ISSN 1818–1015 (Print)
ISSN 2313–5417 (Online)

Министерство образования и науки Российской Федерации
Ярославский государственный университет им. П. Г. Демидова

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ

Том 24 № 4(70) 2017

Основан в 1999 году
Выходит 6 раз в год

Адрес редакции: ЯрГУ, ул. Советская, 14, г. Ярославль, 150003, Россия
Website: http://mais-journal.ru, e-mail: mais@uniyar.ac.ru; телефон (4852) 79-77-73

Научные статьи в журнал принимаются по электронной почте. Статьи должны содержать УДК, аннотации на русском и английском языках и сопровождаться набором текста в редакторе LaTeX. Плата с аспирантов за публикацию рукописей не взимается.

12+

СОДЕРЖАНИЕ

Моделирование и анализ информационных систем. Т. 24, №4. 2017

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. Учредитель – Федеральное государственное бюджетное образовательное учреждение высшего образования "Ярославский государственный университет им. П. Г. Демидова". Подписной индекс – 31907 в Объединенном каталоге "Пресса России". Редактор, корректор А.А. Аладьева. Редактор перевода Э.И. Соколова. Подписано в печать 22.08.2017. Дата выхода в свет 31.08.2017. Формат 60x84$^1/_8$. Усл. печ. л. 15,0. Уч.-изд. л. 13,1. Объем 129 с. Тираж 46 экз. Свободная цена. Заказ 054/017. Адрес типографии: ул. Советская, 14, оф. 109, г. Ярославль, 150003 Россия. Адрес издателя: Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, г. Ярославль, 150003 Россия.

ISSN 1818–1015 (Print)
ISSN 2313–5417 (Online)

P.G. Demidov Yaroslavl State University

MODELING AND ANALYSIS OF INFORMATION SYSTEMS

Volume 24 No 4(70) 2017

Founded in 1999
6 issues per year

Editorial Office Address: P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia
Website: http://mais-journal.ru, e-mail: mais@uniyar.ac.ru

Contents

Modeling and Analysis of Information Systems. Vol. 24, No 4. 2017

Моделирование и анализ информационных систем. Т. 24, № 4 (2017), с. 391–409
Modeling and Analysis of Information Systems. Vol. 24, No 4 (2017), pp. 391–409

DOI: 10.18255/1818-1015-2017-4-391-409

УДК 519.7

О пространственной ограниченности клеточных P-сетей

Башкин В. А.

получена 21 июля 2017

Аннотация. Клеточные P-сети — обобщение концепции двухуровневых ресурсных сетей (сетей Петри) на случай бесконечной регулярной системной решетки. Этот формализм представляет собой гибрид сетей Петри и асинхронных клеточных автоматов и предназначен для моделирования мультиагентных систем с динамической пространственной структурой. Пространственная ограниченность — свойство, гарантирующее сохранение конечности "геометрических размеров" (например, площади) активной части системы на протяжении всей её жизни. Определяются три варианта пространственной ограниченности для клеточных P-сетей: локализованность, ограниченность диаметра и ограниченность площади. Исследуются свойства соответствующих алгоритмических проблем, доказывается их неразрешимость в общем случае. Предлагается нетривиальный критерий локализованности одномерной клеточной сети, основанный на новой концепции графа распространения P-автоматов. Описывается алгоритм построения графа распространения, использующий метод насыщения генерирующих путей. Предлагается способ оценки сверху диаметра одномерной клеточной сети с ограниченным графом распространения.

Ключевые слова: мультиагентные системы, верификация, сети Петри, клеточные автоматы, P-сети, пространственная ограниченность

Для цитирования: Башкин В. А., "О пространственной ограниченности клеточных P-сетей", *Моделирование и анализ информационных систем*, **24**:4 (2017), 391–409.

Об авторах:
Башкин Владимир Анатольевич, orcid.org/0000-0002-2534-1026, д-р физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: v_bashkin@mail.ru

Благодарности:
Работа выполнена при финансовой поддержке РФФИ (проект 17-07-00823).

Введение

Обыкновенные сети Петри представляют собой низкоуровневый формализм с очень простым набором основных элементов: позиция, переход, дуга и фишка. Отсутствуют сколько-нибудь удобные инструменты для высокоуровневых конструкций, таких как модуль и иерархия. Также обыкновенные сети Петри не вполне удобны для моделирования мультиагентных систем с динамической структурой, поскольку в них невозможно изменять в ходе функционирования сети структуру множества переходов. Моделировать возникновение новых агентов и исчезновение старых приходится

изменением разметки вершин-позиций, тем самым разрешая или запрещая срабатывания зависящих от них переходов.

Существует ряд формализмов, построенных на основе обыкновенных сетей Петри, в которых тем или иным способом более явно выделяется понятие агента, а также вносятся конструкции модульности и иерархичности. В частности, в *сетях Петри высокого уровня* (например, в раскрашенных сетях [14], объектных сетях [19] и многих других [11,15]) вводятся охранные функции на переходах и выражения на дугах, позволяющие усложнить условие срабатывания перехода и производимое им действие. Во вложенных сетях Петри [16] усложняется структура ресурса: он сам может быть сетью Петри (вплоть до рекурсивности).

В работе [8] был определен обладающий двухуровневой структурой визуальный язык сетей автоматов с ресурсами (P-сетей), позволяющий описывать мультиагентные системы компактно и в то же время достаточно наглядно. Предложены и исследованы новые методы моделирования и анализа систем, представленных в виде сетей с ресурсами (AP-сетей, P-сетей, клеточных P-сетей). Показано, что модульные AP-сети и вложенные сети обладают рядом конструктивных композициональных свойств (в частности, существуют специальные виды наследуемости важнейших свойств ограниченности и живости).

В работе [9] было определено и исследовано расширение концепции P-сети на случай моделей с бесконечной системной сетью. В связи с близостью по концепции к клеточным автоматам такие системы были названы клеточными сетями. В статье [9] была построена иерархия классов одномерных клеточных сетей (цепочек), основанная на ограничении топологии системной сети. Исследована выразительная мощность ряда базовых классов данной иерархии. Доказано, что: сети с полным набором портов эквивалентны машинам Тьюринга; сети без выходных портов эквивалентны конечным автоматам; сети без входных портов бисимулярны сетям Петри без коммуникаций (communication-free PN). Было продемонстрировано, что такие формализмы пригодны для моделирования и верификации распределенных систем агентов на бесконечной решетке (в частности, пространственно распределенных беспроводных сетей).

Ещё один вариант бесконечных сетей Петри был предложен Д. А. Зайцевым в работе [3]. Однако там рассматриваются обыкновенные одноуровневые сети Петри, и структура решетки сформирована на том же уровне, что и структура поведения отдельного узла. В работе [3] показано, что подобные сети могут быть использованы для моделирования и верификации статичных вычислительных решеток.

Одной из проблем, возникающих при анализе потенциально бесконечных динамичных “перемещающихся в пространстве” клеточных моделей, является выяснение самой возможности таких бесконечных “перемещений”. Например, для агентов-сенсоров обход всей решетки может быть желательным свойством. Напротив, для агентов-базовых станций желательно сохранение постоянного расстояния до станций-соседей (то есть запрет превышения максимального диаметра сети). Кроме того, пространственная ограниченность позволяет рассчитывать на разрешимость многих интересных семантических свойств моделируемых систем, поскольку, как уже было установлено ранее [9], подобные “слабые” клеточные сети слабее машин Тьюринга.

В данной работе исследуются возможности построения конструктивных критериев пространственной ограниченности для существенных подклассов клеточных сетей Петри. При этом рассматриваются различные варианты пространственной ограниченности: от “конечности множества когда-либо населенных ячеек” до более слабой “ограниченности диаметра населенной области”.

Основной результат работы — критерий локализованности клеточной сети, основанный на построении так называемого графа распространения. Соответствующий алгоритм использует метод насыщения, близкий к методу построения покрывающего дерева обыкновенной сети Петри (например, [4]). Однако в данном случае мы насыщаем не конечномерный вектор разметки, а так называемые генерирующие пути в графе распространения.

В работе рассматриваются одномерные клеточные P-сети (системная решетка представляет собой цепочку ячеек), однако все предложенные методы могут быть легко обобщены на более высокие размерности (например, клеточные модели на плоскости и в трехмерном пространстве).

Статья организована следующим образом. В первой главе приводятся основные определения и обозначения, касающиеся мультимножеств, P-сетей и клеточных P-сетей. В частности, приводятся основные свойства выразительности для соответствующих классов формальных моделей. Вторая глава посвящена проблемам пространственной ограниченности одномерных клеточных P-сетей. Вводятся определения, доказывается неразрешимость в общем случае, приводится критерий локализованности на основе графа распространения. Заключение содержит некоторые выводы и направления дальнейших исследований.

1. Предварительные сведения

1.1. Мультимножества

Пусть S — конечное множество. *Мультимножеством* M над множеством S называется отображение $M : S \to Nat$, где Nat — множество неотрицательных целых чисел. Обозначим через $\mathcal{M}(S)$ множество всех конечных мультимножеств над S.

Операции и отношения теории множеств естественно расширяются на конечные мультимножества. Пусть $M_1, M_2, M_3 \in \mathcal{M}(S)$. Полагаем: $M_1 \subseteq M_2 \Leftrightarrow \forall s \in S : M_1(s) \leq M_2(s)$; $M_1 = M_2 + M_3 \Leftrightarrow \forall s \in S : M_1(s) = M_1(s) + M_2(s)$; $M_1 = M_2 \cup M_3 \Leftrightarrow \forall s \in S : M_1(s) = max\{M_1(s), M_2(s)\}$.

1.2. Сети автоматов, управляемых ресурсами (P-сети)

Мы определяем P-сеть как AP-систему [1, 2], в которой в роли фишек выступают специальные расширенные конечные автоматы — так называемые автоматы, управляемые ресурсами (P-автоматы). Таким образом, мы расширяем понятие фишки (ресурса) путем добавления ей собственного поведения. В ходе своей работы автоматная фишка потребляет и производит фишки-ресурсы (в том числе автоматные), находящиеся в узлах, соседних тому, в котором она сама находится. Дуги в AP-сети мы помечаем именами портов, указывая тем самым узлы с доступными ресурсами.

Пусть Ω — конечное множество *типов*, $Const$ — конечное множество типизированных индивидуальных объектов, которые мы называем константами. Тип константы $c \in Const$ обозначается как $Type(c)$. Для $a \in \Omega$ через $Const(a)$ мы обозначаем множество всех констант типа a.

Далее, пусть Π — конечное множество типизированных (элементами Ω) *портов* (имён портов). Тип порта $\pi \in \Pi$ обозначим $Type(\pi)$.

Лежащая в основе P-сети сеть активных ресурсов с дугами, помеченными именами портов, называется *системной сетью*.

Определение 1. Системная сеть — *это набор* $SN = (V, I, O, \pi)$, *где*

- V — *конечное множество вершин (ресурсов);*
- $I : V \times V \to Nat$ — *множество потребляющих дуг;*
- $O : V \times V \to Nat$ — *множество производящих дуг;*
- $\pi : (I \cup O) \to \Pi$ — *функция, помечающая дуги именами портов.*

Заметим, что фактически набор (V, I, O) представляет собой структуру сети активных ресурсов [2]: типы объектов (активных ресурсов) и способы их взаимодействия (потребление и производство). Графически вершины сети изображаются кружками, потребляющие дуги — пунктирными стрелками, производящие дуги — непрерывными стрелками. Потребляющая дуга (u, v) означает, что объекты в вершине v могут (выступая в качестве активных агентов) потреблять объекты из вершины u (выступающие в качестве пассивных ресурсов). Производящая дуга (u, v) означает, что объекты в вершине u могут (выступая в качестве активных агентов) производить объекты в вершине v (выступающие в качестве пассивных ресурсов). При этом один и тот же объект может в рамках различных взаимодействий (определяемых графом системной сети) выступать в любой из 4 ролей: потребляющего агента, потребляемого ресурса, производящего агента и производимого ресурса (в том числе сразу в нескольких ролях одновременно).

В качестве объектов выступают модифицированные конечные автоматы, переходы которых дополнительно к смене внутреннего состояния самого объекта преобразуют ресурсы в инцидентных вершинах системной сети (согласно особым приписанным им ресурсным выражениям).

Определение 2. Разметкой M *системной сети* SN *называется функция*

$$M : V \to \mathcal{M}(Const),$$

сопоставляющая каждому узлу сети мультимножество находящихся в нём объектов.

Размеченной системной сетью *называется пара* (SN, M_0), *где* $SN = (V, I, O, \pi)$ — *системная сеть, а* M_0 — *её* начальная разметка.

Обозначим через Var множество типизированных переменных с типами из Ω. Пусть $a \in \Omega$. Определим язык $L(a)$ *ресурсных выражений* типа a следующим образом.

Пусть $\pi \in \Pi$. *Входной терм* есть терм вида $\pi?e$, где e — переменная или константа типа $Type(\pi)$. Аналогично, *выходной терм* есть терм вида $\pi!e$ с теми же условиями на π и e. *Ресурсное выражение* есть терм вида $\alpha_1; \alpha_2; \ldots; \alpha_k$, где α_j $(j = 1, \ldots, k)$ представляет собой входной или выходной терм (типы подвыражений $\alpha_1, \alpha_2, \ldots, \alpha_k$ могут различаться).

Определение 3. Управляемым ресурсами автоматом типа $a \in \Omega$ *(ресурсным автоматом, P-автоматом) называется набор* $A = (S_A, T_A, l_A)$, *где* S_A — *конечное множество состояний,* $T_A \subseteq S_A \times S_A$ — *отношение перехода и* $l_A : T_A \to L(a)$ — *функция пометки переходов.*

Таким образом, P-автомат — это всего лишь конечный автомат с помеченными переходами и без выделенного начального состояния. В P-сетях P-автоматы будут играть роль фишек сетей Петри. В вырожденном случае, когда P-автомат содержит одно состояние и не содержит переходов, такие фишки представляют собой обычные цветные фишки, которые мы будем называть *элементарными ресурсами.*

Определение 4. *Пусть* $\Omega = \{a_1, \ldots, a_k\}$ — *конечное множество типов,* $\mathcal{A} = (A_1, \ldots, A_k)$ — *конечное множество P-автоматов, такое, что*

1) *для типа* $a_i \in \Omega$ *множество* $Const(a_i)$ *определяется как множество всех состояний P-автомата* A_i; $Const =_{def} \bigcup_{a \in \Omega} Const(a)$;

2) *каждый* A_i *есть P-автомат типа* a_i *над множеством типов* Ω *и множествами* Ω*-типизированных переменных* Var, *имён портов* Π *и констант* $Const$.

