, а также компоненты связности, состоящие из обычных ребер и инцидентные i-ым связанным ребрам всех мультиребер.*

Каждая часть G_i является обычным графом. При этом возможность выделения частей G_i является особенностью делимых графов. В общем случае получить части G_i не удастся.

Пример 1. Рассмотрим кратный граф кратности 2, представленный на рис. 3. Этот граф содержит 3 обычных ребра, 3 кратных ребра и 4 мультиребра:

$$E^o = \left\{ \{x_3, x_4\}, \quad \{x_5, x_6\}, \quad \{x_9, x_{10}\} \right\},$$

$$E^k = \left\{ \{x_1, x_2\}, \quad \{x_7, x_8\}, \quad \{x_{11}, x_{12}\} \right\},$$

$$E^m = \left\{ \{x_2, \{x_3, x_5\}\}, \quad \{x_7, \{x_4, x_6\}\}, \quad \{x_8, \{x_3, x_9\}\}, \quad \{x_{11}, \{x_4, x_{10}\}\} \right\}.$$

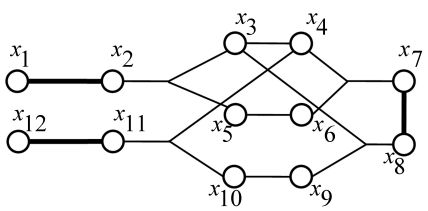

Fig. 3. Divisible graph of multiplicity 2

Рис. 3. Делимый граф кратности 2

В этом графе нет ни одного пути, проходящего только по обычным ребрам, который бы соединял вершины x_3 и x_5, x_4 и x_6, x_3 и x_9 или x_4 и x_{10}. Следовательно, граф делимый. Отнесем обычные вершины x_3 и x_4, а также все инцидентные им обычные и связанные ребра к части G_1 графа, а вершины x_5, x_6, x_9, x_{10} и все инцидентные им обычные и связанные ребра – к части G_2. В итоге получим деление на две части графа, показанные на рис. 4. Связанные ребра кратных и мультиребер отмечены на этом рисунке пунктирными линиями.

Дадим теперь определение кратного пути. Основное отличие кратного пути от пути в обычном графе состоит в том, что связанные ребра каждого кратного и мультиребра должны проходиться в этом пути согласованно.

Определение 5. $S(x,y) = \cup_{i=1}^{k} S^i(x,y)$ *является* кратным путем из вершины x в вершину y *в графе* $G(V,E)$, *если выполнены следующие условия:*

1. $S^i(x,y) = \left(\{x, v_1^i\}, \{v_1^i, v_2^i\}, \ldots, \{v_{l_i-1}^i, v_{l_i}^i\}, \{v_{l_i}^i, y\}\right)$, *где* $l_i \geqslant 0$, – *последовательность ребер, представляющая собой обычный (некратный) путь из x в y, где каждое ребро* $\{a,b\}$ *является либо обычным ребром в графе* $G(V,E)$, *либо i-ым связанным ребром кратного или мультиребра. Значения* l_i *и* l_j ($i \neq j$) *не согласовываются и могут быть как равными, так и различными. Если в путь* $S(x,y)$ *не входит ни одного кратного или мультиребра, то* $S^2(x,y) = S^3(x,y) = \ldots = S^k(x,y) = \varnothing$.

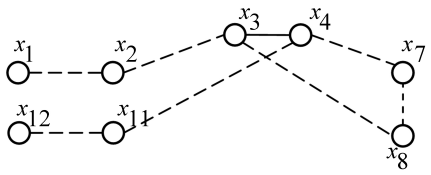

Fig. 4. Partition of a divisible graph

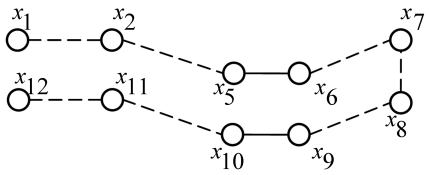

Рис. 4. Части делимого графа

2. *Любая обычная вершина может встретиться в $S^i(x, y)$ несколько раз, то есть $S^i(x, y)$ может содержать циклы.*
3. *Никакая кратная вершина не может встретиться в $S^i(x, y)$ дважды.*
4. *Любое обычное ребро может встречаться в $S^i(x, y)$ несколько раз, причем направления, в которых оно проходится в разных вхождениях, могут не совпадать.*
5. *Обычное ребро, входящее в $S^i(x, y)$, может также входить в любой $S^j(x, y)$, $j \neq i$.*
6. *Все пути $S^i(x, y)$ согласованы (одинаковы) на общей части. Это условие означает, что если связанное ребро какого-то кратного или мультиребра входит в некоторый путь $S^i(x, y)$, то остальные связанные ребра должны входить во все $S^j(x, y)$, $j \neq i$ (по одному связанному ребру в каждый $S^j(x, y)$). При этом порядок вхождения всех кратных и мультиребер во все $S^i(x, y)$ одинаков.*
   Фактически это значит, что если e_1 и e_2 – это два ребра пути $S(x, y)$, каждое из которых либо кратное, либо мультиребро, и в проекции $S^i(x, y)$ связанное ребро из e_1 проходится раньше связанного ребра из e_2, то во всех остальных проекциях $S^j(x, y)$ связанные ребра из e_2 могут проходиться только после связанных ребер из e_1.
7. *Если $S(x, y)$ содержит мультиребро $\{x_0, \{x_1, \ldots, x_k\}\}$, проходимое в направлении от общего конца, то он не может содержать никакого другого мультиребра $\{y_0, \{x_1, \ldots, x_k\}\}$, проходимого в том же направлении. Аналогичное условие должно выполняться и в случае движения к общему концу.*

Определение 6. *Кратный путь $S(x, y)$ является* кратным циклом, *если $x = y$ и $S(x, y) \neq \varnothing$.*

Пример 2. Проиллюстрируем определение кратного пути. Для этого рассмотрим граф, показанный на рис. 3. Будем искать в этом графе путь $S_1(x_1, x_{12})$. Заметим, что часть G_2 графа (справа на рис. 4) фактически представляет собой обычный (некратный) путь из x_1 в x_{12}:

$$S_1^2(x_1, x_{12}) = \big(\{x_1, x_2\},\ \{x_2, x_5\},\ \underline{\{x_5, x_6\}},\ \{x_6, x_7\},\ \{x_7, x_8\},\ \{x_8, x_9\},\ \underline{\{x_9, x_{10}\}},\ \{x_{10}, x_{11}\},\ \{x_{11}, x_{12}\}\big).$$

Все ребра, кроме подчеркнутых, являются связанными ребрами кратных или мультиребер. Поэтому путь в части G_1 обязательно должен проходить эти кратные и мультиребра в том же порядке. Нетрудно убедиться, что путь $S_1^1(x_1, x_{12})$ строится единственным образом:

$$S_1^1(x_1, x_{12}) = \big(\{x_1, x_2\},\ \{x_2, x_3\},\ \underline{\{x_3, x_4\}},\ \{x_4, x_7\},\ \{x_7, x_8\},\ \{x_8, x_3\},\ \underline{\{x_3, x_4\}},\ \{x_4, x_{11}\},\ \{x_{11}, x_{12}\}\big).$$

Тогда $S_1(x_1, x_{12}) = S_1^1(x_1, x_{12}) \cup S_1^2(x_1, x_{12})$. Заметим, что обычное ребро $\{x_3, x_4\}$ проходится в проекции $S_1^1(x_1, x_{12})$ дважды, но это не противоречит нашему определению кратного пути.

Теперь немного изменим граф, добавив ребро $\{x_3, x_9\}$ (рис. 5). Оно свяжет обычные вершины – концы мультиребра $\{x_8, \{x_3, x_9\}\}$. Следовательно, граф перестанет быть делимым.

Проанализируем возможные кратные пути из x_1 в x_{12}. Заметим, что в новом графе сохраняется путь $S_1(x_1, x_{12})$. Кроме того, добавляется путь $S_2(x_1, x_{12})$, состоящий из двух обычных путей:

$$S_2^1(x_1, x_{12}) = \big(\{x_1, x_2\},\ \{x_2, x_3\},\ \underline{\{x_3, x_4\}},\ \{x_4, x_7\},\ \{x_7, x_8\},\ \{x_8, x_3\},\ \underline{\{x_3, x_9\}},\ \underline{\{x_9, x_{10}\}},\ \{x_{10}, x_{11}\},\ \{x_{11}, x_{12}\}\big).$$

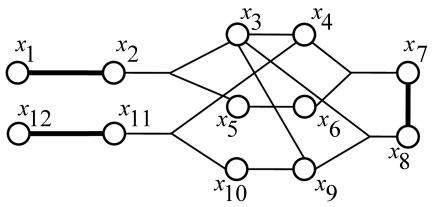

Fig. 5. Multiple graph of multiplicity 2 **Рис. 5.** Кратный граф кратности 2

$S_2^2(x_1, x_{12}) = \big(\{x_1, x_2\},\ \{x_2, x_5\},\ \underline{\{x_5, x_6\}},\ \{x_6, x_7\},\ \{x_7, x_8\},\ \{x_8, x_9\},\ \underline{\{x_9, x_3\}},\ \underline{\{x_3, x_4\}},\ \{x_4, x_{11}\},\ \{x_{11}, x_{12}\}\big)$.

Здесь обычно ребро $\{x_3, x_9\}$ проходится по разу в каждой проекции, причем в противоположных направлениях, а ребро $\{x_3, x_4\}$ также проходится по разу в каждой проекции, но в одном направлении.

Определение 7. *Кратный граф $G(V, E)$ является* связным*, если одновременно выполнены два условия:*

1. *Кратный путь $S(x, y)$ существует для любых двух кратных вершин $x \in V$, $y \in V$.*
2. *Невозможно выделить такой подграф $G' \subset G$, который будет содержать только обычные ребра, и при этом подграфы G' и $G \setminus G'$ не будут соединены ни одним ребром (обычным ребром или связанным ребром мультиребра).*

В отличие от обычных графов связность кратного графа не предполагает наличие кратных путей из каждой вершины в каждую. Фактически в связном кратном графе между каждой парой вершин должен существовать обычный (некратный) путь, использующий связанные ребра кратных и мультиребер несогласованно, а кратные пути обязательно должны существовать только для пар кратных вершин.

Для делимого кратного графа определение связности может быть переписано в более простой форме, что обусловлено структурой графа.

Определение 8. *Делимый кратный граф $G(V, E)$ является* связным*, если одновременно выполнены два условия:*

1. *Кратный путь $S(x, y)$ существует для любых двух кратных вершин $x \in V$, $y \in V$.*
2. *Каждая из частей G_i является связным (некратным) графом.*

Определение 9. Кратное дерево – *это связный кратный граф без циклов.*

Отметим, что оба графа из примеров 1–2 являются кратными деревьями. В кратном дереве количество ребер может быть различным, что видно из указанных примеров. Подробно данный вопрос обсуждался в [2]. Там же был приведен пример кратного дерева, у которого нет ни одного листа.

2. Минимальное остовное дерево в кратном графе

Поставим несколько задач о минимальных остовных деревьях в кратном графе. Для этого сначала приведем определения остовного и полного остовного дерева в кратном графе (детальное обсуждение этих понятий см. в [2]).

Определение 10. Остовным деревом *в кратном графе $G(V, E)$ называется кратное дерево $T(V, E')$, для которого $E' \subseteq E$.*

Заметим, что в остовном дереве заведомо будут существовать кратные пути $S(x, y)$ для всех кратных вершин x, y. Однако если хотя бы одна из вершин x, y является обычной, существование пути $S(x, y)$ не гарантировано даже в том случае, когда такой путь существует в исходном графе $G(V, E)$.

Определение 11. *Остовное дерево $T(V, E')$ в кратном графе $G(V, E)$ является* полным, *если для любой пары вершин $x \in V$, $y \in V$ кратный путь $S_T(x, y)$ в дереве $T(V, E')$ существует тогда и только тогда, когда существует кратный путь $S_G(x, y)$ в исходном графе $G(V, E)$.*

Отметим, что полное остовное дерево существует в любом кратном графе.

Определение 12. *Целочисленная функция $l(e)$, определенная для всех ребер $e \in E$, является* длиной (весом) ребра *в кратном графе $G(V, E)$, если выполнено следующее:*

1. *$l(e) > 0$ для любого ребра e.*
2. *Если e является кратным или мультиребром, то $l(e_1) = l(e_2) = \ldots = l(e_k)$ и $l(e) = k \cdot l(e_1)$, где $e_1, \ldots, e_k$ – это связанные ребра данного ребра e.*

Тогда *вес кратного графа $G(V, E)$* будет определяться по формуле

$$w(G(V, E)) = \sum_{e \in E} l(e).$$

Поставим две задачи о минимальном остовном дереве.

Задача 1 (минимальное остовное дерево)**.** *В кратном графе $G(V, E)$ требуется найти такое остовное дерево $T^{\min}(V, E')$, что для любого другого остовного дерева $T(V, E'')$ выполнено*

$$w(T^{\min}(V, E')) \leqslant w(T(V, E'')).$$

Задача 2 (минимальное полное остовное дерево)**.** *В кратном графе $G(V, E)$ требуется найти такое полное остовное дерево $T^{\min}_{complete}(V, E')$, что для любого другого полного остовного дерева $T_{complete}(V, E'')$ выполнено*

$$w(T^{\min}_{complete}(V, E')) \leqslant w(T_{complete}(V, E'')).$$

Для удобства будем в дальнейшем обозначать задачи 1 и 2 через ОД и ПОД соответственно. Сопоставим оптимизационным задачам ОД и ПОД две задачи распознавания.

Задача 3 (распознавание остовного дерева ограниченного веса)**.**

УСЛОВИЕ. Дан связный кратный граф $G(V, E)$ с положительными целыми весами ребер и положительное целое $K \leqslant w(G)$.

ВОПРОС. Существует ли в кратном графе $G(V, E)$ остовное дерево $T(V, E')$ такое, что

$$w(T(V, E')) \leqslant K?$$

Задача 4 (распознавание полного остовного дерева ограниченного веса)**.**

УСЛОВИЕ. Дан связный кратный граф $G(V, E)$ с положительными целыми весами ребер и положительное целое $K \leqslant w(G)$.

ВОПРОС. Существует ли в кратном графе $G(V, E)$ полное остовное дерево $T_{complete}(V, E')$ такое, что

$$w(T_{complete}(V, E')) \leqslant K?$$

Эти две задачи распознавания в дальнейшем мы будем обозначать через ОДР и ПОДР соответственно.

Лемма 1. *Задача ПОДР* $\in NP$.

Доказательство. Сначала заметим, что объем входных данных любой индивидуальной задачи ПОДР конечен. Действительно, каждая индивидуальная задача определяется конечным множеством вершин кратного графа и конечным множеством ребер с целочисленными весами, а также целым числом $K \leqslant w(G)$.

Значит, для доказательства того, что ПОДР $\in NP$, достаточно показать, что проверка ответа «да» в любой индивидуальной задаче ПОДР осуществляется за полиномиальное время.

Для проверки того, что некий граф $T(V, E')$ дает ответ «да» индивидуальной задаче ПОДР, нужно выполнить следующие действия:

1. Проверить, что $E' \subseteq E$ (не более $|E'| \cdot |E|$ шагов).
2. Проверить связность графа $T(V, E')$ (полиномиальный алгоритм из статьи [1]).
3. Проверить, что граф $T(V, E')$ не содержит циклов. Для этого требуется незначительно модифицировать алгоритм проверки связности графа с предыдущего шага.
   Напомним, что алгоритм сначала ищет все множества достижимости по обычным ребрам (множества обычных вершин таких, что между ними есть пути, состоящие только из обычных ребер) и все множества достижимости по кратным ребрам (множества кратных вершин таких, что между ними есть пути, состоящие только из кратных ребер). На этом шаге нужно дополнительно потребовать, чтобы в графе T на каждом таком множестве был построен ациклический граф (обычное дерево). Для этого достаточно посчитать количество ребер.
   На следующем шаге алгоритма проверки связности определяется список пар смежных множеств достижимости по кратным ребрам (между этими множествами есть кратный путь, содержащий мультиребра). Очевидно, что если выполнено условие ацикличности множеств достижимости, то кратный цикл в графе существует тогда и только тогда, когда в списке пар есть дубликаты или в какой-то паре элементы одинаковы. Поиск дубликатов в списке – квадратичная относительно длины списка процедура, а длина списка заведомо меньше $|V|$. Поиск пар одинаковых элементов в списке – линейная относительно длины списка процедура.
4. Проверить, что для каждой пары вершин $x \in V$, $y \in V$ кратный путь $S_G(x, y)$ существует тогда и только тогда, когда существует кратный путь $S_T(x, y)$ (для этого нужно не более $|V| \cdot |V - 1|$ раз запустить полиномиальный алгоритм проверки существования пути из статьи [1]).
5. Просуммировать веса всех ребер из E' для проверки того, что $w(T) \leqslant K$.

Все шаги описанной процедуры полиномиальны. Следовательно, ПОДР $\in NP$.

Лемма доказана.

Лемма 2. *Задача ОДР* $\in NP$.

Доказательство аналогично. Единственное отличие – не нужен шаг 4 процедуры проверки, поскольку на нем устанавливается полнота остовного дерева.

Для обоснования NP-полноты задач ПОДР и ОДР мы будем выполнять полиномиальное сведение к ним известной NP-полной *задачи о трехмерном сочетании* (см., например, [13, 14]).

Задача 5 (трехмерное сочетание)**.**

УСЛОВИЕ. Дано множество $M \subseteq I \times J \times P$*, где* I*,* J *и* P *– попарно непересекающиеся множества и* $|I| = |J| = |P| = q$.

ВОПРОС. Верно ли, что существует множество $M' \subseteq M$ *такое, что* $|M'| = q$ *и каждое значение каждой координаты встречается в* M' *ровно один раз?*

Эту задачу мы будем в дальнейшем обозначать через 3С.

3. NP-полнота задачи о минимальном остовном дереве в кратном графе при $k \geqslant 3$

Теорема 1. *Задача ПОДР для делимого кратного графа $G(V, E)$ кратности $k \geqslant 3$ является NP-полной.*

Доказательство. Задача ПОДР $\in NP$ (лемма 1). Для доказательства NP-полноты покажем, что 3С $\propto$ ПОДР при любой кратности графа $k \geqslant 3$.

Рассмотрим задачу 3С. Не ограничивая общности рассуждений, можно считать, что каждое значение каждой координаты встречается в M хотя бы один раз. Действительно, если это не так, то задача 3С заведомо имеет ответ «нет», что может быть установлено путем тривиальной линейной проверки.

Фиксируем кратность графа $k \geqslant 3$. Рассмотрим теперь произвольную индивидуальную задачу 3СМ $\in$ 3С с множеством M указанного вида и выполним ее полиномиальное сведение к индивидуальной задаче ПОДРМ $\in$ ПОДР.

Получим по множеству M граф $G(V, E)$ и число $K \leqslant w(G)$:

1. Каждому элементу $i_s \in I$ ($s \in \overline{1, q}$) сопоставим обычную вершину $I_s \in V$.
2. Каждому элементу $j_s \in J$ ($s \in \overline{1, q}$) сопоставим обычную вершину $J_s \in V$.
3. Каждому элементу $p_s \in P$ ($s \in \overline{1, q}$) сопоставим обычную вершину $P_s \in V$.
4. Каждому элементу $(i_s, j_t, p_u) \in M$ сопоставим кратную вершину $x_{stu} \in V$.
5. Добавим в множество V кратную вершину x_0 и обычные вершины $y_4, \ldots, y_k$ (если $k = 3$, добавляем только x_0).
6. Соединим вершину x_0 с каждой вершиной x_{stu} кратным ребром $\{x_0, x_{stu}\}$ веса k.
7. Соединим каждую вершину x_{stu} с вершинами I_s, J_t, P_u, y_4, $\ldots$, y_k мультиребром $\{x_{stu}, \{I_s, J_t, P_u, y_4, \ldots, y_k\}\}$ веса $k(|M| + 1)$ (если $k = 3$, мультиребро имеет вид $\{x_{stu}, \{I_s, J_t, P_u\}\}$).
8. Установим $K = k|M| + kq(|M| + 1) = k|M|(q + 1) + k$.

Очевидно, что шаги 1–8 реализуют полиномиальную процедуру.

Делимость графа также очевидна. Часть G_1 состоит из всех кратных вершин, обычных вершин $I_1, \ldots, I_q$ и соответствующих связанных ребер между ними. Часть G_2 состоит из всех кратных вершин, обычных вершин $J_1, \ldots, J_q$ и соответствующих связанных ребер между ними. Часть G_3 состоит из всех кратных вершин, обычных вершин $P_1, \ldots, P_q$ и соответствующих связанных ребер между ними. Части G_r ($r \in \overline{4, k}$) состоят из всех кратных вершин, обычной вершины y_r и соответствующих связанных ребер между ними (если $k > 3$).

Между любой парой кратных вершин существуют пути, проходящие только по кратным ребрам (шаг 6 алгоритма сведения). При этом $E^o = \varnothing$, а значит, путей между обычными вершинами быть не может. Однако каждая обычная вершина инцидентна хотя бы одному мультиребру (это следует из того, что в множестве M каждое значение каждой координаты встречается хотя бы один раз, и из шага 7 алгоритма сведения). Следовательно, граф связен.

$E^o = \varnothing$, поэтому $|E^o| = 0$.

$|E^k| = |M|$, а $l(e) = k$ для всех $e \in E^k$ (шаги 4, 6 алгоритма сведения), поэтому $\sum_{e \in E^k} l(e) = k|M|$.

$|E^m| \geqslant q$, а $l(e) = k(|M| + 1)$ для всех $e \in E^m$ (шаги 1–3, 7 алгоритма сведения и вышеуказанное условие для множества M), поэтому $\sum_{e \in E^m} l(e) \geqslant kq(|M| + 1)$.

Собирая полученные соотношения, получаем

$$w(G) = \sum_{e \in E} l(e) = \sum_{e \in E^o \cup E^k \cup E^m} l(e) = \sum_{e \in E^o} l(e) + \sum_{e \in E^k} l(e) + \sum_{e \in E^m} l(e) \geqslant 0 + k|M| + kq(|M| + 1) = K.$$

Таким образом, в результате шагов 1–8 получена корректная индивидуальная задача ПОДРМ для делимого графа выбранной кратности $k \geqslant 3$.

Покажем теперь, что любая индивидуальная задача 3СМ имеет ответ «да» тогда и только тогда, когда соответствующая индивидуальная задача ПОДРМ имеет ответ «да».

Пусть сначала индивидуальная задача 3СМ имеет ответ «да». Возьмем $M' \subseteq M$, которое обеспечивает положительный ответ в задаче 3СМ. Построим по M' полное остовное дерево $T(V, E')$, $E' \subseteq E$ в графе $G(V, E)$ таким образом:

1. $E^k \subseteq E'$, то есть все кратные ребра включаются в E'.
2. Включаем мультиребро $\{x_{stu}, \{I_s, J_t, P_u, y_4, \dots, y_k\}\}$ в E' для всех $(i_s, j_t, p_u) \in M'$.

Заметим, что каждая из вершин I_s ($s \in \overline{1, q}$), J_t ($t \in \overline{1, q}$) и P_u ($u \in \overline{1, q}$) инцидентна ровно одному мультиребру, поскольку $|M'| = q$ и для любой пары наборов $(i_s, j_t, p_u) \in M'$, $(i_{s'}, j_{t'}, p_{u'}) \in M'$ выполнено $s \neq s'$, $t \neq t'$ и $u \neq u'$.

Каждая из вершин $y_4, \dots, y_k$ (при наличии) инцидентна ровно q мультиребрам.

Поскольку $E^k \subseteq E'$, то в графе $T(V, E')$ существует путь между каждой парой кратных вершин, проходящий только по кратным ребрам, как и в исходном графе $G(V, E)$. При этом в графе $T(V, E')$ нет ни одного обычного ребра, как и в графе $G(V, E)$.

Следовательно, граф $T(V, E')$ связен.

Поскольку все кратные ребра инцидентны x_0, а у каждой пары мультиребер графа $T(V, E')$ есть отличие в трех парах отдельных концов и при этом нет ни одного обычного ребра, то кратных циклов в графе $T(V, E')$ быть не может. Следовательно $T(V, E')$ есть остовное дерево графа $G(V, E)$. Так как $E^o = \varnothing$, остовное дерево $T(V, E')$ заведомо полное.

Убедимся, что $w(T) \leqslant K$.

$E' = E^k \cup E_T^m$ ($E_T^m \subseteq E^m$). Как уже отмечалось, $\sum_{e \in E^k} l(e) = k|M|$.

$|E_T^m| = |M'| = q$, а $l(e) = k(|M| + 1)$ для всех $e \in E^m$, поэтому $\sum_{e \in E_T^m} l(e) = kq(|M| + 1)$.

В итоге

$$w(T) = \sum_{e \in E'} l(e) = \sum_{e \in E^k \cup E_T^m} l(e) = \sum_{e \in E^k} l(e) + \sum_{e \in E_T^m} l(e) = k|M| + kq(|M| + 1) = K$$

и требуемое неравенство выполнено.

Пусть теперь индивидуальная задача ПОДРМ, построенная по множеству M, имеет ответ «да». Возьмем полное остовное дерево $T(V, E')$, приводящее к ответу «да», и покажем, что по нему можно построить множество M', дающее ответ «да» в задаче 3СМ.

Сначала заметим, что исходный граф $G(V, E)$ является деревом. Действительно, между каждой парой вершин существует единственный кратный путь. Он проходит только по кратным ребрам и обязательно содержит вершину x_0. При этом не существует ни одного кратного пути, содержащего мультиребра, поскольку для любых двух мультиребер обязательно найдется пара несовпадающих отдельных концов (в силу построения графа) и между этими несовпадающими концами нет пути по обычным ребрам, так как $E^o = \varnothing$.

Раз $G(V, E)$ – кратное дерево, то любой его связный подграф будет остовным деревом для $G(V, E)$, а поскольку $E^o = \varnothing$ и $G(V, E)$ – делимый граф, это дерево обязательно будет полным (критерий полноты остовного дерева из статьи [2]).

Рассмотрим дерево $T(V, E')$, дающее ответ «да» в задаче. Множество E^k обязательно включается в E', иначе T не будет деревом (при исключении кратного ребра $\{x_0, x_{stu}\}$ пропадет единственный путь между этими кратными вершинами). Значит, $E' = E^k \cup E_T^m$, $E_T^m \subseteq E^m$. Тогда

$$w(T) = \sum_{e \in E'} l(e) = \sum_{e \in E^k \cup E_T^m} l(e) = \sum_{e \in E^k} l(e) + \sum_{e \in E_T^m} l(e) = k|M| + \sum_{e \in E_T^m} l(e) = k|M| + k(|M| + 1)|E_T^m| \leqslant K,$$

$$k|M| + k(|M| + 1)|E_T^m| \leqslant k|M| + kq(|M| + 1),$$

$$|E_T^m| \leqslant q.$$

Покажем, что в последнем соотношении выполняется точное равенство $|E_T^m| = q$.

Заметим, что для связности графа $T(V, E')$ требуется, чтобы каждая из вершин $I_1, \ldots, I_q, J_1, \ldots, J_q, P_1, \ldots, P_q$ была достижима хотя бы по одному мультиребру. Поскольку каждое из мультиребер множества E^m содержит строго по одной вершине I_s, J_t, P_u (шаг 7 процедуры сведения), то таких мультиребер требуется не меньше q, то есть $|E_T^m| \geqslant q$. Совокупность двух неравенств приводит к требуемому равенству.

Раз $|E_T^m| = q$ и кратный граф $T(V, E')$ связен, то каждая из вершин I_s, J_t, P_u инцидентна ровно одному мультиребру из E_T^m. Поэтому множество M' получается так:

$$M' = \left\{ (i_s, j_t, p_u) \mid \{x_{stu}, \{I_s, J_t, P_u, y_4, \ldots, y_k\}\} \in E_T^m \right\}.$$

Каждое значение каждой координаты встречается в M' ровно один раз, следовательно, задача 3СМ имеет ответ «да».

Таким образом, для произвольного фиксированного $k \geqslant 3$ мы получили полиномиальное сведение известной NP-полной задачи 3С к сужению задачи ПОДР для делимого графа. Следовательно, задача ПОДР NP-полна для делимого графа при любой кратности $k \geqslant 3$.

Теорема доказана.

Следствие 1. *Задача ОДР для делимого кратного графа $G(V, E)$ кратности $k \geqslant 3$ является NP-полной.*

Справедливость данного утверждения следует из того факта, что для полученного в доказательстве теоремы 1 графа $G(V, E)$ любое остовное дерево является полным, а значит, задачи ПОДР и ОДР будут идентичны для этого графа.

Следствие 2. *Задачи ОДР и ПОДР для произвольного кратного графа $G(V, E)$ кратности $k \geqslant 3$ являются NP-полными.*

Данное утверждение следует из того, что делимый кратный граф – частный случай произвольного кратного графа.

Следствие 3. *Оптимизационные задачи ОД и ПОД являются NP-трудными как для произвольного, так и для делимого кратного графа $G(V, E)$ кратности $k \geqslant 3$.*

Следствие 4. *Задачи ОДР и ПОДР являются NP-полными для кратного графа $G(V, E)$ кратности $k \geqslant 3$, у которого $E^o = \varnothing$. Соответствующие оптимизационные задачи ОД и ПОД – NP-трудные.*

Это утверждение следует из того, что при доказательстве теоремы 1 строится граф именно такой структуры. Отметим, что граф из следствия 4 обязательно будет делимым.

Пример 3. Проиллюстрируем процедуру сведения, описанную в доказательстве теоремы. Пусть $q = 3$ и множество M имеет вид

$$M = \left\{ (i_1, j_1, p_1),\ (i_1, j_3, p_2),\ (i_2, j_1, p_3),\ (i_2, j_2, p_1),\ (i_3, j_2, p_1) \right\}.$$

Положим $k = 3$ и будем строить граф кратности 3. Сначала получим множество вершин (шаги 1–5 процедуры сведения):

$$V = \{x_0,\ x_{111},\ x_{132},\ x_{213},\ x_{221},\ x_{321},\ I_1,\ I_2,\ I_3,\ J_1,\ J_2,\ J_3,\ P_1,\ P_2,\ P_3\}.$$

Добавим кратные ребра (шаг 6):

$$E^k = \Big\{ \{x_0, x_{111}\},\ \{x_0, x_{132}\},\ \{x_0, x_{213}\},\ \{x_0, x_{221}\},\ \{x_0, x_{321}\} \Big\}$$

и установим вес этих ребер равным 3. Затем добавим мультиребра (шаг 7):

$$E^k = \Big\{ \{x_{111}, \{I_1, J_1, P_1\}\},\ \{x_{132}, \{I_1, J_3, P_2\}\},\ \{x_{213}, \{I_2, J_1, P_3\}\},\ \{x_{221}, \{I_2, J_2, P_1\}\},\ \{x_{321}, \{I_3, J_2, P_1\}\} \Big\}$$

и установим их вес равным 18. Наконец, установим $K = 69$ (шаг 8). Полученный граф представлен на рис. 6.

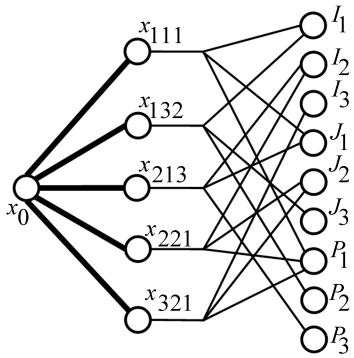

Fig. 6. Multiple graph

Рис. 6. Кратный граф

Нетрудно убедиться, что единственное остовное дерево веса не больше K, которое существует в данном графе, получается исключением из E^m ребер $\{x_{111}, \{I_1, J_1, P_1\}\}$ и $\{x_{221}, \{I_2, J_2, P_1\}\}$ (рис. 7). Это дерево будет иметь вес 69, и каждая обычная вершина инцидентна ровно одному связанному ребру мультиребра. Естественно, это остовное дерево является полным и минимальным в графе. Построим по нему множество M':

$$M' = \Big\{ (i_1, j_3, p_2),\ (i_2, j_1, p_3),\ (i_3, j_2, p_1) \Big\}.$$

Видим, что полученное M' является корректным трехмерным сочетанием, дающим ответ «да» в исходной задаче 3СМ.

Пример 4. Теперь рассмотрим случай, когда индивидуальная задача 3СМ не имеет решения. Пусть $q = 3$ и множество M имеет вид

$$M = \Big\{ (i_1, j_1, p_1),\ (i_1, j_3, p_2),\ (i_2, j_1, p_3),\ (i_2, j_2, p_1),\ (i_3, j_2, p_2) \Big\}.$$

По сравнению с предыдущим примером здесь изменилось значение третьей координаты в последней тройке. Положим $k = 3$ и будем строить граф кратности 3. Сначала получим множество вершин (шаги 1–5 процедуры сведения):

$$V = \{x_0,\ x_{111},\ x_{132},\ x_{213},\ x_{221},\ x_{322},\ I_1,\ I_2,\ I_3,\ J_1,\ J_2,\ J_3,\ P_1,\ P_2,\ P_3\}.$$

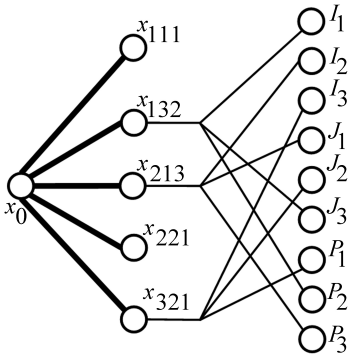

Fig. 7. Spanning tree

Рис. 7. Остовное дерево

Добавим кратные ребра (шаг 6):

$$E^k = \left\{ \{x_0, x_{111}\},\ \{x_0, x_{132}\},\ \{x_0, x_{213}\},\ \{x_0, x_{221}\},\ \{x_0, x_{322}\} \right\}$$

и установим вес этих ребер равным 3. Затем добавим мультиребра (шаг 7):

$$E^k = \left\{ \{x_{111}, \{I_1, J_1, P_1\}\},\ \{x_{132}, \{I_1, J_3, P_2\}\},\ \{x_{213}, \{I_2, J_1, P_3\}\},\ \{x_{221}, \{I_2, J_2, P_1\}\},\ \{x_{322}, \{I_3, J_2, P_2\}\} \right\}$$

и установим их вес равным 18. Наконец, как и в предыдущем примере, установим $K = 69$ (шаг 8). Полученный граф представлен на рис. 8.

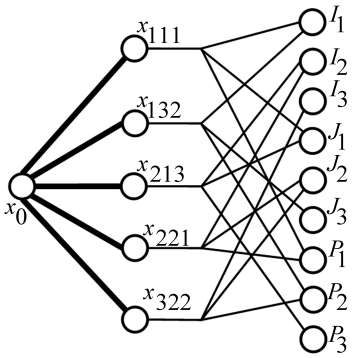

Fig. 8. Multiple graph

Рис. 8. Кратный граф

Нетрудно убедиться, что минимальное остовное дерево в данном графе получается исключением из E^m только одного ребра, например, $\{x_{221}, \{I_2, J_2, P_1\}\}$ (рис. 9). Вес этого дерева равен 87, что больше K, и в нем есть обычные вершины, инцидентные более чем одному ребру (это вершины I_1, J_1 и P_2). Но и задача 3СМ не имеет решения.

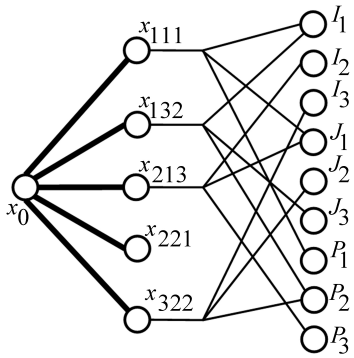

Fig. 9. Spanning tree

Рис. 9. Остовное дерево

Заключение

В данной статье была рассмотрена задача о минимальном остовном дереве в кратном графе в различных вариантах постановки. Для всех вариантов доказано, что при кратности $k \geqslant 3$ задача распознавания остовного дерева ограниченного веса является NP-полной, а соответствующая оптимизационная задача – NP-трудная.

При $k = 1$ кратный граф превращается в обычный, и задача о минимальном остовном дереве разрешима за полиномиальное время. Остается открытым только вопрос о сложности задачи при кратности графа $k = 2$.

References

[1] A. V. Smirnov, “The Shortest Path Problem for a Multiple Graph”, *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 625–633, 2018. DOI: 10.3103/S0146411618070234.

[2] A. V. Smirnov, “The Spanning Tree of a Divisible Multiple Graph”, *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 871–879, 2018. DOI: 10.3103/S0146411618070325.

[3] J. B. Kruskal, “On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem”, *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956. DOI: 10.1090/S0002-9939-1956-0078686-7.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd. The MIT Press, McGraw-Hill Book Company, 2009.

[5] C. Berge, *Graphs and Hypergraphs*. North-Holland Publishing Company, 1973.

[6] A. Basu and R. W. Blanning, “Metagraphs in workflow support systems”, *Decision Support Systems*, vol. 25, no. 3, pp. 199–208, 1999. DOI: 10.1016/S0167-9236(99)00006-8.

[7] A. Basu and R. W. Blanning, *Metagraphs and Their Applications*, ser. Integrated Series in Information Systems. Springer US, 2007, vol. 15.

[8] V. S. Rublev and A. V. Smirnov, “Flows in Multiple Networks”, *Yaroslavsky Pedagogichesky Vestnik*, vol. 3, no. 2, pp. 60–68, 2011.

[9] A. V. Smirnov, “The Problem of Finding the Maximum Multiple Flow in the Divisible Network and its Special Cases”, *Automatic Control and Computer Sciences*, vol. 50, no. 7, pp. 527–535, 2016. DOI: 10.3103/S0146411616070191.

[10] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton University Press, 1962.

[11] V. S. Roublev and A. V. Smirnov, "The Problem of Integer-Valued Balancing of a Three-Dimensional Matrix and Algorithms of Its Solution", *Modeling and Analysis of Information Systems*, vol. 17, no. 2, pp. 72–98, 2010.

[12] A. V. Smirnov, "Network Model for the Problem of Integer Balancing of a Four-Dimensional Matrix", *Automatic Control and Computer Sciences*, vol. 51, no. 7, pp. 558–566, 2017. DOI: 10.3103/S0146411617070185.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[14] R. Karp, "Reducibility among combinatorial problems", in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., Plenum, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 1, 2021
journal homepage: www.mais-journal.ru

COMPUTER SYSTEM ORGANIZATION

On Characteristics of Symbolic Execution in the Problem of Assessing the Quality of Obfuscating Transformations

P. D. Borisov[1], Y. V. Kosolapov[1]

DOI: 10.18255/1818-1015-2021-1-38-51

[1]Southern Federal University, 8a Milchakova str., Rostov-on-Don 344090, Russia.

MSC2020: 68N20
Research article
Full text in Russian

Received February 20, 2021
After revision March 10, 2021
Accepted March 12, 2021

Obfuscation is used to protect programs from analysis and reverse engineering. There are theoretically effective and resistant obfuscation methods, but most of them are not implemented in practice yet. The main reasons are large overhead for the execution of obfuscated code and the limitation of application only to a specific class of programs. On the other hand, a large number of obfuscation methods have been developed that are applied in practice. The existing approaches to the assessment of such obfuscation methods are based mainly on the static characteristics of programs. Therefore, the comprehensive (taking into account the dynamic characteristics of programs) justification of their effectiveness and resistance is a relevant task. It seems that such a justification can be made using machine learning methods, based on feature vectors that describe both static and dynamic characteristics of programs. In this paper, it is proposed to build such a vector on the basis of characteristics of two compared programs: the original and obfuscated, original and deobfuscated, obfuscated and deobfuscated. In order to obtain the dynamic characteristics of the program, a scheme based on a symbolic execution is constructed and presented in this paper. The choice of the symbolic execution is justified by the fact that such characteristics can describe the difficulty of comprehension of the program in dynamic analysis. The paper proposes two implementations of the scheme: extended and simplified. The extended scheme is closer to the process of analyzing a program by an analyst, since it includes the steps of disassembly and translation into intermediate code, while in the simplified scheme these steps are excluded. In order to identify the characteristics of symbolic execution that are suitable for assessing the effectiveness and resistance of obfuscation based on machine learning methods, experiments with the developed schemes were carried out. Based on the obtained results, a set of suitable characteristics is determined.

Keywords: obfuscation, symbolic execution, program similarity, program comprehension

INFORMATION ABOUT THE AUTHORS

Petr D. Borisov | orcid.org/0000-0002-8919-8310. E-mail: borisovpetr@mail.ru
postgraduate student.

Yury V. Kosolapov
correspondence author | orcid.org/0000-0002-1491-524X. E-mail: itaim@mail.ru
PhD.

For citation: P. D. Borisov and Y. V. Kosolapov, “On Characteristics of Symbolic Execution in the Problem of Assessing the Quality of Obfuscating Transformations”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 38-51, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 1, 2021
сайт журнала: www.mais-journal.ru

COMPUTER SYSTEM ORGANIZATION

О характеристиках символьного исполнения в задаче оценки качества обфусцирующих преобразований

П. Д. Борисов[1], Ю. В. Косолапов[1]

DOI: 10.18255/1818-1015-2021-1-38-51

[1]Южный Федеральный Университет, ул. Мильчакова, 8а, г. Ростов-на-Дону, 344090 Россия.