Сеть управляемых ресурсами автоматов *(P-сеть)* $RN = (\Omega, SN, (A_1, \ldots, A_k))$ *состоит из описанного выше конечного множества P-автоматов* $(A_1, \ldots, A_k)$ *и системной сети* SN *над множеством типов* Ω *и множествами* Ω*-типизированных имён портов* Π *и констант* $Const$.

Разметкой (состоянием) P-сети называется разметка лежащей в её основе системной сети.

Допуская некоторую вольность в обозначениях, условимся не различать автомат A и его имя/тип a. Далее мы будем писать $(a|s)$ для обозначения константы, соответствующей состоянию s автомата A типа a. В зависимости от контекста такая константа может называться *агентом*, *ресурсом* или просто *объектом.*

Определим интерливинговую семантику для P-сетей. Срабатыванием P-сети является последовательность срабатываний переходов её агентов. Таким образом, только агенты могут менять состояние системы.

Для срабатывания перехода t в агенте a требуются ресурсы, перечисленные во входных подтермах помечающего t ресурсного выражения. Входной терм $p?e$ описывает ресурс e, который должен быть получен через порт p системной сети (то есть этот ресурс должен находиться в том узле системной сети, из которого ведёт дуга порта p). Аналогично, выходные термы определяют ресурсы, производимые агентом (и узлы-получатели для этих ресурсов).

Режимы срабатывания перехода определяются возможными означиваниями переменных.

Определение 5. Означиванием (переменных) *называется функция*

$$\mathbf{bind} : Var \to Const,$$

такая, что для любой переменной $\varphi \in Var$ *выполняется* $Type(\mathbf{bind}(\varphi)) = Type(\varphi)$.

Пусть M — разметка P-сети $RN = (\Omega, SN, (A_1, \ldots, A_k))$, $a|s$ — P-автомат в состоянии s, находящийся в узле v сети RN при разметке M, t – переход в $a|s$. Пусть b — означивание переменных. Переход $t = (s, s')$ с меткой $l_a(t)$ *активен* при разметке M, если существует взаимно однозначное соответствие между входными подтермами в $l_a(t)$ и объектами (фишками) в M, такое, что для каждого подтерма $p?e$ существует объект $e[b]$ в узле v', соединенном с узлом v входной дугой, помеченной именем порта p.

Описанное выше соответствие определяет "подразметку" $\check{M}$ системной разметки, описывемую входными подтермами выражения $l_a(t)$. Она включает все объекты, потребляемые срабатыванием перехода t при означивании b. Заметим, что для данных t и b разметка $\check{M}$ определяется недетерминированно (может быть несколько вариантов разметок, удовлетворяющих условиям). Через $\mathbf{in}(t[b])$ мы обозначим множество всех таких разметок.

Аналогично, определим множество $\mathbf{out}(t[b])$ разметок, сопоставляющих узлам объекты, производимые переходом t при означивании b в соответствие с выходными подтермами $l_a(t)$.

Определение 6. *Пусть* $(a|s)$ *— агент, находящийся в узле* $v \in V$ *при разметке* M, $t \in T_a$ *— переход, такой, что* $t = (s, s')$ *для некоторого* s'.

Переход t активен *с означиванием переменных* b, *если существует разметка* $\check{M} \in \mathbf{in}(t[b])$, *такая, что* $\forall u \in V : \check{M}(u) \subseteq M(u)$.

Активный означенный переход может (недетерминированно) сработать, *преобразуя* M *в новую разметку* M', *такую, что* $\forall u \in V : M'(u) = M(u) - \check{M}(u) + \hat{M}$, *где* $\check{M} \in \mathbf{in}(t[b])$ *и* $\hat{M} \in \mathbf{out}(t[b])$.

Сети управляемых ресурсами автоматов позволяют моделировать различные аспекты ресурсно-зависимых систем. Рассмотрим простейший классический пример — химическую реакцию (Рис. 1). Единственный активный агент данной системы — автомат R — определяет сам алгоритм реакции ("две молекулы водорода и одна молекула кислорода превращаются в две молекулы воды"). Системная сеть описывает физическую среду: три резервуара и трубу, в которой происходит реакция. Порты *get* и *put* связывают эти два уровня (или два аспекта) модели. С системной точки зрения порты — это физические "отверстия", с точки зрения логики агента — это имена аргументов (параметров).

Несмотря на новые возможности моделирования, P-сети обладают теми же возможностями анализа, что и сети Петри [18]:

Теорема 1. *[8] Класс систем, моделируемых P-сетями, совпадает с классом систем, моделируемых сетями Петри.*

Сети ресурсных автоматов моделируют явное и неявное взаимодействие агентов без введения каких-либо промежуточных слоёв и/или протоколов. Это позволяет

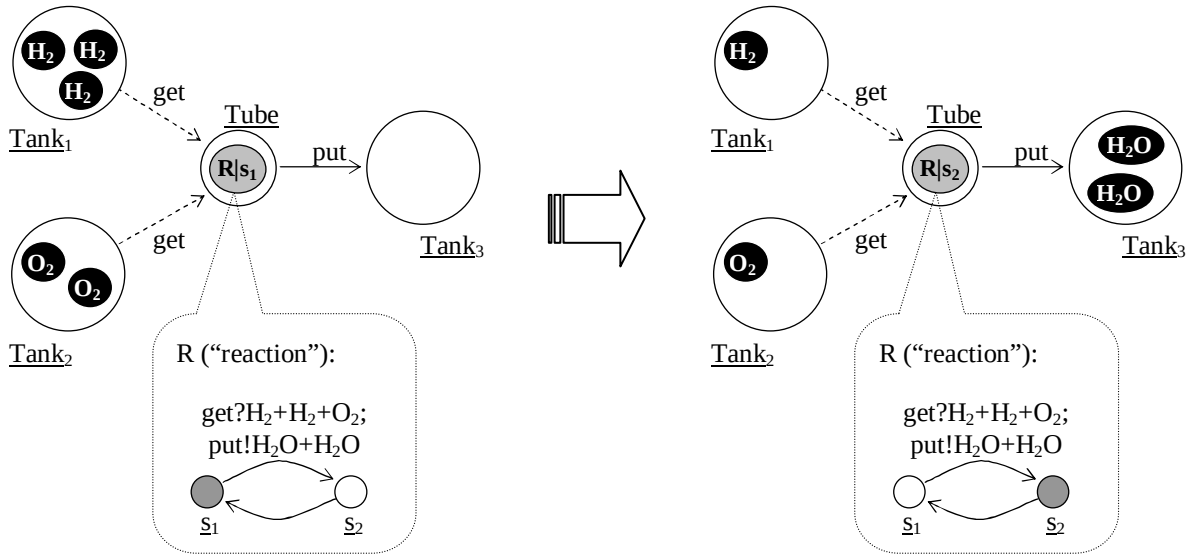


Рис. 1. Пример P-сети: химическая реакция

Fig. 1. RDA-net example: chemical reaction

достаточно адекватно отражать специфику предметной области при помощи единообразного и компактного синтаксиса. Агенты могут быть перемещены, созданы, скопированы и уничтожены другими агентами (внешняя мобильность). Они также могут выполнять все эти действия (над собой) по собственной инициативе (внутренняя мобильность).

Заметим, что выбор в качестве фишек автоматов, а не сетей Петри, нисколько не снижает выразительности формализма по сравнению с обыкновенными и высокоуровневыми сетями Петри (Теорема 1). При этом синтаксис языка получается максимально простым, что позволяет достаточно легко строить различные гибридные и высокоуровневые расширения [10].

1.3. Клеточные P-сети

Одной из интересных сторон P-сетей является возможность удобного моделирования пространственной динамики – агенты могут перемещать друг друга между узлами системной сети (или перемещаться по ней самостоятельно). Естественен переход к бесконечному случаю:

Определение 7. *[9] Клеточная P-сеть — конечная совокупность ресурсных автоматов, расположенных в узлах бесконечной регулярной системной сети (решётки).*

Клеточные P-сети (Celluar Resource Driven Automata, CRDA) — модель, подобная клеточным автоматам, где в отдельной ячейке может находиться не автомат (система с конечным числом состояний), а P-сеть (сеть Петри).

Основные **отличия** клеточных P-сетей от обыкновенных P-сетей (классических **сетей Петри**):

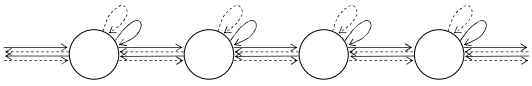
- неограниченное число отдельных сетей (P-автоматов);
- гибкое взаимодействие между ними, возможность создавать и уничтожать автоматы.

Основные **отличия** клеточных P-сетей от классических **синхронных клеточных автоматов**:

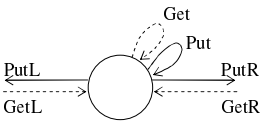
- асинхронность срабатываний ячеек, недетерминированность выбора активной ячейки;
- отдельная ячейка — не автомат, а сеть Петри, то есть система с неограниченным числом состояний.

Заметим, что клеточные автоматы универсально мощны даже в одномерном случае [12], где решетка представляет собой цепочку клеток. Поэтому мы также будем рассматривать только одномерные сети.

Рассматриваемая (бесконечная регулярная одномерная) системная сеть изображена на Рис. 2. Активные фишки (P-автоматы), находящиеся в клетке, могут потреблять/производить ресурсы из трёх клеток — левой соседней, правой соседней и текущей. Ограничений на фишки нет.



a) общий вид решетки



b) порты отдельной ячейки

Рис. 2. Решетка одномерной клеточной P-сети (1-dim CRDA)

Fig. 2. System grid of 1-dim CRDA

Теорема 2. *[9] Класс систем, моделируемых одномерными клеточными P-сетями (1-dim CRDA), совпадает с классом систем, моделируемых машинами Тьюринга.*

2. Пространственная ограниченность

2.1. Неразрешимость

В доказательстве Теоремы 2 в работе [9] использовалось преобразование произвольной машины Тьюринга, представленной в виде двухсчётчикового автомата M

(машины Минского), в одномерную клеточную сеть $N(M)$ с не более чем двумя автоматами в каждой ячейке в любой момент времени. Значения первого и второго счетчиков моделировались количеством "живых" клеток, соответственно слева и справа от центральной ячейки. Конечно, "длина" живой части решетки в такой системе может расти неограниченно.

Очевидно, что такая пространственная неограниченность является необходимым условием универсальности. Если её нет, то система представляет собой обычную, а не клеточную P-сеть, которая является всего лишь сетью Петри (Теорема 1).

Пространственная ограниченность является нетривиальным семантическим свойством, которое требует для своей проверки анализа поведения системы (в общем случае бесконечного).

Сформулируем некоторые возможные варианты определения пространственной ограниченности:

Определение 8. *(Пространственная ограниченность)*

(1) Сеть локализована, *если ни один из автоматов никогда не появляется за пределами некоторого конечного множества ячеек D.*

(2) Сеть имеет ограниченный диаметр, *если в каждый момент времени наибольшее расстояние между "населенными" ячейками (число разделяющих их ячеек) не превосходит некоторого натурального n.*

(3) Сеть имеет ограниченную площадь, *если в каждый момент времени число "населенных" ячеек не превосходит некоторого натурального n.*

Непосредственно из определений следует:

Утверждение 1. *"Локализованность" $\Rightarrow$ "Ограниченный диаметр" $\Rightarrow$ "Ограниченная площадь".*

В то же время легко придумать контрпримеры для (2) $\Rightarrow$ (1) и (3) $\Rightarrow$ (2).

По построению сети $N(M)$ в доказательстве Теоремы 2 (см. [9]) мы имеем:

Утверждение 2. *Если некоторая 2-счетчиковая машина Минского M ограничена (оба её счетчика ограничены), то соответствующая 1-dim CRDA $N(M)$ локализована.*

Тогда из неразрешимости ограниченности для 2-счетчиковых машин Минского [5] имеем:

Следствие 1. *Локализованность неразрешима для 1-dim CRDA.*

Поскольку свойства ограниченности диаметра и площади ещё слабее (Утверждение 1), имеем:

Следствие 2. *Проблемы "ограниченности диаметра" и "ограниченности площади" неразрешимы для 1-dim CRDA.*

Очевидно, это означает, что пространственная ограниченность (во всех трёх вариантах) неразрешима и при более высоких размерностях системной решетки в случае её нетривиальной топологии. Однако, как будет показано далее, использование свойств монотонности сетей Петри позволяет построить достаточно широкие критерии пространственной ограниченности.

2.2. Обозначения для одномерной решетки

Для удобства работы с одномерными клеточными сетями введём ряд новых терминов и обозначений.

Во-первых, будем считать, что все ячейки пронумерованы слева направо целыми числами. За условную точку отсчета примем нулевую ячейку.

Во-вторых, в отличие от [8, 9] мы будем использовать ресурсные выражения без переменных (только константы). Поскольку в каждом типе число констант конечно, это не снижает выразительной мощности модели. В то же время подобное упрощение синтаксиса позволяет более компактно описывать и анализировать бесконечную цепочку ячеек.

Определение 9. *Пусть a — некоторый P-автомат, s — его состояние, M — системная разметка. Обозначим через $M[i](a|s)$ количество автоматов $a|s$ в ячейке одномерной системной сети (цепочки ячеек) с номером i при разметке M (ячейки пронумерованы слева направо последовательными значениями из $\mathbb{Z}$).*

Тогда состояние ячейки с номером i при разметке M представляет собой мультимножество над $Const$:

$$M[i] =_{def} \sum_{a\in\Omega, s\in S_a} M[i](a|s),$$

или, что то же самое, $M[i] = \sum_{c\in Const} M[i](c)$.

Разметка всей системной сети–цепочки представляет собой бесконечномерный вектор вида $M = (\ldots, M[i], M[i+1], \ldots)$, где $M[i] \in \mathcal{M}(Const), i \in \mathbb{Z}$.

Счетное множество всех возможных разметок обозначим как $\mathcal{M}^\infty(Const)$.

Определение 10. *Пусть $M, M' \in \mathcal{M}^\infty(Const)$. Тогда $\forall i \in \mathbb{Z}, c \in Const$:*

$$(M + M')[i](c) =_{def} M[i](c) + M'[i](c),$$

$$(M \cup M')[i](c) =_{def} \mathbf{max}\{M[i](c), M'[i](c)\},$$

$$(M \cap M')[i](c) =_{def} \mathbf{min}\{M[i](c), M'[i](c)\}.$$

Скажем, что $M \subseteq M'$, если $\exists k \in \mathbb{Z} : \forall i \in \mathbb{Z}, c \in Const$ выполняется $M[i](c) \le M'[i+k](c)$.

Таким образом, при сравнении мы допускаем сдвиг нумерации ячеек влево или вправо на произвольную величину. Это естественно, поскольку все связи локальны, а ячейки имеют идентичные свойства, не зависящие от номера.

Набор доступных портов один и тот же для каждой ячейки:

$$\Pi = \{Get, Put, GetL, PutL, GetR, PutR\}.$$

Следовательно, для любого перехода $t \in T_a$ его ресурсное выражение $l(t)$ распадается на шесть подвыражений: $l_{Get}(t), l_{Put}(t), l_{GetL}(t), l_{PutL}(t), l_{GetR}(t)$ и $l_{PutR}(t)$ соответственно. Заметим, что каждое из этих выражений является мультимножеством констант.

Также заметим, что в силу однозначности именования портов множества $\mathbf{in}(t)$ и $\mathbf{out}(t)$ всегда содержат ровно по одному элементу. Обозначим эти элементы как ${}^\bullet t$ и $t^\bullet$ и назовём пред- и постусловием перехода t соответственно. Имеем ${}^\bullet t = (l_{Get}(t), l_{GetL}(t), l_{GetR}(t))$, $t^\bullet = (l_{Put}(t), l_{PutL}(t), l_{PutR}(t))$.

Определение 11. *Пусть автомат $a|s$ находится в ячейке с номером i при разметке M (т.е. $M[i](a|s) > 0$). Пусть также $t \in T_a$ — переход автомата a, такой, что $a|s \xrightarrow{t} a|s'$ для некоторого $s' \in S_a$. Переход t* активен, *если выполняется*

- $l_{GetL}(t) \subseteq M[i-1]$;
- $l_{Get}(t) \subseteq M[i]$;
- $l_{GetR}(t) \subseteq M[i+1]$.

Активный переход t может сработать, *порождая новую разметку M' :*
- $M'[i-1] = M[i-1] - l_{GetL}(t) + l_{PutL}(t)$;
- $M'[i]$:
  - $M'[i](c) = M[i](c) - l_{Get}(t)(c) + l_{Put}(t)(c)$ *для всех* $c \neq a|s$;
  - *при* $M[i](a|s) = l_{Get}(t)(a|s)$ *положим* $M'[i](a|s) = M[i](a|s) - l_{Get}(t)(a|s) + l_{Put}(t)(a|s)$;
  - *при* $M[i](a|s) > l_{Get}(t)(a|s)$ *недетерминированно выберем один из двух вариантов:* $M'[i](a|s) = M[i](a|s) - l_{Get}(t)(a|s) + l_{Put}(t)(a|s)$ *или* $M'[i](a|s) = M[i](a|s) - (a|s) + (a|s') - l_{Get}(t)(a|s) + l_{Put}(t)(a|s)$;
- $M'[i+1] = M[i+1] - l_{GetR}(t) + l_{PutR}(t)$;
- $M'[j] = M[j]$ *при* $j \notin \{i-1, i, i+1\}$

Такое срабатывание обозначим как $M \xrightarrow[i]{t} M'$.

Лемма 1. *Пусть $M, M' \in \mathcal{M}^{\infty}(Const), t \in T_a$ для некоторого $a \in \Omega$. Тогда*

$$M \xrightarrow[i]{t} M' \ \wedge \ t^{\bullet} \subseteq {}^{\bullet}t \ \Rightarrow \ M' \subseteq M.$$

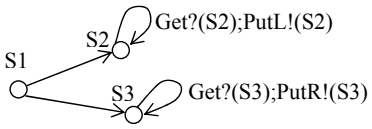
Доказательство. Непосредственной подстановкой в определения активности и срабатывания перехода. □

Лемма 2. *Пусть $M, M' \in \mathcal{M}^{\infty}(Const), t \in T_a$ для некоторого $a \in \Omega$. Тогда*

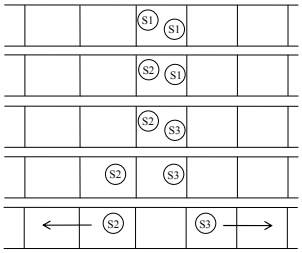
$$M \xrightarrow[i]{t} M' \ \wedge \ |t^{\bullet}| \leq |{}^{\bullet}t| \ \Rightarrow \ |M'| \leq |M|.$$

Доказательство. Аналогично. Заметим также, что с точки зрения количества автоматов не важно, в каком направлении они перемещаются при срабатывании. □

Следствие 3.
1. *Если для каждого перехода t каждого Р-автомата a выполнено ${}^{\bullet}t \subseteq t^{\bullet}$, то сеть локализована.*
2. *Если для каждого перехода t каждого Р-автомата a выполнено $|{}^{\bullet}t| \leq |t^{\bullet}|$, то сеть имеет ограниченную площадь.*



a) Диаграмма переходов автомата



b) “Расползание” автоматов

Рис. 3. Сеть неограниченного диаметра

Fig. 3. Net with unbounded diameter

Сформулированные выше свойства тривиальны: если переходы не увеличивают разметку соседних ячеек, то и разметка всей сети не увеличивается; а если переходы не увеличивают общее количество автоматов, то и занимаемая площадь не может вырасти.

Для свойства ограниченности диаметра таких простых ограничений уже недостаточно: даже “неувеличивающая” сеть может неограниченно разрастаться (Рис. 3).

Следующая теорема устанавливает, что для клеточных сетей существует аналог свойства монотонности сетей Петри (см., например, [4]):

Теорема 3. *Пусть* $M, M', K \in \mathcal{M}^\infty(Const), t \in T_a$ *для некоторого* $a \in \Omega$. *Тогда*

$$M \xrightarrow[i]{t} M' \;\Rightarrow\; M + K \xrightarrow[i]{t} M' + K.$$

Доказательство. Заметим, что при рассмотрении конечного числа срабатываний (в частности, одного) клеточная сеть ведёт себя как обычная P-сеть с конечным числом ячеек системной сети (т.е. как обычная сеть Петри). Далее доказательство аналогично доказательству соответствующей теоремы в [4]. □

Однако, в отличие от обыкновенных сетей Петри, клеточные P-сети не являются вполне структурированными системами переходов [13], так как отношение $\subseteq$ на множестве $\mathcal{M}^\infty(Const)$ (определение 10) не является правильным квазипорядком.

2.3. Граф распространения

Рассмотрим способ описания взаимного перемещения P-автоматов на одномерной решетке при отсутствии сдерживания со стороны *Get*-выражений (то есть в предположении наличия неограниченных ресурсов в каждой ячейке).

Далее представлен алгоритм, строящий так называемый *граф распространения* системы P-автоматов. Этот граф представляет собой (не всегда конечную) комбинацию диаграмм переходов всех автоматов из Ω во всех ячейках сети, до которых они могут "распространиться" при старте процесса из начальной ячейки. При этом различные диаграммы, находящиеся в одной и той же ячейке или в двух соседних, могут быть связаны дополнительными "порождающими дугами". Неформально, если автомат a в состоянии s способен сработать, генерируя (через какой-то из Put-портов) новый автомат a' в состоянии s', то в графе распространения добавляется "генерирующая дуга" из $a|s$ в $a'|s'$. При этом если $a'|s'$ был сгенерирован через $PutL$, то дуга идёт в соседнюю левую ячейку, а если через Put или $PutR$, то, соответственно, в текущую или соседнюю правую. При этом Get-выражения не учитываются (и, следовательно, никак не сдерживают развитие процесса распространения автоматов), то есть получается аппроксимация сверху поведения любого сообщества клеток-автоматов с типами из Ω.

Алгоритм 1. *(граф распространения)*
$PropagationGraph(\Omega)$
Вход: *множество типов* $\Omega = \{a_1, \ldots, a_k\}$.
Выход: *граф распространения* $\mathcal{G}_{prop}(\Omega)$.

1. *Положим G — пустой граф, обозначим все состояния всех P-автоматов признаком "новое".*

2. *Пока в автоматах есть "новые состояния", повторим следующее: выберем "новое" состояние $a|s$, обозначим его признаком "старое", достроим граф распространения G при помощи вызова процедуры $Propagation(G, a|s, 0, \emptyset, \emptyset)$.*

3. *Вернём G.*

Алгоритм 2. *(достройка графа распространения)*
$Propagation(G, a|s, Shift, t, v)$
Вход: *Ориентированный граф G; состояние $a|s$ ($a \in \Omega, s \in S_a$); сдвиг $Shift \in \{-1, 0, +1\}$; переход t ($t \in T_b$ для некоторого $b \in \Omega$); вершина v графа G (помеченная тройкой $i|b|q$, такой, что $b|q \xrightarrow{t} \ldots, i \in \mathbb{Z}$, причем $(a|s) \in l_{PutL}(t)$ при $Shift = -1$, $(a|s) \in l_{Put}(t)$ при $Shift = 0$ и $(a|s) \in l_{PutR}(t)$ при $Shift = +1$).*
Выход: *Ориентированный граф G.*

1. *Добавим (в случае отсутствия) в граф G вершину с меткой $i + Shift|a|s$ и "генерирующую" дугу от $i|b|q$ к $i + Shift|a|s$ с двойной меткой $(Shift, t)$ (меткой сдвига $Shift$ и меткой перехода t).*

2. *Рассмотрим все вершины графа G с метками вида $i + Shift|a|x$ для всех $x \in S_a$. Пока в диаграмме переходов автомата a найдётся состояние $a|y$, достижимое от $a|x$ посредством перехода z (т.е. $a|x \xrightarrow{z} a|y$), для которого в графе G ещё нет вершины с меткой $i + Shift|a|y$ или дуги от $i + Shift|a|x$ к $i + Shift|a|y$ с одинарной меткой перехода z, добавим в G такую вершину и/или дугу.*

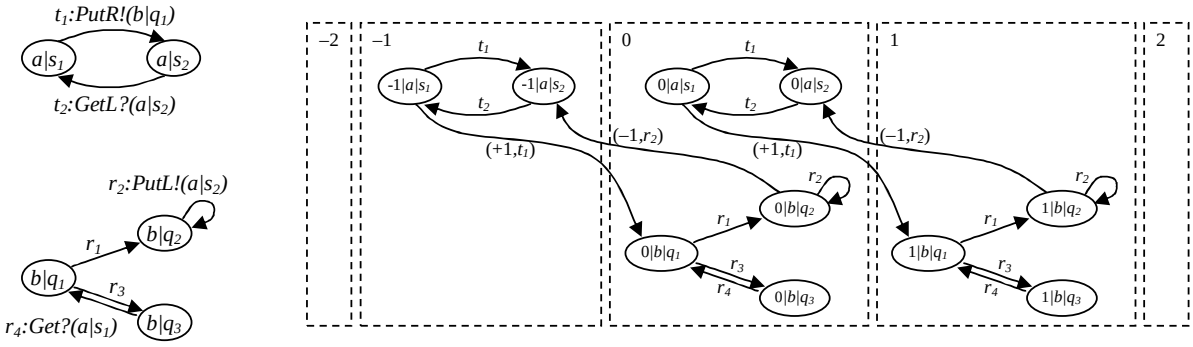


Рис. 4. Множество P-автоматов $\Omega = \{a, b\}$ и его граф распространения

Fig. 4. A set of RDAs $\Omega = \{a, b\}$ and its propagation graph

3а. Для каждой вершины v' с меткой вида $j|a|x$, добавленной в граф G на предыдущем шаге, перехода $t' \in T_a$ и константы $a'|s'$, таких что $a|x \xrightarrow{t'} a|y$ для некоторого y и $(a'|s') \in l_{PutL}(t')$, выполним $Propagation(G, a'|s', -1, t', v')$.

3б. Для каждой вершины v' с меткой вида $j|a|x$, добавленной в граф G на предыдущем шаге, перехода $t' \in T_a$ и константы $a'|s'$, таких что $a|x \xrightarrow{t'} a|y$ для некоторого y и $(a'|s') \in l_{Put}(t')$, выполним $Propagation(G, a'|s', 0, t', v')$.

3в. Для каждой вершины v' с меткой вида $j|a|x$, добавленной в граф G на предыдущем шаге, перехода $t' \in T_a$ и константы $a'|s'$, таких что $a|x \xrightarrow{t'} a|y$ для некоторого y и $(a'|s') \in l_{PutR}(t')$, выполним $Propagation(G, a'|s', +1, t', v')$.

Пример графа распространения приведён на Рис. 4. Здесь рассмотрены автоматы двух типов (a и b), способные создавать друг друга. Правая часть рисунка представляет граф $\mathcal{G}_{prop}(\{a, b\})$, для наглядности разбитый на "ячейки". Видно, что заполнены только ячейки с номерами -1, 0 и 1. Дуги $(+1, t_1)$ и $(-1, r_2)$ соответствуют событиям генерации новых автоматов в соседних ячейках (правой и левой соответственно) при срабатываниях t_1 и r_2.

Довольно очевидно, что любой конечный набор P-автоматов типов a и b, изображенных на рисунке 4, не сможет "расползтись" по решетке системной P-сети более чем на одну ячейку влево и одну ячейку вправо относительно своих стартовых позиций. Это следует из того, что никакая последовательность переходов в соответствующем графе распространения не уходит далеко от той ячейки, в которой она "стартовала." В данном случае ячейки графа $\mathcal{G}_{prop}(\Omega)$ можно рассматривать как абстрактные аналоги ячеек решетки системной сети (использующие не абсолютную нумерацию — номера ячеек, а относительную — сдвиги номеров ячеек относительно исходной).

Утверждение 3. *Число компонент связности графа $\mathcal{G}_{prop}(\Omega)$ не превосходит $|\Omega|$.*

Доказательство. Заметим, что по построению графа каждая вершина достижима из некоторой вершины вида $(0|a|s)$ (т.е. вершины, находящейся в нулевой ячейке). А

в нулевой ячейке находятся в точности все вершины всех диаграмм переходов всех автоматов. Кроме того, в ней также имеются все внутренние дуги диаграмм переходов (с одинарными метками), а также может присутствовать некоторое количество генерирующих дуг с двойными метками. □

Единственный способ получения неограниченного распространения автоматов влево и/или вправо — бесконечные пути в соответствующем графе распространения.

Определение 12. *Бесконечный ациклический путь в графе распространения назовём* генерирующим.

Генерирующий путь, проходящий через все ячейки с индексами, не превышающими некоторого $i \in \mathbb{Z}$, назовём генерирующим влево.

Генерирующий путь, проходящий через все ячейки с индексами, превышающими некоторое $i \in \mathbb{Z}$, назовём генерирующим вправо.