УДК 517,9
Научная статья
Полный текст на русском языке

Получена 20 февраля 2021 г.
После доработки 10 марта 2021 г.
Принята к публикации 12 марта 2021 г.

Обфускация применяется для защиты программ от анализа и обратного проектирования. Несмотря на то, что в настоящее время существуют теоретически стойкие методы обфускации, эти методы пока не могут применяться на практике. В основном это связано либо с затратностью по ресурсам на исполнение обфусцированного кода, либо с ограничением на применение только к конкретному классу программ. С другой стороны, разработано большое количество методов обфускации, которые применяются на практике. Существующие подходы к оценке таких обфусцирующих преобразований в большей степени основаны на статических характеристиках программ. Однако актуальна задача комплексного (учитывающего и динамические характеристики программ) обоснования их эффективности и стойкости. Представляется, что такое обоснование может быть выполнено с помощью методов машинного обучения на основе векторов признаков, описывающих как статические, так и динамические характеристики программ. В настоящей работе такой вектор предлагается строить на основе характеристик пар сравниваемых программ: исходной и обфусцированной, исходной и деобфусцированной, обфусцированной и деобфусцированной. Для получения динамических характеристик программы в работе построена схема, основанная на символьном исполнении. Выбор символьного исполнения обосновывается тем, что такие характеристики могут описать сложность понимания программы при динамическом анализе. В работе предлагается две реализации схемы: расширенная и упрощенная. Расширенная схема приближена к процессу анализа программы аналитиком, так как включает в себя этапы дизассемблирования и трансляции в промежуточный код, в то время как в упрощенной схеме эти этапы исключены. С разработанными схемами проведены эксперименты с целью выявления характеристик символьного исполнения, подходящих для оценки эффективности и стойкости обфускации на основе методов машинного обучения. На основе полученных результатов определен набор подходящих характеристик.

Ключевые слова: обфускация, символьное исполнение, похожесть программ, понимание программ

ИНФОРМАЦИЯ ОБ АВТОРАХ

Петр Дмитриевич Борисов | orcid.org/0000-0002-8919-8310. E-mail: borisovpetr@mail.ru
аспирант.

Юрий Владимирович Косолапов
автор для корреспонденции | orcid.org/0000-0002-1491-524X. E-mail: itaim@mail.ru
канд. техн. наук.

Для цитирования: P. D. Borisov and Y. V. Kosolapov, “On Characteristics of Symbolic Execution in the Problem of Assessing the Quality of Obfuscating Transformations”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 38-51, 2021.

Введение

Обфускация – это модификация программного кода с сохранением его функциональности, затрудняющая анализ, понимание алгоритмов программы и их модификацию. Обфускация широко используется для защиты программ от анализа и обратного проектирования [1]. Несмотря на то, что в настоящее время существуют теоретически стойкие методы обфускации [2], эти методы пока не могут применяться на практике. В основном это связано, либо с затратностью по ресурсам на исполнение обфусцированного кода, либо с ограничением на применение только к конкретному классу программ [3]. Разработано множество методов обфускации, которые на интуитивном уровне затрудняют понимание защищаемых обфускацией алгоритмов, но теоретического обоснования их эффективности нет. Тем не менее предлагается ряд практических способов и метрик для оценки эффективности обфусцирующих преобразований, т.е. стойкости к анализу и пониманию программ [4—8]. Отметим, что понимание исходного кода программы является широко исследуемой областью в программной инженерии [9]. В [10] отмечается, что понимание программы и запутывание кода – это две стороны одной медали, и поэтому метрики для оценки понимания строятся в [10] с использованием знаний из области обфускации. Можно предположить, что методы оценки понимания программы, в свою очередь, также могут быть использованы для оценки эффективности запутывающих преобразований.

В [11] предложена схема оценки стойкости запутывающих преобразований, основанная на сравнении признаков подобия, вычисленных по характеристикам программ. Основным блоком этой схемы является *блок оценки стойкости*, который делает вывод о сходстве программ. Этот блок может быть реализован с помощью методов машинного обучения. Для этого необходимо иметь характеристики программы, описывающие ее с разных сторон анализа: при статическом анализе (структура графа потока управления, полнота дизассемблирования, понятность кода программы и другие) и при динамическом анализе (поведение программы во время выполнения). Для схемы из [11] набор характеристик, предложенных в [4—8], а также характеристик из области понимания программ [9] может быть построен путем статического анализа программы. В то же время собрать характеристики программы с помощью динамического анализа сложнее, так как требуется запустить программу и проанализировать ее поведение, которое может зависеть от среды исполнения и/или входных параметров. В настоящей работе в качестве модели динамического анализа выбрано символьное исполнение [12], которое может характеризовать сложность понимания программы при динамическом анализе, и уже нашло применение в [13] при анализе обфусцированного кода.

Целью настоящей работы является, с одной стороны, получение и оценка характеристик символьного исполнения обфусцированной/деобфусцированной/исходной программ, с другой – оценка схемы получения характеристик символьного исполнения программ, построенной в [11]. Для оценки схемы рассматривается ее упрощенная версия, в которой отсутствуют шаги компиляции в бинарное представление и трансляции обратно в биткод LLVM [14]. Далее, для удобства упрощенную версию будем называть *упрощенной схемой*, а ее полную версию будем называть *расширенной схемой*.

Статья, кроме введения и заключения, содержит три раздела. Первый раздел посвящен обзору работ в области известных методов оценки обфусцирующих преобразований. Во втором разделе предлагаются реализации расширенной и упрощенной схем получения характеристик символьного исполнения программ. Третий раздел посвящен анализу результатов проведенных экспериментов с предложенными схемами.

1. Известные подходы к оценке обфусцирующих преобразований

Один из первых способов комплексной оценки обфусцирующих преобразований предложен К. Колбергом в [4]. Для этого предлагаются четыре индикатора: эффективность (*potency*), стойкость

(*resilience*), стоимость – степень увеличения потребляемых ресурсов обфусцированной программой (*cost*), качество обфускации (*quality*). Эффективность обфускации определяется с использованием метрик качества программ из программной инженерии, таких как длина программы, цикломатическая сложность, сложность потока и структур данных, а также других метрик. Стойкость определяется как функция от времени аналитика на разработку деобфускатора и времени работы самого деобфускатора. Качество обфускации определяется как комбинация трех предыдущих индикаторов: эффективности, стойкости и стоимости. Однако отметим, что в [4] не предлагается способ оценки времени необходимого аналитику для разработки деобфускатора.

В работе [5] предложен метод поиска и выявления зашифрованных данных в программе (в качестве таких данных могут выступать алгоритмы программы). Метод основан на использовании модели N-Gram [15]. С помощью данного метода вычисляется показатель *искусственности*, который используется для выявления участков программы, содержащих данные с большой энтропией. Представляется, что с помощью этого показателя можно оценить эффективность обфусцирующих преобразований, поскольку такие преобразования могут оказывать влияние на энтропию кода программы (как в меньшую, так и в большую сторону).

В работе [6] предложен иной подход: качество обфускации исходного кода оценивается по Колмогоровской сложности. Экспериментально установлено, что чем меньше сходство исходного кода и декомпилированного кода, тем выше Колмогоровская сложность для запутанной программы. Следовательно, чем выше Колмогоровская сложность (оцененная с помощью алгоритмов сжатия), тем лучше обфускация. Этот подход в [7] применяется для оценки запутанности программ, написанных на языке Java. При этом вычисление Колмогоровской сложности выполняется на основе файлов с исходом кодом программ.

Экспериментальный подход для оценки стойкости обфусцирующих преобразований описан в [8]. В этом подходе обфускация рассматривается с точки зрения понимания программного кода аналитиком. Для оценки стойкости была проведена серия контролируемых экспериментов с участием групп аналитиков. Показано, что статический и динамический анализ обфусцированных программ аналитиком занимает значительно больше времени, чем анализ исходной программы. Но этот метод не подходит для автоматического анализа стойкости обфускации.

Проблема автоматической оценки качества обфусцирующих преобразований, как с точки зрения статического анализа, так и с точки зрения динамического анализа, является актуальной. В рассмотренных выше работах, за исключением эксперементального подхода с участием аналитиков, эффективность рассчитывается на основе статического анализа. Для комплексной оценки эффективности необходимо учитывать динамические характеристики программ. Для получения таких характеристик в настоящей работе предлагается использовать символьное исполнение. Для того чтобы определить, какие характеристики символьного исполнения могут быть использованы для оценки эффективности обфускации, строится схема их получения, проводятся эксперименты, выполняется анализ полученных результатов. Отметим, что символьное исполнение находит применение в задачах исследования и анализа обфусцированного кода. В частности, в [13] отмечается, что обфусцирующие преобразования оказывают значительное влияние на эффективность символьного анализа, и предлагается обобщенный подход повышения эффективности такого анализа. Тем не менее в [13] символьное исполнение используется как способ анализа, без рассмотрения применимости в задачах оценки эффективности и стойкости обфускации.

2. Схемы получения характеристик

В [11] предложена схема нахождения характеристик символьного исполнения, которые предполагается использовать для оценки эффективности обфусцирующих преобразований. В этом разделе кратко описывается эта схема, отмечаются некоторые особенности, связанные с шагами трансляции машинного кода в биткод LLVM, а также описывается упрощенная схема.

2.1. Расширенная схема

В соответствии с моделью [11], программа P проходит следующие шаги: 1) компиляция с помощью обфусцирующего компилятора Hikari [16] с различными опциями обфусцирующих преобразований (выбраны 10 различных обфусцирующих преобразований), а также компиляция без применения преобразований; 2) построение графа потока выполнения скомпилированной программы с помощью инструмента mcsema-disass [17]; 3) трансляция полученного на предыдущем шаге представления в биткод LLVM с помощью инструмента mcsema-lift [17]; 4) оптимизация полученного биткода с помощью оптимизатора opt из состава LLVM; 5) символьное исполнение полученных версий биткода с помощью символьного интерпретатора KLEE [18]; 6) обработка полученных характеристик символьного исполнения. Последовательность шагов показана на рис. 1.

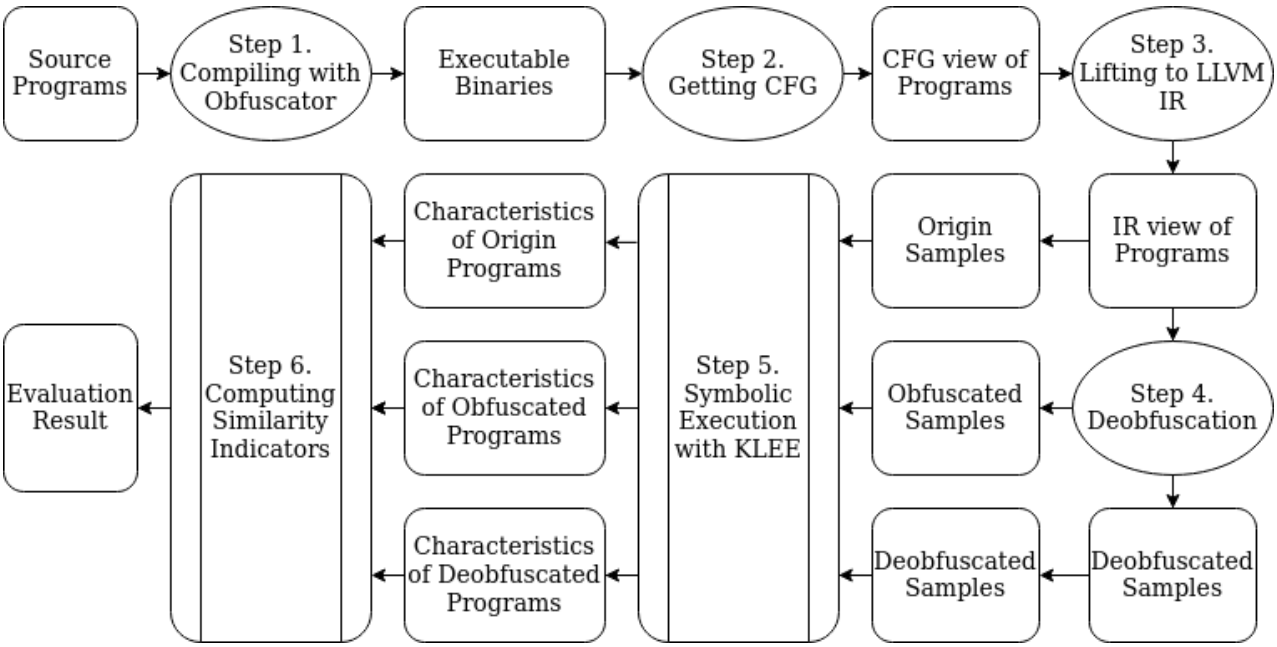

Fig. 1. An extended scheme for finding the characteristics of symbolic execution of programs

Рис. 1. Расширенная схема нахождения характеристик символьного исполнения программ

После первого шага для каждой программы P создается 11 различных *исполнимых* модулей: 10 обфусцированных и один *оригинальный* (без обфускации). В результате выполнения второго и третьего шага получается 11 различных файлов биткода, соответствующих исполнимым модулям. На четвертом шаге для каждого из 10 файлов биткода обфусцированных программ выполняется оптимизация, полученный результат сохраняется в отдельном файле биткода. Заметим, что оптимизатор, используемый на этом шаге, выполняет роль деобфускатора из модели [11], так как оптимизаторы обычно выполняют преобразования обратные к обфусцирующим [19]. Поэтому оптимизированные файлы биткода будем называть деобфусцированными. К началу пятого шага имеется 21 различный файл биткода: оригинальный, 10 обфусцированных, 10 деобфусцированных. Множество полученных файлов биткода, соответствующих программе P, обозначим $B(P)$. На пятом шаге выполняется символьное исполнение каждого файла биткода из $B(P)$.

2.2. Проблемы анализа восстановленного биткода в расширенной модели

Утилиты трансляции McSema, используемые в реализации расширенной схемы, разделяют процесс трансляции кода бинарной программы на два этапа. На первом этапе строится высокоуров-

невое представление программы – граф потока выполнения, содержащий функции, инструкции базовых блоков и другую необходимую информацию. Такая работа выполняется с помощью сторонних утилит, например IDA Pro [20], DynInst [21]. На втором этапе полученное представление транслируется в биткод LLVM внутренней утилитой McSema. Таким образом, обработка программы в представлении машинного кода и подготовка представления удобного для трансляции в большей степени лежит на утилитах бинарного анализа, применяемых на первом этапе. На втором этапе трансляция выполняется практически напрямую: каждая инструкция машинного кода отображается в инструкцию промежуточного представления. Рассмотрим особенности первого и второго этапов трансляции.

При трансляции программ из представления на языке программирования высокого уровня в более низкоуровневое (промежуточное представление компилятора или машинные инструкции, см. шаг 1 на рис. 1) теряется часть информации о программе (например, информация о типах переменных, именах функции, интерфейсах классов). Поэтому при дизассемблировании и декомпиляции, в частности, возникает проблема отличия исполнимого кода от данных [22]. С одной стороны, все исполнимые файлы программы обычно имеют определенный формат [23, 24], в котором, как минимум, прописывается, какие ее участки являются исполнимым кодом, какие данными, где располагается точка входа в программу. Поэтому формат исполнимых файлов частично помогает разрешить проблему определения исполнимого кода. С другой стороны, остаются проблемы с идентификацией адресов переходов при косвенной адресации, определением границ функций. Отметим, что утерянная при компиляции информация важна в задачах обратного проектирования, в частности, при трансляции кода из низкоуровнего представления (машинный код) в высокоуровневое (промежуточное представление компилятора/псевдокод/язык высокого уровня, см. шаги 2,3 рис. 1). Эта информация позволяет более точно и быстро проанализировать код программы [25]. Однако часто бинарные файлы распространяются без нее, поэтому из-за отсутствия полной информации декомпиляция остается трудной задачей.

Применяемая на втором этапе трансляция, на практике реализуется путем интерпретации. Интерпретация подразумевает, что машинные инструкции не напрямую транслируются в инструкции целевого процессора, а транслируются в байт-код так называемой виртуальной машины. При трансляции создается глобальная структура, описывающая целевой процессор (все его регистры, флаги и другая специфичная для целевой архитектуры информация). Машинные инструкции заменяются аналогичными инструкциями уровня промежуточного представления, в которое транслируется программа, но при этом взаимодействие уже выполняется не со структурами настоящего процессора, а со структурой виртуальной машины описывающей процессор [26]. По этой причине оптимизация (деобфускация) биткода такой программы может не принести существенной разницы с исходным кодом, так как оптимизатор с высокой вероятностью не найдет соответствующего преобразования для оптимизации такой структуры программы (другими словами оптимизатор исследует код виртуальной машины, а не код анализируемой программы).

С целью определения влияния промежуточных этапов трансляции на оценку стойкости обфусцирующих преобразований построена упрощенная схема.

2.3. Упрощенная схема

В упрощенной схеме исключены этапы 2 и 3 расширенной схемы, показанной на рис. 1. Таким образом программа транслируется из исходного кода напрямую в биткод LLVM. Обфусцирующие преобразования выполняются корректно, так как работают на уровне промежуточного представления. Далее программа анализируется символьным интерпретатором KLEE. Упрощенная схема анализа изображена на рис. 2.

Благодаря такой организации в упрощенной схеме сохраняется большая часть исходной информации о программе, которая теряется при многочисленных преобразованиях и может быть

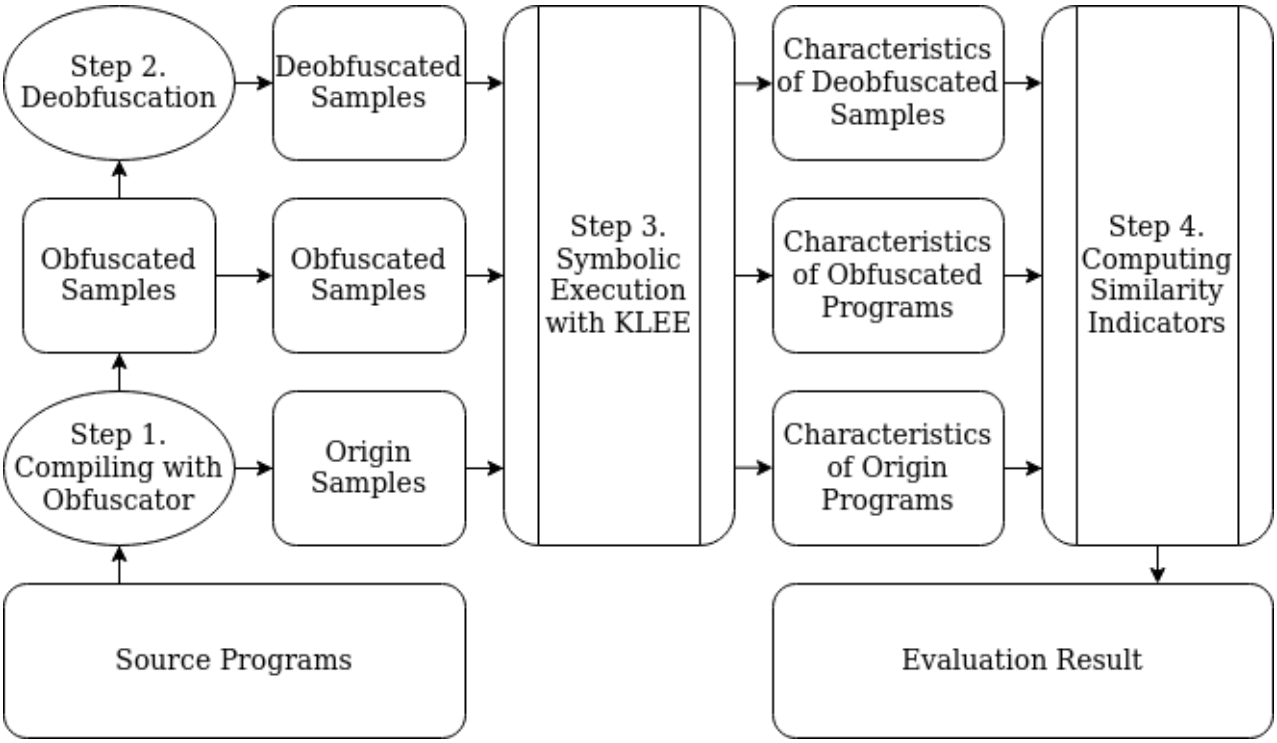

Fig. 2. An simplified scheme for finding the characteristics of symbolic execution of programs

Рис. 2. Упрощенная схема нахождения характеристик символьного исполнения программ

полезна во время ее анализа. Исключение этапов трансляции бинарного представления программы обратно в промежуточное представление LLVM (построение графа потока управления и трансляция полученного представления в биткод LLVM) также устраняет влияние усложненной структуры транслированной программы на работу деобфускатора.

3. Экспериментальное получение характеристик символьного исполнения

Для расширенной и упрощенной схем проведены эксперименты по нахождению характеристик символьного исполнения и вычислению показателей похожести, основанных на таких характеристиках. Эксперименты проводились на компьютере со следующими свойствами: процессор AMD Ryzen 2700U (4/8 ядер/потоков), объем оперативной памяти 16Gb, твердотельный накопитель SSD M.2 PCI-E NVMe 256Gb. Поскольку объем потребляемой оперативной памяти значительно увеличивается во время символьного выполнения, в дополнение к установленной памяти, был увеличен объем файла подкачки (swapfile) до 16Gb. Таким образом, общий объем памяти доступной символьному интерпретатору достиг 32Gb.

В следующих подразделах описываются используемые данные, параметры обфускации, искомые характеристики символьного исполнения, показатели похожести, а также описываются ограничения на эксперименты.

3.1. Описание данных

Для проведения экспериментального исследования была составлена выборка программ $\mathcal{P}$, написанных на языке C. За основу выборки был взят набор программ, использовавшийся в [27] для исследований влияния обфусцирующих преобразований на символьное исполнение. Все программы из $\mathcal{P}$ принимают на вход один параметр командной строки и обрабатывают его. Функционал программ включает: вычисление простых контрольных сумм, сортировку символов входного па-

раметра, поиск символа, проверку свойств заданного числа, преобразование входного параметра, в строку, в число различных систем исчисления, а также выполнение простой операции в зависимости от входного параметра. В случае успешного выполнения программа выводит на экран результат обработки параметра и возвращает 0, иначе возвращает код ошибки.

Большинство программ было изменено, т.к. в них обрабатывался лишь первый символ входного параметра, также добавлено несколько программ, реализующих простой алгоритм. Изначально выборка $\mathcal{P}$ состояла из 50-ти программ. Для ограничения времени, затраченного на символьный анализ $|\mathcal{P}| \cdot |B(P)|$ файлов биткода, было установлено максимальное время символьного исполнения равное 30 минутам. В результате применения схем, изображенных на рис. 1 и рис. 2, для программ из $\mathcal{P}$ было обнаружено, что некоторые программы не могут быть исследованы символьным интерпретатором в течение максимального указанного времени (30 минут). В этом случае время символьного исполнения обфусцированной и исходной программ совпадает и равно максимальному, а другие характеристики символьного исполнения не кажутся объективными, так как символьное исполнение не закончено.

Отметим, что существует также ограничение по объему оперативной памяти, установленной на вычислительном устройстве и доступной символьному интерпретатору. Несмотря на то, что этот объем оперативной памяти был увеличен за счет увеличения объема файла подкачки, остались программы, символьное исполнение которых досрочно завершалось операционной системой из-за того, что процесс символьного интерпретатора занимал всю доступную оперативную память. Такие программы также были исключены из выборки $\mathcal{P}$. Таким образом, в результате для оценки результатов эксперимента было отобрано 35 программ ($|\mathcal{P}| = 35$).

3.2. Параметры обфускации

Используемый обфусцирующий компилятор Hikari является дальнейшим развитием компилятора Obfuscator-LLVM, подробно описанного в [28], но предоставляет расширенный набор возможных преобразований кода программ на уровне промежуточного представления. Для выполнения эксперимента выбраны следующие обфусцирующие преобразования: O_{acd} – встраивание кода, препятствующего анализу структур классов (Anti-Class Dump), O_{sub} – замена инструкций эквивалентными (Substituion), O_{cff} – реализует обфусцирующее преобразование сглаживания графа потока управления программы [29] (Control Flow Flattening), O_{bcf} – встраивание непрозрачных предикатов с целью добавления ложных ветвлений и усложнения графа потока управления программы (Bogus Control Flow), O_{ind} – замена инструкций ветвления косвенными переходами (Indirect Branching), O_{fcw} – создание фиктивных функций-прокси, усложняющих анализ зависимостей между функциями (Function Call Wrapper), O_{fco} – обфускация инструкций вызова функций (Function Call Obfuscation), O_{sbb} – разбиение базовых блоков на семантически эквивалентную последовательность базовых блоков (Split Basic Block), O_{enc} – кодирование статических строк (String Encoding), O_{all} – примение всех обфусцирущих преобразований вместе (All Obfuscation Options). Множество всех обфусцирующих преобразований обозначим $\mathcal{O}$:

$$\mathcal{O} = \{O_{\text{acd}}, O_{\text{all}}, O_{\text{bcf}}, O_{\text{cff}}, O_{\text{enc}}, O_{\text{fco}}, O_{\text{fcw}}, O_{\text{ind}}, O_{\text{sbb}}, O_{\text{sub}}\}.$$

Некоторые преобразования предусматривают дополнительную параметризацию, например, указание вероятности применения к каждому базовому блоку. Для всех таких преобразований использовались параметры по умолчанию.

3.3. Характеристики символьного исполнения

Необходимым условием выбора характеристики является ее чувствительность к изменениям программы. Так как очевидно, что если характеристика не меняется при изменении программы (например, с помощью обфусцирующих преобразований), то по этой характеристике трудно

оценить влияние таких изменений на понимание программы. В качестве набора анализируемых характеристик символьного исполнения выбрано множество

$$\mathcal{F} = \{F_{\text{time}}, F_{\text{iexe}}, F_{\text{icov}}, F_{\text{bcov}}, F_{\text{ilen}}, F_{\text{tsmt}}, F_{\text{qsmt}}, F_{\text{blen}}, F_{\text{qall}}, F_{\text{mem}}\},$$

где F_{iexe} – количество исполненных инструкций в ходе анализа, F_{time} – время символьного исполнения, F_{icov} – процент покрытия инструкций в биткоде, F_{bcov} – процент покрытия инструкций перехода, F_{ilen} – общее количество инструкций в файле биткода, F_{tsmt} – общее время, затраченное решающим модулем SMT (Satisfiability Modulo Theories), F_{qsmt} – количество запросов к решающему модулю в среднем за одно исполнение, F_{blen} – количество операторов перехода в коде программы, F_{qall} – количество запросов к решающему модулю всего в ходе символьного анализа программы, F_{mem} – средний объем потребленной памяти в ходе символьного исполнения.

Для устранения влияния процессов операционной системы на результаты выполнения экспериментов, для каждой программы из $\mathcal{P}$ эксперимент выполняется t раз (на характеристики символьного исполнения могут повлиять такие процессы, как обновление компонент системы, фоновое исполнение служб, плановое выполнение задач). В настоящей работе параметр t равен 5. Для каждой характеристики $F(\in \mathcal{F})$ символом F^i обозначим значение этой характеристики после i-ой итерации алгоритма, а через $\overline{F}$ обозначим усредненное значение характеристики по всем t итерациям: $\overline{F} = (\sum_{i=1}^{t} F^i)t^{-1}$.

3.4. Показатели похожести программ

В [11] предложены показатели похожести программ для заданных характеристик. В настоящей работе для каждой характеристики $F(\in \mathcal{F})$, программы $P(\in \mathcal{P})$, обфусцирующего преобразования $O(\in \mathcal{O})$ вычисляются три меры похожести:

$$\delta^F(P, O(P)) = \frac{|\overline{F}(P) - \overline{F}(O(P))|}{\max\{\overline{F}(P), \overline{F}(O(P))\}}, \quad \delta^F(P, D(O(P))) = \frac{|\overline{F}(P) - \overline{F}(D(O(P)))|}{\max\{\overline{F}(P), \overline{F}(D(O(P)))\}},$$

$$\delta^F(O(P), D(O(P))) = \frac{|\overline{F}(O(P)) - \overline{F}(D(O(P)))|}{\max\{\overline{F}(O(P)), \overline{F}(D(O(P)))\}},$$

которые характеризуют качество обфусцирующего преобразования O в рамках характеристики F. Эти значения усредняются по множеству $\mathcal{P}$ для каждого $O(\in \mathcal{O})$ по каждому $F(\in \mathcal{F})$:

$$\overline{\delta}^F(\mathcal{P}, O(\mathcal{P})) = \frac{\sum_{P(\in\mathcal{P})} \delta^F(P, O(P))}{|\mathcal{P}|}, \quad \overline{\delta}^F(\mathcal{P}, D(O(\mathcal{P}))) = \frac{\sum_{P(\in\mathcal{P})} \delta^F(P, D(O(P)))}{|\mathcal{P}|},$$

$$\overline{\delta}^F(O(\mathcal{P}), D(O(\mathcal{P}))) = \frac{\sum_{P(\in\mathcal{P})} \delta^F(O(P), D(O(P)))}{|\mathcal{P}|}.$$

Для каждой характеристики F из $\mathcal{F}$ набор значений $\overline{\delta}^F(\mathcal{P}, O(\mathcal{P})), \overline{\delta}^F(\mathcal{P}, D(O(\mathcal{P}))), \overline{\delta}^F(O(\mathcal{P}), D(O(\mathcal{P})))$, где $O(\in \mathcal{O})$ позволяет выявить обфусцирующее преобразование, максимально изменяющее исходную программу в рамках рассмотренных показателей похожести. Для фиксированного O трудно сделать предположение о стойкости обфусцирующего преобразования, так как полученные нормированные показатели похожести для каждой F могут сильно различаться друг от друга. Требуется введение весов для характеристик из $\mathcal{F}$ для нахождения интегрального показателя похожести. С другой стороны, к решению этой задачи можно подойти с использованием методов машинного обучения. Для этого необходимо выявить наиболее изменчивые характеристики. С этой целью, значения $\overline{\delta}^F(\mathcal{P}, O(\mathcal{P})), \overline{\delta}^F(\mathcal{P}, D(O(\mathcal{P}))), \overline{\delta}^F(O(\mathcal{P}), D(O(\mathcal{P})))$, усредненные по множеству $\mathcal{O}$, обозначим

$$\overline{\delta}^F(\mathcal{P}, \mathcal{O}(\mathcal{P})), \overline{\delta}^F(\mathcal{P}, D(\mathcal{O}(\mathcal{P}))), \overline{\delta}^F(\mathcal{O}(\mathcal{P}), D(\mathcal{O}(\mathcal{P}))). \tag{1}$$

Показатель $\overline{\delta}^F(\mathcal{P}, \mathcal{O}(\mathcal{P}))$ характеризует среднее влияние обфусцирующего преобразования на значение характеристики F (эффективность). Показатель $\overline{\delta}^F(\mathcal{P}, D(\mathcal{O}(\mathcal{P})))$ характеризует способность деобфускатора приблизить обфусцированную программу к ее исходной версии (стойкость). А показатель $\overline{\delta}^F(\mathcal{O}(\mathcal{P}), D(\mathcal{O}(\mathcal{P})))$ характеризует способность деобфускатора нивелировать обфусцирующие преобразования (контрольное значение). На основании этих значений можно сделать выбор характеристик, которые могут применяться в модели оценки стойкости на основе машинного обучения.

Усредненные по множеству $\mathcal{F}$ значения

$$\overline{\delta}^{\mathcal{F}}(\mathcal{P}, O(\mathcal{P})), \overline{\delta}^{\mathcal{F}}(\mathcal{P}, D(O(\mathcal{P}))), \overline{\delta}^{\mathcal{F}}(O(\mathcal{P}), D(O(\mathcal{P}))) \tag{2}$$

позволяют выявить обфусцирующие преобразования, оказывающие наибольшее влияние на характеристики символьного исполнения программ.

4. Результаты экспериментов

Для рассматриваемых наборов $\mathcal{F}$, $\mathcal{O}$ и $\mathcal{P}$ результаты расчета усредненных значений (1) и (2) для расширенной и упрощенной схем представлены в таблицах 1, 2 и 3, 4 соответственно. Первый столбец в таблицах 1 и 3 показывает насколько обфускация в среднем изменяет программу от исходной в рамках конкретного показателя похожести, т.е. первый столбец показывает эффективность обфускации. Второй столбец показывает насколько деобфусцированная программа отличается от исходной в рамках того же показателя, т.е. этот столбец характеризует стойкость обфусцирующих преобразований в рамках этого показателя похожести. Экспериментально подтверждается, что третий столбец может определяться по значениям первых двух столбцов: если значения первого и второго столбцов близки, то ожидается, что значение третьего столбца будет близко к нулю.

В [30] предлагается значения показателей похожести, которые менее 0,05, считать незначительными. В настоящей работе эта оценка уточняется, основываясь на среднеквадратичном отклонении значений от среднего. Обозначим $M_{\mathcal{F}}$ и $\sigma_{\mathcal{F}}$ соотвественно среднее значение и среднеквадратичное отклонение показателей похожести усредненных по $\mathcal{O}$, а $M_{\mathcal{O}}$ и $\sigma_{\mathcal{O}}$ – соотвественно среднее значение и среднеквадратичное отклонение показателей похожести усредненных по $\mathcal{F}$. Вычисленные значения

$$\Delta_{\mathcal{O}} = \max\{M_{\mathcal{O}} - \sigma_{\mathcal{O}}, 0\}, \Delta_{\mathcal{F}} = \max\{M_{\mathcal{F}} - \sigma_{\mathcal{F}}, 0\}$$

предлагается рассматривать как границы, по которым можно определить значимость показателей. Если для выбранного показателя, его значение меньше соответствующей границы, показатель считается незначимым. Для таблицы 1 и 3 это означает, что значения символьной характеристики практически не изменяются (в среднем по $\mathcal{O}$). Аналогично, для таблицы 2 и 4 это означает, что преобразование O практически не влияет на все символьные характеристики выполнения (в среднем по $\mathcal{F}$).

Анализ таблиц 1 и 3 показывает, что наиболее изменчивыми как при обфускации, так и при деобфускации являются характеристики: F_{time}, F_{iexe}, F_{tsmt}, F_{qsmt}, F_{qall}, F_{mem}. Отметим, что эти характеристики являются наиболее изменчивыми, как в рамках расширенной схемы, так и в рамках упрощенной схемы. Так как общее время символьного исполнения включает в себя время, затраченное модулем SMT, а также общее количество запросов включает в себя запросы к модулю SMT, то из перечисленных выше характеристик оставлены только четыре характеристики: F_{time}, F_{iexe}, F_{qall}, F_{mem}.

Сравнение таблиц для расширенной схемы с таблицами для упрощенной схемы показывает, что значения показателей для расширенной схемы выросли как минимум в два раза по сравнению с упрощенной. Представляется, что это связано с проблемами описанными ранее (см. подраздел 2.2), в частности, с дополнительными этапами дизассемблирования и трансляции, которые исключены

Table 1. Similarity features averaged by $\mathcal{O}$ for extended scheme

Таблица 1. Усредненные по $\mathcal{O}$ показатели похожести расширенной схемы

F	$\overline{\delta}^F(\mathcal{P}, \mathcal{O}(\mathcal{P}))$	$\overline{\delta}^F(\mathcal{P}, D(\mathcal{O}(\mathcal{P})))$	$\overline{\delta}^F(\mathcal{O}(\mathcal{P}), D(\mathcal{O}(\mathcal{P})))$
F_{iexe}	**0.168**	0.165	0.019
F_{time}	**0.224**	0.222	0.045
F_{icov}	**0.108**	0.108	0.001
F_{bcov}	0.057	0.057	0.001
F_{ilen}	**0.109**	0.109	0.001
F_{tsmt}	**0.144**	0.147	0.02
F_{qsmt}	**0.182**	0.179	0.02
F_{blen}	0.039	0.039	0
F_{qall}	**0.203**	0.195	0.025
F_{mem}	**0.123**	0.122	0.012
$\boldsymbol{\Delta_{\mathcal{F}}}$	**0.076**	**0.076**	**0**

Table 2. Similarity features averaged by $\mathcal{F}$ for extended scheme

Таблица 2. Усредненные по $\mathcal{F}$ показатели похожести расширенной схемы

O	$\overline{\delta}^{\mathcal{F}}(\mathcal{P}, O(\mathcal{P}))$	$\overline{\delta}^{\mathcal{F}}(\mathcal{P}, D(O(\mathcal{P})))$	$\overline{\delta}^{\mathcal{F}}(O(\mathcal{P}), D(O(\mathcal{P})))$
O_{acd}	0.015	0.012	0.018
O_{all}	0.715	0.709	0.01
O_{bcf}	0.105	0.103	0.012
O_{cff}	0.019	0.022	0.016
O_{enc}	0.072	0.07	0.015
O_{fco}	0.016	0.011	0.014
O_{fcw}	0.026	0.025	0.021
O_{ind}	0.298	0.3	0.007
O_{sbb}	0.077	0.075	0.013
O_{sub}	0.014	0.019	0.02
$\boldsymbol{\Delta_{\mathcal{O}}}$	**0**	**0**	**0.01**

Table 3. Similarity features averaged by $\mathcal{O}$ for simplified scheme

Таблица 3. Усредненные по $\mathcal{O}$ показатели похожести упрощенной схемы

F	$\overline{\delta}^F(\mathcal{P}, \mathcal{O}(\mathcal{P}))$	$\overline{\delta}^F(\mathcal{P}, D(\mathcal{O}(\mathcal{P})))$	$\overline{\delta}^F(\mathcal{O}(\mathcal{P}), D(\mathcal{O}(\mathcal{P})))$
F_{iexe}	**0.06**	0.059	0.004
F_{time}	**0.128**	0.15	0.066
F_{icov}	**0.015**	0.015	0
F_{bcov}	**0.009**	0.009	0
F_{ilen}	**0.03**	0.03	0
F_{tsmt}	**0.021**	0.021	0.01
F_{qsmt}	**0.071**	0.07	0.001
F_{blen}	0.005	0.005	0
F_{qall}	**0.104**	0.103	0.001
F_{mem}	**0.025**	0.025	0.002
$\boldsymbol{\Delta_{\mathcal{F}}}$	**0.005**	**0.002**	**0**

из упрощенной схемы. Тем не менее, обе схемы могут использоваться для выбора характеристик, в виду того, что соответствующие наборы наиболее изменчивых показателей почти совпадают. Исключение – показатель F_{bcov}, который является изменчивым в рамках упрощенной схемы, в отличие

Table 4. Similarity features averaged by $\mathcal{F}$ for simplified scheme

Таблица 4. Усредненные по $\mathcal{F}$ показатели похожести упрощенной схемы

O	$\overline{\delta}^{\mathcal{F}}(\mathcal{P}, O(\mathcal{P}))$	$\overline{\delta}^{\mathcal{F}}(\mathcal{P}, D(O(\mathcal{P})))$	$\overline{\delta}^{\mathcal{F}}(O(\mathcal{P}), D(O(\mathcal{P})))$
O_{acd}	0.005	0.008	0.007
O_{all}	0.223	0.222	0.009
O_{bcf}	0.015	0.016	0.01
O_{cff}	0.016	0.019	0.009
O_{enc}	0.026	0.028	0.011
O_{fco}	0.006	0.01	0.007
O_{fcw}	0.007	0.01	0.007
O_{ind}	0.143	0.144	0.008
O_{sbb}	0.014	0.017	0.007
O_{sub}	0.014	0.015	0.008
$\Delta_{\mathcal{O}}$	**0**	**0**	**0.007**

от расширенной, но его значение близко к границе. Однако так как расширенная схема в большей степени соответствует процессу динамического анализа программы аналитиком, а расхождение с результатами упрощенной проявляется только в одном показателе, то в качестве показателей выбраны те, которые определены как изменчивые в рамках расширенной схемы.

Анализ таблиц 2 и 4 показывает, как обфусцирующие преобразования влияют на изменение показателей похожести программ. Применение всех обфускаций одновременно ожидаемым образом оказывает максимальный эффект. Из таблиц видно, что практически все обфусцирующие преобразования оказывают различной степени эффект на характеристики символьного исполнения в рамках рассматриваемых показателей похожести. Несмотря на то, что по значениям из таблиц 2 и 4 можно выделить обфусцирующие преобразования, оказывающие наибольший эффект, значения из этих таблиц характеризуют качество каждого обфусцирующего преобразования исключительно в рамках символьного исполнения – без учета характеристик этих преобразований, полученных при статическом анализе.