Утверждение 4. *1. Любой генерирующий путь является либо генерирующим влево, либо генерирующим вправо.*

2. *Никакой путь не может быть одновременно генерирующим влево и генерирующим вправо.*

3. *В ациклическом пути σ найдутся две одинаковые генерирующие дуги с метками $(-1, t)$, такими, что вторая из них находится в графе левее первой (т.е. стартует в ячейке с ме́ньшим индексом) $\Leftrightarrow$ путь σ является генерирующим влево.*

4. *В ациклическом пути σ найдутся две одинаковые генерирующие дуги с метками $(+1, t)$, такими, что вторая из них находится в графе правее первой (т.е. стартует в ячейке с бо́льшим) $\Leftrightarrow$ путь σ является генерирующим вправо.*

Доказательство. (1) Заметим, что в каждой ячейке графа распространения содержится конечное множество вершин и дуг. Следовательно, любой бесконечный ациклический путь должен проходить через бесконечное множество ячеек. Кроме того, все дуги, не являющие внутренними дугами ячейки, связывают между собой исключительно соседние ячейки. Следовательно, путь, проходящий через ячейки с индексами i и $i + 2$, проходит также и через ячейку с индексом $i + 1$.

(2) Предположим противное: такой путь σ существует. Но в этом случае он должен бесконечно много раз проходить через центральную ячейку. Тогда в силу конечности числа вершин в ячейке мы получим нарушение свойства ацикличности.

(3) ($\Rightarrow$) Предположим противное: в графе имеется такой не генерирующий влево ациклический путь:

$$\sigma = \ldots \to (i|a|s) \overset{(-1,t)}{\to} (i-1|a'|s') \to \ldots$$

$$\to (j|a|s) \overset{(-1,t)}{\to} (j-1|a'|s') \to \ldots, \text{ где } \; i > j.$$

Однако в таком случае по построению графа эта дуга должна повториться ещё раз ещё левее: $(j+(j-i)|a|s) \overset{(+1,t)}{\to} (j+1+(j-i)|a'|s')$. Более того, она будет повторяться

бесконечно много раз на границах ячеек с шагом $(j-i)$. Следовательно, путь бесконечен и проходит все ячейки слева от i-ой, то есть является генерирующим влево — противоречие.

(3) ($\Leftarrow$) В силу бесконечности числа пересечений справа налево границ ячеек и конечности множеств переходов автоматов хотя бы один из них должен повторяться.

(4) Симметрично предыдущему случаю. □

Теорема 4. *Граф $\mathcal{G}_{prop}(\Omega)$ конечен тогда и только тогда, когда в нём нет ни одного генерирующего влево или вправо пути.*

Доказательство. ($\Leftarrow$) В силу первой части Утверждения 4 в графе нет ни одного бесконечного ациклического пути. В силу Утверждения 3 число компонент связности также конечно. Следовательно, граф конечен.

($\Rightarrow$) Без ограничения общности достаточно рассмотреть только один из двух типов генерирующих путей. Рассмотрим генерацию вправо.

Предположим противное: в графе имеется путь с двумя такими дугами: $(i|a|s) \overset{(+1,t)}{\to} (i+1|a'|s')$ и $(j|a|s) \overset{(+1,t)}{\to} (j+1|a'|s')$. Однако в таком случае по построению графа эта дуга должна повториться ещё раз ещё правее: $(j+(j-i)|a|s) \overset{(+1,t)}{\to} (j+1+(j-i)|a'|s')$. Более того, она будет повторяться бесконечно много раз на границах ячеек с шагом $(j-i)$. Следовательно, граф бесконечен — противоречие. □

Алгоритм 3. *(проверка конечности графа распространения $\mathcal{G}_{prop}(\Omega)$)*
$GraphIsFinite(\Omega)$
Вход: *множество типов* $\Omega = \{a_1, \ldots, a_k\}$.
Выход: *ответ "конечен" или "бесконечен".*
Основной цикл: *Запустим процедуру* $PropagationGraph(\Omega)$, *контролируя возникновение генерирующих цепочек. Возможные варианты завершения:*

- *Граф построен $\Rightarrow$ ответ "конечен".*
- *В процессе построения возникла генерирующая цепочка $\Rightarrow$ ответ "бесконечен".*

Алгоритм сходится: в противном случае в бесконечном графе отсутствовали бы генерирующие цепочки.

Теорема 5. *Если размеченная клеточная P-сеть с автоматами из Ω не локализована (неограничена в смысле (1)), то граф $\mathcal{G}_{prop}(\Omega)$ бесконечен.*

Доказательство. Предположим противное: граф конечен, хотя сеть N распространяется неограниченно влево и/или вправо. Однако очевидно, что любой (в том числе бесконечный) процесс порождения автоматов (цепочка срабатываний переходов) в размеченной сети должен иметь аналог в графе распространения, поскольку тот содержит все возможные последовательности срабатываний автоматов из Ω в условиях отсутствия каких-либо ограничений по ресурсам. И если сеть не является локальной (например, неограниченно распространяется вправо), то соответствующая её разрастанию бесконечная последовательность переходов возможна и в $\mathcal{G}_{prop}(\Omega)$. Следовательно, граф бесконечен вправо — противоречие. □

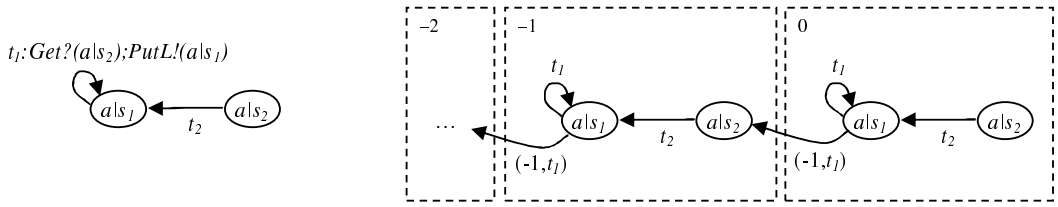


Рис. 5. Р-автомат с бесконечным графом распространения (генерирующий влево путь), который порождает локализованную сеть

Fig. 5. RDA with an infinite propagation graph (left-generating path), producing localized net

Обратное не всегда верно: существует локализованная сеть, граф для которой бесконечен.

На Рисунке 5 изображен Р-автомат, граф распространения которого содержит бесконечный генерирующий влево путь. Однако выполнение такого пути возможно только при наличии в каждой ячейке с отрицательным номером автомата $(a|s_2)$ (т.к. генерирующий переход t_1 помечен соответствующим Get-выражением). Однако начальная разметка всегда конечна, поэтому таких автоматов на решетке может быть только конечное число. И взяться им неоткуда, так как генерируется только константа $(a|s_1)$, а из состояния s_1 состояние s_2 недоступно. Таким образом, бесконечный генерирующий путь никогда не реализуется полностью.

В силу Теоремы 5 Алгоритм 3 позволяет обнаруживать неограниченность сети в смысле (1) (то есть отсутствие локализованности). Действительно, достаточно проверить конечность графа $\mathcal{G}_{prop}(\Omega)$.

Заметим, что данный алгоритм является не разрешающей, а полуразрешающей процедурой для пространственной ограниченности, поскольку допускает “false positive”: для некоторых ограниченных сетей он может дать ответ “не ограничена” (пример на Рисунке 5). В то же время он всегда правильно подтверждает неограниченность, и поэтому является практически значимым.

Кроме того, граф распространения в случае своей конечности позволяет оценивать сверху диаметр локальных клеточных сетей:

Теорема 6. *Пусть граф $\mathcal{G}_{prop}(\Omega)$ конечен и содержит n “ячеек”. Тогда диаметр любой клеточной сети с автоматами из Ω в ходе её функционирования не может вырасти более, чем на n относительно своего начального значения.*

Доказательство. Очевидно, поскольку никакой из начальных автоматов не может породить потомков на бóльшем расстоянии. □

3. Заключение

В работе показано, что проблема пространственной ограниченности клеточных P-сетей (и, следовательно, бесконечных сетей Петри) является в общем случае неразрешимой. Вводятся три варианта определения пространственной ограниченности: локализованность, ограниченность площади и ограниченность диаметра.

Предложено понятие графа распространения P-автоматов в одномерной клеточной сети. Разработан алгоритм построения графа распространения, использующий новое свойство "пространственной монтонности". Предложен нетривиальный критерий локализованности модели, основанный на проверке конечности графа распространения. Предложен метод оценки сверху диаметра локализованной клеточной P-сети с конечным графом распространения.

В качестве возможных направлений дальнейших исследований мы рассматриваем поиск более слабых критериев ограниченности, а также изучение некоторых других "пространственных" свойств клеточных ресурсных моделей: "объема" модели, устойчивых конфигураций, проблем замещения и т.п.

Автор выражает благодарность Горностаевой Екатерине за помощь в подготовке текста статьи.

Список литературы / References

[1] Башкин В. А., "Сети активных ресурсов", *Моделирование и анализ информационных систем*, **14**:4 (2007), 13–19; [Bashkin V.A., "Nets of active resources", *Model. Anal. Inform. Sist.*, **14**:4 (2007), 13–19, (in Russian).]

[2] Башкин В. А., "Формализация семантики систем с ненадежными агентами при помощи сетей активных ресурсов", *Программирование*, **36**:4 (2010), 3–15; English transl.: Bashkin V. A., "Formalization of semantics of systems with unreliable agents by means of nets of active resources", *Progr. and Comp. Soft.*, **36**:4 (2010), 187–196.

[3] Зайцев Д. А., "Верификация вычислительных решеток с особыми краевыми условиями бесконечными сетями Петри", *Моделирование и анализ информационных систем*, **19**:6 (2012), 21–33; English transl.: Zaitsev D. A., "Verification of computing grids with special edge conditions by infinite Petri nets", *Automatic Control and Computer Sciences*, **47**:7 (2013), 403–412.

[4] Котов В. Е., *Сети Петри*, Наука, М., 1984; [Kotov V. E., *Seti Petri*, Nauka, Moskva, 2008, (in Russian).]

[5] Кузьмин Е. В., Чалый Д. Ю., "О разрешимости проблем ограниченности для счетчиковых машин Минского", *Моделирование и анализ информационных систем*, **15**:1 (2008), 16–26; [Kuzmin E. V., Chalyy D. Ju., "On the decidability of boundedness problems for counter Minsky machines", *Model. Anal. Inform. Sist.*, **15**:1 (2008), 16–26. (in Russian).]

[6] Alur R., Dill D., "Automata for modeling real-time systems" (Automata, Languages and Programming), *Lecture Notes In Computer Science*, **443** (1990), 322–335.

[7] Bashkin V. A., "Nets of active resources for distributed systems modeling", *Joint Bulletin of NCC&IIS, Comp. Science*, **28** (2008), 43–54.

[8] Bashkin V. A., Lomazova I. A., "Resource Driven Automata Nets", *Fundamenta Informaticae*, **109**:3 (2011), 223–236.

[9] Bashkin V. A., Lomazova I. A., "Cellular Resource Driven Automata Nets", *Fundamenta Informaticae*, **120**:3-4 (2012), 245–259.

[10] Bashkin V. A., Lomazova I. A., Novikova Yu. A., "Timed Resource Driven Automata Nets for Distributed Real-Time Systems Modelling" (Parallel Computing Technologies), *Lecture Notes In Computer Science*, **7979** (2013), 13–25.

[11] Bednarczyk M. A., Bernardinello L., Pawlowski W., Pomello L., "Modelling mobility with Petri Hypernets" (Recent Trends in Algebraic Development Techniques), *Lecture Notes In Computer Science*, **3423** (2005), 28–44.

[12] Cook M., "Universality in Elementary Cellular Automata.", *Complex Systems*, **15**:1 (2004), 1–40.

[13] Finkel A., Schnoebelen Ph., "Well-Structured Transition Systems Everywhere!", *Theoretical Computer Science*, **256**:1–2 (2001), 63–92.

[14] Jensen K., *Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Springer, 1994.

[15] Köhler-Bußmeier M., "Hornets: Nets within Nets combined with Net Algebra" (ICATPN'2009), *Lecture Notes In Computer Science*, **5606** (2009), 243–262.

[16] Lomazova I. A., "Nested Petri Nets – a Formalism for Specification and Verification of Multi-Agent Distributed Systems", *Fundamenta Informaticae*, **43**:1–4 (2000), 195–214.

[17] Nehaniv C. L., "Asynchronous Automata Networks Can Emulate Any Synchronous Automata Network", *Int. J. of Algebra and Computation*, **14**:5–6 (2004), 719–739.

[18] Petri C. A., *Kommunikation mit Automaten. PhD thesis*, Bonn Institute für Instrumentelle Mathematik, 1962.

[19] Valk R., "Petri Nets as Token Objects: An Introduction to Elementary Object Nets" (ICATPN'98), *Lecture Notes In Computer Science*, **1420** (1998), 1–25.

Bashkin V. A., "On the Spatial Boundedness of Cellular RDA-nets", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 391–409.

DOI: 10.18255/1818-1015-2017-4-391-409

Abstract. Cellular resource driven automata nets (CRDA-nets) is a generalization of the concept of two-level resource nets (Petri nets) with an infinite regular system grid. This formalism is a hybrid of Petri nets and asynchronous Cellular Automata and is designed for modeling multi-agent systems with dynamic spatial structure. Spatial boundedness is a property that guarantees the preservation of the finiteness of "geometric dimensions" of the active part of the system (for example, the living space) during its lifetime. Three variants of spatial boundedness for cellular RDA-nets are defined: localization, bounded diameter and bounded area. The properties of the corresponding algorithmic problems are investigated, their undecidability in the general case is proved. A non-trivial criterion for the localization of an one-dimensional CRDA-net is proposed, based on the new concept of the RDA propagation graph. An algorithm is described for constructing a propagation graph, using the method of saturation of generating paths. A method for estimating the diameter of an 1-dim CRDA with a bounded propagation graph is presented.

Keywords: multiagent systems, verification, Petri nets, cellular automata, resource driven automata nets, spatial boundedness

On the authors:
Vladimir A. Bashkin, orcid.org/0000-0002-2534-1026, PhD,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: v_bashkin@mail.ru

Acknowledgments:
This work was supported by the Russian Fund for Basic Research (17-07-00823).

Моделирование и анализ информационных систем. Т. 24, №4 (2017), с. 410–414
Modeling and Analysis of Information Systems. Vol. 24, No 4 (2017), pp. 410–414

DOI: 10.18255/1818-1015-2017-4-410-414

УДК 004.9

Уточнение свойств центроида дерева

Белов Ю. А., Вовчок С. И.

получена 25 июля 2017

Аннотация. Работа посвящена уточнению свойств центроида дерева. Внимание авторов привлекла популярная задача (бинарного) разбиения графа, для которой неизвестен неперебoрный алгоритм. Выяснено, что для «экономного» разбиения дерева имеет смысл рассматривать разбиения в окрестности центроидных вершин, определение которых представлено. В ходе работы предложены доказательства, связанные с ограничением их веса. Также доказано, что если в дереве имеются две центроидные вершины, то они смежны. В следствии отмечается, что в дереве не могут иметь место три таких вершины. Составлены соответствующие предложения. Согласно первому, любая вершина дерева с определенным ограничением на ее вес является центроидной. По одному из пунктов второго предложения, если в дереве имеются две центроидные вершины, то порядок дерева является чётным числом. Третье предложение гласит, что если в дереве имеется центроидная вершина ограниченного веса, то имеется и другая центроидная вершина того же веса и смежная с первой. Для доказательства предложений рассматривается ветвь наибольшего веса при центроидной вершине и в этой ветви берется другая смежная с центроидной вершина. В работе используется теорема Жордана, при изложении материала представлено три изображения.