Заключение

Показатели похожести, полученные в результате статического анализа программы, можно использовать для оценки эффективности обфусцирующих преобразований, а также для определения их устойчивости к статическим методам деобфускации. Такие показатели могут быть построены на основе различных метрик сложности, рассчитанных, например, от исполнимого файла программы. В настоящей работе для моделирования динамического анализа предложено использовать символьное выполнение. Разработаны и построены две схемы получения характеристик символьного исполнения (расширенная и упрощенная). Для полученных характеристик предложены показатели похожести. С построенными схемами проведены эксперименты с целью определить изменчивые характеристики символьного исполнения с одной стороны, а с другой – сравнить результаты выполнения построенных схем между собой. Сравнение результатов экспериментов обеих схем показывает незначительное отличие в выборе набора изменчивых характеристик символьного исполнения (в упрощенной схеме характеристика F_{bcov} определяется как изменчивая в отличие от расширенной схемы). Также сравнение показывает, что значения показателей упрощенной схемы в среднем меньше значений таких же показателей расширенной схемы, так как символьное исполнение в упрощенной схеме требует меньше ресурсов (времени и памяти) для анализа программы. Тем не менее, характеристики (F_{time}, F_{iexe}, F_{qall}, F_{mem}) являются наиболее чувствительными к изменениям программы в рамках обеих схем. Эти четыре характеристики предлагается использовать

в векторе характеристик для оценки эффективности и стойкости обфускации на основе методов машинного обучения.

Отметим, что расширенная схема нахождения характеристик символьного исполнения является более универсальной по отношению к обфусцирующим преобразованиям, поскольку допускается применение обфускации на уровне машинных инструкций, а также такая схема в большей степени соответствует процессу динамического анализа программы аналитиком.

Дальнейшим направлением исследования является объединение показателей похожести, полученных с помощью статического анализа, с показателями похожести на основе выбранных характеристик символьного исполнения в вектор признаков для проведения экспериментов по оценке эффективности и стойкости обфусцирующих преобразований с помощью методов машинного обучения.

References

[1] C. Collberg and C. Thomborson, "Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection", *IEEE Transactions on Software Engineering*, vol. 28, pp. 735–746, Aug. 2002. DOI: 10.1109/TSE.2002.1027797.

[2] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B.Waters, "Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits", in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 2013, pp. 40–49. DOI: 10.1109/FOCS.2013.13.

[3] H. Xu, Y. Zhou, J. Ming, and M. Lyu, "Layered obfuscation: a taxonomy of software obfuscation techniques for layered security", *Cybersecurity*, vol. 3, p. 9, Apr. 2020. DOI: 10.1186/s42400-020-00049-3.

[4] C. Collberg, C. Thomborson, and D. Low, "A Taxonomy of Obfuscating Transformations", *Tech. Report, N 148, Dept. of Computer Science, Univ. of Auckland*, Jul. 1997.

[5] Y. Kanzaki, A. Monden, and C. Collberg, "Code Artificiality: A Metric for the Code Stealth Based on an N-Gram Model", in *2015 IEEE/ACM 1st International Workshop on Software Protection*, 2015, pp. 31–37. DOI: 10.1109/SPRO.2015.14.

[6] R. Mohsen and A. Pinto, "Algorithmic Information Theory for Obfuscation Security", in *Proceedings of the 12th International Conference on Security and Cryptography - Volume 1: SECRYPT, (ICETE 2015)*, 2015, pp. 76–87. DOI: 10.5220/0005548200760087.

[7] R. Mohsen and A. Pinto, "Evaluating Obfuscation Security: A Quantitative Approach", in *International Symposium on Foundations and Practice of Security*, Springer, Oct. 2015, pp. 174–192, ISBN: 978-3-319-30302-4. DOI: 10.1007/978-3-319-30303-1_11.

[8] M. Ceccato, M. Di Penta, J. Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella, "The Effectiveness of Source Code Obfuscation: an Experimental Assessment", in *2009 IEEE 17th International Conference on Program Comprehension*, May 2009, pp. 178–187. DOI: 10.1109/ICPC.2009.5090041.

[9] J. Siegmund, "Program Comprehension: Past, Present, and Future", in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5, Mar. 2016, pp. 13–20. DOI: 10.1109/SANER.2016.35.

[10] E. Avidan and D. Feitelson, "From Obfuscation to Comprehension", in *2015 IEEE 23rd International Conference on Program Comprehension*, May 2015, pp. 178–181. DOI: 10.1109/ICPC.2015.27.

[11] P. Borisov and Y. Kosolapov, "On the Automatic Analysis of the Practical Resistance of Obfusting Transformations", *Modeling and Analysis of Information Systems*, vol. 26, no. 3, pp. 317–331, Sep. 2019. DOI: 10.18255/1818-1015-2019-3-317-331.

[12] J. King, "Symbolic Execution and Program Testing", *Commun. ACM*, vol. 19, no. 7, pp. 385–394, Jul. 1976. DOI: 10.1145/360248.360252.

[13] B. Yadegari and S. Debray, "Symbolic Execution of Obfuscated Code", in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Oct. 2015, pp. 732–744. DOI: 10.1145/2810103.2813663.

[14] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation", in *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, ser. CGO '04, USA: IEEE Computer Society, 2004, pp. 75–86, ISBN: 0769521029.

[15] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-Based n-Gram Models of Natural Language", *Comput. Linguist.*, vol. 18, no. 4, pp. 467–479, Dec. 1992, ISSN: 0891-2017.

[16] N. Zhang, *Hikari – an improvement over Obfuscator-LLVM*, 2017.

[17] A. Dinaburg and A. Ruef, "Mcsema: Static translation of x86 instructions to llvm", in *ReCon 2014 Conference, Montreal, Canada*, 2014.

[18] C. Cadar and M. Nowack, "KLEE symbolic execution engine in 2019", *International Journal on Software Tools for Technology Transfer*, Jun. 2020. DOI: 10.1007/s10009-020-00570-3.

[19] S. Muchnick, *Advanced Compiler Design Implementation*. 1997, ISBN: 9781558603202.

[20] C. Eagle, *The IDA pro book: the unofficial guide to the world's most popular disassembler*, 2nd ed. No Starch Press, ISBN: 1593273959.

[21] G. Ravipati, A. R. Bernat, N. Rosenblum, B. P. Miller, and J. K. Hollingsworth, "Towards the Deconstruction of Dyninst", UW Madison, Tech. Rep., Jul. 2007, pp. 1–9.

[22] R. N. Horspool and N. Marovac, "An approach to the problem of detranslation of computer programs", *The Computer Journal*, vol. 23, no. 3, pp. 223–229, 1980.

[23] C. Visual and B. Unit, *Microsoft portable executable and common object file format specification*, 1999.

[24] H. Lu, *Elf: From the programmer's perspective*, 1995.

[25] J. Křoustek, P. Matula, J. Končický, and D. Kolář, "Accurate Retargetable Decompilation Using Additional Debugging Information", Jan. 2012.

[26] S. Dasgupta, S. Dinesh, D. Venkatesh, V. S. Adve, and C. W. Fletcher, "Scalable validation of binary lifters", *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 655–671, 2020.

[27] S. Banescu, C. Collberg, V. Ganesh, Z. Newsham, and A. Pretschner, "Code obfuscation against symbolic execution attacks", Dec. 2016, pp. 189–200. DOI: 10.1145/2991079.2991114.

[28] P. Junod, J. Rinaldini, J. Wehrli, and J. Michielin, "Obfuscator-LLVM – Software Protection for the Masses", May 2015, pp. 3–9. DOI: 10.1109/SPRO.2015.10.

[29] T. László and Á. Kiss, "Obfuscating C++ Programs via Control Flow Flattening", *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae. Sectio Computatorica*, vol. 30, no. 1, pp. 3–19, 2009.

[30] Y. Kosolapov and P. Borisov, "Similarity Features For The Evaluation Of Obfuscation Effectiveness", in *2020 International Conference on Decision Aid Sciences and Application (DASA)*, 2020, pp. 898–902. DOI: 10.1109/DASA51403.2020.9317301.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 1, 2021
journal homepage: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

Online Testing of Dynamic Reconfigurations w.r.t. Adaptation Policies

F. Dadeau[1], J. Gros[1], O. Kouchnarenko[1]

DOI: 10.18255/1818-1015-2021-1-52-73

[1]University Bourgogne Franche-Comté, CNRS, FEMTO-ST Institute, 15B avenue des Montboucons, 25030 Besançon, Cedex, France.

MSC2020: 93A30, 68Q60
Research article
Full text in Russian

Received March 3, 2021
After revision March 10, 2021
Accepted March 12, 2021

Self-adaptation of complex systems is a very active domain of research with numerous application domains. Component systems are designed as sets of components that may reconfigure themselves according to adaptation policies, which describe needs for reconfiguration. In this context, an adaptation policy is designed as a set of rules that indicate, for a given set of configurations, which reconfiguration operations can be triggered, with fuzzy values representing their utility. The adaptation policy has to be faithfully implemented by the system, especially w.r.t. the utility occurring in the rules, which are generally specified for optimizing some extra-functional properties (e.g. minimizing resource consumption).

In order to validate adaptive systems' behaviour, this paper presents a model-based testing approach, which aims to generate large test suites in order to measure the occurrences of reconfigurations and compare them to their utility values specified in the adaptation rules. This process is based on a usage model of the system used to stimulate the system and provoke reconfigurations. As the system may reconfigure dynamically, this online test generator observes the system responses and evolution in order to decide the next appropriate test step to perform. As a result, the relative frequencies of the reconfigurations can be measured in order to determine whether the adaptation policy is faithfully implemented. To illustrate the approach the paper reports on experiments on the case study of platoons of autonomous vehicles.

Keywords: component system; adaptation policy; online testing; usage model

INFORMATION ABOUT THE AUTHORS

Frederic Dadeau	orcid.org/0000-0003-0794-5819. E-mail: frederic.dadeau@univ-fcomte.fr Associate professor, Ph.D in Computer Science.
Jean-Philippe Gros	orcid.org/0000-0002-5159-9442. E-mail: jean-philippe.gros@univ-fcomte.fr Ph.D. candidate.
Olga Kouchnarenko correspondence author	orcid.org/0000-0003-1482-9015. E-mail: olga.kouchnarenko@univ-fcomte.fr Professor, Ph.D. in Computer Science.

Funding: RFBR, project No 17-07-01566.

For citation: F. Dadeau, J. Gros, and O. Kouchnarenko, "Online Testing of Dynamic Reconfigurations w.r.t. Adaptation Policies", *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 52-73, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 1, 2021
сайт журнала: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

Онлайн тестирование динамических реконфигураций по отношению к политикам адаптации

Ф. Дадо[1], Ж. Гро[1], О. Б. Кушнаренко[1]

DOI: 10.18255/1818-1015-2021-1-52-73

[1]Университет Бургундия Франш-Комтэ, НЦНИ, Институт ФЕМТО-СТ, 15Б Проспект Монбукон, 25030 Безансон ГПС, Франция.

УДК 519.7
Научная статья
Полный текст на русском языке

Получена 3 марта 2021 г.
После доработки 10 марта 2021 г.
Принята к публикации 12 марта 2021 г.

Самоадаптация сложных систем является активной областью теоретических и прикладных исследований, имеющей чрезвычайно широкий спектр применения.

Компонентно-ориентированные адаптивные системы разрабатываются на базе компонент, которые могут перенастроиться в соответствии с политиками адаптации, описывающими потребности в реконфигурировании. В этом контексте политика адаптации представляет собой набор правил, которые указывают для данного множества конфигураций, какие операции реконфигурирования могут быть инициированы, при этом их полезность представлена нечеткими значениями. Правила обычно разрабатываются для оптимизации некоторых нефункциональных свойств, например, минимизации потребления ресурсов, поэтому реализация системы с политиками адаптации должна быть точной, особенно по отношению к описанной в правилах полезности реконфигурирования.

С целью валидации поведения адаптивных систем в этой статье представлен модельно-ориентированный подход к тестированию, который направлен на создание больших наборов тестов для оценки случаев реконфигурирования и сравнения частоты этих случаев со значениями полезности, описанными в правилах адаптации. Этот процесс основан на модели использования системы в ее среде, для стимулирования ее реконфигурирований. Поскольку система может динамически изменять свою архитектуру, этот генератор тестов наблюдает за откликами системы на события и ее изменениями в режиме онлайн, чтобы решить каким будет следующий подходящий шаг теста. В результате относительные частоты реконфигурирований могут быть измерены, чтобы определить, правильно ли реализована политика адаптации. Чтобы проиллюстрировать предложенный подход, статья описывает эксперименты по моделированию поведения колонн автономных машин.

Ключевые слова: компонентно-ориентированные системы; политика адаптации; онлайн тестирование; модель использования

ИНФОРМАЦИЯ ОБ АВТОРАХ

Фредерик Дадо	orcid.org/0000-0003-0794-5819. E-mail: frederic.dadeau@univ-fcomte.fr доцент информатики.
Жан-Филипп Гро	orcid.org/0000-0002-5159-9442. E-mail: jean-philippe.gros@univ-fcomte.fr аспирант.
Ольга Борисовна Кушнаренко автор для корреспонденции	orcid.org/0000-0003-1482-9015. E-mail: olga.kouchnarenko@univ-fcomte.fr профессор информатики, доктор наук.

Финансирование: РФФИ, проект № 17-07-01566.

Для цитирования: F. Dadeau, J. Gros, and O. Kouchnarenko, “Online Testing of Dynamic Reconfigurations w.r.t. Adaptation Policies”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 52-73, 2021.

Введение

Контекст и мотивация Политики адаптации используются для управления адаптивными системами, указывая, когда могут быть инициированы операции реконфигурирования. В этой статье политики адаптации формально определены как наборы правил, которые указывают для данного множества состояний системы, также называемых конфигурациями, какие операции реконфигурирования могут быть инициированы. Таким образом, правила адаптации используются для определения корректного поведения системы, описывая для каждого реконфигурирования его область, предварительное условие и полезность. Предварительное условие и область могут содержать временные шаблоны [1], при этом область ограничивает применение реконфигурирования (например, на множестве конфигураций или при наступлении событий). Для указания полезности реконфигурирования, т.е. необходимости его активации для системы, используется нечеткое значение (например, high, medium, low), принятое в нечеткой логике. Например, можно указать, что после входа автономного автомобиля в туннель, если его запас энергии ограничен, то очень полезно отключить его компонент GPS.

Как следствие использования такого необязательного характера реконфигурирования возможны различные реализации каждой политики адаптации при условии, что результирующие исполнения адаптивной системы не нарушают никаких временных свойств. Верификация и валидация адаптивных систем является сложной задачей из-за этих множественных, хотя и корректных реализаций системы, подчиняющейся политикам адаптации с временными свойствами. Эти дополнительные артефакты – временные свойства и политики адаптации, дают возможность убедиться в корректности выполнений системы. Например, слежение за временными свойствами при выполнении системы позволяет удостовериться, что никакого нарушения этих свойств не обнаружено [2]. Однако для политик адаптации вопрос об установлении точности их соблюдения (их валидация) не является в настоящее время достаточно изученным. Это объясняется главным образом тем, что политики адаптации не носят предписывающий характер.

Как правило, методы валидации систем опираются на модели их поведения, используемые в качестве опоры для тестирования. Для адаптивных систем с политиками адаптации, это подразумевало бы необходимость выбора некоторой опорной реализации системы под политиками адаптации и требовало бы от системы того же самого выбора для соблюдения политики, а это слишком ограничено в случае описанных выше политик. В этой статье предложен подход для валидации реализации политик адаптации, который использует только сами политики в качестве опоры. Поскольку политики адаптации описывают потребности реконфигурирования по отношению к среде, в которой выполняется система, вместо модели поведения системы мы предлагаем использовать модель среды ее выполнения. Это позволяет принимать во внимание наступление событий в среде с последующими реконфигурированиями, вызванными этими событиями. Для каждого компонента такая модель среды описывает, какие события, в частности внешние, могут стимулировать реконфигурирования при его использовании. Поэтому она называется также моделью использования, и, как правило, она значительно меньше, чем модель самой системы. В предыдущей работе [3] описан подход к валидации системы по отношению к политике адаптации путем проверки того, что реконфигурирования, которые инициированы во время выполнения, соответствуют реконфигурированиям, разрешенным политикой адаптации.

Новые результаты Данная работа продолжает рассмотрение вопроса о соблюдении полезности правил политик адаптации при исполнении системы. Для валидации предлагается провести статистический анализ частот реконфигурирований. Для того, чтобы иметь значительное количество выполнений с осуществлением реконфигурирований, в статье предложено автоматически генерировать множество тестов. Тесты получены случайным исследованием моделей среды компонентов, описывающих, как стимулировать адаптивную компонентно-ориентированную систему,

в частности, посредством внешних событий. Поскольку реконфигурирования могут происходить без управления со стороны специалиста по тестированию, процесс формирования теста происходит в режиме онлайн: предлагаемый нами генератор тестов наблюдает реконфигурирования системы в ответ на события в ее среде и вычисляет следующий этап теста, который должен быть выполнен.

Одним из методов валидации на базе моделей является использование критериев покрытия тестами возможного поведения системы. Они обеспечивают, что случаи тестов дают хорошее представление о возможностях системы, покрывая их. Следуя этому методу, данная работа опирается на результаты [3], где были определены критерии тестового покрытия с акцентом на два артефакта, а именно: (а) политики адаптации и (б) множество временных свойств, которым система должна удовлетворять. Их можно использовать для иллюстрации различных сценариев выполнения. Эти артефакты позволяют обеспечить разнообразность трасс выполнений, которые будут регистрироваться. Таким образом обеспечивается уверенность в результатах анализа частот реконфигурирований. Данная работа содержит следующие новые материалы: (1) *онлайн* процесс формирования тестов, включающий модель среды использования системы, который направлен на создание больших множеств тестовых случаев, удовлетворяющих критериям покрытия, определенным в предыдущей работе; (2) мера встречаемости операций реконфигурирования, определенная на основе частотного анализа и направленная на подтверждение политик адаптации; (3) экспериментальное подтверждение этого процесса на примере автомобилей и колонн автомобилей на дороге.

Данная статья организована следующим образом. Раздел 1 содержит необходимые сведения о компонентно-ориентированных системах и политиках адаптации для них. Процесс генерации тестов, который опирается на модель использования среды системы описан в разделе 2. Основанная на частотном анализе оценка нечетких значений полезности реконфигурирований представлена в разделе 3. Результаты экспериментов по моделированию поведения колонны автономных машин даны в разделе 4. Наконец, библиографические заметки и заключение содержатся в разделе 5.

1. Пример и необходимые определения

1.1. Адаптивные системы на примере компонентно-ориентированной сети VANet

Пример 1. *Начнем с описания примера компонентно-ориентированной системы сети VANet (Vehicular Ad-Hoc Network), который показан на рисунке 1. Сеть VANet состоит из машин, которые либо находятся в одиночном режиме, либо организованы в колонны. Каждая колонна возглавляется лидирующей машиной. Любая машина в одиночном режиме (соло) может попросить присоединиться к колонне или принять решение о создании новой колонны с другой машиной в режиме соло. Каждая машина в колонне может попросить выйти из нее поскольку либо она добралась до места назначения, либо ей необходимо восполнить свои энергические ресурсы. Колонна может поменять лидера, если другая машина имеет большую автономность, либо из-за более удаленного пункта назначения другой машины. В окружении системы могут происходить внешние события. Например, новая машина может выехать на дорогу, или водитель может принять решение покинуть колонну из-за нового места назначения.*

Опираясь на этот пример, введем понятие компонентно-ориентированных систем. Как правило, компоненты – это объекты, которые можно объединять для создания сложных систем. Рассматриваемые компонентно-ориентированные системы являются иерархическими, имеющими два типа компонентов. Примитивные или базовые компоненты предоставляют информацию или услуги, в то время как составные компоненты содержат другие компоненты. Требуемые и предоставляемые интерфейсы являются точками взаимодействия между компонентами. Компонент реализует работу или сервис и поставляет их через предоставляемый интерфейс. Требуемый интерфейс – это интерфейс, необходимый компоненту для работы. Составные компоненты могут делегировать свои интерфейсы внутренним компонентам. Определения компонентов, интерфейсов, переменных, привязок для их совместимой сборки и прочих элементов модели могут быть найдены в [1, 4].

Состояние компонентно-ориентированной системы, также называемое конфигурацией, представляет собой набор вышеупомянутых архитектурных элементов (компонентов, интерфейсов и параметров) вместе с их типами и отношениями, чтобы структурировать и связать их. Реконфигурирование может быть рассмотрено как переход от одной конфигурации к другой.

Компоненты могут быть реализованы независимо друг от друга. Компонент может быть создан и инициирован несколько раз, например, как компоненты колонн и машин.

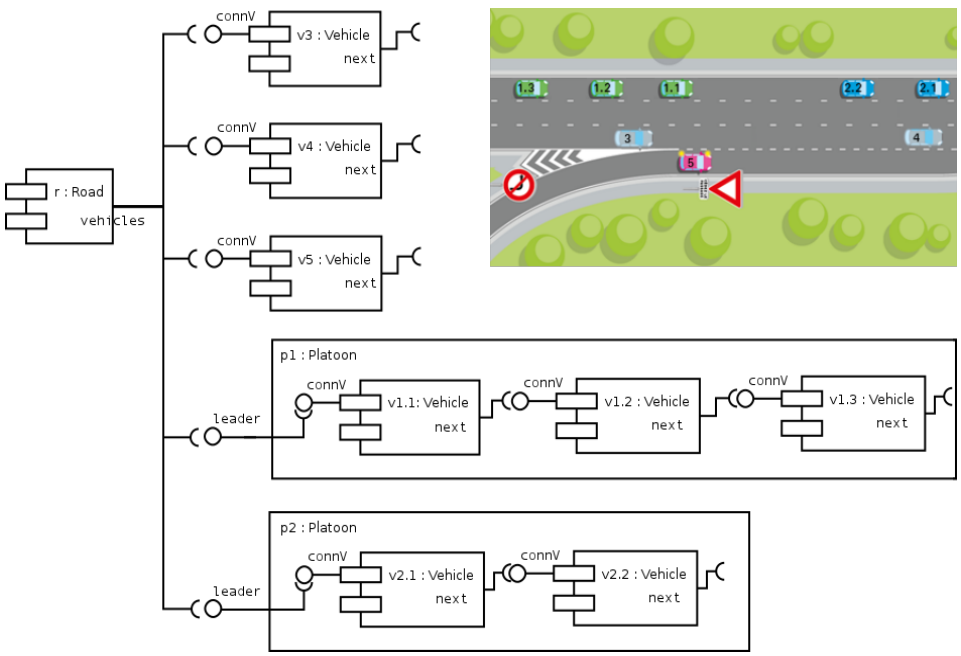

Fig. 1. Component architecture (a) vs. the considered VANet system state (b)

Рис. 1. Компонентная архитектура (a) напротив рассматриваемого состояния сети VANet (b)

Пример 2. *На рисунке 1 слева (a) изображено состояние VANet, соответствующее ситуации на рисунке справа (б). Составной дорожный компонент (Road) содержит компоненты машин (Vehicles), которые могут быть в колоннах (1.X, 2.X) или соло (3, 4, 5). Компоненты машин (Vehicles) соединены друг с другом через интерфейсы, позволяющие обмениваться информацией.*

Политики адаптации для таких систем рассматриваются как артефакты, которые описывают потребности адаптации системы, не являющиеся, однако, обязательными. Например, колонна *может* поменять лидера, когда он не обладает достаточными энергетическими ресурсами, либо когда у другого транспортного средства больше этих ресурсов, чем у лидера, и оно направляется к более удаленному пункту назначения.

1.2. Модели

Пусть $C = \{c, c_1, c_2, \ldots\}$ – множество конфигураций. Определим также множество CP свойств конфигураций, которые являются ограничениями на компоненты и отношения между ними. В частности, эти свойства используются для определения непротиворечивых конфигураций. Например, они используются для описания свойств глобальной инвариантности (например, на компоненты, интерфейсы и их привязки), которым архитектура на базе компонентов должна удовлетворять. Интерпретация $l : C \longrightarrow CP$ дает самую широкую конъюнкцию $cp \in CP$, которая является истинной на $c \in C$, что характеризует текущую конфигурацию наиболее точно.

Определим множество $\mathcal{R}$ операций реконфигурирования, которые делают компонентную архитектуру динамично изменяющейся. Реконфигурирования представляют собой комбинации примитивных операций, таких как инстанциирование/разрушение компонентов, включение/выключение компонентов, привязка/отмена привязки интерфейсов компонентов, запуск/остановка компонентов и т.д. Пусть $\mathcal{R}_{run} = \mathcal{R} \cup \Theta \cup \{run\}$ – множество операций, где $\mathcal{R}$ – конечное множество операций реконфигурирования, Θ – множество операций, запускаемых внешними событиями ($\mathcal{R} \cap \Theta = \emptyset$), и run – название универсального действия, представляющего все текущие операции вне реконфигурирования компонентно-ориентированной системы. Будем считать, что внешние события фиксируются системой и немедленно обрабатываются посредством запуска внутренних методов из Θ, как предложено в [5].

Определение 1 (Маркированная система с переходами (Labelled Transition System (LTS))). *Маркированная система с переходами – это структура $S = \langle C, C^0, \mathcal{R}_{run}, \longrightarrow, l \rangle$, где C множество конфигураций, $C^0 \subseteq C$ – множество начальных конфигураций, $\longrightarrow \subseteq C \times \mathcal{R}_{run} \times C$ – отношение реконфигурирования, и $l : C \longrightarrow CP$ всюду определенная функция интерпретации конфигураций.*

Обозначим $c \xrightarrow{ope} c'$, когда переход $(c, ope, c') \in \longrightarrow$. Переход $c \xrightarrow{ope} c'$ также называется *шагом.*

Определение 2 (Путь). *Для модели S определим путь (реконфигурирования) σ в S как последовательность шагов $c_0 \xrightarrow{ope_0} c_1, c_1 \xrightarrow{ope_1} c_2, ...,$ таких что $\forall i \geq 0.\ (c_i, ope_i, c_{i+1}) \in \longrightarrow$. Для данного пути σ, его трассой является слово $tr(\sigma) = ope_0.ope_1. ... ope_i ...$, составленное из имен операций в σ.*

Пусть c_i или $\sigma(i)$ обозначает i-ю конфигурацию на σ, начальную для i-го шага. Обозначение σ_i используется для суффикса, начинающегося от $\sigma(i)$, и σ_i^j обозначает отрезок пути между $\sigma(i)$ и $\sigma(j)$. Пусть Σ_S обозначает множество путей S, а Σ^f ($\subseteq \Sigma$) – множество конечных путей. Конфигурация c' достижима из c, когда существует путь $\sigma = c_0 \xrightarrow{ope_0} c_1, ... c_{n-1} \xrightarrow{ope_{n-1}} c_n$ в Σ^f такой, что $c = c_0$ и $c' = c_n$ с $n \geq 0$. Выполнение – это путь σ в Σ такой, что $\sigma(0) \in C^0$.

1.3. Политики адаптации

Политики адаптации состоят из правил, которые указывают какие операции реконфигурирования с соответствующим уровнем полезности могут быть инициированы для данного множества конфигураций. Операции реконфигурирования в политиках адаптации охраняются событиями, которые могут либо использовать свойства конфигураций, либо включать свойства временной логики. В этом разделе приведены элементы темпоральной логики линейного времени, основанной на временных шаблонах с областями их применения [1]. Эта логика, названная FTPL[1], используется далее в политиках адаптации.

Временные шаблоны FTPL используются для описания свойств над трассами выполнения, на которых ожидаются операции реконфигурирования. Обозначим $Prop_{FTPL}$ множество формул FTPL, соответствующих грамматике FTPL на рис. 2. В дополнение к свойствам конфигурации (*cp*) в *CP* из раздела 1, FTPL содержит внешние события и события из операций реконфигурирования, временные свойства (*temp*) вместе со свойствами трасс (*trace*), интегрированными во временные свойства.

Семантика FTPL была представлена в [4]. Она является классической для свойств конфигураций, например, как в PLTL[2]. Для других временных шаблонов она использует области или рамки применения (до, после, пока) свойств трасс или возникновения событий. Предположим, что внешние события (такие как события в [5]) происходят немедленно и могут рассматриваться как вызовы

[1]FTPL происходит от слияния TPL (Temporal Pattern Language) и "F"для логики первого порядка (First Order Logic) для ограничений на непротиворечивость при сборки компонентов.

[2]PTPL – Propositional Linear Temporal Logic.

<FTPL>	::=	<temp>|<events>|cp
<temp>	::=	**after** <events> <temp>
	|	**before** <events> <trace>
	|	<trace> **until** <events> | <trace>
<trace>	::=	**always** cp | **eventually** cp
	|	<trace> ∧ <trace>
	|	<trace> ∨ <trace>
<events>	::=	<event>,<events>
	|	<event>
<event>	::=	ope **normal** | ope **exceptional**
	|	ope **terminates** | ext

Fig. 2. FTPL syntax

Рис. 2. Синтаксис FTPL

методов, выполняемых внешними датчиками при обнаружении изменений в окружающей систему среде. Для каждого внешнего события ext, которое может произойти на пути выполнения σ, определим

1) предварительное условие cp_{ext}, представляющее собой формулу логики первого порядка над параметрами, указанными в вызове метода, соответствующего ext, и
2) утверждение $happens_\sigma$ со значением в $\{\top, \bot\}$.

Следуя за семантикой вычисления событий (Event Calculus [6, 7]) определим $happens_\sigma(cp_{ext}, i, j) = \top$, если существует по крайней мере одно вхождение события ext между i-м и j-м состояниями на пути σ такое, что $cp_{ext} = \top$; в противном случае будем считать, что $happens_\sigma(cp_{ext}, i, j) = \bot$.

Определение 3 (Семантика FTPL). *Пусть $\sigma \in \Sigma$. Семантика FTPL определена на $\Sigma \times Prop_{FTPL} \longrightarrow \mathbb{B}_2$ индуктивно по форме формул следующим образом:*

Для свойств конфигураций:		
$\sigma(i) \models cp$	, если	$l(\sigma(i)) \Rightarrow cp$
Для событий:		
$\sigma(i) \models ope$ **normal**	, если	$i > 0 \wedge \sigma(i-1) \neq \sigma(i) \wedge \sigma(i-1) \xrightarrow{ope} \sigma(i) \in \longrightarrow$
$\sigma(i) \models ope$ **exceptional**	, если	$i > 0 \wedge \sigma(i-1) = \sigma(i) \wedge \sigma(i-1) \xrightarrow{ope} \sigma(i) \in \longrightarrow$
$\sigma(i) \models ope$ **terminates**	, если	$\sigma(i) \models ope$ **normal** $\vee\ \sigma(i) \models ope$ **exceptional**
$\sigma(i) \models ext$	, если	$happens_\sigma(cp_{ext}, i, j) = \top$
$\sigma(i) \models event, events$	, если	$\sigma(i) \models event \vee \sigma(i) \models events$
Для свойств трасс:		
$\sigma \models$ **always** cp	, если	$\forall i.(i \geqslant 0 \Rightarrow \sigma(i) \models cp)$
$\sigma \models$ **eventually** cp	, если	$\exists i.(i \geqslant 0 \wedge \sigma(i) \models cp)$
$\sigma \models trace_1 \wedge trace_2$	, если	$\sigma \models trace_1 \wedge \sigma \models trace_2$
$\sigma \models trace_1 \vee trace_2$	, если	$\sigma \models trace_1 \vee \sigma \models trace_2$
Для временных свойств:		
$\sigma \models$ **after** $event\ temp$	, если	$\forall i.(i \geqslant 0 \wedge \sigma(i) \models event \Rightarrow \sigma_i \models temp)$
$\sigma \models$ **before** $event\ trace$	, если	$\forall i.(i > 0 \wedge \sigma(i) \models event \Rightarrow \sigma_0^{i-1} \models trace)$
$\sigma \models trace$ **until** $event$	, если	$\exists i.(i > 0 \wedge \sigma(i) \models event \wedge \sigma_0^{i-1} \models trace)$

Модель реконфигурирования S удовлетворяет свойству $\phi \in Prop_{FTPL}$, что обозначается как $S \models \phi$, *если* $\forall \sigma, (\sigma \in \Sigma_S \wedge \sigma(0) \in C^0 \Rightarrow \sigma \models \phi)$.

Подробная формальная семантика FTPL может быть найдена в [4]. Приведенное выше напоминание FTPL направлено на иллюстрацию временных шаблонов, интегрированных в политики адаптации. Напомним, что эти шаблоны делают политики более выразительными, чем представленные в [8], использующие только свойства состояний и инвариантов над конфигурациями.

Пример 3. *Рассмотрим свойство FTPL для системы VANet:*

φ1: ***after*** Join normal ***always*** $Battery \geq 5 \wedge Dist > 2$ ***until*** Quit normal

Это свойство описывает, что после присоединения автомобиля к колонне его автономность и оставшееся расстояние до цели превышают соответственно 5% и 2 км, до тех пор пока он не покинет колонну. Это относится к каждому транспортному средству.

Определим теперь политики адаптации с временными шаблонами, используя как базу определения в [8] и упрощая обозначения из [4].

Определение 4 (Политика адаптации)**.** *Пусть S – маркированная система с переходами LTS с ее реконфигурированиями $\mathcal{R} \cup \Theta$, CP – множество свойств конфигурации, маркирующих ее состояния, и $Ftype$ – конечное множество нечетких типов. Политика адаптации определяется как $A = \langle R_N, R_R \rangle$, где:*

- *$R_N \subseteq \mathcal{R} \cup \Theta$ – непустое конечное множество операций реконфигурирования,*
- *R_R – конечное множество правил адаптации с элементами $(b, g, ir) \in Prop_{FTPL} \times CP \times I$, где $ir \in I \subseteq R_N \times Ft$ – пара, связывающая нечеткое значение полезности f из области $Ft \in Ftype$ с операцией реконфигурирования $ope \in R_N$.*

Отметим, что I является отношением, поскольку инженер может связать различные нечеткие значения с одним реконфигурированием: иногда полезность высокая (`high`), иногда она низкая (`low`). Кроме того, некоторые операции реконфигурирования из $\mathcal{R} \cup \Theta \setminus R_N$ могут не затрагиваться разработанными политиками и не иметь связанных значений полезности.
Каждое правило адаптации в R_R строится с использованием нескольких ключевых слов: **when** b **if** g **then utility of** ope **is** f. FTPL свойство b после ключевого слова `when` определяет временные рамки операции реконфигурирования шаблоном FTPL. В этой части ключевое слово `and` используется для построения сложных шаблонов. Затем после ключевого слова `if` описывается предикат g на конфигурации системы. Он определяет множество конфигураций в пределах временных рамок, где может быть инициировано реконфигурирование. После этого, чтобы привязать полезность к операции реконфигурирования, $ope \in R_N$ ассоциируется с ее полезностью $f \in Ft$ с помощью ключевых слов `then utility of` и `is`. Полезность определяется нечетким значением (например, `high, medium, low`).

when after *Join normal* until *Quit normal* **and** *VehicleId.battery < 33*
if *state = leader* **then utility of** *PassRelay* **is** *high*

when after *Join normal* until *Quit normal* **and** *VehicleId.battery > Leader.battery*
if *state = platooned* **then utility of** *GetRelay* **is** *medium*

Пример 4. *Рассмотрим два правила адаптации из политики для каждого транспортного средства, включающие реконфигурирования PassRelay и GetRelay. Интуитивно вышеуказанные правила применяются ко всем транспортным средствам и используются для определения тех моментов, когда может произойти смена между лидером и другим автомобилем в колонне. В первом случае реконфигурирование PassRelay происходит, когда батарея лидера недостаточно заряжена. Во втором случае реконфигурирование GetRelay инициируется, когда батарея транспортного средства имеет больше ресурсов, чем у лидера. Заметим, что добавление/удаление компонента не является обязательным, а скорее предлагается с некоторым значением полезности (например, из множества $Ft = \{$ high, medium, low $\}$). Следовательно, нет никакой гарантии, что система в конечном итоге выполнит рассматриваемую операцию реконфигурирования.*

Определим теперь влияние политики адаптации на поведение системы S. Для S и конечного множества AP политик адаптации определим маркированную систему с переходами $S \triangleleft AP$, которая обозначает S в присутствии политик адаптации из множества AP.

Определение 5 (LTS с политиками адаптации)**.** *Ограничение модели S политиками адаптации AP определяется как маркированная система с переходами $S \triangleleft AP = \langle C_{\triangleleft AP}, C^0_{\triangleleft AP}, \mathcal{R}_{run}, \longrightarrow, l \rangle$, где $C_{\triangleleft AP}$ – наименьшее множество конфигураций такое, что если $c \in C$ и $A \in AP$, то $c_{\triangleleft A} \in C_{\triangleleft AP}$, $\mathcal{R}_{run} \cap (\cup_{A \in AP} R_N) \neq$*

$\emptyset, l : \mathcal{C}_{\lhd AP} \longrightarrow CP$ является всюду определенной функцией интерпретации конфигураций, и для каждой операции $ope \in \mathcal{R}_{run}$ отношение реконфигурирования $\longrightarrow \in \mathcal{C}_{\lhd AP} \times \mathcal{R}_{run} \times \mathcal{C}_{\lhd AP}$ определено как наименьшее множество элементов $(c_\lhd, ope, c'_{\lhd A})$, удовлетворяющих следующим правилам:

$$[ACT1] \quad \frac{c \xrightarrow{ope} c'}{c_{\lhd A} \xrightarrow{ope} c'_{\lhd A}} \quad (ope \in \bigcup_{A \in AP} R_N) \wedge b \wedge g$$

$$[ACT2] \quad \frac{c \xrightarrow{ope} c'}{c_{\lhd A} \xrightarrow{ope} c'_{\lhd A}} \quad ope \notin \bigcup_{A \in AP} R_N \ .$$

Это определение означает, что переходы S под AP являются результатом выполнения операций реконфигурирования в соответствии с политиками адаптации (правило *[ACT1]*) или реконфигурирований, незатронутых политиками адаптации (правило *[ACT2]*).

Что касается оценки условия правила *[ACT1]*, а именно его части b, то это может быть сделано децентрализовано, используя прогрессивную семантику, как в [2, 9].

Политики адаптации должны точно соблюдаться системой, особенно по отношению к значениям полезности, встречающимся в правилах, которые обычно определяются для оптимизации некоторых нефункциональных свойств (например, для минимизации потребления ресурсов). С этой целью могут использоваться различные соотношения, например, отношение уточнения [10], симуляции [11], правильного исполнения [12] и т. д. Однако установление этих соотношений для сравнения реализаций или спецификаций в соответствии с политиками адаптации может быть трудной проблемой, которая становится более сложной при рассмотрении событий из окружения системы. Эта проблема является, как правило, неразрешимой для систем с бесконечным числом состояний. Вместо этого в следующем разделе мы предлагаем методологию, основанную на модели среды использования для валидации реализаций адаптивных систем с политиками адаптации.

2. Онлайн-тестирование с моделью использования

В этом разделе сначала обосновывается использование онлайн тестирования, затем описываются модели использования и алгоритм генерации тестов, необходимые для вычисления случаев тестов.

2.1. Онлайн процесс генерации тестов

Адаптивные системы реагируют на внешние события в соответствии с политиками адаптации, которые предоставляют рекомендации для выполнения реконфигурирований системы. Существует множество различных корректных реализаций для системы при заданных политиках адаптации. Назовем частотой реконфигурирований отношение между количеством выполненных реконфигурирований и количеством возможностей их применения. При рассмотрении трасс выполнения и относительных частот реконфигурирований, если операция реконфигурирования с высокой полезностью имеет более низкую относительную частоту, чем операция с низкой полезностью, это означает, что либо реализация системы неправильно учитывает значение полезности, или же необходимо изменить его определение в политике адаптации. Поскольку возможны различные реализации системы при заданных политиках адаптации, иметь единственную модель всех таких реализаций представляется проблематичным: это потребовало бы сделать выбор в отношении поведения системы и заставило бы различные реализации соответствовать этому выбору.

Чтобы избежать описания моделей реализации систем (множество LTS) и их валидации по отношению к спецификации (LTS с политиками адаптации), в данной статье предлагается рассмотреть модель использования тестируемой системы [13]. Таким образом, в нашем случае речь пойдет не о модели адаптивной системы в рамках политик адаптации, а о модели ее среды использования [14], ориентированной на происходящие события, на которые система может реагировать в соответствии

с политиками адаптации. Такие модели обычно меньше моделей, описывающих систему в целом, и инженерам по валидации проще их проектировать вручную.

Напомним, что система может меняться в ответ на внешние события из Θ (также рассматриваемые как контролируемые события, например, посылаемые инженером в целях валидации), ее состояние также изменяется. Это может инициировать внутренние (неконтролируемые) события, которые регистрируются для наблюдения со стороны [3]. Это влияет на генерацию следующих возможных внешних событий, которые будут отправлены в систему, поскольку реконфигурирование может предотвратить происхождение некоторых событий в системе. Таким образом, процесс генерации должен избегать формирования тестов с неактуальными действиями реконфигурирования. Он должен также учитывать внутренние события, которые будут только наблюдаемыми, и соответствующим образом корректировать следующий шаг тестирования. Как следствие, модель среды также принимает во внимание внутренние события, которые соответствуют реконфигурированиям, которые могут произойти в системе.

Чтобы иметь возможность обрабатывать такие ситуации, в предлагаемом подходе онлайн тестирования: (а) каждый шаг теста выполняется непосредственно на тестируемой системе (SUT для System Under Test), (б) изменение тестируемой системы отслеживается алгоритмом генерации тестов и учитывается для генерации следующего шага теста. Предлагаемый процесс показан на рис. 3. Модели использования компонентов исследуются алгоритмом для вычисления следующих шагов теста. Они являются входом для генератора теста (1), который соединен с SUT. В режиме онлайн алгоритм случайным образом выбирает компоненту и вычисляет путем исследования ее модели использования следующий шаг теста (2), а именно событие, которое будет послано тестируемой системе. SUT может отреагировать на событие и выполнить реконфигурирование, которое будет зарегистрировано в трассе выполнения (3). Для вычисления последующего шага генератор тестов также рассматривает обновленную трассу выполнения, которая используется для обновления текущего состояния генератора (4). Параллельно трассы выполнения используются для обнаружения возможного нарушения временных свойств в политиках адаптации, как описано в [2], или для обнаружения несанкционированного реконфигурирования, как описано в [3]. Эти аспекты были уже описаны раньше и не находятся в центре внимания настоящей статьи.

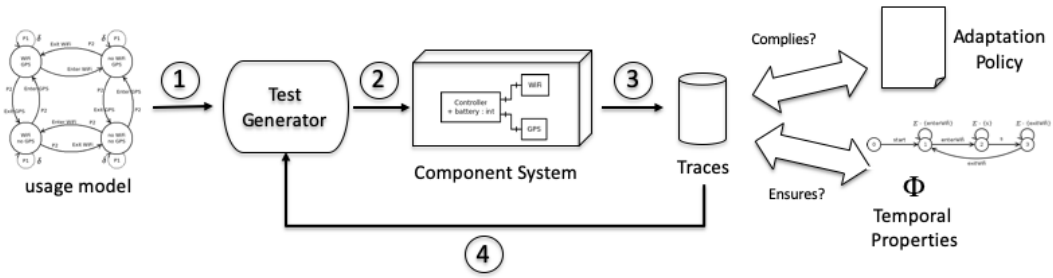

Fig. 3. Online test generation process

Рис. 3. Онлайн процесс генерации тестов

2.2. Тестовые случаи и пути реконфигурирования

Определим тестовые случаи как последовательности контролируемых событий, которые отправляются в тестируемую систему с некоторой периодичностью. В случае адаптивных систем, которые являются гибридными системами, где дискретное и непрерывное время смешиваются, может быть принят дискретизированный подход, подобный описанному в [15]. В этом случае события,

[3] Эти два множества являются непересекающимися, как описано в разд. 1 для операций реконфигурирования и отражено в грамматике FTPL.

используемые для стимулирования системы, посылаются с заданной периодичностью, символизируемой тактами, длительность которых параметризована. Кроме того, введем понятие задержки, обозначенное δ, которая состоит в том, чтобы не предпринимать никаких действий на SUT.

Пример 5 (Тестовый случай для сети VANet). *Для валидации функционирования сети VANet предлагается следующий тестовый случай:*

$$\delta; \delta; V1.join; \delta; ...; \delta; V2.join; \delta; ...V2.quit; ...$$

Он отображает частые вхождения задержки δ, представляющее период времени, в течение которого состояние системы изменяется без какого-либо запроса со стороны окружающей среды, например, в это время уменьшается ресурс аккумуляторных батарей транспортных средств. В этой последовательности join является запросом от автомобиля на вхождение в колонну, а quit отражает выход из колонны.

При выполнении тестового случая на SUT создается трасса реконфигурирования. Затем она может быть проанализирована, чтобы установить, соответствует ли система различным спецификациям, а именно соблюдаются ли политики адаптации и ожидаемые временные свойства.

Пример 6 (Путь реконфигурирования по отношению к внешним событиям). *Рассмотрим две последовательности, показанные на рис. 4: последовательность внешних событий (а именно тестовый случай), состоящий из запросов join и forceQuit в строке верхнего уровня, и путь реконфигурирования (а именно трасса выполнения) системы, сгенерированный в ответ на внешние события, в строке нижнего уровня. Последовательности внешних событий выбираются с одинаковой частотой, а пути реализации сообщают о приеме событий и инициировании реконфигурирований в ответ на эти события.*

В сети VANet могут происходить следующие внешние события. Событие join происходит, когда транспортные средства достаточно близки для слияния в колонну, система может либо принять объединение и вызвать реконфигурирование acceptJoin, либо решить отказаться от запроса refuseJoin. Когда водитель решает выйти из колонны (внешнее событие по отношению к автономной машине), это вызывает событие forceQuit, и система реагирует на него реконфигурированием (quit). Система также может реагировать на внутренние события, например, лидер может поменяться при реконфигурировании (getRelay).

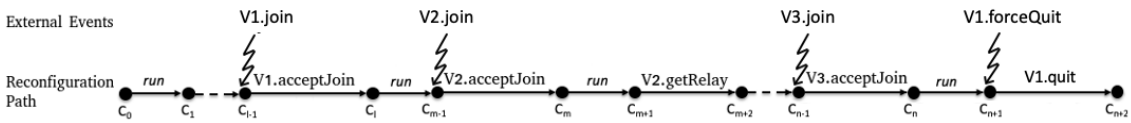

Fig. 4. A reconfiguration path according to external events

Рис. 4. Путь реконфигурирования по отношению к внешним событиям

2.3. Модели использования для онлайн-тестирования

В этом разделе описывается артефакт, который будет использоваться для генерации случаев теста: модели использования компонентов. Такие модели, описанные в [14], в основном направлены на определение различных событий, которые могут происходить в среде системы. Эти события являются контролируемыми и могут быть отправлены в тестируемую систему, на которую она реагирует. Возможно также отправление задержки δ, представляющей отсутствие внешних событий в течение заданного периода времени.

Наиболее распространенным подходом определения таких моделей является использование вероятностных автоматов. В этой статье предложено связать их с использованием компонентов. Интуитивно, каждому состоянию такого автомата соответствует набор событий, которые могут быть инициированы в нем, т.е. на которые компонент способен реагировать. Они будут использоваться генератором теста для создания тестовых случаев.

Как указано в разделе 2.1, неконтролируемые операции реконфигурирования могут быть зарегистрированы на трассах выполнения. Связанные с ними события также учитываются в модели использования для обнаружения и фиксирования изменения текущего состояния компонента, что необходимо для отправки подходящих событий в тестируемую систему.

Определение 6 (Вероятностный автомат модели использования)**.** *Для каждого компонента C его модель использования определена как детерминированный вероятностный автомат $\mathcal{A}_C = \langle Q, q_0, E_\delta \cup O, F, P \rangle$, где Q – множество состояний, $q_0 \in Q$ – начальное состояние, E_δ – набор контролируемых внешних событий*[4] *вместе с δ для задержки (отсутствия внешних событий), O – множество неконтролируемых событий, состоящее из операций реконфигурирования, наблюдаемых на системе* [5]*, F – отношение перехода $F \in Q \times (E_\delta \cup O) \times Q$, и P – вероятность*[6] *перехода $P : Q \times E_\delta \longrightarrow [0;1]$ такая, что $\forall q \in Q \Longrightarrow \Sigma_{e \in E_\delta} P(q, e) = 1$.*

Каждому компоненту соответствует вероятностный автомат, будь то составной виртуальный компонент высокого уровня (например, дорога в нашем примере), или базовый компонент (например, транспортное средство).

Пример 7 (Модели использования транспортных средств в VANet)**.** *В примере VANet каждое транспортное средство реагирует на три внешних события: enter, обозначающее, что транспортное средство въезжает на дорогу; join, которое обозначает, что транспортное средство запрашивает присоединение к другому транспортному средству или существующей колонне; и forceQuit, которое указывает, что машина покидает колонну из-за вмешательства водителя. Кроме того, имеются три неконтролируемых, но наблюдаемых события относительно автомобилей: leave означает, что транспортное средство покидает дорогу; acceptJoin представляет реконфигурирование колонны для принятия запроса от машины; и quit представляет операцию реконфигурирования для выхода машины из колонны. Модель использования для этого примера показана на рис. 5. Предполагается, что события происходят с определенной инженером вероятностью (число в скобках). На рисунке δ-маркированные переходы представляют задержку в одну единицу времени, а другие переходы с метками представляют либо контролируемые события create, join и forceQuit, либо наблюдаемые события leave, acceptJoin и quit, обозначающие реконфигурирования в системе. Наблюдаемые события представлены пунктирными линиями.*

На основе моделей использования отдельных компонентов можно построить композицию моделей использования компонентов, используя, например, операции инкапсуляции и рафинирования компонентов, композицию расширенных автоматов интерфейсов или иерархические input/output автоматы и т.д. Чтобы избежать такого построения, требующего времени и объемов памяти, можно работать децентрализованно, например, как это сделано в [2, 9] для оценки свойств во время выполнения. В нашем подходе модели использования компонентов обрабатываются децентрализованно.

[4] вызывающие реконфигурирования из Θ, см. раздел 1.2.

[5] реконфигурирования из $\mathcal{R}$, см. раздел 1.2.

[6] Предполагается, что если никакой переход из текущего состояния не отмечен событием, то вероятность этого события равна 0.

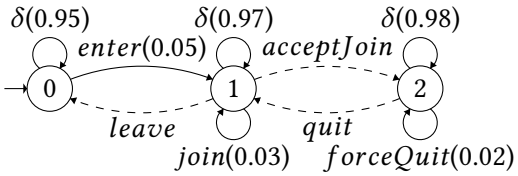

Fig. 5. Usage model of the VANet vehicle

Рис. 5. Модель использования автомобиля в сети VANet

2.4. Генерация тестов на базе моделей использования

Процесс генерации тестов опирается на модели использования компонентов, составляющих SUT, как представлено на рис. 3. Поскольку компоненты могут эволюционировать неконтролируемым образом, онлайн подход также опирается на трассу выполнения системы, регистрирующую происходящие реконфигурирования. Тестовый случай определяется как последовательность событий, полученных при прохождении переходов вероятностных автоматов моделей использования компонентов системы. Пусть $A_\delta(C)$ – модель использования, связанная с компонентом C. Для данной компонентно-ориентированной системы тестовый случай, основанный на множестве моделей использования, представляет собой конечную последовательность событий $C_0^j.e_0^j; C_1^k.e_1^k; ...C_n^l.e_n^l$ (длины $n+1$), в которой на i-м шаге теста запускаемое событие e^j связано с компонентом C^j и имеет вероятность $P_{C^j}(q^j, e^j) > 0$.

Для вычисления множества тестовых случаев, также называемого набором тестов, случайный обход Маркова [16] выполняется по вероятностным моделям использования компонентов, при этом исследование моделей использования компонентов выполняется случайным образом. Предлагаемый алгоритм генерации тестов, представленный алгоритмом 1, повторяет следующие шаги до тех пор, пока не будет достигнута максимальная длина тестовых примеров. Сначала случайным образом выбирается один из компонентов (строка 6). Текущее состояние автомата обновляется по отношению к трассе выполнения системы (строка 7). Этот этап представляет собой учет различных операций реконфигурирования (на базе наблюдаемых событий), которые могли произойти с момента последнего выбора компонента. Затем среди переходов, исходящих из текущего состояния, случайным образом выбирается переход и связанная с ним вероятность (строка 8). Если переход не является задержкой (строка 9), событие передается в тестируемую систему через рассматриваемый компонент (строка 10-12), и текущее состояние обновляется. В конце (строка 14) при учете реального времени выполнения системы, процесс генерации теста ожидает (в зависимости от частоты дискретизации событий), прежде чем перейти к вычислению следующего шага теста.

Этот алгоритм может быть использован для генерации наборов тестов произвольного размера и тестовых случаев произвольной длины. Это позволяет провести оценку того, правильно ли реализованы нечеткие значения, связанные с операциями реконфигурирования в политике адаптации. С этой целью производится анализ относительных частот реконфигурирований, описанный в следующем разделе.

3. Соблюдение политик адаптаций

Нашей целью является оценка соответствия реализации политики адаптации по отношению к полезности реконфигурирования, определенной в спецификации нечеткими значениями. В предыдущей работе [3] представлено использование правил политик адаптации в качестве метрики (критериев) покрытия модели для оценки релевантности набора тестов. В настоящей работе представлено новое дополнительное использование такой метрики покрытия, чтобы оценить, как реализация

системы соблюдает инициирования операций реконфигурирования при применении правил политик адаптации.

Algorithm 1. Algorithm for online test case generation

1: **for all** A_C **do**
2: state(A_C) ⟵ $q_0(A_C)$
3: **end for**
4: i ⟵ 0
5: **while** i < n **do**
6: C ⟵ selectComponent()
7: state(A_C) ⟵ update(A_C, $trace$)
8: e ⟵ pick(E_δ, state(A_C))
9: **if** $e \neq \delta$ **then**
10: C.send(e)
11: state(A_C) ⟵ update(A_C, [e])
12: i ⟵ i + 1
13: **end if**
14: await()
15: **end while**

Algorithm 1. Алгоритм генерации тестов в онлайн

3.1. Покрытие, пригодность и частота правила

Покрытие правила Начнем с определения функции, которая подсчитывает количество выполнений правила на заданном пути реконфигурирования.

Определение 7 (Число выполнений правила)**.** *Пусть σ – путь реконфигурирования, и $actual_{\sigma(i)}$ – фактическая операция реконфигурирования, происходящая в состоянии $\sigma(i)$. Число выполнений правила $r \in R_R$ на σ определено следующим образом:*

$$\#actual^r(\sigma) = \sum\nolimits_i f_a^r(\sigma(i)),$$

где $f_a^r(\sigma(i))$ – характеристическая функция предиката $actual_{\sigma(i)} \in dom(I)$[7], в котором I – отношение, связывающее операции реконфигурирования с нечеткими значениями.

Эта информация может использоваться для оценки того, охватывается ли данное правило тестовым случаем или, в более общем плане, набором тестов. Однако если правило никогда не охватывается, это может происходить по нескольким причинам. Во-первых, тестовые примеры не достигают конфигурации, в которой применимо реконфигурирование. В этом случае набор тестов должен быть уточнен, чтобы охватить правило. Во-вторых, некоторые части правил адаптации могут быть написаны неправильно и содержать слишком строгий или даже недостижимый/недопустимый триггер (свойство b или охранный предикат g в определении 4). Для таких случаев мы предлагаем подсчитать количество раз, когда эти различные части правил удовлетворяются.

Пригодность правила Определим теперь число переключений условия инициирования правила как число конфигураций, в которых свойство запуска правила становится верным.

Определение 8 (Количество инициирований правила)**.** *Пусть σ – путь реконфигурирования, и $trig_{\sigma(i)}$ – множество операций реконфигурирования, которые могут быть инициированы в состоянии $\sigma(i)$, то есть b-триггеры которых истинны в $\sigma(i)$. Число инициирования правила $r \in R_R$ на σ определяется следующим образом: $\#trig^r(\sigma) = \sum_i f_t^r(\sigma(i))$, где $f_t^r(\sigma(i))$ – характеристическая функция предиката*

$$r \in trig_{\sigma(i)} \wedge r \notin trig_{\sigma(i-1)} \wedge actual_{\sigma(i-1)} \notin dom(I).$$

[7] которая равна 1, если предикат истинен, и 0 в противном случае

Поскольку реконфигурирование может происходить в области применения правила, пока свойство b является верным, подсчет количества таких конфигураций не дает информации о реальной ситуации с реконфигурированием. Поэтому для данного пути σ определим число пригодности (число допусков) правила, которое является числом состояний на пути σ, когда правило стало пригодным. Назовем такие правила, чьи триггеры b и охранные предикаты g истинны в конфигурации $\sigma(i)$, приемлемыми в данном состоянии.

Определение 9 (Количество пригодности (допусков) правила)**.** *Пусть σ – путь реконфигурирования, а $elig_{\sigma(i)}$ является множеством правил, которые могли бы быть применены в состоянии $\sigma(i)$. Число пригодности правила $r \in R_R$ определяется как $\#elig^r(\sigma) = \sum_i f_e^r(\sigma(i))$, где $f_e^r(\sigma(i))$ – характеристическая функция предиката*

$$r \in elig_{\sigma(i)} \wedge r \notin elig_{\sigma(i-1)} \wedge actual_{\sigma(i-1)} \notin dom(I).$$

Для каждого правила эти метрики помогают определить, какая часть правила не была удовлетворена во время выполнения тестовых случаев. Однако в некоторых случаях возможно, что правило являлось пригодным, но предлагаемая реализация системы намеренно игнорировала его. Для анализа таких случаев в качестве дополнительной метрики мы вычисляем частоту правил адаптации.

Частота правила Для данного набора тестов периодичность или частота активации правила измеряется как количество его активаций, соотнесенное к количеству конфигураций, в которых это правило становилось пригодным.

Определение 10 (Частота правила)**.** *Для данного набора тестов TS, состоящего из тестовых примеров tc, частота правила $r \in R_R$ определена следующим образом:*

$$freq^r(TS) = \frac{\Sigma_{tc \in TS} \#actual^r(exec(tc))}{\Sigma_{tc \in TS} \#elig^r(exec(tc))}.$$

Если правило не является приемлемым ни в одном состоянии, то есть число его допусков равно 0, то его частота также равна 0.

Чтобы получить частоты, имеющие смысл, мы полагаемся на процесс генерации тестов, описанный в разделе 2. С его помощью можно генерировать большие наборы тестов, охватывая при этом правила политик адаптации. Эта метрика полезна для оценки того, как часто активируется интересующее правило. После вычисления эту частоту можно сравнить с нечетким значением, специфицированным в правиле, чтобы обнаружить потенциальное несоответствие в реализации политики адаптации. Примечательно, что, начиная с определенного множества начальных конфигураций и предполагаемого поведения системы, анализ частот позволяет обнаружить правило с высокополезным реконфигурированием (полезность `high`), которое применяется реже, чем правило с низкополезным реконфигурированием (полезность `low`).

3.2. Соблюдение политики адаптации

Интуитивно соответствие реализации по отношению к политике адаптации может быть объяснено с помощью значений полезности, используемых в правилах адаптации. По определению 4, в правиле адаптации $r = (b, g, ir)$ пара $ir = (ope, f)$ присваивает реконфигурированию ope полезность f. Предположим, что нечеткие значения полезности упорядочены. Тогда инженер по валидации может ожидать, что каждое правило с полезностью N будет применяться с большей частотой, чем любое правило с полезностью $N - 1$. Формально, для данной политики адаптации A и данного набора тестов TS, политика адаптации считается точно реализованной, если

$$\forall r, r' \in A, f^r > f^{r'} \Longrightarrow freq^r(TS) > freq^{r'}(TS).$$

Пусть $\sqsubseteq$ обозначает известное отношение погружения слов. Рассматривая множество $E_\delta \cup O$ контролируемых и неконтролируемых событий, после удаления δ из рассматриваемых слов будем использовать отношение $\sqsubseteq$ не учитывая δ, с обозначением $\sqsubseteq_\delta$.

Утверждение 1. *Если политика адаптации A точно реализована, то $\forall tc \in TS$, $\exists \sigma \in \Sigma_{S \lhd A}$, такой что $tc \sqsubseteq_\delta tr(\sigma)$. Более того, $\forall i \geq 0$, конфигурация $\sigma(i)$ достижима посредством выполнения операций реконфигурирования, заданных множеством $elig_{\sigma(i)}$ приемлемых правил адаптации.*

Доказательство Каждый случай теста $tc \in TS$ генерируется с использованием алгоритма 1 и на базе определения 6. По определению 5 для $S \lhd A$ каждое состояние $\sigma(i)$ на пути σ, соответствующем тестовому случаю tc, получено путем применения либо правила *[ACT1]* в случае реконфигурирования по одному из правил политики адаптации, либо правила *[ACT2]* в случае наблюдаемого реконфигурирования. При этом в случае правила *[ACT1]*, начиная с начальных конфигураций, каждое следующее состояние является результатом одной ($actual_{\sigma(i)}$) из операций реконфигурирования, выбранной в соответствии с ее полезностью из множества правил $elig_{\sigma(i)}$, приемлемых в данной конфигурации. Выбор этого реконфигурирования при исполнении системы увеличивает его частоту, что соответствует точной реализации политики адаптации. Достижимость конфигураций на пути σ обеспечивается этой конструкцией. Отношение $tc \sqsubseteq_\delta tr(\sigma)$ погружения слов между случаем теста и трассой пути основано на использовании определений 6, 1 и 5.

4. Экспериментирование

Чтобы подтвердить предложенный в этой статье подход, следующий раздел приводит примеры экспериментов, где были выявлены несоответствия в реализации политик адаптации по сравнению с определенной в них полезностью. Мы начинаем этот раздел с вопросов исследования (RQ), затем опишем эксперименты, их результаты, а также их обоснованность.

[RQ1.] В какой степени наш подход эффективен для генерации большого количества соответствующих тестовых случаев? Целью является оценка способности генераторов тестов производить большие наборы тестов с хорошим охватом правил политик адаптации. Особенно мы хотим гарантировать, что тестовые случаи способны достичь состояний системы, в которых реконфигурирования являются приемлемыми.

[RQ2.] В какой степени анализ частот реконфигурирований позволяет оценить нечеткие значения? Чтобы иметь возможность измерять частоты достаточно точно, мы стремимся проверить, что выполняется значительное количество реконфигурирований.

[RQ3.] В какой степени можно обнаружить подозрительные результаты? Наша общая цель состоит в том, чтобы оценить, способен ли процесс обнаружить несоответствия в реконфигурированиях, которые выполняются с полезностью, заданной нечеткими значениями в политиках адаптаций.

Экспериментальная процедура Для ответа на эти вопросы был разработан следующий эксперимент. Для сети VANet были разработаны различные реализации (на языке Java), которые отличаются друг от друга стратегией выполнения реконфигурирований. Типичные стили реализации взяты нами из ранее опубликованных по этой тематике статей. Например, в [1] авторы преобразовывают нечеткие значения для операций реконфигурирования, используя упорядочение, при этом полезность применения операций гарантируется пороговым значением. Правила политик адаптации, реализованных в Tangram4Fractal [8], рассматриваются в порядке объявления: для применения выбирается правило, которое является первым применяемым в порядке описания в политике адаптации. В [17] авторы классифицируют реконфигурирования в соответствии с определенными приоритетами: если допустимы две операции с одинаковым приоритетом, выбирается первая из них.

Разработанный эксперимент для сети VANet содержит 9 операций реконфигурирования, из которых 3 (GetRelay, PassRelay и Quit) инициируются 8 правилами политики адаптации (R1-R8). Правило

R1 определяет, когда ведущее транспортное средство может передать реле, правило R6 определяет, когда транспортное средство может заменить нынешнего лидера. Другие правила определяют различные случаи, когда машина может выйти из своей колонны из-за исчерпания энергии или достижения пункта назначения. Правила R1-R4 имеют высокую полезность, полезность правил R5-R6 средняя, а правила R7-R8 описываются с низкой полезностью.

В проведенном нами эксперименте транспортные средства могут динамически добавляться на дорогу и удаляться, как только они достигают места назначения. Во время различных экспериментов количество транспортных средств колебалось между 100 и 250. В итоге машины в индивидуальном режиме сопровождались колоннами (от 4 до 25), в каждой из которых было до 8 машин. Такое функционирование обеспечивало обновление транспортных средств на дороге и повышало шансы на достижение конфигураций, в которых возможны реконфигурирования. Заметим, что генерирование приемлемых начальных конфигураций, которые минимизируют количество компонентов, не находилось в центре внимания данного эксперимента.

Процесс генерации тестов, описанный алгоритмом 1, использовался для создания и выполнения случаев теста на различных реализациях. Основываясь на трассах выполнения, были вычислены частоты применения правил в соответствии с формулами, приведенными в разделе 3. Затем эти частоты были сравнены с нечеткими значениями, описанными в правилах политики адаптации. В нашем эксперименте анализ был выполнен на случаях теста, состоящих из 100 000 шагов. Чтобы обеспечить активацию каждого правила адаптации по меньшей мере один раз на рассматриваемом наборе тестов, мы использовали критерии покрытия, определенные в [3].

Измерения частот реконфигурирований на каждом шаге выполнения позволяют нарисовать графики, которые показывают эволюцию частот по мере выполнения. Для каждого правила реконфигурирования эти графики можно использовать для сравнения его частоты с полезностью реконфигурирования, заданной в правиле адаптации. В конце выполнения эксперимента можно сравнить частоты, взятые в совокупности, чтобы проверить фактическое инициирование реконфигурирований. Как представлено на рис. 6, частоты стабилизируются с увеличением числа шагов, показывая, что большое количество генерируемых тестов позволяет выполнить сравнение частот, которое заслуживает доверия (RQ1).

Использование графиков позволяет визуальный анализ, который делает возможным обнаружение подозрительного поведения, например, потенциальную нестабильность на одной из частот (RQ3).

Для оценки способности к обнаружению потенциальных проблем при реализации политики адаптации было разработано шесть реализаций (E1-E6), которые отличаются друг от друга способом выбора реконфигурирования. Первая реализация E1 выбирает реконфигурирования в соответствии с уровнем приоритета. Кроме того, если приемлемое реконфигурирование не было выполнено, его приоритет увеличивается для следующего шага. Результаты выполнения E1 можно найти в левой части рис. 6 и в таблице 1. При рассмотрении рисунка можно заметить, что график реконфигурирования R5 находится под графиками реконфигурирований R7 и R8. Эти графики показывают несоответствие, поскольку R5 имеет средний приоритет и должен иметь более высокую частоту, чем низкоприоритетные реконфигурирования R7 и R8. Затем мы изменили E1, чтобы создать реализацию E2, увеличив уровень приоритета реконфигурирования R5, результаты которого можно найти в правой части рис. 6 и в таблице 1. В результате график R5 стал расположен выше, чем R7 и R8, соответствуя заданным приоритетам. Таким образом, этот пример показывает, что предложенный подход помогает обнаружить несогласованный выбор реализации и затем подтвердить произведенную корректировку (RQ2).

Для продолжения экспериментов были смоделированы и имитированы другие варианты реализации со скорректированным уровнем приоритета, как в E2. Их результаты приведены в таблице 1.

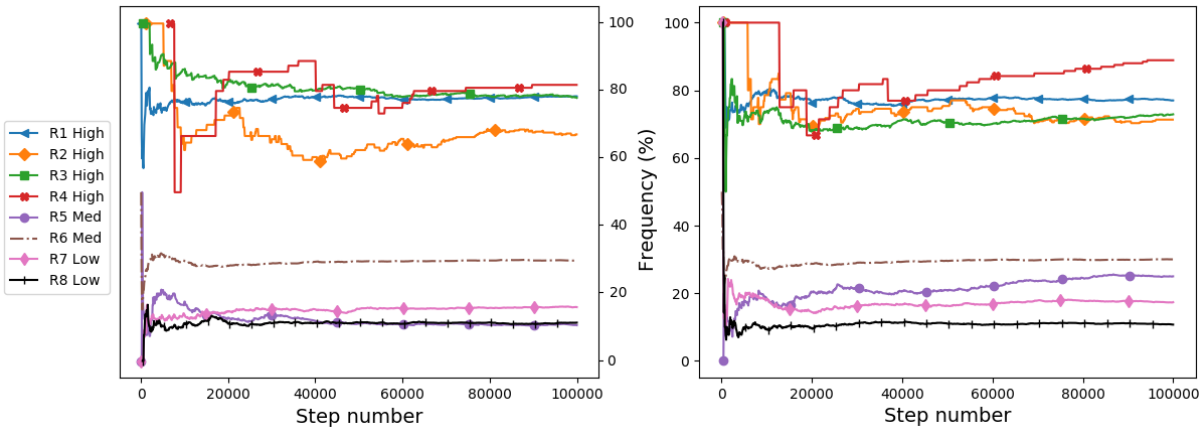

Fig. 6. Frequency analysis of implementations E1 (left) vs. E2 (right)

Рис. 6. Анализ частот реализации E1 (слева) и E2 (справа)

При реализации E3 использовалась гипотеза справедливости: если два реконфигурирования с одинаковым приоритетом являются приемлемыми, то реконфигурирование, которое не было выбрано на данном шаге, будет выбрано в следующий раз, когда сложится такая же ситуация. При моделировании E4 реконфигурирования выбираются на основе их уровня приоритета. Реализация E5 выбирает правило с низкой полезностью в 20% случаев. Наконец, реализация E6 выбирает первое реконфигурирование политики адаптации, которое может быть инициировано.

Результаты моделирования E2 и E3 показывают, что сделанный выбор реализации соответствует указанным нечетким значениям полезности. Реализация E4 показывает приемлемые результаты, но реконфигурирование R4 происходит слишком часто по сравнению с другими реконфигурированиями, имеющими такую же полезность. Реализация E5 является несогласованной, поскольку реконфигурирование R5 среднего приоритета находится на графике ниже реконфигурирований R7 и R8 низкого приоритета. Наконец, моделирование E6 показывает несколько несоответствий в частотах. Эти эксперименты показывают интерес предложенного метода, позволяющего пользователю как подтвердить реализацию системы в присутствии политик адаптации, так и идентифицировать подозрительное поведение. В этом случае последующий анализ позволяет разработать реализацию сложной системы, соответствующую политике адаптации (RQ2).

Table 1. Average results for 100 000-step executions

Таблица 1. Средние результаты для выполнений из 100 000 шагов

	R1(%)	R2(%)	R3(%)	R4(%)	R5(%)	R6(%)	R7(%)	R8(%)
	high	high	high	high	med.	med.	low	low
E1	78.4	67.2	78	82.8	10.6	29.7	16	11.5
E2	77	71.3	72.9	87.9	25.5	30	17.3	10.9
E3	68.7	64.7	70.8	76	23.1	31.4	14.9	9.4
E4	79.2	72.3	74.1	92.9	25	30.2	17.2	11.4
E5	64.4	66	56.6	68.2	17	29.7	19.1	24.4
E6	22.9	14.9	0.7	0	4.2	26.2	10.4	23.3

Обоснованность результатов Первая угроза обоснованности относится к анализу частот, который может опираться на слишком малое количество инициирований правил, чтобы иметь возможность выполнять точное упорядочение реальных значений полезности правил. Заметим в связи с этим, что во время экспериментов каждое реконфигурирование запускалось от нескольких десятков до нескольких тысяч раз в случаях тестов, состоящих из 100 000 шагов. Таким образом, было обеспечено значительное количество допусков для каждого правила, что дает релевантность вы-

числяемым частотам. Вторая угроза обоснованности заключается в том, что эксперименты были проведены только для одной сложной системы, хотя и с многочисленными реализациями. Расширение экспериментов на другие примеры систем является частью будущих работ. Наконец, еще одной угрозой обоснованности является тот факт, что мы сами разработали все реализации, что может вызвать некоторую предвзятость. Тем не менее, как упоминалось выше, выбор сделанных нами реализаций опирается на реальные стратегии применений адаптаций, уже описанные в соответствующей литературе.

5. Библиографические заметки и заключение

5.1. Библиографические заметки

Данная работа продолжает рассмотрение вопроса о самоадаптации сложных систем. Специализированные обзоры [18, 19] подчеркивают, что адаптивные и самоадаптивные системы должны обеспечивать безопасность на разных уровнях, учитывая автоматические реконфигурирования и важность их согласованности. Авторы предлагают обратный контроль поведения системы и изменение системы в случае необходимости. Наш подход с онлайн-тестированием также предлагает инструментирование системы. Однако вместо изменения системы, мы предлагаем информировать инженеров по валидации или пользователей о потенциальных проблемах, поскольку 1) прямой контроль не предполагается при тестировании закрытых систем (black-box testing), и 2) мы рассматриваем, что ошибки могут появляться на фазе разработки, нежели происходить из среды выполнения системы. В статье [20] был обсужден подход к онлайн-тестированию, и авторы выступили за использование онлайн-обнаружения сбоя и диагностики для восстановления до правильного состояния системы после него. В работе [21] авторы подвели итог по использованию моделей для обеспечения уверенности в самоадаптивных системах. Они определили методы, которые опираются на эволюционные алгоритмы для автоматической генерации случаев тестов. Другие подходы [22] использовали цикл MAPE-T для мониторинга применяемости и полезности случаев тестов во время выполнения системы. Наш подход также основан на принципах мониторинга, но не для анализа случаев теста. Нашей целью является проверка реализаций системы в присутствии политик адаптации, поэтому мы генерируем случаи тестов непосредственно в режиме онлайн на основе выполнения системы и модели среды использования системы.

Среди моделей взаимодействия для распределенных систем отметим язык взаимодействия в [23], чья операционная семантика основана на пошаговом выполнении. В отличие от данной работы, позволяющей валидацию трасс выполнения посредством чтения событий одного за другим в офлайн режиме, наш подход к тестированию на базе моделей использования компонентов позволяет онлайн обработку событий. Подход, описанный в [24], также используется офлайн для управления состояниями компонентно-ориентированных систем. Для этого используются политики адаптации с подмножеством временных шаблонов над состояниями. В отличие от нашего подхода политики применяются сразу, позволяя построить множество состояний системы и затем планировать в режиме офлайн реконфигурирования, имеющие целью оптимизацию некоторых нефункциональных требований к системе.

В области систем на основе компонентов онлайн-тестирование использовалось в [25] для обнаружения нарушений свойств системы после реконфигурирования с целью возвращения в нормальное состояние. Несколько статей относятся к встроенным тестам для компонентных систем с акцентом на их архитектуру. В работе [26] использовался онлайн-подход к тестированию на базе моделей, где тестируемая система (SUT) стимулировалась марковским процессом принятия решений (MDP). Состояния модели и поведение системы были связаны, чтобы генерировать действия модели использования, которые приведут к намеченному поведению. В отличие от вышеупомянутых работ,

наш подход основан на извлечении информации из записей поведения системы для генерации соответствующих тестов для SUT на базе модели ее использования.

Правила адаптации и политики адаптации полезны для уменьшения или даже устранения неопределенности, когда несколько разных действий возможны в адаптивной системе. В области динамического дельта-моделирования для адаптивных компонентных систем работа [27] использовала множество правил с приоритетами в качестве технологических маршрутов. В статьях [17, 28] политики адаптации содержат правила со значениями приоритета, тогда как в [29] функции полезности используются для определения целей системы. С одной стороны, наше понятие полезности близко к ним, так как оно позволяет управление приоритетами, чтобы выбрать правило с наибольшей полезностью по отношению к намеченным целям. С другой стороны, вышеупомянутые работы не подтверждали точного выполнения правил. В [30] авторы улучшили реактивность самоадаптивных систем с помощью предиктивных алгоритмов. Наш подход является более общим, поскольку он подтверждает полезность правил реконфигурирования для разнообразных систем в присутствии политик адаптации.

Заключение

В настоящей статье был представлен подход и метод для автоматической генерации тестовых случаев для валидации политик адаптации компонентно-ориентированных систем. Этот подход направлен на получение больших наборов тестов на основе вероятностных моделей среды использования компонентов, чтобы иметь возможность оценивать метрики во время выполнения системы и тем самым сравнивать выполнение операций реконфигурирования по отношению к их формальной спецификации в политиках адаптации. Проведенные эксперименты показали, что этот подход позволяет обнаружить реализации, которые не соблюдают должным образом полезность реконфигурирований. Подчеркнем, что этот подход, применяемый в данной статье к системам компонентов, может быть легко адаптирован к другим типам систем с использованием правил и политик адаптации.

Одним из будущих направлений работы на основе этого подхода является предоставление пользователю средств для проверки того, что политика адаптации, которая правильно реализуется, выполняет такие нефункциональные свойства, как оптимизированное потребление ресурсов и т. д. Эта работа имела целью генерацию случаев теста в виде последовательностей событий. Одним из улучшений может быть автоматическая генерация большого числа имеющих смысл начальных конфигураций, которые могут быть решающими в процессе тестирования для достижения конкретных конфигураций во время выполнения адаптивной системы.

References

[1] J. Dormoy, O. Kouchnarenko, and A. Lanoix, “Using Temporal Logic for Dynamic Reconfigurations of Components”, in *FACS*, ser. LNCS, L. Barbosa and M. Lumpe, Eds., vol. 6921, Springer Berlin Heidelberg, 2012, pp. 200–217, ISBN: 978-3-642-27268-4. DOI: 10.1007/978-3-642-27269-1_12.

[2] O. Kouchnarenko and J.-F. Weber, “Decentralised Evaluation of Temporal Patterns over Component-Based Systems at Runtime”, in *Formal Aspects of Component Software*, I. Lanese and E. Madelaine, Eds., ser. LNCS, vol. 8997, Bertinoro, Italy: Springer, Sep. 2015, pp. 108–126.

[3] F. Dadeau, J.-P. Gros, and O. Kouchnarenko, “Testing Adaptation Policies for Software Components”, *Software Quality Journal*, no. 28, pp. 1347–1378, 2020. DOI: 10.1007/s11219-019-09487-w.

[4] O. Kouchnarenko and J.-F. Weber, “Adapting Component-Based Systems at Runtime via Policies with Temporal Patterns”, in *FACS, 10th Int. Symp. on Formal Aspects of Component Software*, ser. LNCS, J. L. Fiadeiro, Z. Liu, and J. Xue, Eds., vol. 8348, Springer, 2014, pp. 234–253, ISBN: 978-3-319-07601-0. DOI: 10.1007/978-3-319-07602-7_15.

[5] M. Kim, I. Lee, U. Sammapun, J. Shin, and O. Sokolsky, "Monitoring, checking, and steering of real-time systems", *ENTCS*, vol. 70, no. 4, pp. 95–111, 2002. DOI: 10.1016/S1571-0661(04)80579-6.

[6] R. A. Kowalski and M. J. Sergot, "A Logic-based Calculus of Events", *New Gener. Comput.*, vol. 4, no. 1, pp. 67–95, 1986. DOI: 10.1007/BF03037383. [Online]. Available: https://doi.org/10.1007/BF03037383.

[7] R. Miller and M. Shanahan, "The Event Calculus in Classical Logic - Alternative Axiomatisations", *Electron. Trans. Artif. Intell.*, vol. 3, no. A, pp. 77–105, 1999. [Online]. Available: http://www.ep.liu.se/ej/etai/1999/016/.

[8] F. Chauvel, O. Barais, I. Borne, and J.-M. Jézéquel, "Composition of Qualitative Adaptation Policies", in *23rd IEEE/ACM Int. Conf. on Automated Software Engineering (ASE 2008)*, IEEE Computer Society, 2008, pp. 455–458, ISBN: 978-1-4244-2187-9.

[9] A. Bauer and Y. Falcone, "Decentralised LTL monitoring", in *FM 2012: Formal Methods*, ser. LNCS, vol. 7436, Springer, 2012, pp. 85–100.

[10] K. Larsen and B. Thomsen, "A Modal Process Logic", in *LICS'88, 1988*, IEEE Computer Society, 1988, pp. 203–210. DOI: 10.1109/LICS.1988.5119.

[11] R. Milner, *Communication and concurrency*, ser. PHI Series in computer science. Prentice Hall, 1989, ISBN: 978-0-13-115007-2.

[12] B. Jonsson and K. Larsen, "Specification and Refinement of Probabilistic Processes", in *Proc. LICS'91*, IEEE Computer Society, 1991, pp. 266–277, ISBN: 0-8186-2230-X. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome/360/proceeding.

[13] J. A. Whittaker and M. G. Thomason, "A Markov chain model for statistical software testing", *IEEE Trans. on Software Engineering*, vol. 20, no. 10, pp. 812–824, 1994, ISSN: 0098-5589.

[14] G. H. Walton, J. H. Poore, and C. J. Trammell, "Statistical testing of software based on a usage model", *Software: Practice and Experience*, vol. 25, no. 1, pp. 97–108, 1995.

[15] G. Dupont, Y. Aït Ameur, M. Pantel, and N. Singh, "Proof-Based Approach to Hybrid Systems Development: Dynamic Logic and Event-B", in *Int. Conf. Abstract State Machines, Alloy, B, TLA, VDM, and Z (ABZ 2018)*, M. Butler, A. Raschke, and K. Reichl, Eds., ser. LNCS, vol. 10817, Springer-Verlag, 2018, pp. 155–170.

[16] A. Sinclair, *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Basel, Switzerland, Switzerland: Birkhauser Verlag, 1993, ISBN: 0-8176-3658-7.

[17] D. Romero, C. Quinton, L. Duchien, L. Seinturier, and C. Valdez, "SmartyCo: Managing Cyber-Physical Systems for Smart Environments", in *Software Architecture – 9th European Conference, ECSA 2015*, 2015, pp. 294–302. DOI: 10.1007/978-3-319-23727-5_25.

[18] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, and T. Vogel, "Software engineering for self-adaptive systems: A second research roadmap", in *Software Engineering for Self-Adaptive Systems II*, Springer, 2013, pp. 1–32.

[19] R. De Lemos, D. Garlan, C. Ghezzi, H. Giese, J. Andersson, M. Litoiu, B. Schmerl, D. Weyns, L. Baresi, and N. Bencomo, "Software engineering for self-adaptive systems: Research challenges in the provision of assurances", in *Software Engineering for Self-Adaptive Systems III. Assurances*, Springer, 2017, pp. 3–30.

[20] S. Gupta, A. Ansari, S. Feng, and S. A. Mahlke, "Adaptive online testing for efficient hard fault detection", in *27th Int. Conf. on Computer Design*, 2009, pp. 343–349. DOI: 10.1109/ICCD.2009.5413132.

[21] B. H. C. Cheng, K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Müller, P. Pelliccione, A. Perini, N. A. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, and N. M. Villegas, "Using Models at Runtime to Address Assurance for Self-Adaptive Systems", in *Models@run.time: Foundations, Applications, and Roadmaps*, N. Bencomo, R. France, B. H. C. Cheng, and U. Aßmann, Eds. Cham: Springer International Publishing, 2014, pp. 101–136, ISBN: 978-3-319-08915-7. DOI: 10.1007/978-3-319-08915-7_4.