Ключевые слова: центроид, центроид дерева

Для цитирования: Белов Ю. А., Вовчок С. И., "Уточнение свойств центроида дерева", *Моделирование и анализ информационных систем*, **24**:4 (2017), 410–414.

Об авторах:
Белов Юрий Анатольевич, orcid.org/0002-0007-1896-0951, канд. физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: belov45@yandex.ru

Вовчок Сергей Иванович, orcid.org/0000-0003-0535-5786, аспирант,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: vof4ok@gmail.com

Введение

Поводом для рассмотрения указанного в заголовке вопроса явилась популярная задача (бинарного) разбиения графа: для данного простого графа указать способ разбиения его вершин на два подмножества такие, что, во-первых, подмножества равны по количеству вершин (или равны с некоторой заданной погрешностью) и, во-вторых, количество «перекрёстных» рёбер графа (то есть рёбер, соединяющих вершины из различных множеств) при этом разбиении минимально. Конечно, задача допускает различные варианты постановки — для нагруженных графов, для мультиграфов, для ориентированных графов и т. п., но даже для простейшего случая точный не переборный алгоритм её решения неизвестен.

Обычное эвристическое соображение, используемое при этом, состоит в том, что для связного графа имеется набор остовных деревьев и достаточно построить разбиение для некоторого остовного дерева, чтобы автоматически получить разбиение всего графа. Конечно, при этом возникают вопросы, какое остовное дерево «выгоднее» взять, чтобы получить в итоге более качественное разбиение графа. Кроме того, возникает задача «экономного» разбиения дерева, в связи с чем и имеет смысл рассматривать разбиения дерева, «в окрестности центроидных вершин», то есть разбиения, в которых исключаются только рёбра, примыкающие к центроидной вершине.

Доказательства и предложения

Рассматривается дерево — связный граф без циклов. Порядок графа $n(T)$ — количество его вершин. Как известно, количество рёбер при этом равно $n-1 = R(T)$, оно иногда называется размером дерева. Конечно, можно отметить, что это практически то же самое, что порядок графа, но некоторые формулы с использованием этого понятия будут проще, поэтому понятие размера имеет какой-то смысл. Для произвольной вершины v дерева можно рассмотреть набор поддеревьев, называемых ветвями данной вершины. Ветвь при вершине v — максимальный подграф графа T, для которого данная вершина является концевой.

Количество ветвей, примыкающих к данной вершине v, равно, конечно, степени этой вершины. Понятно также, что ветви, примыкающие к данной вершине v, не имеют общих рёбер и имеют только одну общую вершину — вершину v, сумма размеров всех ветвей, примыкающих к произвольной вершине, равна количеству всех рёбер дерева, то есть размеру графа T (см. рис. 1):

$$n-1 = R(T). \qquad (*)$$

На рисунке 1 при вершине v имеются три ветви: размеров 1, 3 и 9.

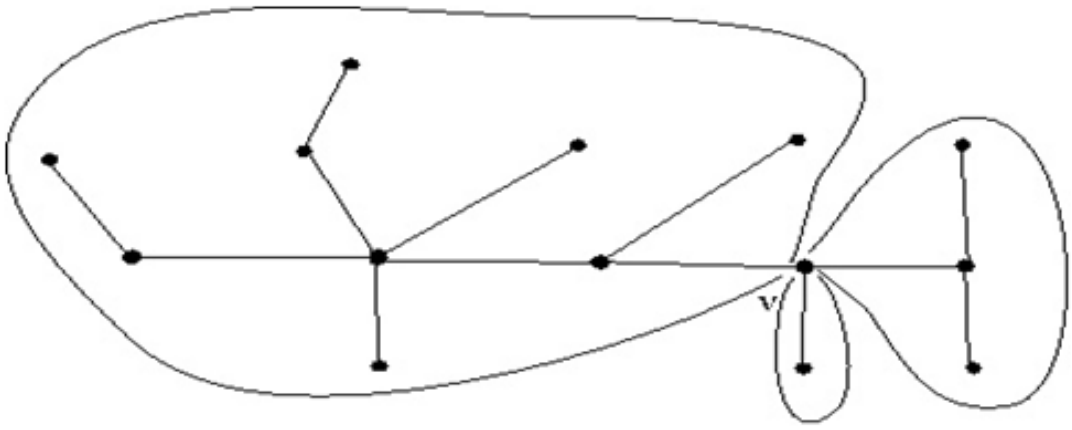


Рис. 1. Изображение графа T
Fig.1 Graph T image

Определение 1. *Для произвольной вершины v дерева T весом вершины называется максимум размеров ветвей, примыкающих к данной вершине v; будем обозначать его через $w(v)$.*

Например, если v — концевая вершина дерева, то к ней примыкает ровно одна ветвь, размер её равен $n-1$ и она просто совпадает со всем исходным графом. Таким образом, вес любой концевой вершины равен $n-1$. Вообще вес вершин, очевидно, находится в пределах от 1 до $n-1$. На рисунке 1 $w(v) = 9$.

Определение 2. *Вершина называется центроидной, если её вес наименьший среди всех вершин дерева.*

Теорема (Жордан). В произвольном дереве имеются ровно две центроидные вершины, смежные друг с другом, или ровно одна центроидная вершина.

Докажем это утверждение и некоторые его уточнения.

1. Докажем, что вес любой центроидной вершины не превосходит $\frac{n}{2}$. Предположим противное, пусть имеется ветвь, примыкающая к центроидной вершине v, имеющая размер больше $\frac{n}{2}$. Тогда, во-первых, такая ветвь может быть только одна, в силу (*). Пусть (v, a) — ребро, входящее в эту ветвь. Тогда, как легко понять, вес вершины a будет меньше веса вершины v, что противоречит центроидности вершины v.

   Действительно, сумма размеров всех ветвей, при вершине a, кроме одной ветви, включающей ребро (a, v), на единицу меньше размера наибольшей ветви при вершине v, а одна оставшаяся ветвь, примыкающая к вершине a, будет иметь размер не более $\frac{n}{2}$, так как все ветви, примыкающие к вершине v, кроме наибольшей ветви, имели в сумме размер не более $n-1-\frac{n}{2} = \frac{n}{2}-1$, а теперь к этой величине лишь добавилась единица (см. рис. 2).

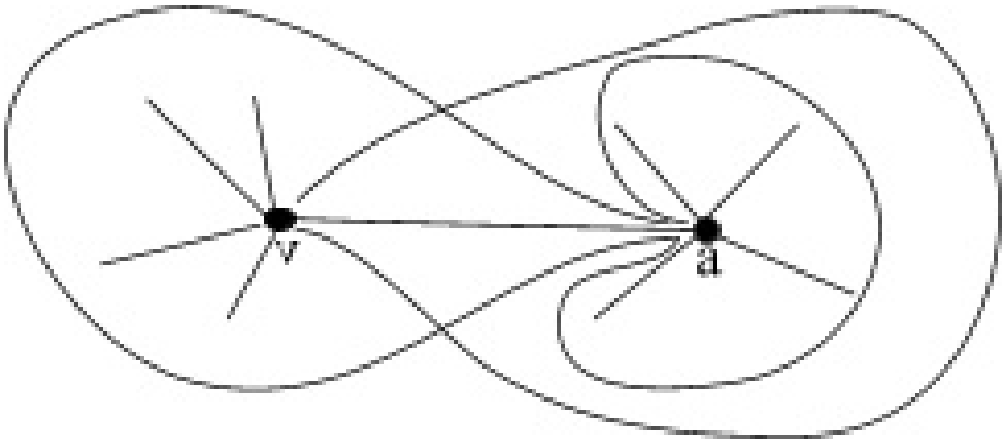


Рис. 2. Изображение графа для доказательства
Fig.2. Graph image for proof

2. Докажем, что если в дереве имеются две центроидные вершины, то они смежны. Предположим противное: пусть в дереве имеются две центроидные несмежные вершины v_1 и v_2. В дереве любые две вершины соединены единственной простой цепью. Так как v_1 и v_2 не смежны, имеется вершина a, лежащая между v_1 и v_2. Тогда нетрудно увидеть, что сумма размеров ветвей, примыкающих к вершине a должна быть меньше $n-1$, что противоречит (*).

Действительно, сумма размеров всех ветвей, примыкающих к вершине a, кроме ветви, включающей v_1, не более $\frac{n}{2}-1$, так как эта сумма есть часть размера одной ветви, примыкающей к v_1 (а именно ветви, включающей вершину a), уменьшенная по крайней мере на единицу. (Напомним, что вес каждой ветви при центроидной вершине не более $\frac{n}{2}$). Вес оставшейся ветви, примыкающей к вершине a и содержащей v_1, также не превосходит $\frac{n}{2}-1$, так как это часть ветви, идущей от вершины v_2 , содержащей вершину a. Значит, сумма размеров всех ветвей, примыкающих к вершине a, не превосходит $\frac{n}{2}-1+\frac{n}{2}-1=\frac{n}{2}$ — действительно, противоречие с (*).

Следствие. В произвольном дереве может быть только одна или только две смежные центроидных вершины. Действительно, три вершины, будучи попарно смежными, образовывали бы треугольник, но в дереве циклов нет.

Предложение 1. Любая вершина v дерева, вес которой не более $\frac{n}{2}$, является центроидной.

Доказательство. Предположим противное: v — не центроидная вершина, тогда в дереве имеется центроидная вершина c, вес её, по предположению, меньше веса вершины v: $w(c) < w(v) \leq \frac{n}{2}$. В дереве имеется единственный путь от c к v. Пусть r — последнее ребро этого пути. Рассмотрим ветвь при вершине c, содержащую вершину v. Вес этой ветви, как и любой ветви при вершине c, меньше $\frac{n}{2}$. Эта ветвь включает в себя все ветви при вершине v, кроме ветви, содержащей c. Таким образом, сумма весов всех ветвей при вершине v, кроме ветви, содержащей вершину c, имеет вес меньше чем $\frac{n}{2}-1$, так как хотя бы ребро r в эту сумму не входит (см. рис. 3). Рассмотрим теперь оставшуюся ветвь при вершине v, содержащую вершину c. Она имеет вес не более $w(v)$, то есть не более $\frac{n}{2}$. Тогда сумма весов всех ветвей при вершине v будет меньше $\frac{n}{2}+\frac{n}{2}-1$, то есть меньше $n-1$. Противоречие с (*).

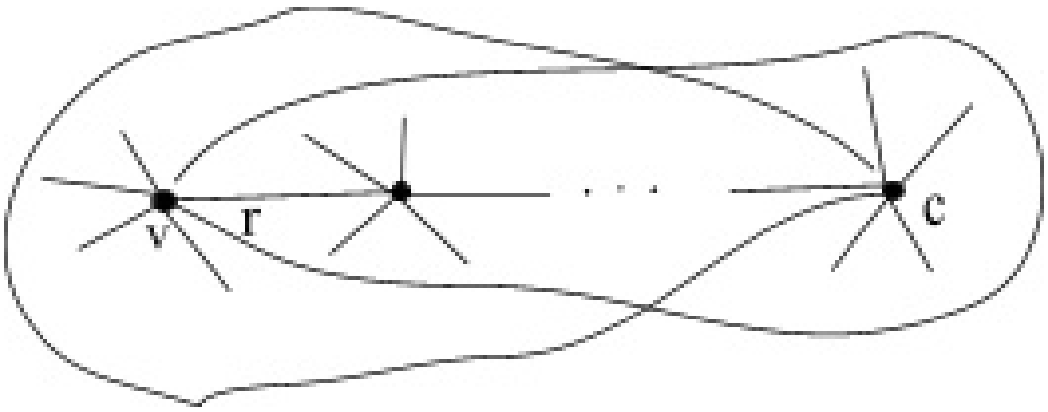


Рис. 3. Изображение графа для доказательства предложения 1
Fig. 3. Graph image for suggestion 1 proof.

Предложение 2. Если в дереве имеются две центроидные вершины c_1 и c_2, тогда порядок дерева $n(T)$ является чётным числом и $w(c_1)=w(c_2)=\frac{n}{2}$.

Доказательство. Пусть имеются две центроидные вершины v_1 и v_2. Они смежны в силу утверждения теоремы. Можно отметить, что ветвь при v_1, содержащая

v_2, имеет размер $\frac{n}{2}$. Действительно, её размер не более $\frac{n}{2}$ в силу теоремы, но он не может быть и меньше $\frac{n}{2}$, так как иначе сумма размеров остальных ветвей при вершине v_1, была бы более $\frac{n}{2} - 1$, а тогда ветвь при вершине v_2, содержащая v_1, имела бы вес больше, чем $\frac{n}{2} - 1 + 1 = \frac{n}{2}$, что противоречит теореме.

Предложение 3. Если в дереве имеется центроидная вершинav веса $\frac{n}{2}$, то имеется и другая центроидная вершина (естественно, того же веса и смежная с первой).

Для доказательства достаточно (аналогично предыдущим случаям) рассмотреть ветвь наибольшего веса при вершине v и взять в этой ветви вершину, смежную с v.

Отметим, что для деревьев чётного порядка утверждения, обратные к предложениям 2 и 3, вообще говоря, не обязательны.

Список литературы / References

[1] Евстигнеев В. А., Касьянов В. Н., *Теория графов. Алгоритмы обработки деревьев*, Наука, Новосибирск, 1994; [Yevstigneev V. A., Kas'anov V. N., *Teoriya grafov. Algoritmy obrabotki derev'ev*, Nauka, Novosibirsk, 1994, (in Russian).]

[2] Ловас Л., Пламмер М., *Прикладные задачи теории графов. Теория паросочетаний в математике, физике, химии*, Мир, М., 1998; In English: Lovasz L., Plummer M.D., *Matching Theory*, Akademiai Kiado, North Holland, Budapest, 1986.

Belov Y. A., Vovchok S. I., "Tree Centroid Properties Clarification", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 410–414.

DOI: 10.18255/1818-1015-2017-4-410-414

Abstract. The paper is devoted to the tree centroid properties clarification. Attention of the authors was attracted by the popular problem of (binary) partition of a graph. The solution is known only by brute force algorithm. It was found that for a "economical" partition of a tree it makes sense to consider partitions in the neighborhood of centroid vertices, the definition of which is presented. In the paper, we proposed proofs connected with the limitation of their weight. It is also proved that if there are two centroid vertices in a tree, they are adjacent. In what follows, it is noted that three such vertices can not be in the tree. The corresponding statements are made. According to the first one, any vertex of a tree with a certain restriction on its weight is centroid. According to one of the points of the second statement, if there are two centroid vertices in the tree, the order of the tree is an even number. The third statement says that if a tree has a centroid vertex of limited weight, there is another centroid vertex of the same weight and adjacent to the first one. To prove the propositions, we consider the branch of greatest weight with a centroid vertex and take in this branch another vertex adjacent to the centroid. In this paper, Jordan's theorem is used, three images are used in the presentation of the material.

Keywords: centroid, tree centroid

On the authors:
Yurii A. Belov, orcid.org/0002-0007-1896-0951, PhD,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: belov45@yandex.ru

Sergei I. Vovchok, orcid.org/0000-0003-0535-5786, graduate student,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: vof4ok@gmail.com

Моделирование и анализ информационных систем. Т. 24, № 4 (2017), с. 415–433
Modeling and Analysis of Information Systems. Vol. 24, No 4 (2017), pp. 415–433

DOI: 10.18255/1818-1015-2017-4-415-433

УДК 517.9

О задаче минимизации последовательных программ

Захаров В. А.[1], **Жайлауова Ш. Р.**

получена 17 июля 2017

Аннотация. Стандартные схемы программ — это одна из наиболее простых моделей последовательных императивных программ, предназначенная для решения задач оптимизации и верификации программ. Мы рассматриваем разрешимое отношение логико-термальной эквивалентности стандартных схем программ и задачу минимизации их размера при условии сохранения отношения логико-термальной эквивалентности. Нами доказано, что эта задача является алгоритмически разрешимой. Далее показано, что стандартные схемы программ с отношением логико-термальной эквивалентности и конечные детерминированные автоматы-преобразователи, работающие над полугруппами подстановок, взаимно транслируются друг в друга. Отсюда следует, что также разрешимы задачи проверки эквивалентности и минимизации для преобразователей указанного вида. Кроме того, на основе обнаруженной взаимосвязи выделен подкласс стандартных схем программ, минимизация которых осуществима за полиномиальное время при помощи ранее известных методов минимизации автоматов-преобразователей, работающих над полугруппами. В заключении приведен пример, свидетельствующий о том, что в общем случае задача минимизации автоматов-преобразователей над полугруппой подстановок может иметь несколько неизоморфных решений.

Ключевые слова: последовательная программа, автомат-преобразователь, минимизация, подстановка, полугруппа, эквивалентность

Для цитирования: Захаров В. А., Жайлауова Ш. Р., "О задаче минимизации последовательных программ", *Моделирование и анализ информационных систем*, **24**:4 (2017), 415–433.

Об авторах:
Захаров Владимир Анатольевич, orcid.org/0000-0002-3794-9565,
д-р физ.-мат. наук, ст. науч. сотр.,
Национальный исследовательский университет "Высшая школа экономики", факультет компьютерных наук,
ул. Мясницкая, 20, г. Москва, 101000 Россия, e-mail: zakh@cs.msu.ru

Жайлауова Шынар Рустамбековна, магистр,
Московский государственный университет им. М.В. Ломоносова, факультет ВМК,
Ленинские горы, д. 1, стр. 52, ГСП-1, Москва, 119991 Россия, e-mail: gulgaisha93@mail.ru

Благодарности:
[1]Работа выполнена при финансовой поддержке грантов РФФИ №16-01-00546 и №15-01-05742.

Введение

Теория схем программ — первая математическая теория, объектами изучения которой являются компьютерные программы (не путать с алгоритмами!), — возникла в конце 50-х годов с целью создания математических методов, облегчающих

разработку программ. Компьютеры того времени обладали очень ограниченными вычислительными ресурсами, и поэтому задача оптимизации программ по размеру, объему рабочей памяти и производительности была чрезвычайно актуальной. Часто отлаженную и протестированную программу нужно было модифицировать вручную, чтобы сократить ее размер или оптимизировать рабочую память. После проведения оптимизирующих преобразований измененная программа должна была вновь подвергаться тестовым испытаниям. Чтобы облегчить этот процесс, А.А. Ляпунов в статье [12] предложил следующий подход к решению задачи оптимизации программ:

1) создать математическую модель, в которой компьютерные программы могли быть представлены выражениями, подобными логическим или алгебраическим формулам;

2) определить поведение (семантику) моделей программ так, чтобы эквивалентные модели программ (т.е. модели, имеющие одинаковое поведение) соответствовали программам, вычисляющим одну и ту же функцию;

3) использовать для оптимизации программ только такие преобразования, которые сохраняют эквивалентность предложенных моделей программ.

Модели программ, предложенные в статье [12], были названы схемами программ по аналогии с терминами «схема формулы» или «схема правила вывода», используемыми в математической логике. Позднее Р.И. Подловченко предложила назвать модели программ, удовлетворяющие требованию п. 2, аппроксимирующими моделями [13].

Если использовать указанный подход, то для подтверждения корректности программной оптимизации можно не проводить повторного тестирования оптимизированной программы, а всего лишь проверить эквивалентность схемы исходной программы и схемы модифицированной программы. При наличии алгоритма проверки эквивалентности схем программ эту проверку можно автоматизировать, и поэтому на первый план была выдвинута задача создания алгоритмов проверки эквивалентности схем программ. Она была успешно решена Ю.И. Яновым в работе [14]. Достигнутый успех привлек к теории схем программ внимание со стороны математиков, занимавшихся решением задач системного программирования. Спустя короткое время на основе этой теории были предложены алгоритмы оптимизации линейных участков программ [10], рабочей памяти программ [2, 11]; эти и другие подобные алгоритмы были использованы в первом оптимизирующем трансляторе для языка программирования ALGOL-60 [16]. Первые положительные результаты исследования алгоритмических задач в теории схем программ показали, что с помощью этой теории можно создавать эффективные методы оптимизации программ.

Другое важное открытие в теории схем программ совершила Дж. Ратледж [22]; она показала, что схемы программ Ляпунова–Янова и конечные автоматы Рабина–Скотта можно взаимно транслировать друг в друга — вычисления схем программ можно представить как вычисления автоматов и наоборот. Отсюда следует, что разрешающие алгоритмы для задач теории автоматов можно перенести в теорию схем программ. В частности, многочисленные алгоритмы минимизации детерминированных конечных автоматов [26] можно использовать для сокращения размера

программ. На основании обнаруженной взаимосвязи теории схем программ и теории автоматов был предложен следующий метод построения алгоритмов оптимизации программ (см. [3]). Предположим, что имеется аппроксимирующая модель программ $\mathcal{P}$ с отношением эквивалентности $\sim_{\mathcal{P}}$ и мерой сложности $||\cdot||_{\mathcal{P}}$ на множестве схем программ. Предположим также, что удалось обнаружить класс автоматов $\mathcal{A}$ с отношением эквивалентности $\sim_{\mathcal{A}}$ и мерой сложности $||\cdot||_{\mathcal{A}}$, для которого существует эффективная взаимнооднозначная трансляция $\varphi : \mathcal{P} \to \mathcal{A}$, удовлетворяющая двум условиям:

1) для любой пары схем программ π_1, π_2 справедливо соотношение $\pi_1 \sim_{\mathcal{P}} \pi_2 \Leftrightarrow \varphi(\pi_1) \sim_{\mathcal{A}} \varphi(\pi_2)$;

2) для любой схемы программ π верно равенство $||\pi||_{\mathcal{P}} = ||\varphi(\pi)||_{\mathcal{A}}$.

При соблюдении этих условий любой алгоритм минимизации автоматов из класса $\mathcal{A}$ в метрике сложности $||\cdot||_{\mathcal{A}}$ можно легко адаптировать для решения задачи минимизации программ в метрике сложности $||\cdot||_{\mathcal{P}}$. Здесь теория автоматов предоставляет абстрактную модель вычислений, в которой удобно решать проблемы эквивалентности и минимизации, а теория схем программ — средство для приложения полученных результатов в решении задач анализа и оптимизации программ.

Помимо работы [22], описанный прием был успешно применен А.А. Летичевским в работе [1] для решения задачи оптимизации программ по быстродействию; в качестве схем программ использовались дискретные преобразователи Глушкова, обобщающие схемы программ Ляпунова–Янова. Достигнутый успех стимулировал поиск других взаимосвязей между теорией схем программ и теорией автоматов. В статье [19] было показано, что схемы последовательных программ с перестановочными операторами можно транслировать в детерминированные многоленточные автоматы, а в статье [17] было установлено, что подобная же трансляция имеет место для рекурсивных схем программ и детерминированных автоматов с магазинной памятью. Однако проблема эквивалентности для этих двух разновидностей автоматов оказалась твердым орешком — лишь два десятилетия спустя алгоритмическая разрешимость этой проблемы была доказана в статьях [18, 24] (вопрос о ее сложности открыт до сих пор). Большой интерес специалистов в теории схем программ, проявленный к проблеме эквивалентности для многоленточных и магазинных автоматов, оказал отрицательное влияние на развитие этой теории — были обойдены вниманием другие виды схем программ и классы автоматов, для которых осуществима описанная выше трансляция, не удалось построить практичных алгоритмов проверки эквивалентности и оптимизации схем программ. В итоге намеченные в статье [3] цели развития теории схем программ и ее применения в системном программировании не были своевременно достигнуты. Эти неудачи послужили одной из причин последующего упадка этой теории; более подробно об этом периоде развития теории схем программ можно прочесть в книге [7].

И хотя в настоящее время задачи, исследовавшиеся в теории схем программ в 70–80 гг., уже не относятся к числу наиболее актуальных, потенциал некоторых моделей и методов, созданных и развитых в рамках этой теории, еще далек от исчерпания. В частности, мы предлагаем исследовать еще одну сочетаемую пару «схема программ — автомат», пригодную для решения задачи оптимизации последовательных императивных программ. В качестве аппроксимирующей модели программ $\mathcal{P}$

рассматривается модель стандартных схем программ, впервые предложенная в статье [19], с отношением логико-термальной эквивалентности $\sim_{\mathcal{P}}$, впервые описанным в статье [8]. Выбор этой модели программ обусловлен тем, что

1) проблема логико-термальной эквивалентности стандартных схем программ разрешима за полиномиальное время (см. [4, 23]);

2) семантику стандартных схем программ можно описать при помощи конечных подстановок, образующих полугруппу, которая обладает многими полезными алгебраическими свойствами и хорошо подходит для решения задачи минимизации автоматов.

Как будет показано далее в данной статье, выбранной модели программ соответствует класс конечных автоматов-преобразователей над полугруппами, введенный в статье [28]. Для автоматов этого вида эффективно разрешимы проблема проверки эквивалентности [28] и проблема минимизации [6].

В данной статье представлены три основных результата исследования проблемы минимизации стандартных схем программ. Во-первых, показано, что задача минимизации стандартных схем программ с логико-термальной эквивалентностью алгоритмически разрешима и сводится к задаче минимизации конечных автоматов-преобразователей над полугруппой конечных подстановок. Во-вторых, выделена подполугруппа консервативных ортогональных подстановок, для которой эффективно разрешима задача минимизации автоматов-преобразователей при помощи метода минимизации, описанного в статье [6]. В-третьих, установлено, что в общем случае задача минимизации конечных автоматов-преобразователей над полугруппой произвольных конечных подстановок не имеет однозначного решения подобно тому, как не имеет однозначного решения задача минимизации дизъюнктивных нормальных форм в булевой алгебре.

Статья организована следующим образом. Поскольку описание модели стандартных схем программ опирается на понятие подстановки, в следующем разделе приводятся основные понятия алгебры подстановок. Далее следует описание устройства стандартных схем программ и логико-термальной эквивалентности. В следующем разделе доказывается алгоритмическая разрешимость задачи минимизации стандартных схем программ с отношением логико-термальной эквивалентности. Затем дается описание модели вычислений конечных автоматов-преобразователей над полугруппами и показывается, что классы стандартных схем программ конечных детерминированных автоматов-преобразователей над полугруппой подстановок взаимно транслируются друг в друга. Отсюда следует, что задачи проверки эквивалентности и минимизации для преобразователей указанного вида также разрешимы. В следующем разделе выделена подполугруппа консервативных ортогональных подстановок и показано, что задача минимизации автоматов-преобразователей, работающих над этой подполугруппой, имеет однозначное решение, вычислимое за полиномиальное время. Отсюда следует, что для класса схем программ, соответствующих преобразователям указанного вида, существует эффективное решение задачи минимизации. В заключении приводится пример автомата-преобразователя над полугруппой подстановок, для которого задача минимизации имеет несколько неизоморфных решений. Этот пример призван обозначить некоторые трудности,

с которыми придется столкнуться при изучении сложности задачи минимизации стандартных схем программ.

1. Алгебра подстановок

Для произвольных заданных множеств переменных X, функциональных символов $\mathcal{F}$ и предикатных символов $\mathcal{P}$ будем использовать запись $Term(X, \mathcal{F})$ для обозначения множества термов, построенных из переменных и функциональных символов, а запись $Atom(X, \mathcal{F}, \mathcal{P})$ — для обозначения множества атомарных формул (атомов), построенных из предикатных символов и термов. Для каждого терма или атома T обозначим записью Var_T множество переменных, входящих в выражение T. Высота терма или атома определяется обычным образом как высота дерева в древесном представлении этих выражений.

Пусть X и Y — два конечных множества переменных. Подстановкой назовем всякое отображение $\theta : X \to Term(Y, \mathcal{F})$, сопоставляющее каждой переменной из X некоторый терм из $Term(Y, \mathcal{F})$. Множество всех таких подстановок условимся обозначать записью $Subst(X, Y, \mathcal{F})$. Если $X = \{x_1, \ldots, x_n\}$ — конечное множество переменных и $\theta(x_i) = t_i$ для всех i, $1 \le i \le n$, то такая подстановка называется конечной и определяется списком пар $\{x_1/t_1, \ldots, x_n/t_n\}$. Запись Var_θ будет использоваться для обозначения множества переменных $\bigcup\limits_{i=1}^{n} Var_{t_i}$, входящих в состав всех термов t_i подстановки θ. Результатом применения подстановки θ к терму t является терм $t\theta$, получающийся одновременной заменой в терме t каждого вхождения любой переменной x_i термом t_i. Композиция $\eta \circ \theta$ подстановок $\theta \in Subst(X, Y, \mathcal{F})$ и $\eta \in Subst(Y, Z, \mathcal{F})$ — это такая подстановка из множества $Subst(X, Z, \mathcal{F})$ (ее традиционно обозначают записью $\theta\eta$), которая определяется равенством $\theta\eta(x) = \theta(x)\eta$ для каждой переменной x, $x \in X$. Множество подстановок $Subst(X, X, \mathcal{F})$ с операцией композиции образует полугруппу, в которой нейтральным элементом служит тождественная подстановка $\varepsilon = \{x_1/x_1, \ldots, x_n/x_n\}$.