[22] E. M. Fredericks, A. J. Ramirez, and B. H. C. Cheng, "Towards run-time testing of dynamic adaptive systems", in *Proc. Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, May 2013, pp. 169–174. DOI: 10.1109/SEAMS.2013.6595504.

[23] E. Mahe, C. Gaston, and P. L. Gall, "Revisiting Semantics of Interactions for Trace Validity Analysis", in *FASE 2020, Proceedings*, ser. LNCS, vol. 12076, Springer, 2020, pp. 482–501. DOI: 10.1007/978-3-030-45234-6_24.

[24] F. Alvares, E. Rutten, and L. Seinturier, "Behavioural Model-Based Control for Autonomic Software Components", in *IEEE Int. Conf. on Autonomic Computing, ICAC'15*, IEEE Computer Society, 2015, pp. 187–196. DOI: 10.1109/ICAC.2015.31.

[25] M. Greiler, H.-G. Gross, and A. van Deursen, "Evaluation of online testing for services: a case study", in *Proc. Int. Workshop on Principles of Engineering Service-Oriented Systems, PESOS 2010*, 2010, pp. 36–42. DOI: 10.1145/1808885.1808893.

[26] M. Camilli, C. Bellettini, A. Gargantini, and P. Scandurra, "Online Model-Based Testing under Uncertainty", in *29th IEEE International Symposium on Software Reliability Engineering, ISSRE 2018*, 2018, pp. 36–46. DOI: 10.1109/ISSRE.2018.00015.

[27] M. Helvensteijn, "Dynamic delta modeling", in *16th International Software Product Line Conference, SPLC'12*, E. S. de Almeida, C. Schwanninger, and D. Benavides, Eds., ACM, 2012, pp. 127–134, ISBN: 978-1-4503-1095-6. DOI: 10.1145/2364412.2364434. [Online]. Available: http://dl.acm.org/citation.cfm?id=2364412.

[28] F. Trollman, J. Fähndrich, and S. Albayrak, "Hybrid adaptation policies: towards a framework for classification and modelling of different combinations of adaptation policies", in *Proc. Int. Conf. SEAMS@ICSE 2018, Gothenburg, Sweden, May 28-29, 2018*, 2018, pp. 76–86. DOI: 10.1145/3194133.3194137.

[29] J. O. Kephart and W. E. Walsh, "An Artificial Intelligence Perspective on Autonomic Computing Policies", in *5th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2004), 7-9 June 2004, Yorktown Heights, NY, USA*, 2004, pp. 3–12. DOI: 10.1109/POLICY.2004.1309145.

[30] V. Poladian, D. Garlan, M. Shaw, M. Satyanarayanan, B. R. Schmerl, and J. P. Sousa, "Leveraging Resource Prediction for Anticipatory Dynamic Configuration", in *Proc. Int. Conf. on Self-Adaptive and Self-Organizing Systems, SASO 2007*, IEEE Computer Society, 2007, pp. 214–223, ISBN: 0-7695-2906-2. DOI: 10.1109/SASO.2007.35. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome/4274871/proceeding.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 1, 2021
journal homepage: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

An Algorithm for Correcting Levels of Useful Signals on Interpretation of Eddy-Current Defectograms

E. V. Kuzmin[1], O. E. Gorbunov[2], P. O. Plotnikov[2], V. A. Tyukin[2], V. A. Bashkin[1,2]

DOI: 10.18255/1818-1015-2021-1-74-88

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya str., Yaroslavl 150003, Russia.
[2]Center of Innovative Programming, NDDLab, 144 Soyuznaya str., Yaroslavl 150008, Russia.

MSC2020: 68T09
Research article
Full text in Russian

Received October 21, 2020
After revision February 17, 2021
Accepted March 12, 2021

To ensure traffic safety of railway transport, non-destructive tests of rails are regularly carried out by using various approaches and methods, including eddy-current flaw detection methods. An automatic analysis of large data sets (defectograms) that come from the corresponding equipment is an actual problem. The analysis means a process of determining the presence of defective sections along with identifying structural elements of railway tracks in defectograms. This article continues the cycle of works devoted to the problem of automatic recognizing images of defects and structural elements of rails in eddy-current defectograms. In the process of forming these images, only useful signals are taken into account, the threshold levels of amplitudes of which are determined automatically from eddy-current data. The previously used algorithm for finding threshold levels was focused on situations in which the vast majority of signals coming from the flaw detector is a rail noise. A signal is considered useful and is subject to further analysis if its amplitude is twice the corresponding noise threshold. The article is devoted to the problem of correcting threshold levels, taking into account the need to identify extensive surface defects of rails. An algorithm is proposed for finding the values of threshold levels of rail noise amplitudes with their subsequent correction in the case of a large number of useful signals from extensive defects. Examples of the algorithm's operation on real eddy-current data are given.

Keywords: nondestructive testing, eddy current testing, rail flaw detection, automated analysis of defectograms

INFORMATION ABOUT THE AUTHORS

Egor V. Kuzmin correspondence author	orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru, kuzminev@nddlab.com Professor, Doctor of Science.
Oleg E. Gorbunov	orcid.org/0000-0001-6274-9971. E-mail: gorbunovoe@nddlab.com General Director, PhD.
Petr O. Plotnikov	orcid.org/0000-0001-5687-7969. E-mail: plotnikovpo@nddlab.com Production Engineer.
Vadim A. Tyukin	orcid.org/0000-0001-9149-7435. E-mail: tyukinva@nddlab.com Head of Software Development.
Vladimir A. Bashkin	orcid.org/0000-0002-2534-1026. E-mail: bashkinva@nddlab.com Professor, Doctor of Science.

Funding: This work was supported by P. G. Demidov Yaroslavl State University Project № VIP-016.

For citation: E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, V. A. Tyukin, and V. A. Bashkin, "An Algorithm for Correcting Levels of Useful Signals on Interpretation of Eddy-Current Defectograms", *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 74-88, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 1, 2021
сайт журнала: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

Алгоритм корректировки уровней полезных сигналов при расшифровке вихретоковых дефектограмм

Е. В. Кузьмин[1], О. Е. Горбунов[2], П. О. Плотников[2], В. А. Тюкин[2], В. А. Башкин[1,2]

DOI: 10.18255/1818-1015-2021-1-74-88

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, д. 14, г. Ярославль, 150003 Россия.
[2]ООО «Центр инновационного программирования», NDDLab, ул. Союзная, д. 144, г. Ярославль, 150008 Россия.

УДК 004.021
Научная статья
Полный текст на русском языке

Получена 21 октября 2020 г.
После доработки 17 февраля 2021 г.
Принята к публикации 12 марта 2021 г.

Для обеспечения безопасности движения на железнодорожном транспорте регулярно проводится неразрушающий контроль рельсов с применением различных подходов и методов, включая методы вихретоковой дефектоскопии. Актуальной задачей является автоматический анализ больших массивов данных (дефектограмм), которые поступают от соответствующего оборудования. Под анализом понимается процесс определения по дефектограммам наличия дефектных участков наряду с выявлением конструктивных элементов рельсового пути. Данная статья продолжает цикл работ, посвященных задаче автоматического распознавания образов дефектов и конструктивных элементов железнодорожных рельсов по вихретоковым дефектограммам. При формировании этих образов принимаются в расчет только полезные сигналы, пороговые уровни амплитуд которых определяются автоматически по вихретоковым данным. Применяемый ранее алгоритм нахождения пороговых уровней был ориентирован на ситуации, при которых подавляющее большинство поступающих от дефектоскопа сигналов составляет рельсовый шум. Сигнал считается полезным и подлежит дальнейшему анализу, если его амплитуда в два раза превосходит соответствующий пороговый уровень шума. Статья посвящена задаче корректировки пороговых уровней с учётом необходимости выявления протяжённых поверхностных дефектов рельсов. Предлагается алгоритм нахождения значений пороговых уровней амплитуд рельсового шума с их последующей корректировкой в случае наличия большого количество полезных сигналов от протяженных дефектов. Приводятся примеры работы алгоритма на реальных вихретоковых данных.

Ключевые слова: неразрушающий контроль рельсов, вихретоковая дефектоскопия, обнаружение дефектов, автоматический анализ дефектограмм

ИНФОРМАЦИЯ ОБ АВТОРАХ

Егор Владимирович Кузьмин автор для корреспонденции	orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru, kuzminev@nddlab.com профессор, доктор физ.-мат. наук.
Олег Евгеньевич Горбунов	orcid.org/0000-0001-6274-9971. E-mail: gorbunovoe@nddlab.com генеральный директор, канд. физ.-мат. наук.
Петр Олегович Плотников	orcid.org/0000-0001-5687-7969. E-mail: plotnikovpo@nddlab.com инженер-технолог.
Вадим Александрович Тюкин	orcid.org/0000-0001-9149-7435. E-mail: tyukinva@nddlab.com руков. сектора разработки.
Владимир Анатольевич Башкин	orcid.org/0000-0002-2534-1026. E-mail: bashkinva@nddlab.com профессор, доктор физ.-мат. наук.

Финансирование: Работа выполнена в рамках инициативной НИР ЯрГУ им. П. Г. Демидова № VIP-016.

Для цитирования: E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, V. A. Tyukin, and V. A. Bashkin, “An Algorithm for Correcting Levels of Useful Signals on Interpretation of Eddy-Current Defectograms”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 74-88, 2021.

Введение

Для обеспечения безопасности движения на железнодорожном транспорте регулярно проводится неразрушающий контроль рельсов с применением различных подходов и методов, включая методы вихретоковой дефектоскопии. Актуальной задачей является автоматический анализ [1—3] больших массивов данных (дефектограмм), которые поступают от соответствующего оборудования. Под анализом понимается процесс определения по дефектограммам наличия дефектных участков наряду с выявлением конструктивных элементов рельсового пути. При этом в условиях значительных объемов поступающей на обработку информации наибольший интерес представляют быстрые и эффективные алгоритмы анализа данных.

Данная статья продолжает цикл работ [4—7], посвященных задаче автоматического распознавания образов дефектов и конструктивных элементов железнодорожных рельсов по дефектограммам вихретоковых дефектоскопов. Дефектограммы разбиваются на фрагменты (блоки анализа), каждый из которых обрабатывается отдельно. В текущем блоке анализа с использованием алгоритма из статей [4, 5] происходит выделение полезных сигналов (на фоне рельсового шума), которые группируются в отметки. Найденные отметки подлежат дальнейшей классификации с применением нейронных сетей. В статье [6] решалась задача распознавания записей небольших конструктивных элементов (длиной до 157 мм) следующих трёх типов: 1) болтовой стык с прямым или скошенным соединением рельсов, 2) электроконтактная сварка рельсов и 3) алюминотермитная сварка рельсов. В статье [7] проводилось распознавание записей длинных (от 420 мм до 3220 мм) конструктивных элементов рельсового пути двух классов: 1) счётчик осей подвижного состава, 2) пересечение рельсовых путей. Отметки, которые не были отнесены к тому или иному типу конструктивных элементов, классифицируются как условные дефекты.

Отметим, что применяемый алгоритм определения порогового уровня амплитуд полезных сигналов [4, 5] рассчитан на фрагменты вихретоковых дефектограмм, подавляющее большинство сигналов в которых составляет рельсовый шум. Практическое использование этого алгоритма в рамках аппаратно-программного комплекса рельсовой дефектоскопии показало, что при анализе рельсов с длинными поверхностными дефектами (образующимися обычно из-за контактной усталости металла), которые могут иметь протяжённость в десятки и сотни метров, выдаваемые пороговые значения амплитуд полезных сигналов являются завышенными (по абсолютному значению) и не позволяют формировать в отметки ни записи конструктивных элементов, ни записи дефектных участков рельсов.

Данная статья посвящена задаче корректировки порогового уровня амплитуд полезных сигналов при автоматической расшифровке дефектограмм вихретоковых дефектоскопов с учётом необходимости выявления протяжённых поверхностных дефектов рельсов.

В статье рассматривается обобщённое устройство в виде 12-разрядного вихретокового дефектоскопа с 15 каналами данных (на один рельс). Каналы данных соответствуют физическим датчикам, которые последовательно располагаются на поверхности рельса перпендикулярно направлению движения дефектоскопа. Значения амплитуд сигналов каждого канала регистрируются дефектоскопом в виде целых чисел от -2048 до 2047.

Интерес представляют амплитуды только полезных сигналов. Сигнал считается полезным (и подлежит дальнейшему анализу), если отклонение от нуля как минимум в два раза превосходит соответствующий пороговый уровень шума рельсов.

В статье предлагается алгоритм нахождения положительного и отрицательного значений пороговых уровней амплитуд рельсового шума с их последующей корректировкой в сторону нуля в случае наличия большого количества полезных сигналов (например, от протяженных поверхностных дефектов рельсов) на анализируемом фрагменте вихретоковой дефектограммы; приводятся примеры работы алгоритма на реальных данных.

1. Нахождение пороговых уровней полезных сигналов

При автоматическом анализе дефектограммы обычно разбиваются на фрагменты, которые, например, могут соответствовать 50-метровым участкам рельсового пути, т. е. при снятии показаний 15-канального дефектоскопа с каждого миллиметра пути блок анализа представляет собой матрицу размером 15 строк на 50000 столбцов, где элемент матрицы — это значение амплитуды сигнала соответствующего канала данных.

Пример графического представления 50-метрового блока анализа приведён на рис. 1 (15-канальные данные с 50-метрового участка одного рельса; нумерация каналов данных производится сверху вниз). В этом фрагменте дефектограммы помимо записей конструктивных элементов (сварных рельсовых стыков), сливающихся на рисунке в сплошные вертикальные линии, можно видеть, что начиная приблизительно с 18-го метра и до конца рассматриваемого фрагмента присутствуют сильные полезные сигналы от продолжительного поверхностного дефекта.

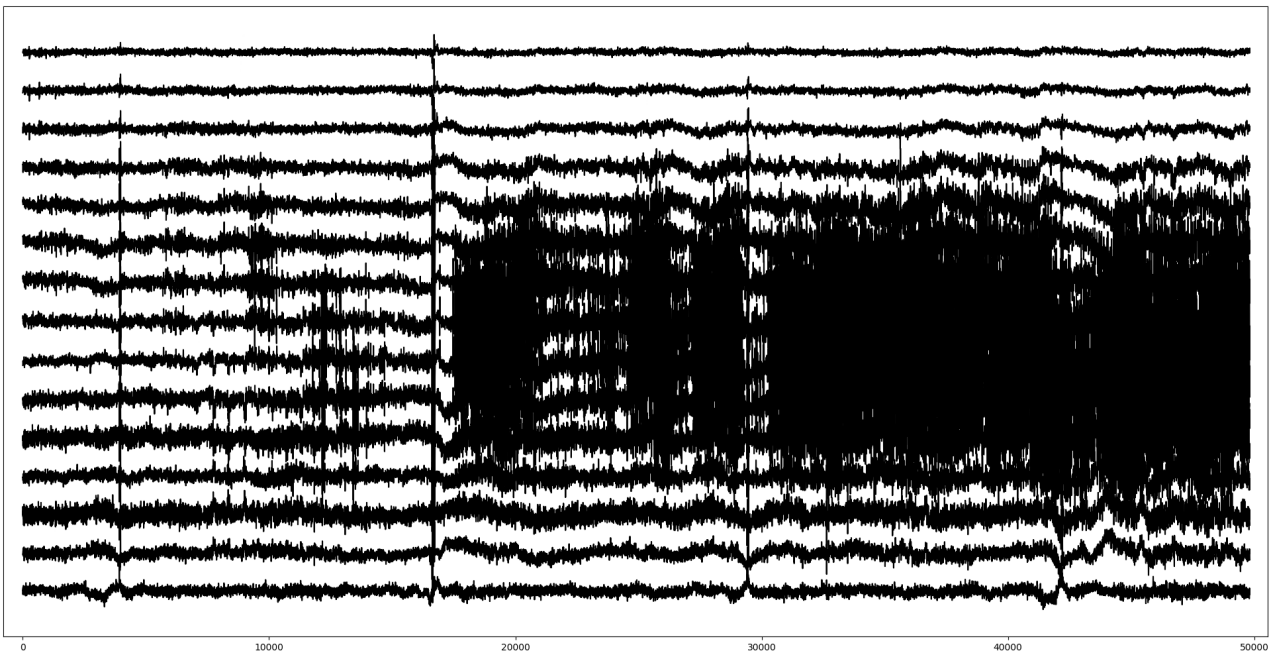

Fig. 1. Eddy-current data (15 channels). An example of an analysis block (50 m)

Рис. 1. Вихретоковые данные по 15 каналам. Пример 50-метрового блока анализа

При этом первые три канала вихретоковых данных никак не захватывают этот дефект, регистрируя главным образом рельсовый шум. Основной объём сигналов от дефекта приходится на центральные каналы. В качестве примера на рис. 2 показаны отдельно данные седьмого и восьмого каналов, которые содержат наиболее сильные сигналы от дефекта (т. е. наиболее сильные амплитудные отклонения от нуля).

Приведённый пример специально подобран таким образом, чтобы было видно отличие участков рельсового шума от участков продолжительного дефекта поверхности катания. Дефект может охватывать всю поверхность рельса на протяжении нескольких сотен метров. В таком случае могут возникнуть трудности в визуальном восприятии того, содержит ли дефектограмма запись дефекта или же это рельсовый шум, который был зарегистрирован датчиками с некорректными настройками чувствительности.

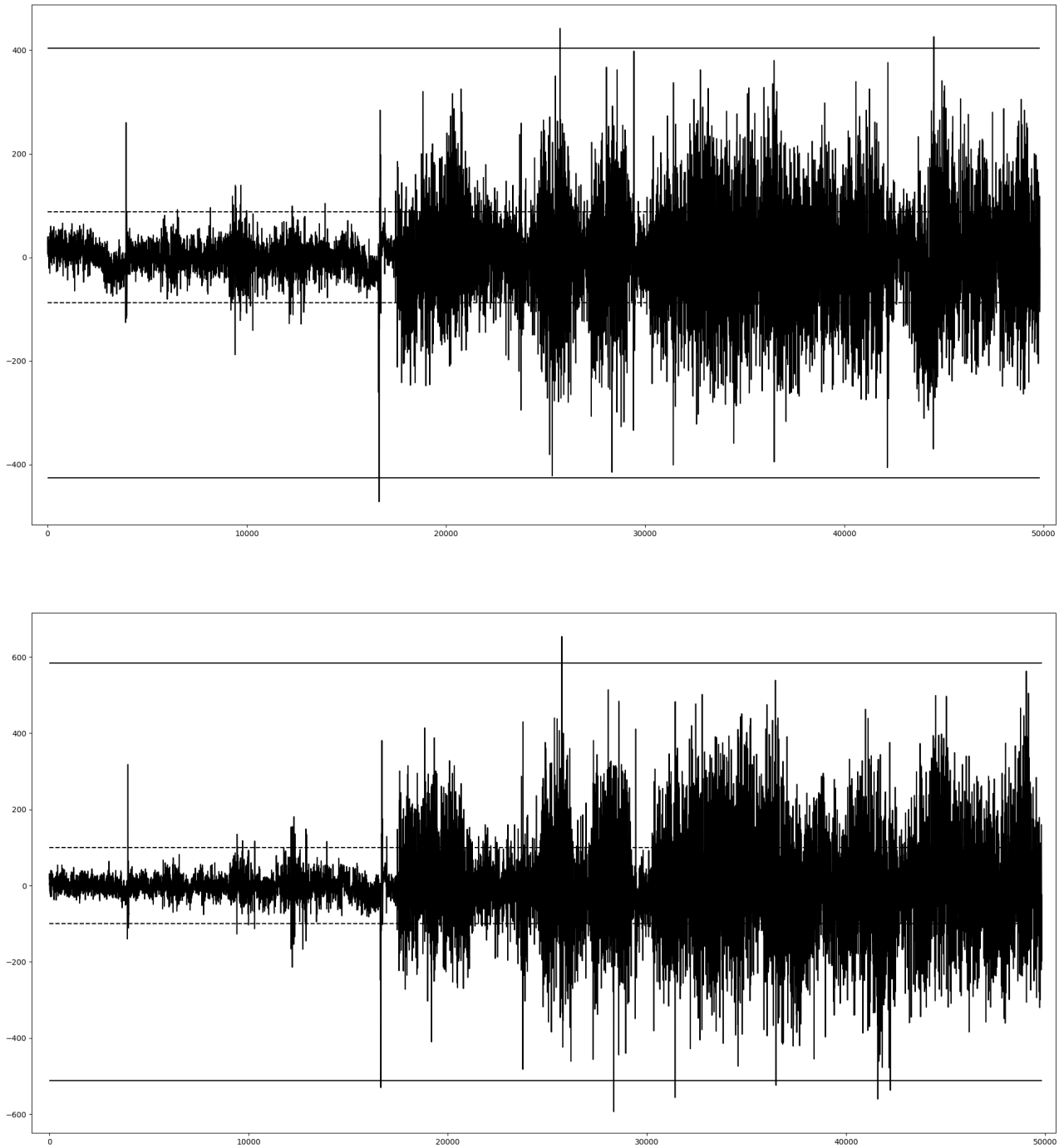

Fig. 2. Eddy-current data of the seventh (top) and the eighth (bottom) channels (50-meter analysis block). Useful signal levels before (solid lines) and after (dashed lines) correcting

Рис. 2. Вихретоковые данные 50-метрового блока анализа по седьмому (вверху) и восьмому (внизу) каналам. Уровни полезных сигналов до (сплошные линии) и после (пунктирные линии) корректировки

В одном блоке анализа для каждого канала данных отдельно вычисляются положительное и отрицательное пороговые значения амплитуд сигналов рельсового шума. При вычислении положительного порога принимаются во внимание только неотрицательные амплитуды сигналов. Аналогичным образом для вычисления отрицательного порога берутся в расчет только те амплитуды, значение которых меньше или равно нулю.

Следующая функция на языке Python 3 возвращает абсолютное значение искомого порога рельсового шума для одного канала данных. В этой функции параметр $A[0:2047]$ — это вспомогательный массив, который хранит частоты соответствующие его индексам амплитуд сигналов одной направленности (положительной или отрицательной). Элемент $A[k]$ представляет собой количество сигналов (одной направленности) с амплитудой k по абсолютному значению. Параметр *end* — максимальное абсолютное значение амплитуды рассматриваемой направленности, т. е. максимальный задействованный индекс массива A, а *total* — сумма значений элементов массива A.

```
def GetNoiseLevel(A, end, total):
    all = total - (A[0] / 2)
    sum = 0
    ctr = A[0] / 2
    sig = 0
    for k in range(1, end + 1):
        ctr += A[k]
        sum += A[k] * k * k
        oldsig = sig
        if ctr > 0:
            sig = math.sqrt(sum / ctr)
        else:
            sig = 0
        if ctr > (0.3 * all) and (3 * sig) < k and (3 * oldsig) >= (k - 1):
            break
    return int(3 * sig + 0.5)
```

Эта функция реализует алгоритм, предложенный и обоснованный в статьях [4, 5]. Идея алгоритма основывается на том, что вероятность появления сигнала с некоторой амплитудой в бездефектных рельсах на участке без конструктивных элементов подчиняется (в приближении) закону нормального распределения. При этом предполагается, что такие сигналы, которые по своей сути являются рельсовым шумом, будут составлять подавляющее большинство от общего числа сигналов, поступающих от вихретокового дефектоскопа при неразрушающем контроле рельсов. В таком случае может быть задействовано правило трёх сигм с предварительным исключением из анализируемой выборки сильных сигналов от дефектов и конструктивных элементов, которые не укладываются в рамки закона нормального распределения.

Практика показала, что при появлении на дефектограммах записей продолжительных поверхностных дефектов этот алгоритм выдаёт завышенные (по абсолютному значению) амплитудные пороги рельсового шума. Пороговые уровни полезных сигналов также оказываются завышенными, поскольку представляют собой удвоение найденных пороговых значений рельсового шума.

Например, на рис. 2 вычисленные с использованием указанного алгоритма положительный и отрицательный пороговые уровни полезных сигналов для седьмого и восьмого канала данных представлены в виде сплошных линий отсечки. Здесь почти все сигналы от конструктивных элементов и дефектов находятся между найденными пороговыми значениями, т. е. эти сигналы не будут учтены в последующих алгоритмах анализа текущего фрагмента дефектограммы.

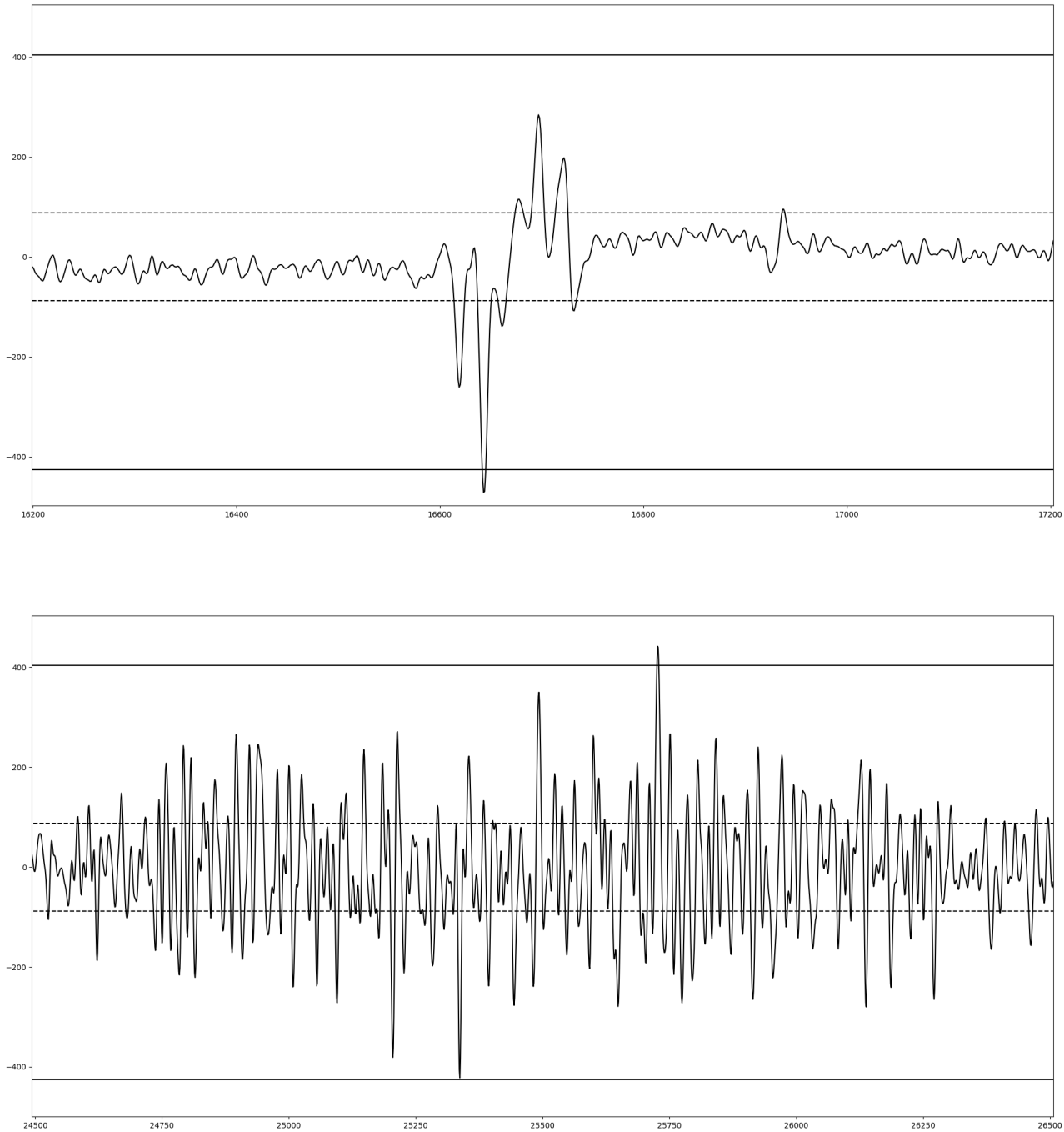

Fig. 3. Eddy-current data of the seventh channel. The aluminothermic weld (top) and the surface defect (bottom). Useful signal levels before (solid lines) and after (dashed lines) correcting

Рис. 3. Вихретоковые данные по седьмому каналу. Алюминотермитная сварка (вверху) и поверхностный дефект (внизу). Уровни полезных сигналов до (сплошные линии) и после (пунктирные линии) корректировки

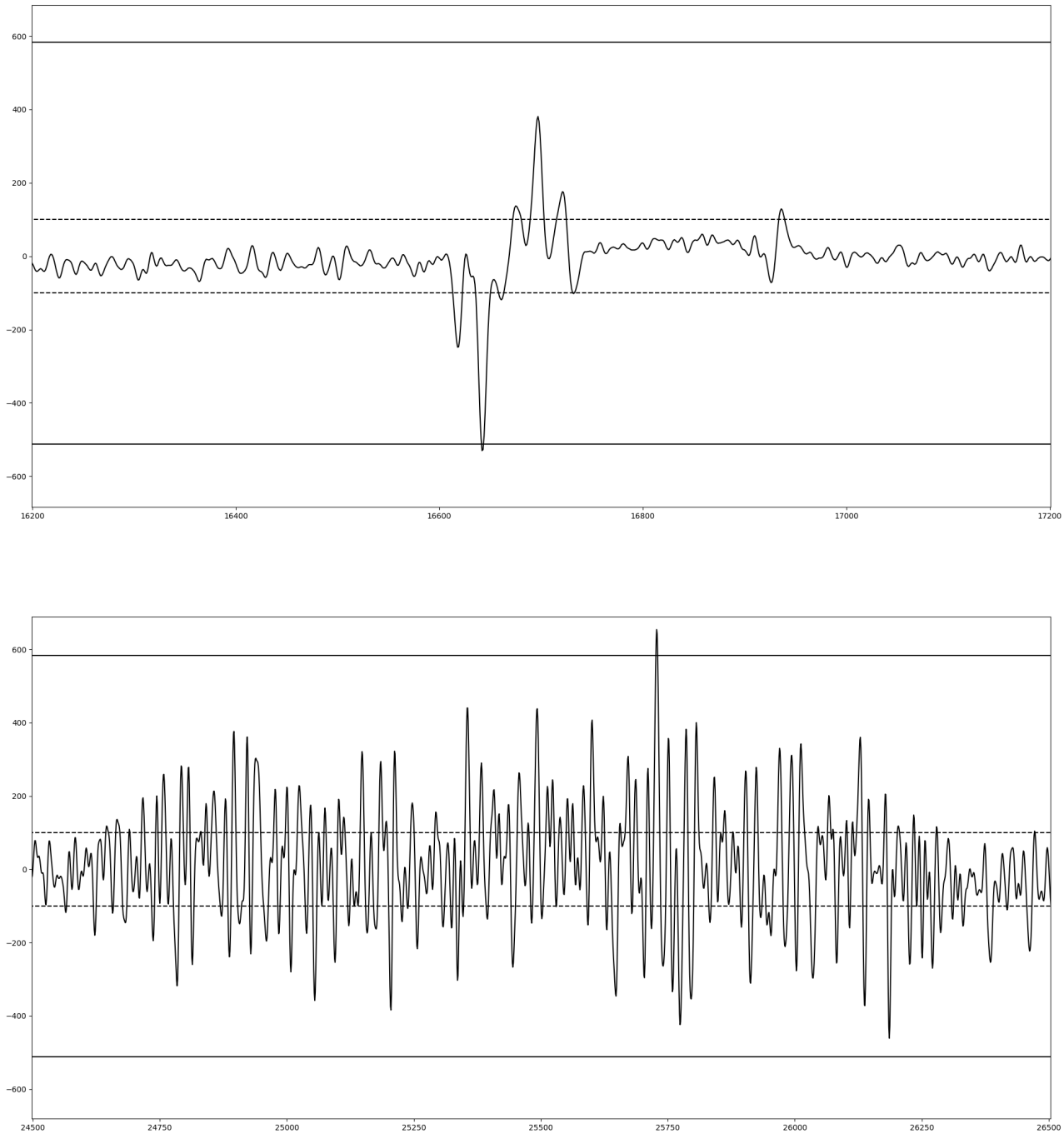

Fig. 4. Eddy current data of the eighth channel. The aluminothermic weld (top) and the surface defect (bottom). Useful signal levels before (solid lines) and after (dashed lines) correcting

Рис. 4. Вихретоковые данные по восьмому каналу. Алюминотермитная сварка (вверху) и поверхностный дефект (внизу). Уровни полезных сигналов до (сплошные линии) и после (пунктирные линии) корректировки

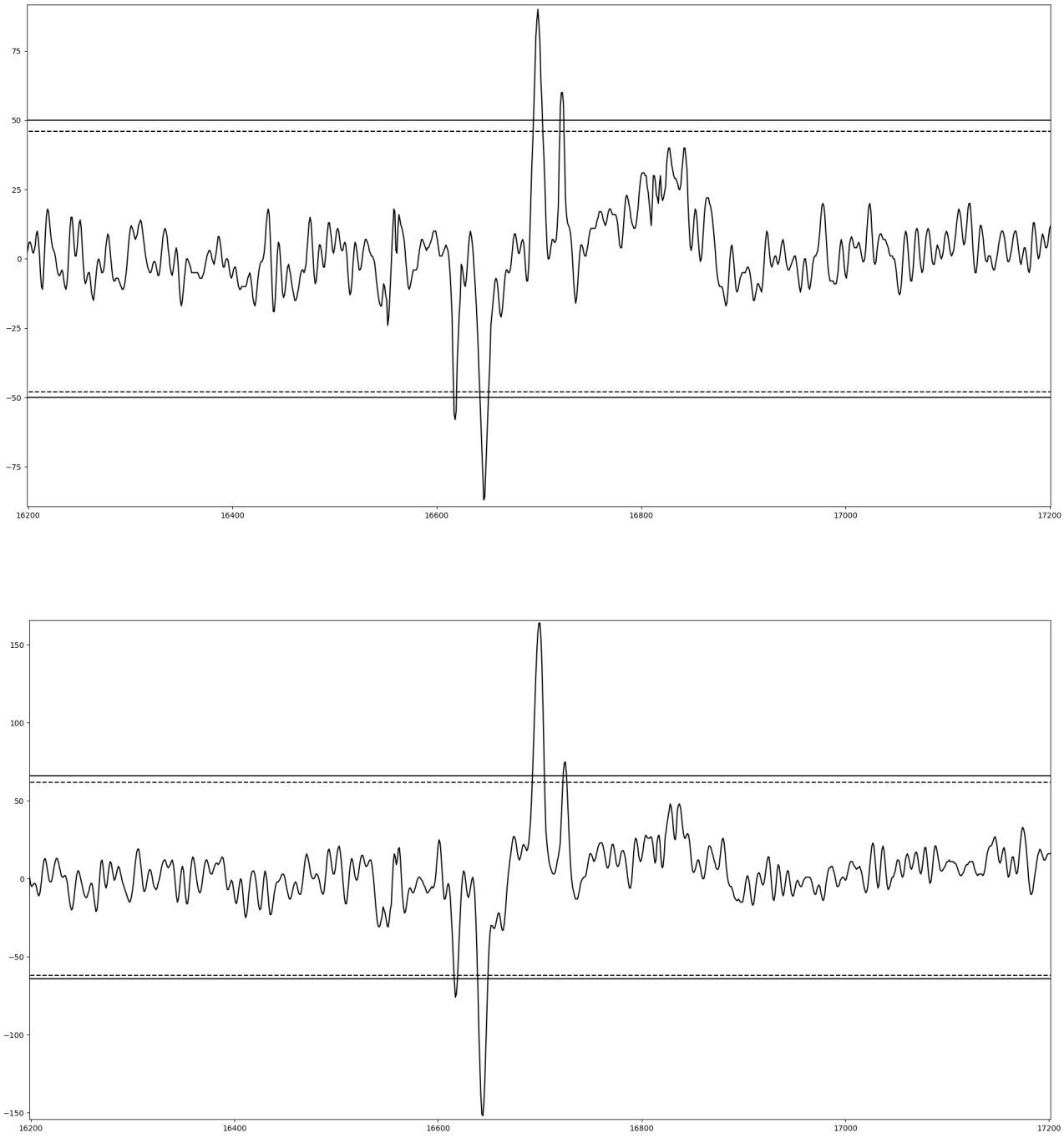

Fig. 5. Eddy current data of the first (top) and the second (bottom) channels. The aluminothermic weld. Useful signal levels before (solid lines) and after (dashed lines) correcting

Рис. 5. Вихретоковые данные по первому (вверху) и второму (внизу) каналам. Алюминотермитная сварка. Уровни полезных сигналов до (сплошные линии) и после (пунктирные линии) корректировки

На рис. 3 и 4 для седьмого и восьмого каналов данных показаны отдельно участки дефектограммы, содержащие запись сварного алюминотермитного рельсового стыка (вверху) и запись сигналов от поверхностного дефекта (внизу). На этих рисунках найденные пороговые уровни полезных сигналов показаны также в виде сплошных линий отсечки.

Заметим, что первыми двумя каналами дефектоскопа наличие каких-либо поверхностных дефектов на рассматриваемом отрезке рельсового пути не регистрируется (см. рис. 1). Полезными сигналами по этим двум каналам выступают только сигналы от сварных рельсовых стыков. Остальные данные представляют собой рельсовый шум. И как можно видеть на рис. 5, в этом случае сплошные линии найденных пороговых уровней довольно точно отсекают полезные сигналы, поступившие от алюминотермитной сварки. Другими словами, для этих каналов алгоритм нахождения пороговых значений рельсового шума отработал успешно (в соответствии с заложенной в него идеей).

Таким образом, если канал данных текущего блока анализа содержит запись продолжительного поверхностного дефекта, то найденные пороговые значения рельсового шума нуждаются в корректировке в сторону нуля. Решению задачи корректировки этих пороговых уровней посвящён следующий раздел.

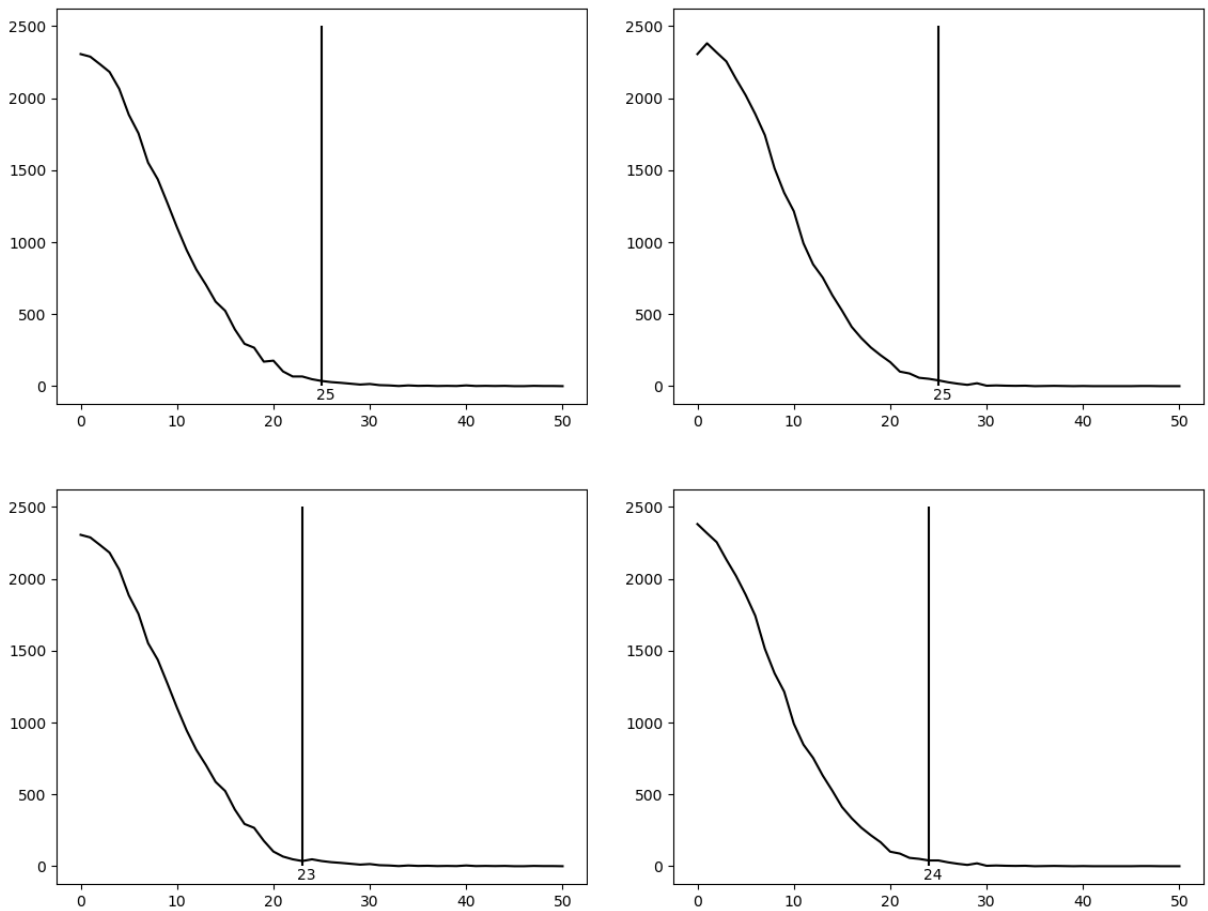

Fig. 6. Graphs of the density of the amplitude frequencies distribution for the first channel before (top) and after (bottom) correcting the noise threshold levels

Рис. 6. Графики плотности распределения частот амплитуд первого канала до (вверху) и после (внизу) корректировки пороговых уровней шума

2. Корректировка пороговых уровней полезных сигналов

Практический опыт работы с вихретоковыми данными позволил выявить характерный признак наличия на записи сигналов от продолжительного поверхностного дефекта.

В блоке анализа для одного канала, содержащего сигналы от такого дефекта, на графике плотности распределения частот появления амплитуд одной направленности на отрезке от нуля до порогового значения рельсового шума будут находиться точки локальных минимумов, количество которых зависит от степени опасности поверхностного дефекта, т. е. от количества полезных сигналов и значений их амплитуд. Для «бездефектного» канала данных этот график на том же отрезке будет значительно более гладким, в идеале не имея ни одного локального минимума. При этом в любом случае форма графика на указанном отрезке повторяет в приближении форму графика плотности нормального распределения вероятностей на участке от 0 до $3{\cdot}\sigma$, где σ — среднеквадратическое отклонение от нулевого математического ожидания.

Локальные минимумы представляют интерес как ресурс, который может быть использован для уменьшения абсолютного значения порогового уровня рельсового шума. Итерационное удаление точек локальных минимумов на рассматриваемом отрезке графика с последующим сдвигом остальных данных в сторону нуля позволяет произвести также и сдвиг порогового уровня шума. Процедура останавливается, как только на графике от нуля до текущего значения порога не останется ни одного локального минимума. Получившееся в результате таких сдвигов новое значение порогового уровня и будет искомым результатом, т. е. будет являться скорректированным порогом рельсового шума.

Следующая функция на языке Python 3 реализует описанную идею, возвращая в качестве результата скорректированный пороговый уровень. Здесь параметр $A[0:2047]$, как и ранее, представляет собой массив (график) частот амплитуд сигналов одной направленности, т. е. $A[k]$ — это количество сигналов с амплитудой k по абсолютному значению. Параметр *level* — найденный предыдущим алгоритмом пороговый уровень шума (для амплитуд одной направленности), который подлежит корректировке.

```
def GetCorrectedNoiseLevel(level, A):
    end = level
    flg = 1
    while flg:
        flg = 0
        ctr = 0
        for k in range(0, end):
            if A[k] <= A[k + 1]:
                flg = 1
            else:
                A[ctr] = A[k]
                ctr += 1
        A[ctr] = A[end]
        end = ctr
        newlevel = end
    return newlevel
```

На рис. 6, 7, 8 и 9 представлены графики плотности распределения частот положительных (справа) и отрицательных (слева) амплитуд сигналов первого, второго, седьмого и восьмого каналов данных до (сверху) и после (снизу) корректировки порогового уровня рельсового шума, который отмечен вертикальной линией отсечки с указанием его абсолютного значения.

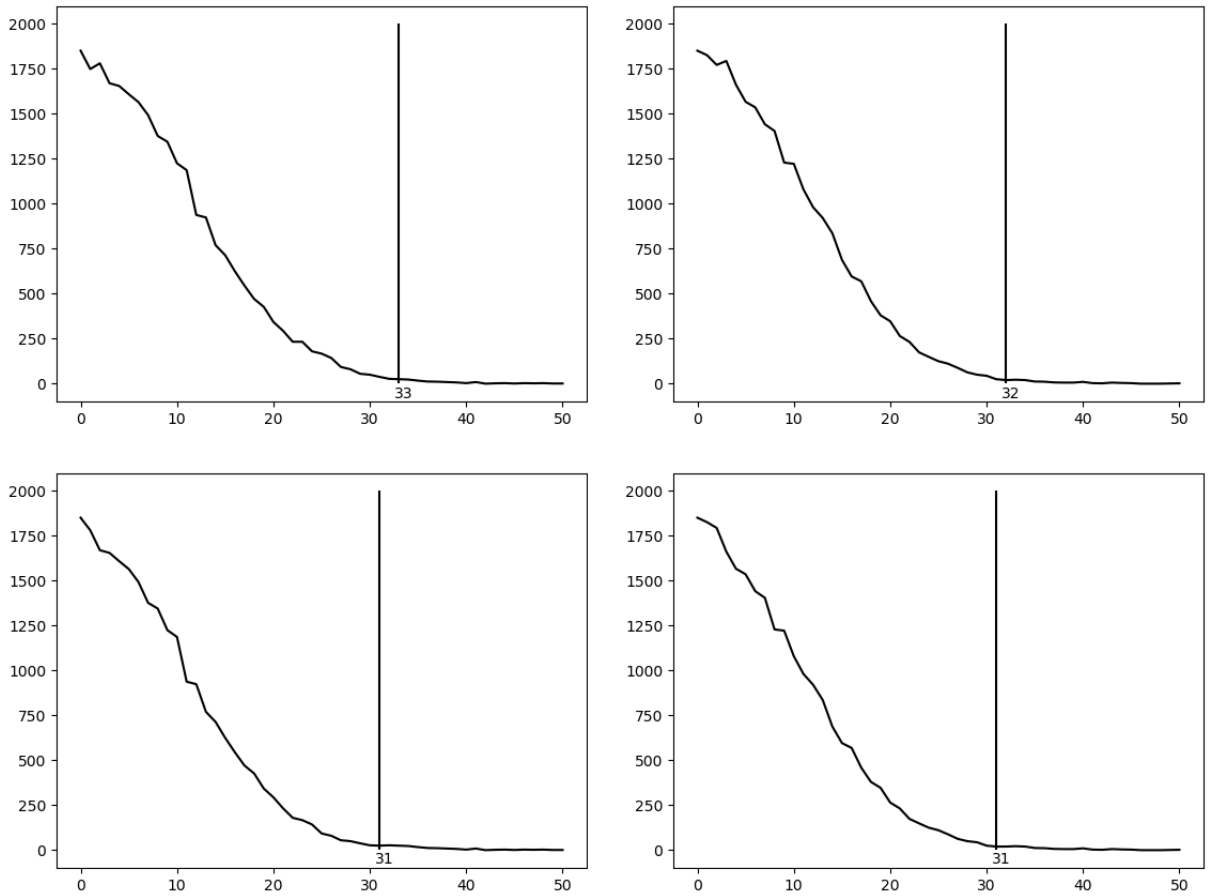

Fig. 7. Graphs of the density of the amplitude frequencies distribution for the second channel before (top) and after (bottom) correcting the noise threshold levels

Рис. 7. Графики плотности распределения частот амплитуд второго канала до (вверху) и после (внизу) корректировки пороговых уровней шума

На этих рисунках для канала данных текущего блока анализа значения амплитуд сигналов одинаковой направленности откладываются по оси X, а суммарное количество зарегистрированных сигналов одной амплитуды отмечается по оси Y.

На рис. 6 и 7 видно, что графики распределения частот амплитуд сигналов являются «гладкими», почти не имеют точек локальных минимумов. И, соответственно, для этих каналов произведена незначительная корректировка пороговых уровней.

Исходные графики частот амплитуд на рис. 8 и 9 содержат большое количество локальных минимумов и подвергаются значительному сжатию, приближённо сохраняя в результате сглаживания начальную форму на отрезке от нуля до отметки порогового уровня.

Результат работы алгоритма корректировки продемонстрирован на рис. 2, 3 и 4. Новый положительный/отрицательный пороговый уровень амплитуд полезных сигналов, равный положительному/отрицательному удвоенному скорректированному порогу шума, показан в виде пунктирной линии отсечки. На этих рисунках значения полезных сигналов от сварных стыков, в частности от алюминотермитной сварки, и продолжительного поверхностного дефекта выходят за границы, обозначенные пунктирными линиями. Это означает, что такие сигналы впоследствии будут участвовать в формировании соответствующих отметок как для конструктивных элементов, так и для участка продолжительного дефекта поверхности катания рельсов.

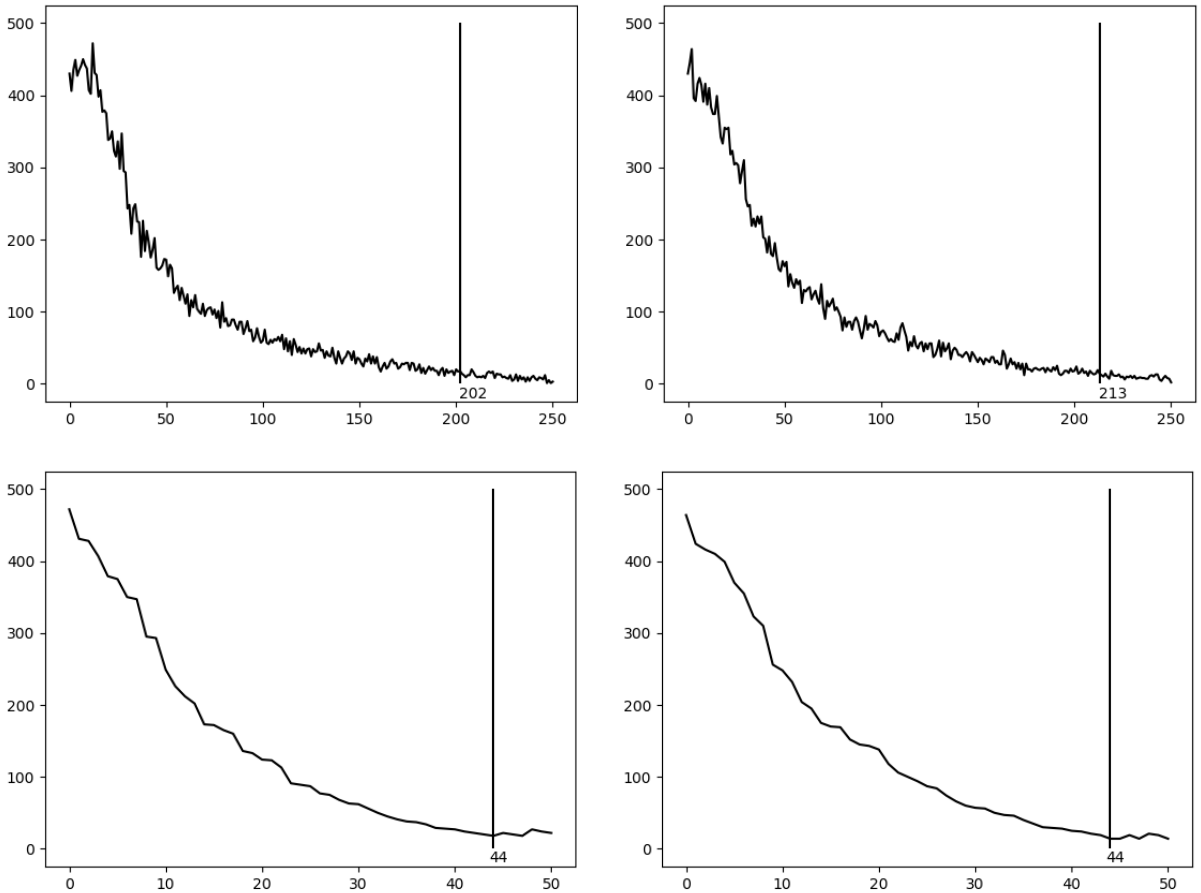

Fig. 8. Graphs of the density of the amplitude frequencies distribution for the seventh channel before (top) and after (bottom) correcting the noise threshold levels

Рис. 8. Графики плотности распределения частот амплитуд седьмого канала до (вверху) и после (внизу) корректировки пороговых уровней шума

Отметим, что на основе скорректированных пороговых уровней также будет производиться и оценка степени опасности найденного поверхностного дефекта. Вопросы оценки степени опасности выделенных на дефектограмме дефектов в данной статье не рассматриваются.

В случае с первым и вторым каналами данных ситуация с пороговыми уровнями полезных сигналов кардинально не изменилась. На рис. 5 пунктирная линия нового порога находится рядом со сплошной линией соответствующего исходного порогового уровня, отсекая всё те же полезные сигналы, поступившие от сварного алюминотермитного рельсового стыка.

Таким образом, предложенный в статье алгоритм корректировки порогового уровня шума может быть применён для любого вихретокового канала без предварительного анализа данных на предмет присутствия сигналов от продолжительного поверхностного дефекта. Корректировка порогов будет проведена по необходимости в зависимости от количества точек локальных минимумов на графиках плотности распределения частот амплитуд.

Алгоритм гарантированно останавливается. Полученный в результате пороговый уровень сохраняет прежний смысл, отмечая на сжатом (сглаженном) графике плотности распределения частот границу амплитуд рельсового шума.

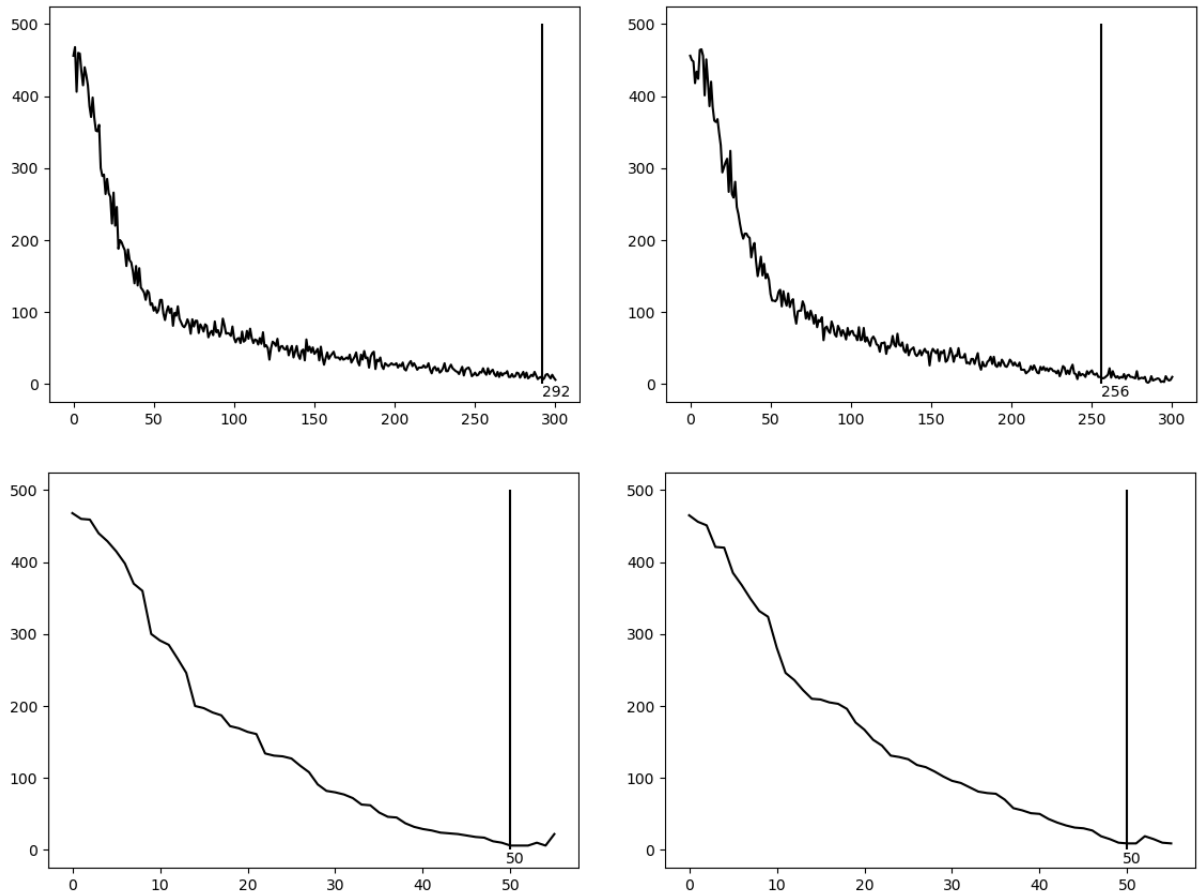

Fig. 9. Graphs of the density of the amplitude frequencies distribution for the eighth channel before (top) and after (bottom) correcting the noise threshold levels

Рис. 9. Графики плотности распределения частот амплитуд восьмого канала до (вверху) и после (внизу) корректировки пороговых уровней шума

Сжатый график на отрезке от нуля до порогового значения в приближении повторяет форму графика плотности нормального распределения вероятностей. Если запустить рассмотренный в предыдущем разделе алгоритм определения порогового уровня шума на этом фрагменте сжатого графика, будут выданы сопоставимые результаты. Например, для седьмого канала оба искомых уровня (и положительный, и отрицательный) будут иметь одинаковое абсолютное значение, равное 47 (найденные алгоритмом корректировки пороговые уровни имели значение 44).

Идея алгоритма основывается на результатах наблюдений, полученных на практике при обработке большого массива вихретоковых данных, а сам алгоритм корректировки приводится без строгого математического обоснования. В связи с этим представляет интерес построение математической модели влияния протяжённых поверхностных дефектов на плотность распределения частот амплитуд сигналов, зафиксированных на дефектограмме.

Заключение

Предложенный в статье алгоритм корректировки пороговых уровней амплитуд полезных сигналов показал свою эффективность на практике при неразрушающем контроле рельсов. В рамках аппаратно-программного комплекса вихретоковой дефектоскопии алгоритм успешно применяется для выделения сигналов, поступающих от протяжённых дефектов поверхности катания рельсов.

References

[1] A. A. Markov and E. A. Kuznetsova, *Rails flaw detection. Formation and analysis of signals. Book 1. Principles*. St. Petersburg: KultInformPress, 2010.

[2] A. A. Markov and E. A. Kuznetsova, *Rails flaw detection. Formation and analysis of signals. Book 2. Data interpretation*. St. Petersburg: Ultra Print, 2014.

[3] V. F. Tarabrin, A. V. Zverev, O. E. Gorbunov, and E. V. Kuzmin, "About Data Filtration of the Defectogram Automatic Interpretation by Hardware and Software Complex ASTRA", *NDT World*, vol. 64, no. 2, pp. 5–9, 2014.

[4] E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, and V. A. Tyukin, "Finding the Level of Useful Signals on Interpretation of Magnetic and Eddy-Current Defectograms", *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 658–666, 2018.

[5] E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, and V. A. Tyukin, "An Efficient Algorithm for Finding the Level of Useful Signals on Interpretation of Magnetic and Eddy Current Defectograms", *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 867–870, 2018.

[6] E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, V. A. Tyukin, and V. A. Bashkin, "Application of Neural Networks for Recognizing Rail Structural Elements in Magnetic and Eddy Current Defectograms", *Automatic Control and Computer Sciences*, vol. 53, no. 7, pp. 628–637, 2019.

[7] E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, V. A. Tyukin, and V. A. Bashkin, "Application of Convolutional Neural Networks for Recognizing Long Structural Elements of Rails in Eddy-Current Defectograms", *Modeling and analysis of information systems*, vol. 27, no. 3, pp. 316–329, 2020.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 1, 2021
journal homepage: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

Automated Teaching System "Sets" (Research for Organizing the 1st Part of the Project)

V. S. Rublev[1], M. D. Kondakov[1]

DOI: 10.18255/1818-1015-2021-1-90-103

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya, Yaroslavl 150003, Russia.

MSC2020: 03B35, 03F03, 97E60
Research article
Full text in English

Received December 4, 2020
After revision December 14, 2020
Accepted March 12, 2021

The issues of building an automated learning system "Sets" which will allow students to master one of the important topics of the discipline "Discrete Mathematics" and to develop logical and mathematical thinking in this direction are studied. The corresponding topic of the 1st part of the project includes materials related to the concept of a set, operations on sets, algebra of sets, proofs of statements for sets, and the derivation of formulas for the number of set elements. The system is based on a construction of the statements proof editor for a set and of the formulas derivation editor for the number of set elements, both editors are to be used for teaching. The first of these allows students to split the original statement into a number of simpler statements, taken together equivalent to the original statement, to choose a method of proving each simple statement and to conduct their step-by-step proof. The second editor allows (using the inclusion-exclusion principle and the formula of the number of complement elements) to derive a step-by-step formula for the number of set elements through the specified numbers of elements for sets from which the resulting set is constructed.

An important part of the system is to monitor the correctness of all actions of students, and on this basis the entire learning system is developed. The logical supervision over the correctness of the selected action in the first editor is performed by a Boolean function created by the system and corresponding to this action and by checking it for identical truth. In the second editor, invariants such as characteristic strings of the set and of its number of elements are used for verification. The rest of the system is related to learning of set algebra and to preparation to editors usage. The main focus here is on the learning strategy in which testing the understanding of the learned material is rather rigorous and eliminating the random choice of answers. The division of the material into sections with verification of the success of teaching not only by tests, but also by exercises and tasks, allows students to master the complex logical and mathematical techniques of proving statements for sets and derivation of formulas for the number of set elements.

Keywords: computer learning, discrete sets, set algebra, statements for sets, step-by-step proof, number of set elements, formula derivation, correctness control

INFORMATION ABOUT THE AUTHORS

Vadim S Rublev
correspondence author
orcid.org/0000-0002-0252-9958. E-mail: roublev@mail.ru
PhD, Professor.

Maxim D Kondakov
orcid.org/0000-0002-7238-3675. E-mail: kondakovmd99@mail.ru
student of the Department of Theoretical Informatics.

Funding: This work was supported by P. G. Demidov Yaroslavl State University Project № VIP-016.

For citation: V. S. Rublev and M. D. Kondakov, "Automated Teaching System "Sets" (Research for Organizing the 1st Part of the Project)", *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 90-103, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 1, 2021
сайт журнала: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

Автоматизированная обучающая система «Множества» (исследования организации 1-й части проекта)

В. С. Рублев[1], М. Д. Кондаков[1]

DOI: 10.18255/1818-1015-2021-1-90-103

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, г. Ярославль, 150003 Россия.

УДК 510.52:372.851
Научная статья
Полный текст на английском языке

Получена 4 декабря 2020 г.
После доработки 14 декабря 2020 г.
Принята к публикации 12 марта 2021 г.

Исследуются вопросы построения автоматизированной обучающей системы «Множества», которая позволит учащемуся освоить одну из важных тем дисциплины «Дискретная математика» и развить логико-математическое мышление в этом направлении. Соответствующая тема 1-й части проекта включает материал, связанный с понятием множества, операциями над множествами, алгеброй множеств, доказательствами утверждений для множеств, выводом формул для количества элементов множества. В основе системы лежит построение с целью использования для обучения редактора доказательства утверждений для множества и редактора вывода формул для количества элементов множества. Первый из них позволяет студенту разбить исходное утверждение на ряд более простых утверждений, в совокупности эквивалентных исходному утверждению, выбрать метод доказательства каждого простого утверждения и провести их пошаговое доказательство. Второй редактор позволяет, используя формулу включения и исключения и формулу количества элементов дополнения, вывести пошагово формулу для количества элементов множества через заданные количества элементов, связанных с ним множеств.
Важной частью системы является контроль правильности всех действий студента, и на этой основе разработана вся система обучения. Логический контроль правильности выбранного действия в первом редакторе осуществляется созданием системой булевой функции, соответствующей этому действию, и проверкой ее на тождественную истинность. Во втором редакторе для контроля используются такие инварианты, как характеристическая строка множества и характеристическая строка количества элементов множества. Остальная часть системы связана с обучением алгебре множеств и подготовке к использованию редакторов. При этом основное внимание уделяется стратегии обучения, при которой проверка понимания усвоенного материала является довольно строгой, исключающей случайный выбор ответов. Разбиение материала на секции с контролем успешности обучения не только тестами, но и упражнениями и задачами, позволяет студенту овладеть сложным логико-математическим аппаратом доказательства утверждений для множеств и вывода формулы для количества элементов множества.

Ключевые слова: компьютерное обучение, дискретные множества, алгебра множеств, утверждения для множеств, пошаговое доказательство, количество элементов множества, вывод формулы, контроль корректности

ИНФОРМАЦИЯ ОБ АВТОРАХ

Вадим Сергеевич Рублев
автор для корреспонденции
orcid.org/0000-0002-0252-9958. E-mail: roublev@mail.ru
кандидат физ.-мат. наук, профессор.

Максим Дмитриевич Кондаков
orcid.org/0000-0002-7238-3675. E-mail: kondakovmd99@mail.ru
студент кафедры теоретической информатики.

Финансирование: Работа выполнена в рамках инициативной НИР ЯрГУ им. П. Г. Демидова № VIP-016.

Для цитирования: V. S. Rublev and M. D. Kondakov, “Automated Teaching System “Sets” (Research for Organizing the 1st Part of the Project)”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 90-103, 2021.

Introduction

The challenge of teaching mathematical and computer sciences to the majority of students is caused by the low quality of school education, which is focused not on the thinking development, but on memorizing algorithms for solving some typical problems. This problem in the field of teaching students to develop new computer technologies can only be solved by individual teaching. But this approach requires a huge amount of additional time from the teacher. The solution is to develop computer-based automated teaching systems (ATS), which can be used not only to check knowledge, but also to conduct basic level teaching.

Most software systems called learning systems (for example, Moodle, Claroline, Dokeos, ATutor [1–11]) do not support the full learning cycle (methodologies) – they are just applications that provide access to texts, issue tests, and check student's memory ability. The best solution is to use various methods of adaptive learning in the learning processes, which are focused on a specific subject of study and on the individual characteristics of each student. Their features are:

- intelligent analysis of problem solutions;
- interactive support for problem solving;
- support in solving problems using examples;
- adaptive navigation support;
- adaptive view;
- adaptive support for collaboration of system users (students).

This provides the system with flexibility in relation to users and in relation to the presentation of a study material. In addition, the theoretical and methodological basis is the statement that teaching can be reduced to a set of the following components:

- information needed to study;
- control activities that allow you to test your knowledge of given materials;
- the way to assess the level of knowledge gained;
- follow-up management – an important and complex mechanism that makes the system namely teaching.

When developing such a system, you need to solve a number of problems: how to divide the material, how to check knowledge, and how flexible the system will be in relation to users.

The check by testing of student's memory is insufficient for the ATS for branches of exact sciences, which should teach data analysis, formalization, analysis, reasoning, and transformation. The use of computer algebra [12–15] is the basis for the construction of such systems. For example, one of the paper authors used this to construct ATS of computational complexity of algorithms [16–21].

In this paper, we consider computer-based learning models for proving statements for sets and deriving formulas for the number of set elements through specified numbers of other sets elements. These models can be divided into 2 groups: models for conducting of proofs for assertions or deriving of formulas for the number of set elements, and learning models that prepare students to use the first group of models.

1. The problem of constructing a proof editor of statements for sets

To solve the problem specified in the headline, select the following task sequence:

1. Limitations on the type of statements for sets for whose prove you need to build an editor.
2. Equivalent transformation of the sets included in the statement.
3. Splitting the basic statement into an equivalent set of simple statements.
4. The choice of the method of proving a simple statement and the selection of an initial set of premises in it.
5. Definition of the elementary step of the proof.
6. Control of the correctness proof by the editor.

1.1. Statement type constraints for sets

In the general case of the statement we will consider some *universal* set U and its subsets $X_1, X_2, \dots, X_n$. The statement uses formulas for these subsets, constructed with operations *complement, intersection, union*, and brackets, changing the order of these operations, if necessary.

We restrict ourselves to statements for sets of the following form

<*conjunction of set relations*> $\{\longrightarrow \mid \leftrightarrow\}$ <*conjunction of set relations*>.

So in example (1)

$$\overline{X_1 \cap \overline{X}_2 \cup \overline{X}_3} = (X_2 \cup X_3) \cap \overline{X}_1 \;\leftrightarrow\; X_1 \cap X_2 \cap X_3 = \varnothing \;\wedge\; X_2 \subseteq X_1 \cup X_3 \tag{1}$$

it is argued that the equality of two sets given by expressions takes place if and only if the intersection of the subsets is empty and the second subset is included in the union of the other two.

1.2. Equivalent transformation of sets included in the statement

To simplify the process of a proof conducted by a student, it is recommended to simplify the complex expressions of some sets of statements. In this case, we mean the representation of a complex expression in the form of a union of sets (and their intersections) in some cases or as the intersection of sets (and their unions) in other cases. So in example (1) transformation of the set on the left side of the set equality (it by A) denote

$$A \equiv \overline{X_1 \cap \overline{X}_2 \cup \overline{X}_3} = (\overline{X}_1 \cup X_2) \cap X_3 = (\overline{X}_1 \cap X_3) \cup (X_2 \cup X_3) \tag{2}$$

gives both views (2). The set of the right part of the equation (it by B) denote is already represented in (1) by an intersection, and the union is obtained by the following transformation

$$B \equiv (X_2 \cup X_3) \cap \overline{X}_1 = X_2 \cap \overline{X}_1 \cup X_3 \cap \overline{X}_1. \tag{3}$$

The reason why such representations of the set are important is the possibilities of simplifying the conduct of the proof. The representation of a set as a union of its subsets allows us to divide the further proof into separate branches, where the membership of an element to a set becomes easier by dividing it into cases of belonging to a particular subset, and the proof for each such branch is simplified and can be conducted separately. Representation in the form of intersection simplifies the proof of the conclusion about non-belonging to some set due to the fact that it is enough to obtain first the result about non-belonging to one of its subsets of intersection.

1.3. Splitting the basic statement into an equivalent set of simple statements

If there is an equivalent operation in the basic statement, the statement can be replaced by a conjunction of two statements with implications in one direction and the other. So the statement example (1) can be replaced by the conjunction of the following statements (with the replacement of the parts of the set equality by the introduced notations A and B):

$$A = B \;\longrightarrow\; X_1 \cap X_2 \cap X_3 = \varnothing \;\wedge\; X_2 \subseteq X_1 \cup X_3 \tag{4}$$

$$X_1 \cap X_2 \cap X_3 = \varnothing \;\wedge\; X_2 \subseteq X_1 \cup X_3 \;\longrightarrow\; A = B \tag{5}$$

Note that statement (4), having in conclusion the conjunction, can be spitted into 2 statements. In statement (5), the ratio of equality of 2 sets at the end of the implication can be replaced by the conjunction of 2 inclusion, and therefore the statement (5) can also be divided into 2. As a result, we obtain the following partition of the basic statement (1):

$$A = B \;\longrightarrow\; X_1 \cap X_2 \cap X_3 = \varnothing \tag{6}$$

$$A = B \longrightarrow X_2 \subseteq X_1 \cup X_3 \tag{7}$$

$$X_1 \cap X_2 \cap X_3 = \emptyset \wedge X_2 \subseteq X_1 \cup X_3 \longrightarrow A \subseteq B \tag{8}$$

$$X_1 \cap X_2 \cap X_3 = \emptyset \wedge X_2 \subseteq X_1 \cup X_3 \longrightarrow B \subseteq A \tag{9}$$

An example of reducing the proof of the basic statement (1) to the proof of 4 simple statements (6)-(9) shows that this can be done in other cases of the basic statements.

1.4. The choice of a method of proving a simple statement and the selection of the initial set of premises in it

In a simple statement, when performing the premises of the left part of the implication, it is necessary to prove the truth of the right part of the implication (call it a *target* statement). There are 2 methods of proof when the target is the ratio of inclusion of sets:

- **direct** method when we prove for an *arbitrary* element of the universal set, that from the belonging of an element to the included set follows its belonging to the including set (call it a *target output*), that is, the accuracy of the inclusion.
- **indirect** method, when we assume the opposite in the target statement (*exists an element of the universal set that belongs to the included set of the target relation of the inclusion of sets, but does not belong to the including set*) and by a consistent output come to a contradiction with the conjunction of premises left part of the implication to this simple statement.

For example, in simple statements (7)-(9) both methods are possible. However, for statement (7) it is more rational to apply the method (perhaps indirect fewer steps of proof), and for statements (8)and (9) the direct method is more rational.

If the target statement is the equality of a set to an empty set or to a universal set, then only the indirect method is rational. In this case, the existence of its element is opposite for an empty set, and the existence of an element that does not belong to this set is opposite for the equality of a certain set to a universal set. For a simple statement (6) you need to use an indirect method and show the contradiction with the premises of the statement.

The proof of any method begins with sequential steps, each of which is based on a premise. The initial premise in the direct method of proof is that an arbitrary element of the universal set belongs to the included set of the target statement. For example, for statement (8), the initial premise is the expression $\forall x : x \in A$.

The initial premise in the indirect method of proof is the denial of the target statement, which is expressed by the existence of an element that contradicts the target statement. For example, for (6), the initial statement premise is the expression $\exists x : x \in X_1 \cap X_2 \cap X_3$, and for (7) – statement $\exists x : x \in X_2, x \notin X_1 \cup X_3$.

In addition to the initial premise, conclusions can be based on assumptions statement related to the conditions (left side of the implication). The statement system prepares them as the initial set of premises, adding to it the initial premise. Meanwhile

1) to the equality of sets (for example, $C = D$) there correspond 4 premises (in the example, $x \in C \longrightarrow x \in D$, $x \in D \longrightarrow x \in C$, $x \notin C \longrightarrow x \notin D$, $x \notin D \longrightarrow x \notin C$);
2) to the inclusion case (for example, $X_2 \subseteq X_! \cup X_3$) there correspond 2 premises (in the example, $x \in X_2 \longrightarrow x \in X_1 \cup X_3$ and $x \notin X_1 \cup X_3 \longrightarrow x \notin X_2$);
3) to the equality of a set to the empty set (for example, $X_1 \cap X_2 \cap X_3 = \emptyset$) there correspond 1 premise (in the example, $x \notin X_1 \cap X_2 \cap X_3$);
4) and the equality of a set to the universal set (for example, $X_1 \cup X_2 \cup X_3 = U$) also matches 1 premise (in the example, $x \in X_1 \cup X_2 \cup X_3$).

For each elementary conclusion of the proof, if it is true, the system adds the conclusion as a premise to the set of premises of the proof or branch of the proof (more on that in the next section).

1.5. Definition of the elementary step of the proof

The proof is conducted with the help of a sequence of steps, at each of which an elementary conclusion is drawn, based on the indicated premises for conclusion. For educational purposes, we limit ourselves to only elementary conclusions based on no more than 2 premises. For elementary conclusions, the following ideas are used:

1. If an element belongs to a certain set (for example, premises $x \in C$), the output can be its belonging to any set, covering the set of the premises (in the example, the output $x \in C \cup D$).
2. If an element belongs to two sets (for example, 2 premises $x \in C$ and $x \in D$), the output can be an element belonging to the intersection of these sets (in the example output $x \in C \cap D$).
3. If an element belongs to a set (for example , $x \in C \cap D$), it belongs to each part of it (in the example, 2 outputs $x \in C$ and $x \in D$).
4. If an element belongs to a set (for example, the premise $x \in C$), it does not belong to its complement (in the example the output $x \notin \overline{C}$).
5. If an element does not belong to a set (for example, premise $x \notin C$), it belongs to its complement (in the example, the output $x \in \overline{C}$)).
6. If an element does not belong to two sets (for example, 2 premises $x \notin C$ and $x \notin D$), it does not belong to the union of these sets (in the example the output $x \notin C \cup D$).
7. If an element does not belong to a union of sets (for example, the premise $x \notin C \cup D$), it does not belong to any of these union sets (in the example, the output $x \notin C$ and the output $x \notin D$).
8. If an element belongs to a union of sets (for example, the premise $x \in C \cup D$), it can belong to one of these union sets (in the example, the output $x \in C$ and the output $x \in D$) – this output is called *the splitting of the cases*.

Note that the splitting into cases can be conducted in different ways. A partition where the sets of cases do not intersect is called *alternative*. As an example of the premise $x \in C \cup D$ you can write the following division into cases: the output $x \in C$ and the output $x \in D \cap \overline{C}$. This is especially important when for one case, the further conclusion is easily built. Then, for the second case, additional information is obtained, which can help in the further conclusion. If the first case is also difficult, it can be divided into 3 cases: the output $x \in C \cap \overline{D}$, the output $x \in C \cap D$ and the output $x \in D \cap \overline{C}$.

Note also that the division into alternative 2 cases of belonging to a certain or set non-belonging to this set can always be done without relying on premises (for example, the output $x \in C$ and the output $x \in \overline{C}$ form 2 cases and do not require a premise).

Each conclusion, if made correctly, is added as a premise to the preceding set of premises. But when cases appear, each of them is associated with its own independent branch of proof and many premises of this branch, which is formed from the previous set of premises by adding a new premise – the case output.

Each branch of the proof must end with either a white square denoting the receipt of the target output, or a black square denoting the receipt of a contradiction. If all branches of the proof from the opposite ended with a contradiction, the proof of the statement was successful. If in the direct method the proof of all branches ended, but there are branches that ended in success (a white square), the proof was successful.

1.6. Editor control of the proof

The system allows the trainee to build a proof of the basic statement. But the system has to control all his or her actions, starting from splitting into simple statements, selecting the method of proof with the organization of an initial premise, performing each elementary step of the proof up to the completion of each branch of the proof.

To this purpose, the system, at the end of the above actions (by pressing a button **Verify**), builds a Boolean function corresponding to a statement for sets (simple statements into which the basic statement is splitted, or statements of the original premise, or the statement of an elementary step, as implications of

conjunctions of the premises and output of the elementary step or implication of the premise and disjunction of the output of cases) and verifies its identity truth. This Boolean function (let us call it *BIF* Boolean Identity Function) is constructed as follows:

1. Each set $x \in X_i$ is replaced with a boolean variable y_i, whose value is equal to the truth of the statement that the element x belongs to this set ($y_i \equiv x \in X_i$). Other sets are replaced in the same way. For example, the set A is replaced with a boolean variable a,whose name is a lowercase letter, corresponding to the name of the set and its true value coincides with the value of the statement of the belonging of an element x to this set, i.e. $a \equiv x \in A$.
2. Operations for sets of union, intersection and complement are replaced accordingly on operations of negation, conjunction and disjunction for the corresponding subsets of statements.
3. The equality for sets is replaced by the equivalence operation of the corresponding statements.
4. The inclusion for sets is replaced by the operation of implication of the corresponding statements.
5. The equality of the set to the empty set is replaced by the negation of the corresponding statement for the set.
6. The equality of the set to the universal set is replaced by the corresponding statement for the set.

So in the considered example (2) of identical representations of the set A we obtain the following BIF. $f_a = (\overline{y_1 \wedge \overline{y_2} \vee \overline{y_3}} \leftrightarrow (\overline{y_1} \vee y_2) \wedge y_3) \wedge (\overline{y_1 \wedge \overline{y_2} \vee \overline{y_3}} \leftrightarrow (\overline{y_1} \wedge y_3) \vee (y_2 \wedge y_3))$ and since it is identically true we, then by theorem *on the connection of expressions of the algebra of sets and the algebra of statements* and its consequences (see, for example [22] the system confirms the correctness of transformations of the set A.

Next, for a simple statement (6), the BIF will look like this: $f_6 \equiv (\overline{y_1 \wedge \overline{y_2} \vee \overline{y_3}} \leftrightarrow (y_2 \vee y_3) \wedge \overline{y_1}) \longrightarrow \overline{y_1 \wedge y_2 \wedge y_3}$. Its identical truth also confirms the correctness of the simple statement (6).

When conducting proof from the contrary of this statement, the student receives the initial premise. $\exists x : x \in X_1 \cap X_2 \cap X_3$ from the negation of the proved statement $\overline{X_1 \cap X_2 \cap X_3 = \emptyset}$ and BIF for verification gets the following expression: $f_{u1} \equiv y_1 \wedge y_2 \wedge y_3 \leftrightarrow \overline{\overline{y_1 \wedge y_2 \wedge y_3}}$, and since it is identically true, the initial premise is built correctly by the student.

From the received initial premise follows a conjunction of 3 elementary conclusions. $x \in X_1 \wedge x \in X_2 \wedge x \in X_3$, what is verified by the identical truth of BIF of the function of implication of the function of premise and conjunction of functions of elementary conclusions $y_1 \wedge y_2 \wedge y_3 \longrightarrow y_1 \wedge y_2 \wedge y_3$.

2. Formulas derivation editor for the number of set elements

The student's work with this editor can be divided into the following stages:

1. Derivation of the formula for the desired set determined by the problem through the original sets of the problem.
2. Derivation of the formula for the desired set determined by the problem through the original sets of the problem.
3. Derivation of the formula for the number of the desired set elements through the specified numbers of elements from other sets of the problem using the inclusion-exclusion principle, or using formulas for complements to the sets of the problem.

As an example, consider the following problem:

> *85% of citizens annually attend entertainment events: theaters, museums, cinemas. Moreover, 43% visit theaters, 25% visit museums, 7% visit cinemas and museums, 15% visit theaters and cinemas, and 5% visit all 3 shows, but 45% do not visit theaters or museums. What is the percentage of citizens who attend only 1 type of entertainment?*

First of all, students must enter the notations for the source sets and write a formula for the desired set. Let's introduce the notations T, M, and K, respectively, for the sets of citizens who visit theaters, museums, and cinemas.