На множестве подстановок $Subst(X, X, \mathcal{F})$ можно определить отношения предпорядка $\preceq$. Для пары подстановок θ_1, θ_2 отношение $\theta_1 \preceq \theta_2$ выполняется, если есть такая подстановка η, что $\theta_2 = \theta_1\eta$. В случае выполнимости отношения $\theta_1 \preceq \theta_2$ подстановку θ_1 называют прототипом подстановки θ_2, а подстановку θ_2 — примером подстановки θ_1. Квазиупорядоченное множество $(Subst(X, X, \mathcal{F}), \preceq)$ образует квазирешетку, наименьшим элементом которой является тождественная подстановка ε. Эта квазирешетка удовлетворяет условию обрыва убывающих цепей. Более подробные сведения об алгебре подстановок могут быть почерпнуты из работы [15].

2. Стандартные схемы программ

В качестве математической модели последовательных императивных программ мы используем стандартные схемы программ; они были введены в статье [19] и получили такое название в статье [3]. Подробное описание этой разновидности схем программ приведено в монографии [9]. Мы опишем вкратце устройство этой модели, опираясь на понятия теории графов и алгебры подстановок. Пусть задано

некоторое конечное множество переменных X. Стандартная схема программ π представляет собой размеченный ориентированный граф; его вершины будем называть узлами. Каждому узлу v графа π приписана атомарная формула A_v из множества $Atom(X, \mathcal{F}, \mathcal{P})$. Один из узлов v_0 графа π особо выделен в качестве входа в программу, а некоторые другие узлы $u_1, \ldots, u_k$ являются выходами из программы. Из каждого невыходного узла исходят не более двух дуг, помеченных разными символами из множества $\{0, 1\}$. Кроме того, каждой дуге, ведущей в графе π из узла u в узел v, приписана подстановка θ_{uv} из множества $Subst(X, X, \mathcal{F})$. Из выходов схемы не исходит ни одной дуги. Предполагается также, что через каждый узел графа π проходит некоторый маршрут, ведущий из входа схемы в один из ее выходов. Размером $||\pi||$ схемы программ π назовем число ее узлов.

Узлы графа π соответствуют точкам программы, в которых проводится проверка условий и ветвление потока управления программы. Атомарные формулы, приписанные узлам, соответствуют условиям ветвления, а дуги графа π — линейным участкам программы. Если линейный участок между точками ветвления u и v представляет собой последовательность операторов присваивания $x_{i_1} := t_1; \ldots; x_{i_m} := t_m$, то соответствующей дуге приписана подстановка $\theta_{uv} = \{x_{i_m}/t_m\} \cdots \{x_{i_1}/t_1\}$, представляющая собой композицию односвязочных подстановок для всех операторов этого линейного участка.

Пусть задан некоторый маршрут в схеме программ π из узла v_0 в узел v_m

$$w = v_0 \xrightarrow{\sigma_1,\theta_1} v_1 \xrightarrow{\sigma_2,\theta_2} \cdots \xrightarrow{\sigma_m,\theta_m} v_m \ . \tag{1}$$

Если v_0 — вход схемы π, то маршрут w называется *начальным*, а если v_m — один из выходов схемы π, то маршрут w называется *финальным*. Маршрут w, который является одновременно начальным и финальным, называется *полным маршрутом*. Множество всех полных маршрутов в схеме программ π обозначим записью $Path(\pi)$. Записью θ_w будем обозначать композицию $\theta_1 \circ \cdots \circ \theta_m$ всех подстановок, приписанных дугам этого маршрута.

Двоичная последовательность $lh(w) = \sigma_1, \sigma_2, \ldots, \sigma_m$ называется *логической историей* маршрута (1); вместе с узлом v_0 она однозначно определяет маршрут w в схеме программ π. Последовательность пар

$$lth(w) = (A_{v_0}\eta_0, \sigma_1), (A_{v_1}\eta_1, \sigma_2), \ldots, (A_{v_{m-1}}\eta_{m-1}, \sigma_m), (A_{v_m}\eta_m, 1),$$

где $\eta_0 = \varepsilon$ и $\eta_i = \theta_i\eta_{i-1}$ для каждого i, $1 \le i \le m$, называется *логико-термальной историей* (л-т историей) маршрута w. *Детерминантом* $Det(\pi)$ схемы программ π называется множество л-т историй всех полных маршрутов в схеме программ π, т. е. $Det(\pi) = \{lth(w) \ : \ w \in Path(\pi)\}$. Две схемы программы π_1 и π_2 считаются *логико-термально (л-т) эквивалентными* (отношение обозначается записью $\pi_1 \sim_{lt} \pi_2$), если справедливо равенство $Det(\pi_1) = Det(\pi_2)$.

Схема программ π' называется *S-минимальной*, если неравенство $||\pi'|| \le ||\pi||$ выполняется для любой схемы программ π, л-т эквивалентной схеме π'. Задача минимизации схем программ состоит в том, чтобы для произвольной заданной схемы программ π построить л-т эквивалентную ей минимальную схему программ π'.

Функциональная эквивалентность стандартных схем программ определяется на множестве всех интерпретаций сигнатуры $\Sigma = (X, \mathcal{F}, \mathcal{P})$. Для каждой интерпре-

тации I в схеме программ π выбирается единственный (если он существует) полный маршрут $w(\pi, I)$, который в каждом узле v_i, $0 \le i < m$, удовлетворяет условию $I \models A_{v_i}\eta_i = \sigma_{i+1}$. Тогда атомарная формула $A_{v_m}\eta_m$ объявляется значением $\pi(I)$ схемы программ π в интерпретации I. Две схемы программ π_1 и π_2 считаются функционально эквивалентными, если для любой интерпретации I выполняется соотношение $I \models \pi_1(I) = \pi_2(I)$.

В статье [19] было показано, что отношение функциональной эквивалентности алгоритмически неразрешимо, но, как было установлено в работе [8], оно аппроксимируется разрешимым отношением л-т эквивалентности. Таким образом, алгоритмы минимизации стандартных схем программ с л-т эквивалентностью можно корректно использовать для оптимизации последовательных императивных программ. Далее мы покажем, что указанная задача минимизации стандартных схем программ с л-т эквивалентностью алгоритмически разрешима и, кроме того, сводима к задаче минимизации конечных автоматов-преобразователей над полугруппой подстановок. Для этого целесообразно обратиться к следующему простому критерию л-т эквивалентности схем программ.

Сокращенной логико-термальной историей маршрута (1) в схеме программ π назовем пару $rlth(w) = (lh(w), A_{v_m}\theta_w)$, состоящую из логической истории $lh(w) = \sigma_1, \sigma_2, \ldots, \sigma_m$ этого маршрута и примера атомарной формулы A_{v_m}, приписанной последнему узлу v_m маршрута w, которая специализирована композицией подстановок $\theta_1 \circ \cdots \circ \theta_m$, приписанных всем дугам этого маршрута. *Сокращенным детерминантом* схемы программ π назовем множество

$$RDet(\pi) = \{rlth(w) \ : \ w \text{ — начальный маршрут в схеме } \pi\}$$

сокращенных л-т историй всех начальных маршрутов в схеме программ π.

Утверждение 1. *Для любой пары схем программ π_1 и π_2, через каждый узел которых проходит хотя бы один полный маршрут, справедливо соотношение*

$$\pi_1 \sim_{lt} \pi_2 \ \Leftrightarrow \ RDet(\pi_1) = RDet(\pi_2) \ .$$

Доказательство. Достаточно заметить, что в случае, когда через каждый узел схемы программ проходит хотя бы один полный маршрут, каждой л-т истории $lth(w) = (A_{v_0}\eta_0, \sigma_1), (A_{v_1}\eta_1, \sigma_2), \ldots, (A_{v_{m-1}}\eta_{m-1}, \sigma_m), (A_{v_m}\eta_m, 1)$ взаимнооднозначно соответствует набор, состоящий из $m + 1$ сокращенных л-т историй $rlth(w_i) = (\sigma_1, \ldots, \sigma_i, A_{v_i}\eta_i), 0 \le i \le m$. □

3. Минимизация стандартных схем программ

Как уже было упомянуто, отношение л-т эквивалентности стандартных схем программ разрешимо (см. [4, 8, 23]). Если в модели вычислений разрешима проблема эквивалентности, то для решения задачи минимизации вычислительных систем (программ) в этой модели можно использовать переборный алгоритм: для минимизации заданной программы π нужно перебирать в порядке возрастания сложности все программы π' и проверять эквивалентность $\pi \sim \pi'$. Если для каждого значения

сложности k существует лишь конечное множество программ π', имеющих сложность $||\pi'|| = k$, то мы получаем готовый (хотя и далеко не самый эффективный) алгоритм минимизации программ в рассматриваемой модели вычислений.

К сожалению, если мерой сложности стандартной схемы программ π является число ее узлов, то для каждого натурального $k, k > 1$, существует бесконечно много схем программ сложности k. Причина тому — неограниченность множества подстановок, которыми можно помечать дуги программы. Чтобы преодолеть это препятствие, можно попытаться для каждой схемы программы π выделить такое конечное подмножество подстановок, которого достаточно для построения минимальной схемы программ, эквивалентной схеме π.

Пусть задана схема программ π с множеством узлов V, дуги которой помечены конечными подстановками из множества $Subst(X, X, \mathcal{F})$, где $|X| = n$. Мы будем полагать, что каждый узел схемы π лежит на некотором маршруте из входа схемы в один из ее выходов. Как можно легко заметить, не удовлетворяющие этому требованию узлы можно удалить вместе с инцидентными им дугами, сохранив при этом детерминант $Det(\pi)$ схемы. Рассмотрим произвольный узел v в этой схеме и переменную x. Переменная x считается *существенной* в узле v схемы π, если из этого узла исходит хотя бы один такой маршрут w, сокращенная л-т история $rlth(w) = (lh(w), A)$ которого удовлетворяет условию $x \in Var_A$. В противном случае переменную x будем называть *несущественной* в узле v.

Как показывает следующее утверждение, для несущественных переменных вид подставляемых термов в соответствующих связках тех подстановок, которыми помечены дуги схемы программы, не имеет значения.

Утверждение 2. *Предположим, что схема программ π содержит дугу $u \xrightarrow{\sigma,\theta} v$, и подстановка θ' отличается от подстановки θ только связкой для переменной x, т.е. $\theta(y) = \theta'(y)$ для любой переменной $y \in X \setminus \{x\}$. Предположим, что схема программ π' получена из π заменой дуги $u \xrightarrow{\sigma,\theta} v$ на дугу $u \xrightarrow{\sigma,\theta'} v$. Тогда если переменная x является несущественной в узле v, то $\pi \sim \pi'$.*

Доказательство. Рассмотрим произвольный маршрут $w = w_1; u \xrightarrow{\sigma,\theta} v; w_2$, проходящий в схеме π через выделенную дугу, и сокращенные л-т истории $rlth(w) = (lh(w), A)$ и $rlth(w_2) = (lh(w_2), A_2)$. Из определения л-т истории маршрутов в схеме программ следует, что $A = A_2\theta\theta_{w_1}$. Поскольку переменная x несущественна в узле v, атом A_2 не содержит вхождений переменной x. А так как подстановки θ и θ' отличаются только связкой для переменной x, верно равенство $A_2\theta = A_2\theta'$. Таким образом, маршрут $w' = w_1; u \xrightarrow{\sigma,\theta'} v; w_2$ будет иметь ту же самую сокращенную л-т историю, что и маршрут w. Значит, $RDet(\pi) = RDet(\pi')$, и поэтому согласно утверждению 1 схемы π и π' л-т эквивалентны. □

Свойство существенности переменной x в узле v схемы программ π можно описать иным способом при помощи графа зависимости переменных в программе, использующегося при анализе потоков данных (см. [21]). Рассмотрим ориентированный граф D_π, вершинами которого служат все пары (x, v), $x \in X$, $v \in V$. В этом графе из вершины (y, u) в вершину (x, v) исходит дуга в том и только том случае, если в схеме программ π есть такая дуга $u \xrightarrow{\sigma,\theta} v$, что $y \in Var_{\theta(x)}$. В определен-

ном таким образом графе зависимости переменных выделим множество вершин $U_\pi = \{(x, v) : x \in Var_{A_v}\}$.

Утверждение 3. *Переменная x существенна в узле v схемы программ π тогда и только тогда, когда в графе зависимости переменных D_π существует путь из вершины (x, v) в одну из вершин множества U_π.*

Доказательство. Утверждения такого рода хорошо известны в теории статического анализа программ. В данном случае оно легко доказывается индукцией по длине маршрута в схеме программ π, подтверждающего существенность переменной x в узле v (необходимое условие), и по длине пути в графе зависимости D_π из вершины (x, v) в вершину из множества U_π (достаточное условие). □

Используя граф зависимости, покажем, как оценить высоту термов, присутствующих в разметках дуг в эквивалентных схемах программ.

Утверждение 4. *Предположим, что схемы программ π_1 и π_2 над множеством переменных X, где $|X| = n$, обладают следующими свойствами:*

1. *$||\pi_1|| = m$, и глубина всех термов в подстановках, которыми помечены дуги в схеме π_1, не превосходит некоторого натурального числа k;*
2. *$||\pi_2|| \le m$, и в схеме π_2 есть такая дуга и $\xrightarrow{\sigma,\eta} v$, что переменная x является существенной в узле v, а высота терма $\eta(x)$ превосходит величину $km(n+1)$.*

Тогда $\pi_1 \not\sim_{lt} \pi_2$.

Доказательство. Согласно утверждению 3 существенность переменной x в узле v влечет за собой существование пути в графе зависимости переменных D_{π_2} из вершины (x, v) в одну из вершин множества U_{π_2}. Поскольку $||\pi_2|| \le m$ и $|X| = n$, кратчайший путь такого вида имеет длину, не превосходящую величины mn. Тогда в схеме программ π_2 существует соответствующий этому пути маршрут w'' такой же длины из узла v в некоторый узел v'', который подтверждает существенность переменной x в узле v. Последнее означает, что $rlth(w'') = (lh(w''), A)$ и $x \in Var_A$. Рассмотрим в схеме программ π_2 кратчайший маршрут w' из входа схемы в вершину u. Длина этого маршрута не превосходит числа m. Тогда начальный маршрут $w_2 = w', u \xrightarrow{\sigma,\eta} v, w''$ имеет длину, меньшую величины $m(n+1)$, и сокращенную л-т историю $rlth(w_2) = (lh(w_2), A\eta\theta_{w'})$. Поскольку $x \in Var_A$ и высота терма $\eta(x)$ превосходит величину $km(n+1)$, высота атома $A\eta\theta_{w'}$ также превосходит эту величину.