A student can write a formula for the desired set based on the complement of the set of those who attend only one show:

$$X \equiv \overline{T \setminus (M \cup K) \cup M \setminus (T \cup K) \cup K \setminus (T \cup M)}, \tag{10}$$

but this student can write another formula based on taking union of those who do not attend anything and those who attend at least 2 types of shows:

$$X \equiv \overline{T \cup M \cup K} \cup T \cap M \cup T \cap K \cup M \cap K. \tag{11}$$

The system must check whether the entered formula matches the task conditions correctly. To do this we divide the universal set U into 8 pairwise disjoint subsets in the following order:

$$(\overline{T} \cup \overline{M} \cup \overline{K}, \overline{T} \cup \overline{M} \cup K, \overline{T} \cup M \cup \overline{K}, \overline{T} \cup M \cup K, T \cup \overline{M} \cup \overline{K}, T \cup \overline{M} \cup K, T \cup M \cup \overline{K}, T \cup M \cup K).$$

Let us assign each set of U to its characteristic string (CS) of 0,1, in which all its parts are defined as units. For example, for the set from the problem described above CS=(10010111). For sets T, M, and K, their CSs are respectively equal to (00001111), (00110011), (01010101). For the desired set X, its CS is obtained by applying to CSs of initial sets operations of bitwise conjunction, disjunction, and negation corresponding to the intersection, union, and complement operations included in the set formula. The ground for this follows, for example, from [22, 23]. Therefore, the correctness of any transformation of a set is controlled by the system by comparison with the CS of this set.

While deriving a formula for the number of elements, the inclusion-exclusion principle is used, as well as the formula for the number of elements for the complement set. The correctness of their application should also be monitored. To do this, the characteristic string of the quantity (CSQ) for the set elements which is similar to the CS is introduced. Its elements are equal to the multiplier with which the number of elements for the corresponding part of the set is included in the sum of the number of the set elements. While adding (or subtracting) such strings, their elements are added (subtracted) bitwise. Note that if for the number of set elements in CSQ all elements are from 0,1, then CSQ coincides with the CS of this set.

Let us illustrate this by the example under consideration. Firstly, let us introduce the numbers of elements for the sets given initially:

$$|T \cup M \cup K| = 85, |T| = 43, |M| = 25, |T \cap K| = 15, |M \cap K| = 7, |\overline{T} \cap \overline{M}|.$$

Let us take a simpler formula (11) for determining the numbers of elements of the set X. It is a union of 2 disjoints sets: complement to the union of main sets X1, X2, X3, and the union of pairwise intersections of these sets. Therefore, using the inclusion-exclusion principle, we get:

$$|X| = |\overline{T \cup M \cup K}| + |T \cap M \cup T \cap K \cup M \cap K|. \tag{12}$$

The first term in (12), using the number of elements of the complement, is replaced by the difference of sets:

$$|\overline{T \cup M \cup K}| = |U| - |T \cup M \cup K|, \tag{13}$$

and the second one is expanded by the inclusion-exclusion principle

$$|T \cap M \cup T \cap K \cup M \cap K| = |T \cap M| + |T \cap K| + |M \cap K| - 2|T \cap M \cap K|. \tag{14}$$

Checking the CSQ for (14) also gives a valid equality

$$(00010111) = (00000011) + (00000101) + (00010001) - 2(00000001).$$

Using the equalities (13) and (14), we make a substitution in (12)

$$|X| = |U| - |T \cup M \cup K| + |T \cap M| + |T \cap K| + |M \cap K| - 2|T \cap M \cap K|. \tag{15}$$

In the resulting formula (15), all the terms are known except . Let us express the formula for it using the complement

$$|T \cap M| = |U| - |\overline{T \cap M}| = |U| - |\overline{T} \cup \overline{M}|. \tag{16}$$

Transform the subtractive to (16) using the inclusion-exclusion principle and then, using the formula for complements, we obtain

$$|\overline{T} \cup \overline{M}| = |\overline{T}| + |\overline{M}| - |\overline{T} \cap \overline{M}| = 2|U| - |T| - |M| - |\overline{T} \cap \overline{M}|. \tag{17}$$

Substitute the result in formula (16) and then after substituting in (15) we get the final formula (18)

$$|X| = |T| + |M| + |\overline{T} \cap \overline{M}| + |T \cap K| + |M \cap K| - 2|T \cap M \cap K| - |T \cup M \cup K|. \tag{18}$$

Its check by performing actions on the CSQ of terms

$$(00001111) + (00110011) + (11000000) + (00000101) + (00010001) - (00000002) - (01111111) = \\ = (11121224) - (01111113) = (10010111)$$

shows a coincidence with the CSQ of the desired set. Substituting the values of summands in formula (18)

$$|X| = 43 + 25 + 45 + 15 + 7 - 2 \cdot 5 - 85 = 40. \tag{19}$$

We get the answer to the question of the problem: *40% of citizens, except for those who attend only 1 type of entertainment.*

3. Checking by the formula derivation correctness editor

The example above for solving a problem with supervising shows how the editor can check the logic of the formula derivation. But this check is not yet complete, because, firstly, the check of syntactic correctness of the entered expressions is not defined, and, secondly, the exact steps of the formula derivation are not defined. For entering expressions, it is needed to use syntax verification using the finite state grammar. For step-by-step derivation, it is needed to introduce a system of elementary transformations of formulas for a set and for the number of its elements. In this case, an elementary transformation is performed for the selected fragment of the transformed formula.

For sets formulas, we include the following elementary transformations in the system:

1. Removing the double complement.
2. Permutation of sets in the intersection.
3. Permutation of the sets in the union.
4. Absorption for intersection-repeats of sets in an intersection or an intersection with the universal set are removed.
5. Absorption for union-repeats of sets in a union or a union with an empty set are removed.
6. Removal of parentheses-usage of the distributive law for intersection with the union of sets.
7. Factorization using the distributive law for unions of intersections.
8. Transformation of the difference, that is the change of the subtraction to the intersection with the complement.
9. Removing the complementation sign-using the duality law.
10. Inclusion of the complementation sign-using the duality law.

11. Notation for a part of the set-introduction of a new formula.
12. Formula substitution-replacing the notation of a set by its formula.
13. Replacing of a union with a complement with the universal set.
14. Replacing of the universal set by a union of an arbitrary set with its complement.
15. Replacing of an intersection with a complement with an empty set.
16. Replacing of an empty set by an intersection of an arbitrary set with its complement.

Using elementary set transformations can lead to a long derivation. Therefore, along with the list of elementary transformations, we introduce a list of grouped transformations:

1. Multiple operation application-if there are several fragments selected in the formula, the elementary operation is applied to each fragment;
2. Multiple absorption-in the selected fragment of an intersection or a union of sets, repetitions of sets are removed, as well as unions with an empty set or an intersection with the universal set.

When we derive the formula for the number of set elements, we use the following transformations:

1. The inclusion-exclusion principle for a union of a given number of sets;
2. Replacing the number of elements in a difference of sets by the difference of their elements numbers-if the subtracted set is included in the reduced set;
3. Replacing the number of set elements with the difference between the quantities of the universal set and the complement of a set;
4. Mutual deletion of similar terms with different signs;
5. Deleting of terms with empty intersections;
6. Collecting terms-checks the similarity of all selected terms and the correctness of the coefficient calculation.

4. Teaching models

4.1. Teaching strategy

All teaching material can be divided into two large categories. The first category includes sections that provide basic theoretical knowledge and concepts on set definition, operations with sets, and set algebra. The main purpose of these sections is to develop and train a student's memory using memorization skills. These sections are most often checked by testing.

However, the course program also includes teaching the informal usage of mathematical techniques, and at this point there are additional difficulties. These include technical difficulties associated with entering of formulas and their transformation, and difficulties associated with an insufficient level of the logical thinking of a student. This student must achieve the goal by breaking a derivation or a proof into parts, choosing methods for these parts, and constructing a sequence of derivations leading to the goal. Thus, defined is the second category of sections related to the mathematical methodology of conducting a proof of an assertion for sets or deriving a formula for the number of set elements.

The following features distinguish the second category of sections from the first one. Firstly, the check of the material of sections from the second category cannot be based only on testing, because it is necessary to check the student's ability to use various mathematical techniques. Secondly, when studying the material, it is necessary to teach the student to link individual techniques into a purposeful process by constructing a sequence of studied techniques. Thirdly, you need to check the ability to link several processes when solving the final problem. One of the tools used in checking the sections material from the second category is the algorithm for checking the syntactic correctness of the input and the algorithm for checking the correctness of the derivation.

The material and the check interact with the third component of the system, which is responsible for determining the volume of the material in a single session, and sets a set of tasks and their number, as well as other system parameters. It is namely this component that gives flexibility to the system and distinguishes

ATS from an application that can only output text and a set of tests based on it. Further we give a description of the proposed scenario for the interaction between the system and a user.

After logging in, the user sees a list of course sections that are divided into accessible and yet not accessible ones according to the course plan. Sections must be completed one after the other in their order (the principle of linear learning). Upon the entry to the section, the student is provided with its material for studying. After studying the material, the student proceeds to the check of the acquired knowledge, for which he or she must complete the initial number of tasks defined in the section (tests, exercises, tasks). Each question in the test offers several possible answers (usually 6), randomly selected from a pre-defined set of answers for each task. Among the suggested answers, there may be several correct answers, or all correct answers, or none correct answers. The student must mark all correct answers. But if, in his opinion, there are no correct answers, then he should choose this answer. The system considers the task completed if the student has marked all the correct answers and none of the incorrect ones. Otherwise, it displays one of the texts: "Not all correct answers are marked", "Some of the marked answers are not correct", or a combination of them, and provides a text fragment of the section material associated with the error. After studying this fragment or re-studying the entire material of the section, the student has the opportunity to re-answer the question of the task. If the user does not complete the task again, it is replaced with two additional tasks. Thus, the number of tasks to complete the section may increase. The session with the student is terminated when a certain number of tasks in the section is reached, and the student is able to return to learning only after a certain break. If a student reaches a session interruption several times in a row, his or her account in the system is temporarily blocked, and he or she is called to the teacher. In the case when the student first earned a lot of additional tasks, and then began to answer correctly, the number of control tasks begins to decrease in some progression. This approach makes the student carefully and thoughtfully treat the material of the section. In this way, not only the check of the section knowledge is organized, but also the learning. Only after completing all the tasks the student is able to proceed to the study of the material in the next section.

However, there are sections whose material is based on the previous sections and can not be mastered without absolute possession of the material of these previous sections. In such sections, an additional check of the knowledge of previous sections is carried out. In this case, the number of initial tasks for repeating the section is reduced to one.

Note also that custom teaching parameters are defined for configuring ATS. For example, the initial number of test tasks in each section or sub-section, the number of tasks to be added if the answer is incorrect, the number of tasks to be reduced for every 2 consecutive correct answers, and so on.

4.2. Splitting into sections

All the material is divided into the following sections and subsections.

1. Set definition. Tests to determine correct or incorrect answers.
2. Relations of sets:
   (a) The equality relation. Tests to determine correct or incorrect answers.
   (b) The inclusion relation. Tests to determine correct or incorrect answers.
3. Operations on sets:
   (a) Sets union. Tests to determine correct or incorrect answers.
   Proof and transformation exercises.
   (b) Intersection and complement of sets. Tests to determine correct or incorrect answers. Proof and transformation exercises.
4. Set algebra. Tests to determine correct or incorrect answers.
   Exercises on specification of sets formulas.
5. Methodology of proving statements for sets.

   (a) Premises and a conclusion of a proof step. Tests to determine correct or incorrect answers. Exercises for drawing conclusions from given premises. Exercises for determining premises for a given conclusion.
   (b) Splitting into cases. Tests to determine whether the answers are correct or incorrect in relation to the branches of a proof. Exercises for splitting into normal and alternative cases.
   (c) Direct method of proof. Tests to determine whether the answers are correct or incorrect regarding the completion of proof branches. Exercises on determination of the initial premise and the conclusion derived from it.
   (d) Indirect method of proof. The tests to determine correct and incorrect answers relative to the branches of a proof and their completion. Exercises for setting the initial premise. Exercises for double division into cases.
6. Errors in a proof. Tests to determine correct and incorrect answers.
7. System of transformations for formulas for a set.
   (a) Elementary transformations. Tests to determine correct and incorrect answers.
      i. Permutations and absorptions. Exercises on transformations.
      ii. Transformations with parentheses. Exercises on transformations.
      iii. Transformations with complement sets. Exercises on transformations.
      iv. Notation and substitution of formulas. Exercises on transformations.
   (b) Multiple transformations.
      i. Multiple application of an operation. Exercises on transformations.
      ii. Multiple absorption. Exercises on transformations.
8. Final section for assertion proof. The problem of proving a statement for sets.
   (a) Numerical illustrations with general cases when all conditions of a statement are met and when each of the conditions is not met.
   (b) Venn Diagrams with general cases of fulfillment of all statement conditions and non-fulfillment of each of the conditions.
9. Formulas connecting the numbers of sets elements.
   (a) The inclusion-exclusion principle. Tests to determine correct and incorrect answers. Exercises for derivations of a formula for sets with singularities.
   (b) Formula for the number of elements for the difference of sets and the complement of a set. Tests to determine correct and incorrect answers.
10. System of transformations for formulas for the number of set element. Tests to determine correct or incorrect answers.
   (a) The inclusion-exclusion principle for a union of a given number of sets. Exercises on transformations.
   (b) Replacing the number of elements in sets difference by the difference of their elements numbers. Exercises on transformations.
   (c) Replacing the number of set elements by the difference between the quantities of the universal set and the set complement. Exercises on transformations.
   (d) Mutual deletion of similar terms with different signs. Exercises on transformations.
   (e) Deleting terms with empty intersections. Exercises on transformations.
   (f) Collecting similar terms. Exercises on transformations.
   (g) Notation and substitution of formulas. Exercises on transformations.
11. Final section for formulas for the number of set elements. Task to derive a formula for the number of set elements.

Conclusion

We hope that the described approach to construct an automated teaching system for proving statements for sets will be implemented and will show effectiveness in teaching this subject and developing students' logical and mathematical thinking.

References

[1] A. Büchner, *Moodle 3 Administration. Third Edition*. Packt Publishing, 2016.

[2] S. K. Udaya and T. V. Vamsi Krishna, "E-Learning: Technological Development in Teaching for school kids", *International Journal of Computer Science and Information Technologies*, vol. 5, no. 5, pp. 6124–6126, 2014.

[3] M. Kerres, D. Meister, F. von Gross, and U. Sander, *Enzyklopädie Erziehungswissenschaft*. 2012.

[4] S. Clark and J. Baggaley, "Assistive Software for Disabled Learners", *The International Review of Research in Open and Distributed Learning*, vol. 5, no. 3, 2004. [Online]. Available: https://doi.org/10.19173/irrodl.v5i3.198.

[5] A. Nagy, "The Impact of E-Learning", in *E-Content: Technologies and Perspectives for the European Market*, Berlin: Springer-Verlag, 2005, pp. 79–96. [Online]. Available: https://doi.org/10.1007/3-540-26387-X_4.

[6] I. E. Allen and J. Seaman, *Changing Course: Ten Years of Tracking Online Education in the United States*. Babson Survey Research Group, 2013.

[7] D. Burgos *et al.*, Eds., *Higher Education Learning Methodologies and Technologies Online*, Springer, 2019, pp. 6–7.

[8] A. Holzinger *et al.*, Eds., *Machine Learning and Knowledge Extraction*. Springer, 2019.

[9] Y. Zhang and D. Cristol, *Handbook of Mobile Teaching and Learning*. Springer, 2019.

[10] S. Kong and H. Abelson, *Computational Thinking Education*. Springer Nature, 2019.

[11] G. Hanna, D. Reid, and M. de Villiers, *Proof Technology in Mathematics Research and Teaching*. Springer, 2019.

[12] M. England and other, Eds., *Computer Algebra in Scientific Computing*, Springer, 2019.

[13] J. H. Davenport, Y. Siret, and E. Tournier, *Computer algebra: systems and algorithms for algebraic computation*. Academic Press, 1988.

[14] J. Gathen and J. Gerhard, *Modern Computer Algebra*. Cambridge University Press, 2013.

[15] V. B. Taranchuk, *The main functions of computer algebra systems*, in Russian. Minsk: BSU, 2013.

[16] A. V. Ermilova and V. S. Rublev, "Problems of mathematical thinking development for students basing on example of teaching system to the course "Algorithms and computational complexity analysis"", in Russian, in *Proceedings of IX International theoretical and practical conference Information Technologies and IT-Education*, Moscow: INTUIT.RU, 2014, pp. 297–304.

[17] V. S. Rublev and M. T. Yusufov, "Automated system for teaching computational complexity analysis of algorithms", in Russian, *Modern Information Technologies and IT-Education*, vol. 12, no. 1, pp. 135–145, 2016.

[18] V. S. Rublev and M. T. Yusufov, "Automated teaching system "Analysis of algorithms computational complexity" (research for organizing the 1st part of the project)", in Russian, *Modern Information Technologies and IT-Education*, vol. 13, no. 2, pp. 170–178, 2017.

[19] V. S. Rublev and M. T. Yusufov, “Automated System for Teaching Computational Complexity of Algorithms Course”, in Russian, *Modeling and Analysis of Information Systems*, vol. 24, no. 4, pp. 481–495, 2017.

[20] V. S. Rublev and M. T. Yusufov, “Models for teaching analysis of algorithm computational complexity”, in Russian, in *3rd International on Computer Algebra and Information Technologies*, Odessa: ONSU, 2018, pp. 157–160.

[21] V. S. Rublev and D. R. Vakhmyanin, “Automated system for teaching «Proof of statements for sets»”, in Russian, *Modern Information Technologies and IT-Education*, vol. 15, no. 4, pp. 1014–1027, 2019.

[22] V. S. Rublev, *Boolean functions (individual work #4 and #5 for the course “Discrete Mathematics”)*, in Russian. Yaroslavl: YSU, 2018.

[23] V. S. Rublev, *Sets (individual work #1 for the course “Discrete Mathematics”)*, in Russian. Yaroslavl: YSU, 2018.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 1, 2021
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

LTL-Specification of Counter Machines

E. V. Kuzmin[1]

DOI: 10.18255/1818-1015-2021-1-104-119

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya str., Yaroslavl 150003, Russia.

MSC2020: 68Q60, 68N30, 68Q04, 03B44, 03B70, 03D10
Research article
Full text in Russian

Received January 11, 2021
After revision February 12, 2021
Accepted March 12, 2021

The article is written in support of the educational discipline "Non-classical logics". Within the framework of this discipline, the objects of study are the basic principles and constructive elements, with the help of which the formal construction of various non-classical propositional logics takes place. Despite the abstractness of the theory of non-classical logics, in which the main attention is paid to the strict mathematical formalization of logical reasoning, there are real practical areas of application of theoretical results. In particular, languages of temporal modal logics are widely used for modeling, specification, and verification (correctness analysis) of logic control program systems. This article demonstrates, using the linear temporal logic LTL as an example, how abstract concepts of non-classical logics can be reflected in practice in the field of information technology and programming. We show the possibility of representing the behavior of a software system in the form of a set of LTL-formulas and using this representation to verify the satisfiability of program system properties through the procedure of proving the validity of logical inferences, expressed in terms of the linear temporal logic LTL. As program systems, for the specification of the behavior of which the LTL logic will be applied, Minsky counter machines are considered. Minsky counter machines are one of the ways to formalize the intuitive concept of an algorithm. They have the same computing power as Turing machines. A counter machine has the form of a computer program written in a high-level language, since it contains variables called counters, and conditional and unconditional jump operators that allow to build loop constructions. It is known that any algorithm (hypothetically) can be implemented in the form of a Minsky three-counter machine.

Keywords: non-classical logic, linear temporal logic, counter machines, LTL-specification

INFORMATION ABOUT THE AUTHORS

Egor V. Kuzmin
correspondence author

orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru
Professor, Doctor of Science.

Funding: This work was supported by P. G. Demidov Yaroslavl State University Project № VIP-016.

For citation: E. V. Kuzmin, "LTL-Specification of Counter Machines", *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 104-119, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 1, 2021
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

LTL-спецификация счётчиковых машин

Е. В. Кузьмин[1]

DOI: 10.18255/1818-1015-2021-1-104-119

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, д. 14, г. Ярославль, 150003 Россия.

УДК 519.7
Научная статья
Полный текст на русском языке

Получена 11 января 2021 г.
После доработки 12 февраля 2021 г.
Принята к публикации 12 марта 2021 г.

Статья написана в поддержку учебной дисциплины “Неклассические логики”. В рамках этой дисциплины объектами изучения являются базовые принципы и конструктивные элементы, с помощью которых происходит формальное построение различных неклассических логик высказываний. Несмотря на абстрактность теории неклассических логик, в которой основное внимание уделяется строгой математической формализации логических рассуждений, существуют реальные прикладные области применения теоретических результатов. В частности, языки темпоральных модальных логик широко используются для моделирования, спецификации и верификации (анализа корректности) программных систем логического управления. В этой статье на примере линейной темпоральной логики LTL демонстрируется, как абстрактные понятия неклассических логик могут находить отражение на практике в области информационных технологий и программирования. Показывается возможность представления поведения программной системы в виде набора LTL-формул и использования этого представления для проверки выполнимости программных свойств системы через процедуру доказательства справедливости логических выводов, выраженных в терминах линейной темпоральной логики LTL. В качестве программных систем, для спецификации поведения которых будет применяться логика LTL, рассматриваются счётчиковые машины Минского. Счётчиковые машины Минского — один из способов формализации интуитивного понятия алгоритма. Они обладают той же вычислительной мощностью, что и машины Тьюринга. Счётчиковая машина имеет вид компьютерной программы, написанной на языке высокого уровня, поскольку содержит переменные, называемые счётчиками, и операторы условного и безусловного перехода, позволяющие строить конструкции циклов. Известно, что любой алгоритм (гипотетически) может быть реализован в виде трёхсчётчиковой машины Минского.

Ключевые слова: неклассическая логика, линейная темпоральная логика, счётчиковые машины, LTL-спецификация

ИНФОРМАЦИЯ ОБ АВТОРАХ

Егор Владимирович Кузьмин
автор для корреспонденции

orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru
профессор, доктор физ.-мат. наук.

Финансирование: Работа выполнена в рамках инициативной НИР ЯрГУ им. П. Г. Демидова № VIP-016.

Для цитирования: E. V. Kuzmin, “LTL-Specification of Counter Machines”, *Modeling and analysis of information systems*, vol. 28, no. 1, pp. 104-119, 2021.

Введение

Статья написана в поддержку учебной дисциплины «Неклассические логики» [1, 2]. В рамках этой дисциплины объектами изучения являются базовые принципы и конструктивные элементы, с помощью которых происходит формальное построение различных неклассических логик высказываний. В качестве основных конструктивных элементов рассматриваются возможные миры (как ключевой инструмент выхода за пределы классической логики высказываний), отношение достижимости между мирами и модальные операторы. Логики определяются относительно семантического понятия *логического следования*, отвечающего на вопрос, каким образом из набора предпосылок Σ выводится заключение A.

Несмотря на абстрактность теории неклассических логик, в которой основное внимание уделяется строгой математической формализации логических рассуждений, существуют реальные прикладные области применения теоретических результатов. Например, языки темпоральных модальных логик широко используются для моделирования, спецификации и верификации (анализа корректности) программных систем логического управления.

Цель статьи — наглядно на примере линейной темпоральной логики LTL продемонстрировать, как абстрактные понятия неклассических логик могут находить отражение на практике в области информационных технологий и программирования. В частности, в данной работе будет показана возможность представления поведения программной системы в виде набора LTL-формул и использования этого представления для проверки выполнимости программных свойств системы через процедуру доказательства справедливости логических выводов, выраженных в терминах линейной темпоральной логики LTL.

В статье в качестве программных систем, для спецификации поведения которых будет применяться логика LTL, рассматриваются счётчиковые машины Минского.

Счётчиковые машины Минского [3—5] — это один из способов формализации интуитивного понятия алгоритма. Они обладают той же вычислительной мощностью, что и машины Тьюринга. Однако счётчиковая машина имеет более наглядный программный вид, т. е. вид компьютерной программы, написанной на языке высокого уровня, поскольку содержит переменные, называемые счётчиками, и операторы условного и безусловного перехода, позволяющие строить конструкции циклов. Известно, что произвольная машина Тьюринга моделируется машиной Минского, которая имеет всего лишь три счётчика. Это означает, что для реализации любого алгоритма (гипотетически) достаточно формализма трёхсчётчиковых машин Минского. Более того, для каждой машины Минского может быть построена моделирующая ее работу двухсчётчиковая машина при использовании специальной кодировки входа и выхода [3].

1. Предварительные сведения

1.1. Счётчиковые машины Минского

Счётчиковая машина Минского M представляет собой набор (q_0, q_n, Q, V, Δ), где $Q = \{q_0, \dots, q_n\}$ — конечное непустое множество состояний машины; $q_0 \in Q$ — начальное состояние; $q_n \in Q$ — финальное состояние; $V = \{x_1, \dots, x_m\}$ — конечное непустое множество счётчиков, которые могут принимать значения из $\mathbb{N} \cup \{0\}$; $\Delta = \{\delta_0, \dots, \delta_{n-1}\}$ — набор правил переходов по состояниям машины; δ_i — правило переходов для состояния q_i. Состояния q_i, $0 \leqslant i \leqslant n-1$, подразделяются на два типа. Состояния первого типа имеют правила переходов вида:

$$(\delta_i)\ \ q_i:\ \ x_j := x_j + 1;\ \text{goto}\ q_k,$$

где $1 \leqslant j \leqslant m, 0 \leqslant k \leqslant n$. Для состояний второго типа имеем, $1 \leqslant j \leqslant m, 0 \leqslant k, l \leqslant n$:

$$(\delta_i)\ \ q_i:\ \ \text{if}\ x_j > 0\ \text{then}\ (x_j := x_j - 1;\ \text{goto}\ q_k)\ \text{else goto}\ q_l.$$

Для финального состояния q_n правило перехода не предусмотрено. Это означает, что при попадании в состояние q_n машина Минского M завершает свою работу.

Конфигурация счётчиковой машины Минского — это набор $(q_i, c_1, \dots, c_m)$, где q_i — состояние машины, $c_1, \dots, c_m$ — натуральные числа (включая ноль), являющиеся значениями соответствующих счётчиков.

Рассмотрим множество состояний $Q = \{q_0, \dots, q_n\}$ как набор булевых переменных. Будем считать, что некоторая переменная $q_i \in Q$ принимает значение 1, если счётчиковая машина находится в состоянии q_i $(0 \leqslant i \leqslant n)$. В других случаях q_i будет иметь значение 0. Тогда конфигурацию машины Минского можно представить в виде вектора значений соответствующего набора переменных

$$(q_0, \dots, q_n, x_1, \dots, x_m).$$

Исполнением машины Минского называется последовательность конфигураций $s_0\ s_1\ s_2\ s_3\ s_4\ \dots$, индуктивно определяемая в соответствии с правилами переходов. Счётчиковая машина имеет одно исполнение из начальной конфигурации s_0, так как для каждого состояния предусмотрено не более одного правила переходов. Машина, получив на вход некоторый набор значений счётчиков, стартует из состояния q_0 и либо останавливается в состоянии q_n с выходным набором значений счётчиков, либо зацикливается, реализуя тем самым частичную числовую функцию.

Добавим к описанию счётчиковой машины отображение $\mathrm{bnd}\colon V \longrightarrow \mathbb{N}$, устанавливающее предельное значение $\mathrm{bnd}(x)$ для каждого счётчика $x \in V$. Добавив ограничивающее условие, изменим правило переходов первого типа следующим образом, $1 \leqslant j \leqslant m, 0 \leqslant k \leqslant n$:

$$(\delta_i)\ \ q_i:\ \ \text{if } x_j < \mathrm{bnd}(x_j) \text{ then } x_j := x_j + 1;\ \text{goto } q_k.$$

В результате получили счётчиковую машину с ограничениями. Если при срабатывании правила переходов первого типа оказывается, что значение соответствующего счётчика достигло предельного значения, работа машины останавливается. Далее в статье под счётчиковой машиной будет пониматься счётчиковая машина с ограничениями.

1.2. Счётчиковая машина возведения числа в квадрат

Рассмотрим в качестве примера трехсчётчиковую машину Минского $3cM$ с ограничениями, которая реализует функцию возведения числа n в квадрат, где $0 \leqslant n \leqslant 10$. Эта счётчиковая машина имеет восемь состояний $q_0, q_1, \dots, q_7$, где q_0 является начальным состоянием, q_7 — финальное состояние. Множество счётчиков $V = \{a, b, c\}$. В начальной конфигурации счётчик a получает значение n, а начальные значения двух других счётчиков b и c равны нулю. В финальной конфигурации результат вычисления будет содержаться в счётчике c при нулевых значениях остальных счётчиков a и b. На значения счётчиков накладываются ограничения $\mathrm{bnd}(a) = 11$, $\mathrm{bnd}(b) = 11$ и $\mathrm{bnd}(c) = 101$. Правила переходов по состояниям машины $3cM$ представлены ниже [4]:

$(\delta_0)\ \ q_0:\ \ \text{if } a > 0 \text{ then } (a := a - 1;\ \text{goto } q_1) \text{ else goto } q_7;$
$(\delta_1)\ \ q_1:\ \ \text{if } c < \mathrm{bnd}(c) \text{ then } c := c + 1;\ \text{goto } q_2;$
$(\delta_2)\ \ q_2:\ \ \text{if } a > 0 \text{ then } (a := a - 1;\ \text{goto } q_3) \text{ else goto } q_5;$
$(\delta_3)\ \ q_3:\ \ \text{if } b < \mathrm{bnd}(b) \text{ then } b := b + 1;\ \text{goto } q_4;$
$(\delta_4)\ \ q_4:\ \ \text{if } c < \mathrm{bnd}(c) \text{ then } c := c + 1;\ \text{goto } q_1;$
$(\delta_5)\ \ q_5:\ \ \text{if } b > 0 \text{ then } (b := b - 1;\ \text{goto } q_6) \text{ else goto } q_0;$
$(\delta_6)\ \ q_6:\ \ \text{if } a < \mathrm{bnd}(a) \text{ then } a := a + 1;\ \text{goto } q_5.$

При построении этой счётчиковой машины использовался тот факт, что $n^2 = (2n-1)+(2n-3)+\dots+3+1$.

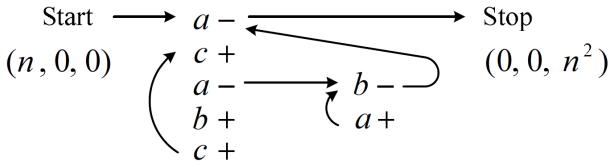

Fig. 1. Graphical representation of the counter machine $3cM$ of squaring a number

Рис. 1. Графическое представление счётчиковой машины $3cM$ возведения числа в квадрат

Правила переходов трехсчётчиковой машины $3cM$ наглядно представлены в графическом виде на рис. 1, где для некоторой переменной $v \in V$ обозначение «$v+$» соответствует увеличению счётчика на единицу, а «$v-$» используется для обозначения условного вычитания единицы с переходом в другое состояние по правосторонней стрелке в случае нулевого значения счётчика v.

2. Линейная темпоральная логика

Линейная темпоральная логика LTL (linear temporal logic), или темпоральная логика линейного времени (linear-time temporal logic), представляет собой расширение классической логики высказываний с помощью модальных операторов, позволяющих учитывать временной аспект в последовательностях событий и объектов. Темпоральная логика LTL нашла важное применение в области формальной верификации, где она используется для описания требований к аппаратным и программным системам [6]. В данной статье формализм логики LTL будет задействован для описания поведения счётчиковых машин, а также спецификации их свойств, подлежащих дальнейшему анализу на выполнимость методом проверки модели (model checking) [7, 8].

Прежде чем рассматривать LTL в качестве программной логики, дадим её определение как некоторой абстрактной модальной темпоральной логики, являющейся одним из представителей внушительного ряда неклассических логик [2].

Символами языка логики LTL являются пропозициональные переменные p, p_0, p_1, p_2, ... (элементарные высказывания), константы true и false, логические связки $\neg$ (отрицание), $\wedge$ (конъюнкция), $\vee$ (дизъюнкция), $\Longrightarrow$ (импликация), темпоральные модальные операторы **X** (neXt, непосредственное следование), **U** (Until, условное ожидание), **F** (Future, неизбежность), **G** (Globally, инвариантность) и знаки пунктуации «(» и «)».

Формулами языка логики LTL являются те и только те строки символов, которые могут быть рекурсивно построены из пропозициональных переменных и констант по следующему правилу:

если φ и ψ — формулы, то $\neg\varphi$, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(\varphi \Longrightarrow \psi)$, $\mathbf{X}\,\varphi$, $(\varphi\,\mathbf{U}\,\psi)$, $\mathbf{F}\,\varphi$ и $\mathbf{G}\,\varphi$ также являются формулами.

Значение (истинность или ложность) формул темпоральной логики LTL определяется через понятие интерпретации.

Интерпретация формул логики LTL представляет собой набор (w_0, W, T, v), где W обозначает некоторое непустое множество объектов, интуитивно понимаемое как множество *возможных миров*, $w_0 \in W$ — это начальный мир, а $T : W \longrightarrow W$ — функция переходов между мирами.

Если возможные миры w и w' принадлежат W и связаны функцией переходов $T(w) = w'$, то для простоты будем писать $w \longrightarrow w'$. Запись $w \longrightarrow w'$ читается как «мир w' непосредственно достижим из мира w» и означает, что существует прямой переход из мира w в мир w'.

Обозначим $w \overset{*}{\longrightarrow} w'$ транзитивную достижимость мира w' из мира w за ноль или более шагов (применений функции T). Другими словами $w \overset{*}{\longrightarrow} w'$ означает, что $T^i(w) = w'$, где $T^i(w) = T(T^{i-1}(w))$

и $T^0(w) = w$ при $i \in \mathbb{N} \cup \{0\}$. Запись $w \xrightarrow{i} w'$ будет обозначать достижимость w' из w ровно за i шагов. В случае $i = 0$ считается, что мир w транзитивно достигает сам себя за ноль шагов, т. е. $w' = w$.

Функция ν сопоставляет истинное (возвращается результат 1) или ложное (результат 0) значение каждой паре вида (p, w), где p — пропозициональная переменная, а $w \in W$. Запишем это как $\nu_w(p) = 1$ и $\nu_w(p) = 0$ соответственно.

Интуитивно, $\nu_w(p) = 1$ означает, что в мире w высказывание p истинно, а $\nu_w(p) = 0$ — в мире w высказывание p ложно.

Функция ν_w для каждого мира $w \in W$ расширяется на все множество формул по рекурсивным правилам. Рекурсивные правила расширения функции ν_w для логических связок $\neg$, $\wedge$, $\vee$, $\Longrightarrow$ следующие. Для каждого мира $w \in W$ имеем:

$\nu_w(\neg\varphi) = 1$, если $\nu_w(\varphi) = 0$, иначе $\nu_w(\neg\varphi) = 0$;

$\nu_w(\varphi \wedge \psi) = 1$, если $\nu_w(\varphi) = \nu_w(\psi) = 1$, иначе $\nu_w(\varphi \wedge \psi) = 0$;

$\nu_w(\varphi \vee \psi) = 1$, если $\nu_w(\varphi) = 1$ или $\nu_w(\psi) = 1$, иначе $\nu_w(\varphi \vee \psi) = 0$;

$\nu_w(\varphi \Longrightarrow \psi) = 1$, если $\nu_w(\varphi) = 0$ или $\nu_w(\psi) = 1$, иначе $\nu_w(\varphi \Longrightarrow \psi) = 0$.

Из этих правил видно, что классические логические связки действуют в рамках одного текущего мира w.

Для темпоральных модальных операторов логики LTL рекурсивные правила расширения функции ν_w выглядят следующим образом. Для каждого мира $w \in W$ имеем:

$\nu_w(\mathbf{X}\,\varphi) = 1$, если $\exists\, w' \in W$ такой, что $w \longrightarrow w'$ и $\nu_{w'}(\varphi) = 1$, иначе $\nu_w(\mathbf{X}\,\varphi) = 0$,

т. е. формула φ должна выполняться в следующем достижимом мире;

$\nu_w(\mathbf{F}\,\varphi) = 1$, если $\exists\, w' \in W$ такой, что $w \xrightarrow{*} w'$ и $\nu_{w'}(\varphi) = 1$, иначе $\nu_w(\mathbf{F}\,\varphi) = 0$,

т. е. формула φ должна выполняться в некотором мире w', который транзитивно достижим из текущего мира w;

$\nu_w(\mathbf{G}\,\varphi) = 1$, если $\forall\, w' \in W$ таких, что $w \xrightarrow{*} w'$, имеем $\nu_{w'}(\varphi) = 1$, иначе $\nu_w(\mathbf{G}\,\varphi) = 0$,

т. е. формула φ должна выполняться в текущем мире w и во всех мирах w', которые транзитивно достижимы из мира w;

$\nu_w(\varphi\,\mathbf{U}\,\psi) = 1$, если $\exists\, w' \in W$ такой, что $w \xrightarrow{i} w'$ и $\nu_{w'}(\psi) = 1$, а $\forall\, w'' \in W$ таких, что $w \xrightarrow{j} w''$, имеем $\nu_{w''}(\varphi) = 1$, где $j < i$ и $i \in \mathbb{N} \cup \{0\}$, иначе $\nu_w(\varphi\,\mathbf{U}\,\psi) = 0$,

т. е. формула ψ должна выполняться в некотором мире w', который транзитивно достижим из текущего мира w, но при этом во всех мирах, которые предшествуют миру w' на пути из w, должна выполняться формула φ.

В логике LTL операторы **F** и **G** являются двойственными $\mathbf{G}\,\varphi = \neg\mathbf{F}\neg\varphi$. При этом сам оператор **F** представляет собой специальный случай применения оператора **U**, а именно $\mathbf{F}\varphi = \text{true}\,\mathbf{U}\,\varphi$.

Семантика темпоральных модальных операторов схематично поясняется на рис. 2.

Отметим, что каждую интерпретацию в логике LTL можно представить в виде бесконечной последовательности миров (с начальным миром w_0), связанных между собой функцией переходов, со своей оценочной функцией ν.

Для обозначения того, что из набора LTL-формул φ_1, φ_2, …, φ_m (предпосылок), где $m \in \mathbb{N}$, логически выводится формула ψ (заключение), используется металингвистический символ «$\vDash$». Вывод в логике LTL является справедливым, если он сохраняет свою истинность в начальных мирах всех интерпретаций. Пусть Σ — произвольное множество формул (набор предпосылок). Тогда заключение ψ будет *логически следовать* из набора формул Σ, т. е. $\Sigma \vDash \psi$, тогда и только тогда, когда не существует такой интерпретации, при которой все формулы из Σ в начальном мире являются

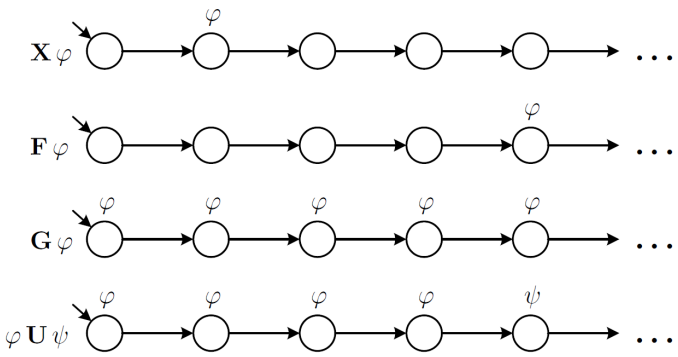

Fig. 2. Semantics of the temporal modal operators **X**, **F**, **G**, and **U**

Рис. 2. Семантика темпоральных модальных операторов **X**, **F**, **G** и **U**

истинными, а формула ψ в нём ложна. Другими словами, каждая интерпретация, обращающая в начальном мире все формулы из Σ в истину, в этом же мире обращает в истину и формулу ψ.

Более формально, логический вывод $\Sigma \models \psi$ справедлив тогда и только тогда, когда при всех возможных интерпретациях (w_0, W, T, ν) для мира $w_0 \in W$ выполняется условие, что если $\nu_{w_0}(\varphi) = 1$ для всех предпосылок $\varphi \in \Sigma$, то $\nu_{w_0}(\psi) = 1$.

Запись $\Sigma \not\models \psi$ означает, что условие $\Sigma \models \psi$ не выполняется. В этом случае будет существовать хотя бы одна интерпретация (w_0, W, T, ν), в рамках которой в начальном мире w_0 все формулы из Σ выполняются, а формула ψ в мире w_0 является ложной. Такая интерпретация называется контрпримером или контрмоделью.

3. LTL как программная логика

Пусть Cl(3cM) — это класс счётчиковых машин с восемью булевыми переменными-состояниями q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, тремя переменными-счётчиками a, b, c и соответствующими ограничениями bnd(a), bnd(b) и bnd(c). Тогда применительно к этому классу трёхсчётчиковых машин Cl(3cM) линейная темпоральная логика LTL может быть рассмотрена следующим образом.

Сопоставим возможному миру $w \in W$ логики LTL конфигурацию некоторой счётчиковой машины из класса Cl(3cM) на некотором шаге её *исполнения*. Тогда каждый возможный мир будет соответствовать вектору значений переменных $(q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, a, b, c)$. При этом разные миры могут иметь одни и те же наборы значений указанных переменных.

Непосредственную достижимость $w \longrightarrow w'$ возможного мира w' из мира w будем понимать как осуществление перехода из одной конфигурации счётчиковой машины в другую после срабатывания одного из правил переходов. Если для текущей конфигурации ни одно из правил переходов не выполняется, то предполагается, что происходит «пустой» переход в новый мир w', который будет соответствовать прежней конфигурации счётчиковой машины.

Поскольку интерпретация в логике LTL представляет собой бесконечную последовательность миров, связанных между собой функцией переходов, каждый мир (кроме первого по порядку) в этой последовательности в конкретный момент времени имеет своего непосредственного предшественника. Расширим множество переменных, значения которых будут отслеживаться в возможных мирах. Пусть в интерпретации в некоторый момент времени мир w является непосредственным предшественником мира w'. Для каждой переменной $v \in \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, a, b, c\}$ введём ещё одну переменную $\_v$ для хранения в мире w' предыдущего значения v, т. е. того значения, которое переменная v имела в мире w. Тогда каждый мир может быть представлен в виде вектора значений переменных $(q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, a, b, c, \_q_0, \_q_1, \_q_2, \_q_3, \_q_4, \_q_5, \_q_6, \_q_7, \_a, \_b, \_c)$, который будет

уникальным, т. е. два мира с одним и тем же набором значений соответствующих переменных считаются одним и тем же миром.

Для определённости положим, что в начальном мире любой интерпретации все переменные вида $\_v$ имеют значение 0.

Отметим, что при таком представлении возможных миров в виде векторов значений указанных переменных для класса Cl(3cM) трёхсчётчиковых машин с ограничениями множество миров W в каждой интерпретации (w_0, W, T, ν) логики LTL будет конечным. Этот факт обеспечивает возможность проверки справедливости всех логических выводов в рамках логики LTL, рассмотренной применительно к классу счётчиковых машин Cl(3cM).

В качестве элементарного высказывания логики LTL будем рассматривать выражение над переменными, построенное с использованием операторов сравнения, арифметических операторов и констант (целых неотрицательных чисел). Элементарное высказывание p формулируется таким образом, чтобы в результате оно могло принимать только два логических значения 1 (истина) или 0 (ложь). Результат оценочной функции $\nu_w(p)$ для элементарного высказывания p и мира w будет зависеть от значений задействованных в p переменных, которые они имеют в мире w.

Примером простого элементарного высказывания является выражение в виде одной булевой переменной-состояния.

Теперь для любой счётчиковой машины M из класса Cl(3cM) может быть построен такой набор LTL-формул, который будет описывать в точности все возможные исполнения счётчиковой машины M и только их. Всякий раз когда все LTL-формулы из набора будут одновременно обращаться в 1 (иметь значение «истина») при некоторой интерпретации (w_0, W, T, ν), это будет означать, что интерпретация (w_0, W, T, ν) соответствует одному из возможных исполнений машины M, т. е. повторяет последовательность конфигураций в исполнении. И наоборот, для любого исполнения счётчиковой машины M будет существовать повторяющая его интерпретация, которая будет приводить к одновременному выполнению всех LTL-формул из рассматриваемого набора.

Пусть набор LTL-формул $\Sigma = \{\varphi_1, \ldots, \varphi_m\}$, где $m \in \mathbb{N}$, описывает все возможные исполнения некоторой счётчиковой машины M из класса Cl(3cM), у которой q_0 — это начальное состояние, а состояние q_7 является финальным. Рассмотрим LTL-формулу $\psi = «q_0 \Longrightarrow \mathbf{F}\,\mathbf{G}\,q_7»$. Эта формула является истинной только для тех интерпретаций, которые в начальном мире имеют $q_0 = 0$ или же если в начальном мире выполняется $q_0 = 1$, то в последовательности миров этих интерпретаций рано или поздно появится мир, начиная с которого для всех следующих миров будет выполняться $q_7 = 1$. Тогда справедливость логического вывода $\varphi_1, \ldots, \varphi_m \vDash \psi$ будет означать, что счётчиковая машина M, стартуя из состояния q_0 при любых начальных значениях счётчиков, обязательно завершит свою работу в финальном состоянии q_7, т. е. машина M никогда не зацикливается, но при этом всегда срабатывает некоторое правило переходов до тех пор, пока машина не окажется в финальном состоянии. Если же имеет место $\varphi_1, \ldots, \varphi_m \nvDash \psi$, то существует интерпретация (контрпример), соответствующая такому исполнению счётчиковой машины, которое для некоторых входных данных не приводит к требуемому результату вычислений, поскольку не попадает в финальное состояние. При этом набор формул $\varphi_1, \ldots, \varphi_m, \neg\psi$ будет описывать все интерпретации, являющиеся контрпримерами, т. е. те интерпретации, для которых все формулы из этого набора одновременно являются истинными.

Получили, что задача проверки выполнимости для счётчиковой машины некоторого свойства, заданного LTL-формулой, может быть сведена к задаче доказательства справедливости логического вывода в рамках линейной темпоральной логики LTL.

В следующих разделах будет показано, каким образом поведение счётчиковых машин может быть представлено в виде набора LTL-формулы, а также будут рассмотрены примеры LTL-свойств счётчиковых машин.

4. LTL-спецификация счётчиковой машины с ограничениями

Основная идея представления поведения счётчиковой машины в виде набора LTL-формул состоит в описании (на языке логики LTL) того, каким образом происходит изменение значения каждой переменной счётчиковой машины на каждом шаге её исполнения.

Для любой переменной-состояния или переменной-счётчика изменение её значения задаётся с помощью трёх LTL-формул. Первая LTL-формула описывает ситуации, при которых происходит возрастание значения соответствующей переменной, вторая LTL-формула определяет условия, приводящие к уменьшению значения переменной. Третья LTL-формула имеет жёсткую конструкцию, составленную из элементов первых двух формул. Она описывает условия, при которых переменная не меняет своего значения во время работы счётчиковой машины.

Для описания ситуаций, приводящих к увеличению и уменьшению значения целочисленной переменной-счётчика v используются LTL-формулы вида

$$\mathbf{G}\,\mathbf{X}(\ v > \_v \Longrightarrow \mathit{FiringCond}\ \wedge \_v < \mathrm{bnd}(v) \wedge v = \_v + 1\ );$$

$$\mathbf{G}\,\mathbf{X}(\ v < \_v \Longrightarrow \mathit{FiringCond}' \wedge \_v > 0 \qquad \wedge v = \_v - 1\ ).$$

Символ лидирующего подчеркивания «_» в обозначении переменной $\_v$ можно воспринимать как псевдооператор, позволяющий обратиться к значению переменной v, которое она имела в предыдущей конфигурации счётчиковой машины. При этом псевдооператор может использоваться только под действием темпорального оператора **X**.

Выражения $\mathit{FiringCond}$ и $\mathit{FiringCond}'$ описывают состояния счётчиковой машины, при которых возникает необходимость изменения значения переменной v, если это, конечно, допускается соответствующим условием $\_v < \mathrm{bnd}(v)$ или $\_v > 0$.

Для спецификации поведения булевой переменной-состояния v будут использоваться LTL-формулы другого вида

$$\mathbf{G}\,\mathbf{X}(\ \neg\_v \wedge\ \ v \Longrightarrow \mathit{FiringCond}\ );$$

$$\mathbf{G}\,\mathbf{X}(\ \ \_v \wedge \neg v \Longrightarrow \mathit{FiringCond}'\ ).$$

Первая формула означает, что всякий раз, когда состояние v счётчиковой машины активируется, из этого следует, что было выполнено условие $\mathit{FiringCond}$ перехода в состояние v, т. е. сработало некоторое правило переходов. Выражение $\mathit{FiringCond}'$ во второй формуле описывает условия выхода из состояния v.

Третья LTL-формула, описывающая сохранение прежнего значения для счётчика v, имеет вид

$$\mathbf{G}\,\mathbf{X}(\ v = \_v\ \Longrightarrow\ \neg(\mathit{FiringCond} \wedge \_v < \mathrm{bnd}(v)) \wedge \neg(\mathit{FiringCond}' \wedge \_v > 0)\ ).$$

Для булевой переменной-состояния v аналогичная LTL-формула выглядит как

$$\mathbf{G}\,\mathbf{X}(\ v = \_v\ \Longrightarrow\ \neg(\neg\_v \wedge \mathit{FiringCond}) \wedge \neg(\_v \wedge \mathit{FiringCond}')\ ).$$

Итак, опираясь на формальное определение счётчиковой машины с ограничениями, запишем в общем виде требуемое поведение каждой её переменной с помощью тройки LTL-формул.

Начнём с описания поведения переменных-счётчиков. LTL-формула, учитывающая ситуации, которые приводят к увеличению значения переменной-счётчика `xj`, имеет следующий вид:

$$\mathbf{G}\,\mathbf{X}(\ \texttt{xj} > \texttt{\_xj}\ \Longrightarrow\ (\texttt{\_qi} \vee \texttt{\_qr} \vee \cdots \vee \texttt{\_qs}) \wedge \texttt{\_xj} < \texttt{bnd(xj)} \wedge \texttt{xj} = \texttt{\_xj+1}\ ),$$

где переменные `_qi`, `_qr`, ... , `_qs` соответствуют таким состояниям первого типа $q_i, q_r, \ldots, q_s$, в правилах переходов которых участвует счётчик x_j.

Например, для состояния первого типа q_i в описании счётчиковой машины должно быть правило переходов (δ_i) q_i : if $x_j < \mathrm{bnd}(x_j)$ then $x_j := x_j + 1$; goto q_k, которое порождает условие LTL-спецификации вида `_qi` ∧ `_xj<bnd(xj)` ∧ `xj=_xj+1`.

Если в описании машины для счётчика x_j правил перехода первого типа нет, то LTL-формула «возрастания» для программной переменной xj следующая:

$$\mathbf{G}\,\mathbf{X}(\ \texttt{xj} > \texttt{\_xj} \implies \text{false}\ ).$$

LTL-формула, учитывающая ситуации, которые приводят к уменьшению значения переменной-счётчика xj, имеет вид

$$\mathbf{G}\,\mathbf{X}(\ \texttt{xj} < \texttt{\_xj} \implies (\texttt{\_qk} \vee \texttt{\_ql} \vee \cdots \vee \texttt{\_qt}) \wedge \texttt{\_xj}{>}0 \wedge \texttt{xj}{=}\texttt{\_xj}{-}1\ ),$$

где переменные _qk, _ql, ..., _qt соответствуют состояниям второго типа q_k, q_l, ..., q_t, в правилах переходов которых участвует счётчик x_j, т. е. для состояния второго типа q_k в описании машины должно быть правило переходов (δ_k) q_k : if $x_j > 0$ then ($x_j := x_j - 1$; goto q_p) else goto q_h, которое порождает условие LTL-спецификации вида _qk ∧ _xj>0 ∧ xj=_xj−1.

Если в описании машины для счётчика x_j правил перехода второго типа нет, то LTL-формула «убывания» для переменной xj строится просто как

$$\mathbf{G}\,\mathbf{X}(\ \texttt{xj} < \texttt{\_xj} \implies \text{false}\ ).$$

Ситуации, не приводящие к изменению значения переменной-счётчика xj, описываются следующей LTL-формулой, которая строится автоматически по уже рассмотренным двум формулам «возрастания» и «убывания»:

$$\mathbf{G}\,\mathbf{X}(\ \texttt{xj} = \texttt{\_xj} \implies \neg((\texttt{\_qi} \vee \texttt{\_qr} \vee \cdots \vee \texttt{\_qs}) \wedge \texttt{\_xj}{<}\texttt{bnd(xj)}) \wedge \neg((\texttt{\_qk} \vee \texttt{\_ql} \vee \cdots \vee \texttt{\_qt}) \wedge \texttt{\_xj}{>}0)\ ),$$

Эта формула справедлива только в том случае, когда счётчик x_j участвует в правилах переходов обоих типов. Если же для счётчика x_j имеются правила переходов только одного типа, то в этой формуле после оператора импликации будет присутствовать только один соответствующий конъюнктивный член с отрицанием.

Теперь опишем построение LTL-спецификации поведения бинарных переменных-состояний. LTL-формула, учитывающая условия, при которых счётчиковая машина переходит из некоторого текущего состояния в новое состояние q_k, т. е. логическая переменная qk получает значение 1, имеет следующий вид:

$$\mathbf{G}\,\mathbf{X}(\ \neg\texttt{\_qk} \wedge \texttt{qk} \implies \texttt{\_qi} \wedge \texttt{\_xj}{>}0 \vee \cdots \vee \texttt{\_qr} \wedge \neg(\texttt{\_xt}{>}0) \vee \cdots \vee \texttt{\_qs} \wedge \texttt{\_xl}{<}\texttt{bnd(xl)}\ ),$$

где условия вида _qi ∧ _xj>0 соответствуют правилам переходов второго типа

(δ_i) q_i : if $x_j > 0$ then ($x_j := x_j - 1$; goto $\mathbf{q_k}$) else goto q_h,

а условия вида _qr ∧ ¬(_xt>0) — правилам

(δ_r) q_r : if $x_t > 0$ then ($x_t := x_t - 1$; goto q_p) else goto $\mathbf{q_k}$.

Условия вида _qs ∧ _xl<bnd(xl) соответствуют правилам переходов первого типа

(δ_s) q_s : if $x_l < \mathrm{bnd}(x_l)$ then $x_l := x_l + 1$; goto $\mathbf{q_k}$.

Важно отметить, что все переменные приведённой LTL-формулы, стоящие в условиях после оператора импликации, должны быть отличными от переменной-состояния qk, т. е. переход в то же самое состояние здесь не специфицируется.

Если в состояние q_k счётчиковой машины нет ни одного перехода, то для переменной qk LTL-формула «активации» строится как

$$\mathbf{G}\,\mathbf{X}(\ \neg\texttt{\_qk} \wedge \texttt{qk} \implies \text{false}\ ).$$

LTL-формула «деактивации» (выхода из) состояния q_k, которому соответствует правило переходов первого типа со счётчиком x_j, для переменной qk имеет вид:

$\mathbf{G}\,\mathbf{X}(\ \_\mathtt{qk} \wedge \neg\mathtt{qk} \implies \_\mathtt{xj}<\mathtt{bnd(xj)}\ )$.

Для правила переходов второго типа без петель получаем LTL-формулу

$\mathbf{G}\,\mathbf{X}(\ \_\mathtt{qk} \wedge \neg\mathtt{qk} \implies \mathrm{true}\ )$,

где логическая константа true означает, что переменная qk должна быть обязательно сброшена в ноль уже на следующем шаге после её активации, т. е. после присваивания значения 1.

Если правило перехода для q_k имеет вид петли (δ_k) $\mathbf{q_k}$: if $x_j < \mathrm{bnd}(x_j)$ then $x_j := x_j + 1$; goto $\mathbf{q_k}$, состояние q_k соответствует двум петлям (δ_k) $\mathbf{q_k}$: if $x_l > 0$ then ($x_j := x_j - 1$; goto $\mathbf{q_k}$) else goto $\mathbf{q_k}$ или же q_k является финальным состоянием, то переменная qk будет иметь следующую LTL-формулу:

$\mathbf{G}\,\mathbf{X}(\ \_\mathtt{qk} \wedge \neg\mathtt{qk} \implies \mathrm{false}\ )$.

Если состояние второго типа q_k имеет в правиле перехода со счётчиком x_j только одну петлю, то для переменной qk выбирается соответствующий вариант LTL-формулы

$\mathbf{G}\,\mathbf{X}(\ \_\mathtt{qk} \wedge \neg\mathtt{qk} \implies \_\mathtt{xj}>0\ )$ или $\mathbf{G}\,\mathbf{X}(\ \_\mathtt{qk} \wedge \neg\mathtt{qk} \implies \neg(\_\mathtt{xj}>0)\ )$.

LTL-формула, описывающая ситуации, при которых переменная-состояние q_k сохраняет своё значение после применения некоторого правила переходов, так же строится автоматически по уже рассмотренным формулам «активации» и «деактивации»:

$\mathbf{G}\,\mathbf{X}(\ \_\mathtt{qk} = \mathtt{qk} \implies \neg(\neg\_\mathtt{qk} \wedge \mathit{FiringCondA}) \wedge \neg(\_\mathtt{qk} \wedge \mathit{FiringCondD})\ )$,

где *FiringCondA* — условие, которое стоит после оператора импликации в LTL-формуле «активации» состояния q_k, а *FiringCondD* — соответствующее условие из формулы деактивации этого состояния.

Далее будет приведена LTL-спецификация поведения конкретной счётиковой машины, а также рассмотрены примеры LTL-свойств, требующих проверки их выполнимости для этой счётчиковой машины.

5. LTL-спецификация трёхсчётчиковой машины возведения числа в квадрат

Построим LTL-спецификацию трехсчётчиковой машины $3cM$ (с ограничениями) возведения числа n в квадрат. В этой LTL-спецификации переменная-счётчик a при инициализации получает значение n, $0 \leqslant n \leqslant 10$. Переменная-состояние q0 инициализируется единицей, т. е. изначально имеем q0 = 1. Остальные переменные при инициализации обнуляются. В LTL-формулах в качестве предельных значений переменных-счётчиков будем использовать конкретные числа: bnd(a) = 11, bnd(b) = 11 и bnd(c) = 101.

$\mathtt{Sinit}$: $\mathtt{a}=\mathtt{n} \wedge \mathtt{b}=0 \wedge \mathtt{c}=0 \wedge \mathtt{q0}=1 \wedge \mathtt{q1}=0 \wedge \mathtt{q2}=0 \wedge \mathtt{q3}=0 \wedge \mathtt{q4}=0 \wedge \mathtt{q5}=0 \wedge \mathtt{q6}=0 \wedge \mathtt{q7}=0$;

$\mathtt{Sa}$: $\mathbf{G}\,\mathbf{X}(\ \mathtt{a}>\_\mathtt{a} \implies \_\mathtt{q6} \wedge \_\mathtt{a}<11 \wedge \mathtt{a}=\_\mathtt{a}+1\ ) \wedge$
$\mathbf{G}\,\mathbf{X}(\ \mathtt{a}<\_\mathtt{a} \implies (\_\mathtt{q0} \vee \_\mathtt{q2}) \wedge \_\mathtt{a}>0 \wedge \mathtt{a}=\_\mathtt{a}-1\ ) \wedge$
$\mathbf{G}\,\mathbf{X}(\ \mathtt{a}=\_\mathtt{a} \implies \neg(\_\mathtt{q6} \wedge \_\mathtt{a}<11) \wedge \neg((\_\mathtt{q0} \vee \_\mathtt{q2}) \wedge \_\mathtt{a}>0)\ )$;

$\mathtt{Sb}$: $\mathbf{G}\,\mathbf{X}(\ \mathtt{b}>\_\mathtt{b} \implies \_\mathtt{q3} \wedge \_\mathtt{b}<11 \wedge \mathtt{b}=\_\mathtt{b}+1\ ) \wedge$
$\mathbf{G}\,\mathbf{X}(\ \mathtt{b}<\_\mathtt{b} \implies \_\mathtt{q5} \wedge \_\mathtt{b}>0 \wedge \mathtt{b}=\_\mathtt{b}-1\ ) \wedge$
$\mathbf{G}\,\mathbf{X}(\ \mathtt{b}=\_\mathtt{b} \implies \neg(\_\mathtt{q3} \wedge \_\mathtt{b}<11) \wedge \neg(\_\mathtt{q5} \wedge \_\mathtt{b}>0)\ )$;

$\mathtt{Sc}$: $\mathbf{G}\,\mathbf{X}(\ \mathtt{c}>\_\mathtt{c} \implies (\_\mathtt{q1} \vee \_\mathtt{q4}) \wedge \_\mathtt{c}<101 \wedge \mathtt{c}=\_\mathtt{c}+1\ ) \wedge$
$\mathbf{G}\,\mathbf{X}(\ \mathtt{c}<\_\mathtt{c} \implies \mathtt{false}\ ) \wedge$
$\mathbf{G}\,\mathbf{X}(\ \mathtt{c}=\_\mathtt{c} \implies \neg((\_\mathtt{q1} \vee \_\mathtt{q4}) \wedge \_\mathtt{c}<101)\ )$;

$\mathtt{Sq0}$: $\mathbf{G}\,\mathbf{X}(\ \neg\_\mathtt{q0} \wedge \ \mathtt{q0} \implies \_\mathtt{q5} \wedge \neg(\_\mathtt{b}>0)\ ) \wedge$
$\mathbf{G}\,\mathbf{X}(\ \ \_\mathtt{q0} \wedge \neg\mathtt{q0} \implies \mathtt{true}\ ) \wedge$
$\mathbf{G}\,\mathbf{X}(\ \ \_\mathtt{q0} = \ \mathtt{q0} \implies \neg(\neg\_\mathtt{q0} \wedge \_\mathtt{q5} \wedge \neg(\_\mathtt{b}>0)) \wedge \neg\_\mathtt{q0}\ )$;

$$
\begin{aligned}
\texttt{Sq1}:\ & \mathbf{G}\mathbf{X}(\ \neg\texttt{\_q1} \wedge\ \texttt{q1}\ \Rightarrow\ \texttt{\_q0} \wedge \texttt{\_a} > 0 \vee \texttt{\_q4} \wedge \texttt{\_c} < 101\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q1} \wedge \neg\texttt{q1}\ \Rightarrow\ \texttt{\_c} < 101\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q1} =\ \texttt{q1}\ \Rightarrow\ \neg(\neg\texttt{\_q1} \wedge (\texttt{\_q0} \wedge \texttt{\_a} > 0 \vee \texttt{\_q4} \wedge \texttt{\_c} < 101)) \wedge \neg(\texttt{\_q1} \wedge \texttt{\_c} < 101)\ ); \\
\texttt{Sq2}:\ & \mathbf{G}\mathbf{X}(\ \neg\texttt{\_q2} \wedge\ \texttt{q2}\ \Rightarrow\ \texttt{\_q1} \wedge \texttt{\_c} < 101) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q2} \wedge \neg\texttt{q2}\ \Rightarrow\ \texttt{true}\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q2} =\ \texttt{q2}\ \Rightarrow\ \neg(\neg\texttt{\_q2} \wedge \texttt{\_q1} \wedge \texttt{\_c} < 101) \wedge \neg\texttt{\_q2}\ ); \\
\texttt{Sq3}:\ & \mathbf{G}\mathbf{X}(\ \neg\texttt{\_q3} \wedge\ \texttt{q3}\ \Rightarrow\ \texttt{\_q2} \wedge \texttt{\_a} > 0\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q3} \wedge \neg\texttt{q3}\ \Rightarrow\ \texttt{\_b} < 11\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q3} = \neg\texttt{q3}\ \Rightarrow\ \neg(\neg\texttt{\_q3} \wedge \texttt{\_q2} \wedge \texttt{\_a} > 0) \wedge \neg(\texttt{\_q3} \wedge \texttt{\_b} < 11)\ ); \\
\texttt{Sq4}:\ & \mathbf{G}\mathbf{X}(\ \neg\texttt{\_q4} \wedge\ \texttt{q4}\ \Rightarrow\ \texttt{\_q3} \wedge \texttt{\_b} < 11\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q4} \wedge \neg\texttt{q4}\ \Rightarrow\ \texttt{\_c} < 101\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q4} =\ \texttt{q4}\ \Rightarrow\ \neg(\neg\texttt{\_q4} \wedge \texttt{\_q3} \wedge \texttt{\_b} < 11) \wedge \neg(\texttt{\_q4} \wedge \texttt{\_c} < 101)\ ); \\
\texttt{Sq5}:\ & \mathbf{G}\mathbf{X}(\ \neg\texttt{\_q5} \wedge\ \texttt{q5}\ \Rightarrow\ \texttt{\_q2} \wedge \neg(\texttt{\_a} > 0) \vee \texttt{\_q6} \wedge \texttt{\_a} < 11\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q5} \wedge \neg\texttt{q5}\ \Rightarrow\ \texttt{true}\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q5} = \neg\texttt{q5}\ \Rightarrow\ \neg(\neg\texttt{\_q5} \wedge (\texttt{\_q2} \wedge \neg(\texttt{\_a} > 0) \vee \texttt{\_q6} \wedge \texttt{\_a} < 11)) \wedge \neg\texttt{\_q5}\ ); \\
\texttt{Sq6}:\ & \mathbf{G}\mathbf{X}(\ \neg\texttt{\_q6} \wedge\ \texttt{q6}\ \Rightarrow\ \texttt{\_q5} \wedge \texttt{\_b} > 0\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q6} \wedge \neg\texttt{q6}\ \Rightarrow\ \texttt{\_a} < 11\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q6} = \neg\texttt{q6}\ \Rightarrow\ \neg(\neg\texttt{\_q6} \wedge \texttt{\_q5} \wedge \texttt{\_b} > 0) \wedge \neg(\texttt{\_q6} \wedge \texttt{\_a} < 11)\ ); \\
\texttt{Sq7}:\ & \mathbf{G}\mathbf{X}(\ \neg\texttt{\_q7} \wedge\ \texttt{q7}\ \Rightarrow\ \texttt{\_q0} \wedge \neg(\texttt{\_a} > 0)\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q7} \wedge \neg\texttt{q7}\ \Rightarrow\ \texttt{false}\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \ \texttt{\_q7} = \neg\texttt{q7}\ \Rightarrow\ \neg(\neg\texttt{\_q7} \wedge \texttt{\_q0} \wedge \neg(\texttt{\_a} > 0))\ ).
\end{aligned}
$$

Рассмотрим LTL-свойства счётчиковой машины $3cM$, которые обязательно должны выполняться для любого её исполнения.

$\texttt{P1}:\ \mathbf{G}(\ \texttt{q7}\ \Rightarrow\ \texttt{c} = \texttt{n} * \texttt{n} \wedge \texttt{a} = 0 \wedge \texttt{b} = 0\ ).$

Это свойство означает, что если счётчиковая машина $3cM$ оказывается в финальном состоянии q_7, то значения счётчиков a и b будут равны 0, а счётчик c будет содержать искомый результат n^2.

$\texttt{P2}:\ \mathbf{G}(\ \texttt{a} + \texttt{b} \leqslant \texttt{n}\ ).$

Это свойство требует, чтобы суммарное значение счётчиков a и b было ограничено константой n на протяжении всего исполнения счётчиковой машины $3cM$.

$\texttt{P3}:\ \mathbf{G}(\ \texttt{c} \leqslant \texttt{n} * \texttt{n}\ ).$

Требуется, чтобы значение счётчика c на протяжении всего исполнения машины $3cM$ не выходило за пределы n^2.

$\texttt{P4}:\ \texttt{q0}\ \Rightarrow\ \mathbf{F}\mathbf{G}(\ \texttt{q7}\ ).$

Если счётчиковая машина $3cM$ стартует из состояния q_0, то она обязательно должна остановиться в своём финальном состоянии q_7.

Добавим возможность обращаться в каждом мире логики LTL к значению ещё одной переменной n. Переменная n будет использоваться как константа на протяжении одного исполнения машины $3cM$, т. е. значение n будет одним и тем же во всех мирах одной интерпретации, соответствующей этому исполнению. Чтобы зафиксировать значение переменной n, которое она будет иметь при инициализации счётчиковой машины $3cM$, построим следующую LTL-формулу:

$$
\begin{aligned}
\texttt{Sn}:\ & 0 \leqslant \texttt{n} \wedge \texttt{n} \leqslant 10 \wedge \\
& \mathbf{G}\mathbf{X}(\ \texttt{n} > \texttt{\_n}\ \Rightarrow\ \texttt{false}\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \texttt{n} < \texttt{\_n}\ \Rightarrow\ \texttt{false}\ ) \wedge \\
& \mathbf{G}\mathbf{X}(\ \texttt{n} = \texttt{\_n}\ \Rightarrow\ \texttt{true}\ ).
\end{aligned}
$$

Тогда справедливость логического вывода (в рамках логики LTL)

$$\texttt{Sn}, \texttt{Sinit}, \texttt{Sa}, \texttt{Sb}, \texttt{Sc}, \texttt{Sq0}, \texttt{Sq1}, \texttt{Sq2}, \texttt{Sq3}, \texttt{Sq4}, \texttt{Sq5}, \texttt{Sq6}, \texttt{Sq7} \models \texttt{P1}, \texttt{P2}, \texttt{P3}, \texttt{P4}$$

будет означать, что счётчиковая машина $3cM$ для любого n, $0 \leqslant n \leqslant 10$, стартуя из начального состояния q_0 при $a = n$, $b = 0$ и $c = 0$, обязательно завершит работу в финальном состоянии q_7 с результатом $a = 0$, $b = 0$ и $c = n^2$. При этом на протяжении всего исполнения будет выполняться $a + b \leqslant n$ и $c \leqslant n^2$. Действительно, левая часть логического вывода описывает те и только те интерпретации, которые соответствуют всем возможным исполнениям счётчиковой машины $3CM$ при условии $0 \leqslant n \leqslant 10$, а правая часть вывода обязывает, чтобы эти интерпретации/исполнения удовлетворяли требуемым свойствам машины $3cM$.

Рассмотрим пример логического вывода, который не будет являться справедливым в логике LTL применительно к счётчиковой машине $3cM$. Потребуем, чтобы в каждом возможном исполнении машины $3cM$ итоговый результат её работы в состоянии q_7 был отличным от $c = 2 * n$ при $n > 0$:

$$\texttt{P5} : \ \mathbf{G}(\ \texttt{q7} \wedge \texttt{n} > 0 \implies \neg(\texttt{c} = 2 * \texttt{n})\ ).$$

В этом случае получим, что формула P5 не будет выводиться из приведённых выше посылок

$$\texttt{Sn}, \texttt{Sinit}, \texttt{Sa}, \texttt{Sb}, \texttt{Sc}, \texttt{Sq0}, \texttt{Sq1}, \texttt{Sq2}, \texttt{Sq3}, \texttt{Sq4}, \texttt{Sq5}, \texttt{Sq6}, \texttt{Sq7} \not\models \texttt{P5},$$

так как существует контрпример — интерпретация, при которой все формулы левой части логического вывода являются истинными, а правая часть в виде формулы-заключения P5 становится ложной. Эта интерпретация представляет собой цепочку из 17 различных миров, последний из которых имеет петлю, т. е. достижим сам из себя. Контрпример соответствует исполнению счётчиковой машины $3cM$ при $n = 2$.

Проверку справедливости рассмотренных логических выводов можно выполнить, например, с помощью программного средства верификации Cadence SMV [7, 9, 10].

6. Проверка справедливости логических выводов

Ниже приводится код на языке программного средства верификации Cadence SMV [9], позволяющий проверить выполнимость свойств счётчиковой машины $3cM$ с использованием линейной темпоральной логики LTL.

На вход верификатора Cadence SMV подаются описания переменных-состояний, переменных-счётчиков и «константы» n, а также описания вспомогательных переменных, применяющихся для запоминания предыдущих значений состояний и счётчиков машины $3cM$. С помощью оператора `init` проводится инициализация вспомогательных переменных для начальных миров всех интерпретаций, а оператор `next` позволяет явно задать значение переменной вида $\_v$ в каждом следующем мире текущей интерпретации. Эти операторы устанавливают базовые правила переходов между мирами в рамках интерпретаций логики LTL. Запись формул логики LTL происходит через ключевое слово `assert`.

Проверка выполнимости каждого LTL-свойства P1, P2, P3, P4 и P5 проводится по отдельности посредством вызова соответствующей команды главного меню программы Cadence SMV.

Поскольку имена свойств задействованы в конструкции `using ... prove`, а точнее, указаны в разделе `prove`, то проверка выполнимости выбранного свойства будет проводиться только для тех интерпретаций логики LTL, которые одновременно удовлетворяют всем LTL-формулам, перечисленным в разделе `using`.

Контрпример для свойства P5 выводится в виде таблицы, каждый столбец которой содержит вектор значений всех переменных в соответствующем мире найденной интерпретации.

На языке Cadence SMV символы «&», «|», «~» и «->» означают логические «$\wedge$», «$\vee$», «$\neg$» и «$\Longrightarrow$» соответственно. А логические константы «true» и «false» имеют вид «1» и «0».

```
module main()
{ /* ------- описание переменных ------- */
   q0,  q1,  q2,  q3,  q4,  q5,  q6,  q7: 0..1;
  _q0, _q1, _q2, _q3, _q4, _q5, _q6, _q7: 0..1;
  a, _a, b, _b, n, _n: 0..15;
  c, _c: 0..127;
  /* ------- инициализация вспомогательных переменных ------- */
  init(_q0):= 0; init(_q1):= 0; init(_q2):= 0; init(_q3):= 0;
  init(_q4):= 0; init(_q5):= 0; init(_q6):= 0; init(_q7):= 0;
  init(_a):= 0; init(_b):= 0; init(_c):= 0; init(_n):= 0;
  /* ------- запоминание предыдущих значений переменных ------- */
  next(_q0):= q0; next(_q1):= q1; next(_q2):= q2; next(_q3):= q3;
  next(_q4):= q4; next(_q5):= q5; next(_q6):= q6; next(_q7):= q7;
  next(_a):= a; next(_b):= b; next(_c):= c; next(_n):= n;
  /* ------- LTL-спецификация ------- */
  Sn: assert /* формулы для константы n */
      0 <= n & n <= 10 &
      G X( n > _n  ->  0 ) &
      G X( n < _n  ->  0 ) &
      G X( n = _n  ->  1 );
  Sinit: assert  /* формула инициализации */
         (a = n) & (b = 0) & (c = 0) &
         (q0 = 1) & (q1 = 0) & (q2 = 0) & (q3 = 0) &
         (q4 = 0) & (q5 = 0) & (q6 = 0) & (q7 = 0);
  Sa: assert  /* формулы для переменной a */
      G X( a > _a  ->  _q6 & _a<11 & a=_a+1 ) &
      G X( a < _a  ->  (_q0 | _q2) & _a>0 & a=_a-1 ) &
      G X( a = _a  ->  ~(_q6 & _a<11) & ~((_q0 | _q2) & _a>0) );
  Sb: assert  /* формулы для переменной b */
      G X( b > _b  ->  _q3 & _b<11 & b=_b+1 ) &
      G X( b < _b  ->  _q5 & _b>0 & b=_b-1 ) &
      G X( b = _b  ->  ~(_q3 & _b<11) & ~(_q5 & _b>0) );
  Sc: assert  /* формулы для переменной c */
      G X( c > _c  ->  (_q1 | _q4) & _c<101 & c=_c+1 ) &
      G X( c < _c  ->  0 ) &
      G X( c = _c  ->  ~((_q1 | _q4) & _c<101) );
  Sq0: assert  /* формулы для переменной q0 */
       G X( ~_q0 &  q0  ->  _q5 & ~(_b>0) ) &
       G X(  _q0 & ~q0  ->  1 ) &
       G X(  _q0 =  q0  ->  ~(~_q0 & _q5 & ~(_b>0)) & ~_q0 );
  Sq1: assert  /* формулы для переменной q1 */
       G X( ~_q1 &  q1  ->  _q0 & _a>0 | _q4 & _c<101 ) &
       G X(  _q1 & ~q1  ->  _c<101 ) &
       G X(  _q1 =  q1  ->  ~(~_q1 & (_q0 & _a>0 | _q4 & _c<101)) & ~(_q1 & _c<101) );
  Sq2: assert  /* формулы для переменной q2 */
       G X( ~_q2 &  q2  ->  _q1 & _c<101 ) &
       G X(  _q2 & ~q2  ->  1 ) &
       G X(  _q2 =  q2  ->  ~(~_q2 & _q1 & _c<101) & ~_q2 );
  Sq3: assert  /* формулы для переменной q3 */
       G X( ~_q3 &  q3  ->  _q2 & _a>0 ) &
       G X(  _q3 & ~q3  ->  _b<11 ) &
       G X(  _q3 =  q3  ->  ~(~_q3 & _q2 & _a>0) & ~(_q3 & _b<11) );
  Sq4: assert  /* формулы для переменной q4 */
       G X( ~_q4 &  q4  ->  _q3 & _b<11 ) &
```

```
      G X(  _q4 & ~q4  ->  _c<101 ) &
      G X(  _q4 =  q4  ->  ~(~_q4 & _q3 & _b<11) & ~(_q4 & _c<101) );
  Sq5: assert  /* формулы для переменной q5 */
      G X( ~_q5 &  q5  ->  _q2 & ~(_a>0) | _q6 & _a<11 ) &
      G X(  _q5 & ~q5  ->  1 ) &
      G X(  _q5 =  q5  ->  ~(~_q5 & (_q2 & ~(_a>0) | _q6 & _a<11)) & ~_q5 );
  Sq6: assert  /* формулы для переменной q6 */
      G X( ~_q6 &  q6  ->  _q5 & _b>0 ) &
      G X(  _q6 & ~q6  ->  _a<11 ) &
      G X(  _q6 =  q6  ->  ~(~_q6 & _q5 & _b>0) & ~(_q6 & _a<11) );
  Sq7: assert  /* формулы для переменной q7 */
      G X( ~_q7 &  q7  ->  _q0 & ~(_a>0) ) &
      G X(  _q7 & ~q7  ->  0 ) &
      G X(  _q7 =  q7  ->  ~(~_q7 & _q0 & ~(_a>0)) );
  /* ------- проверяемые свойства счётчиковой машины ------- */
  P1: assert G( q7  ->  c=n*n & a=0 & b=0 );
  P2: assert G( a+b<=n );
  P3: assert G( c<=n*n );
  P4: assert q0  ->  F G(q7);
  P5: assert G( q7 & n>0  ->  ~(c = 2*n) );
  /* ------- предпосылки и заключения в логических выводах ------- */
  using  /* используя предпосылки */
    Sn, Sinit,
    Sa, Sb, Sc,
    Sq0, Sq1, Sq2, Sq3, Sq4, Sq5, Sq6, Sq7
  prove  /* доказать заключения */
    P1, P2, P3, P4, P5;
}
```

Заключение

В данной статье описывается способ LTL-спецификации поведения уже построенных счётчиковых машин с ограничениями. Однако применяемый для этого подход может быть задействован и для создания новых программ. Действительно, после задания с помощью LTL-формул требуемого поведения программы и проверки на выполнимость ряда ключевых программных свойств по полученной LTL-спецификации может быть построен соответствующий код на языке высокого уровня. Такой подход, например, был применён в работах [11—16] для спецификации, построения и верификации программ логических контроллеров (ПЛК). При этом кроме LTL-спецификации исполнений программы ПЛК и проверяемых программных свойств, на языке логики LTL проводилось описание согласованного поведения датчиков, т. е. накладывались ограничения на возможные входные данные ПЛК-программы.

References

[1] E. V. Kuzmin, *Non-Classical Propositional Logics*, in Russian. Yaroslavl: P.G. Demidov Yaroslavl State University, 2016, p. 160.

[2] G. Priest, *An Introduction to Non-Classical Logic. From if to is.* Cambridge University Press, 2008, p. 648.

[3] M. Minsky, *Computation: Finite and Infinite Machines.* Prentice-Hall, Inc., 1967.

[4] R. Schroeppel, "A Two Counter Machine Cannot Calculate 2^N", Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Artificial Intelligence Memo #257, 1972, p. 32.

[5] E. V. Kuzmin, *Counter Machines*, in Russian. Yaroslavl: Yaroslavl State University, 2010, p. 128.

[6] A. Pnueli, “The temporal logic of programs”, in *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, IEEE Computer Society Press, 1977, pp. 46–57.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. The MIT Press, 2001.

[8] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[9] *Cadence SMV*. [Online]. Available: http://www.kenmcmil.com/smv.html.

[10] E. Clarke, O. Grumberg, and K. Hamaguchi, “Another Look at LTL Model Checking”, Carnegie Mellon University, Tech. Rep. CMU-CS-94-114, 1994.

[11] E. V. Kuzmin and V. A. Sokolov, “On Construction and Verification of PLC Programs”, *Automatic Control and Computer Sciences*, vol. 47, no. 7, pp. 443–451, 2013.

[12] E. V. Kuzmin and V. A. Sokolov, “Modeling, Specification and Construction of PLC-programs”, *Automatic Control and Computer Sciences*, vol. 48, no. 7, pp. 554–563, 2014.

[13] E. V. Kuzmin, V. A. Sokolov, and D. A. Ryabukhin, “Construction and Verification of PLC-programs by LTL-specification”, *Automatic Control and Computer Sciences*, vol. 49, no. 7, pp. 453–465, 2015.

[14] E. V. Kuzmin, V. A. Sokolov, and D. A. Ryabukhin, “Construction and Verification of PLC LD Programs by the LTL Specification”, *Automatic Control and Computer Sciences*, vol. 48, no. 7, pp. 424–436, 2014.

[15] E. V. Kuzmin, D. A. Ryabukhin, and V. A. Sokolov, “Modeling a Consistent Behavior of PLC-Sensors”, *Automatic Control and Computer Sciences*, vol. 48, no. 7, pp. 602–614, 2014.

[16] E. V. Kuzmin, D. A. Ryabukhin, and V. A. Sokolov, “On the Expressiveness of the Approach to Constructing PLC-programs by LTL-Specification”, *Automatic Control and Computer Sciences*, vol. 50, no. 7, pp. 510–519, 2016.