Обратимся теперь к схеме программ π_1 и рассмотрим в ней произвольный начальный маршрут w_1, длина которого не превосходит величины $m(n+1)$. Сокращенная л-т история этого маршрута имеет вид $rlth(w_1) = (lt(w_1), B\theta_{w_1})$, причем подстановка θ_{w_1} представляет собой композицию не более чем $m(n+1)$ подстановок, в которых все подставляемые термы имеют высоту, не превосходящую числа k. Значит, высота атома $B\theta_{w_1}$ не превосходит величины $km(n+1)$. Отсюда следует, что $rlth(w_2) \notin RDet(\pi_1)$. Согласно утверждению 1 это означает, что $\pi_1 \not\sim_{lt} \pi_2$. □

На основе приведенных выше утверждений может быть доказана

Теорема 1. *Проблема минимизации стандартных схем программ алгоритмически разрешима.*

Доказательство. Предположим, что требуется минимизировать схему программ π размера $||\pi|| = m$ над множеством переменных X, где $|X| = n$, причем все термы в подстановках, которыми помечены дуги этой схемы, имеют глубину, не превосходящую некоторого натурального числа k. Тогда в минимальной схеме программ π_0, л-т эквивалентной схеме программ π, для каждой дуги $u \xrightarrow{\sigma,\eta} v$ и для каждой переменной x верно одно из двух:

- если переменная x несущественна в узле v, то согласно утверждению 2 терм $\eta(x)$ можно задавать произвольно, сохраняя при этом л-т эквивалентность схемы программ; поэтому можно полагать, что в этом случае $\eta(x) = x$;
- если переменная x существенна в узле v, то согласно утверждению 3 высота терма $\eta(x)$ не превосходит величины $km(n+1)$.

Поскольку проблема л-т эквивалентности стандартных схем программ разрешима, поиск минимальной схемы программ π_0 можно проводить полным перебором по конечной совокупности всех схем программ, размер которых меньше числа m, и при этом таких, что во всех подстановках, приписанных дугам схемы, все термы имеют высоту, не превосходящую величины $km(n+1)$. □

Конечно, такой переборный алгоритм минимизации совершенно непрактичный — он имеет, по меньшей мере, тройную экспоненциальную сложность. Далее мы обсудим перспективы создания более эффективных способов минимизации стандартных схем программ при помощи методов минимизации других систем вычислений.

4. Автоматы-преобразователи над полугруппами

Пусть заданы два конечных множества $\mathcal{C}$ и $\mathcal{A}$. Элементы множества $\mathcal{C}$ будем называть *входными сигналами*. Конечные последовательности входных сигналов (слова в алфавите $\mathcal{C}$) будем называть *потоками сигналов*. Обозначим записью $\mathcal{C}^*$ множество всевозможных потоков сигналов. Элементы множества $\mathcal{A}$ будем называть *простыми действиями*. Конечные последовательности простых действий, представляющие собой слова в алфавите $\mathcal{A}$, будем называть *составными действиями*. Рассмотрим полугруппу $(S, e, \circ)$, порожденную множеством простых действий $\mathcal{A}$, в которой e обозначает нейтральный элемент, а $\circ$ — полугрупповую операцию. Элементы полугруппы S назовем *состояниями данных*. Применив простое действие a к состоянию данных s, получаем результат $s \circ a$. Составное действие $h = a_1 a_2 \dots a_k$ представляет собой композицию $[h]_S = a_1 \circ a_2 \circ \dots \circ a_k$ простых действий системы. Индекс S может быть опущен, если из контекста понятно, о какой полугруппе идет речь.

Детерминированный *автомат-преобразователь* с конечным числом состояний над множеством сигналов $\mathcal{C}$ и множеством базовых действий $\mathcal{A}$ — это размеченная система переходов $\Pi = (\mathcal{C}, \mathcal{A}, Q, q_0, F, T, h_0, E)$, состоящая из

- конечного множества *состояний управления* Q,
- *начального состояния* $q_0 \in Q$,
- множества *состояний выхода* $F \subseteq Q$,
- *функции переходов* $T : Q \times \mathcal{C} \to Q \times \mathcal{A}^*$,
- *инициализирующего действия* $h_0 \in \mathcal{A}^*$,
- *функции финализации* $E : F \to \mathcal{A}^*$.

Каждая четверка (q, c, q', h), удовлетворяющая равенству $T(q, c) = (q', h)$, называется *переходом* и обозначается записью $q \xrightarrow{c,h} q'$. *Размером* $||\Pi||$ автомата-преобразователя Π называется число $|Q|$ его состояний управления.

Прогоном автомата-преобразователя Π на потоке сигналов $\alpha = c_1 c_2 \dots c_n$ из состояния управления q называется последовательность переходов

$$q \xrightarrow{c_1,h_1} q_1 \xrightarrow{c_2,h_2} q_2 \xrightarrow{c_3,h_3} \cdots \xrightarrow{c_m,h_m} q' \ , \tag{2}$$

которую будем обозначать записью $q \xrightarrow{\alpha,h}_* q'$, где $h = h_1 h_2 \dots h_m$. Если состояние управления q является инициальным, то прогон также называется *инициальным*, а если $q' \in F$, то прогон называется *финальным*. Прогон, являющийся одновременно инициальным и финальным, называется *полным*. Для полного прогона $q \xrightarrow{\alpha,h}_* q'$ автомата-преобразователя Π, действия которого интерпретируются в полугруппе S, состояние данных $[h_0 h E(q')]_S$ считается *результатом* этого прогона. Поведение автомата-преобразователя Π, характеризуется частичной функцией $\Pi : \mathcal{C}^* \to S$; ее значения для потока сигналов α определяются соотношением

$$\Pi(\alpha) = \begin{cases} [h_0 h E(q')], \text{ если преобразователь } \Pi \text{ имеет полный прогон } q \xrightarrow{w,h}_* q', \\ \text{не определено, в противном случае.} \end{cases}$$

Для заданной полугруппы $(S, e, \circ)$ преобразователи Π_1 и Π_2 называются *S-эквивалентными*, если равенство $\Pi_1(\alpha) = \Pi_2(\alpha)$ выполняется для любого потока сигналов α. Отношение S-эквивалентности условимся обозначать записью $\Pi_1 \sim_S \Pi_2$. Автомат-преобразователь Π' называется *S-минимальным*, если неравенство $||\Pi'|| \leq ||\Pi||$ выполняется для любого преобразователя Π, S-эквивалентного преобразователю Π'. Задача минимизации автоматов-преобразователей над полугруппой S состоит в том, чтобы для произвольного заданного преобразователя Π построить S-эквивалентный ему S-минимальный преобразователь Π'.

Взаимосвязь между схемами программ и автоматами-преобразователями обеспечивается следующими двумя утверждениями.

Утверждение 5. *Существует такая эффективная трансляция φ стандартных схем программ в автоматы-преобразователи, работающие над полугруппой подстановок, которая удовлетворяет условиям:*

1) *для любой пары схем программ π' и π'' верно соотношение $\pi' \sim_{lt} \pi'' \ \Leftrightarrow \varphi(\pi') \sim_{Subst} \varphi(\pi'')$;*

2) *для любой схемы программ π верно равенство $||\pi|| = ||\varphi(\pi)||$.*

Доказательство. Отображение φ определяется следующим образом. Предположим, что схема программ π сигнатуры $\Sigma = (X, \mathcal{F}, \mathcal{P})$ содержит множество узлов V, и в том числе входной узел v_0. Не ограничивая общности, будем полагать, что каждый узел схемы лежит на некотором маршруте из входа схемы в один из ее выходов. Введем в рассмотрение n новых 1-местных функциональных символов $f_1, \dots, f_n$, где $n = |X|$.

Автомат-преобразователь $\Pi = \varphi(\pi)$ работает над множеством входных сигналов $\mathcal{C} = \{0, 1\}$ и множеством состояний данных $Subst(X, X, \mathcal{F} \cup \{f_1, \dots, f_n\} \cup \mathcal{A})$, имеет множество состояний управления V, совпадающее с множеством состояний выхода,

и начальное состояние v_0. Каждая дуга $u \xrightarrow{\sigma,\theta} v$ в схеме программ π становится переходом в преобразователе $\varphi(\Pi)$. Инициализирующим действием служит тождественная подстановка ε. Функция финализации приписывает каждому состоянию управления v подстановку $\eta_v = \{x_1/f_1(A_v), \ldots, x_n/f_n(A_v)\}$.

Из описания преобразователя Π видно, что между множеством начальных маршрутов в схеме π и множеством полных прогонов в автомате-преобразователе Π существует взаимно-однозначное соответствие: каждому начальному маршруту w с сокращенной л-т историей $rlth(w) = (\alpha, A)$ соответствует полный прогон на потоке сигналов α с результатом $\Pi(\alpha) = \{x_1/f_1(A), \ldots, x_n/f_n(A)\}$. Поэтому из утверждения 1 следует справедливость доказываемого утверждения. □

Утверждение 6. *Существует такая эффективная трансляция ψ автоматов-преобразователей, работающих над множеством входных сигналов $\mathcal{C} = \{0, 1\}$ и полугруппой подстановок, в стандартные схемы программ, которая удовлетворяет следующим условиям:*

1) *для любой пары автоматов-преобразователей Π' и Π'' верно соотношение $\Pi' \sim_{Subst} \Pi'' \Leftrightarrow \psi(\Pi') \sim_{lt} \psi(\Pi'')$;*

2) *для любого автомата-преобразователя Π верно равенство $|\Pi| = |\psi(\Pi)|$.*

Доказательство. Пусть задана полугруппа подстановок $Subst(X, X, \mathcal{F})$ и пусть R — некоторый n-местный предикатный символ, где $n = |X|$, а P — некоторый 0-местный предикатный символ. Рассмотрим произвольный детерминированный автомат-преобразователь $\Pi = (\mathcal{C}, Subst(X, X, \mathcal{F}), Q, q_0, F, T, h_0, E)$, причем, не ограничивая общности, будем полагать, что $h_0 = \varepsilon$, а из состояний выхода не исходит ни одного перехода. Стандартная схема программ $\pi = \psi(\Pi)$ устроена так. Ее узлами являются все состояния управления преобразователя Π, входом — начальное состояние q_0, а выходами — все состояния выхода преобразователя Π. Каждый переход $q' \xrightarrow{\sigma,\theta} q''$ в преобразователе Π становится дугой в схеме программ π. Каждому выходу q, $q \in F$, приписана атомарная формула $A_q = R(x_1, \ldots, x_n)E(q)$, а всем остальным узлам схемы приписан предикат P.

Из описания схемы программ π видно, что между множеством полных прогонов в автомате-преобразователе Π и множеством полных маршрутов в схеме π существует взаимно-однозначное соответствие: каждому полному прогону на потоке сигналов $\alpha = \sigma_1 \ldots \sigma_m$, завершающемуся с результатом η, соответствует полный маршрут в схеме π с л-т историей $lth(w) = (P, \sigma_1), \ldots, (P, \sigma_m), (R(x_1, \ldots, x_n)\eta, 1)$. Отсюда следует справедливость доказываемого утверждения. □

Утверждения 5 и 6 обеспечивают переносимость методов и результатов, полученных в одной из теорий — теории схем программ или теории автоматов-преобразователей над полугруппами, — в другую теорию. Например, из утверждения 6 и ранее полученных результатов о разрешимости за полиномиальное время задачи проверки л-т эквивалентности схем программ (см. [4, 23]) следует

Теорема 2. *Проблема эквивалентности конечных детерминированных автоматов-преобразователей, работающих над полугруппой подстановок, разрешима за полиномиальное время.*

Что касается задачи минимизации, то вопрос о ее сложности для схем программ остается открытым; поэтому на основании теоремы 1 и утверждения 5 справедливо более слабое утверждение

Теорема 3. *Проблема минимизации конечных детерминированных автоматов-преобразователей, работающих над полугруппой подстановок, разрешима.*

В следующем разделе мы покажем, что утверждение 5 о транслируемости автоматов-преобразователей в схемы программ вкупе с эффективными методами минимизации автоматов-преобразователей над полугруппами дает эффективное решение задачи минимизации для некоторых классов схем программ.

5. Минимизация автоматов-преобразователей над полугруппой подстановок

Первый алгоритм минимизации конечных детерминированных автоматов-преобразователей над свободной полугруппой слов был описан в статье [20]. В работе [5] была исследована задача минимизации автоматов-преобразователей, работающих над группами; было показано, что если множество состояний данных S образует группу с разрешимой проблемой тождеств, то задача минимизации автоматов-преобразователей имеет однозначное решение, которое может быть эффективно вычислено. Затем в статье [6] было получено решение задачи минимизации автоматов-преобразователей над упорядоченными полугруппами. Поскольку полугруппа подстановок $Subst(X, X, F)$ является квазирешеткой, метод, предложенный в этой статье, применим для решения задачи минимизации автоматов-преобразователей, работающих над $Subst(X, X, F)$. Рассмотрим требования, предъявляемые в статье [6] к полугруппе S, и оценим, в какой мере полугруппа подстановок $Subst(X, X, F)$ удовлетворяет им.

На полугруппе S определим бинарное отношение $\preceq_S$ следующим образом: для любой пары элементов s_1, s_2 отношение $s_1 \preceq_S s_2$ выполняется тогда и только тогда, когда для некоторого элемента s имеет место равенство $s_1 \circ s = s_2$. Полугруппа называется *упорядоченной*, если $\preceq_S$ является отношением частичного порядка на множестве S. Первое требование, предъявляемое к рассматриваемой полугруппе, таково.

Req1: Частично упорядоченное множество $(S, \preceq)$ является фундированной решеткой, в которой для каждой пары элементов $[h_1]$ и $[h_2]$ эффективно вычислима их точная нижняя грань.

Второе требование касается разрешимости линейных уравнений в рассматриваемой полугруппе $(S, e, \circ)$.

Req2: Существует алгоритм решения уравнений вида $[g] \circ X = [h]$ для любой заданной пары действий $g, h \in \mathcal{A}^*$.

Последнее требование свойства сократимости. Полугруппа называется *левосократимой*, если всякое уравнение вида $s \circ X = \widehat{s}$ имеет не более одного решения, т.е. соотношение $s \circ s' = s \circ s'' \;\Rightarrow\; s' = s''$ верно для любой тройки s, s', s''.

Req3: $(S, e, \circ)$ — левосократимая полугруппа.

Полугруппа подстановок $Subst(X, X, F)$ удовлетворяет требованию **Req1** — она является фундированной квазирешеткой с отношением предпорядка $\preceq$, в которой точная верхняя грань подстановок вычислима при помощи процедуры унификации, а точная нижняя грань вычислима при помощи процедуры антиунификации (устройство этой квазирешетки подробно описано в [15]). Как известно, обе эти процедуры можно выполнить за полиномиальное относительно размера обрабатываемых подстановок время. Эквивалентные подстановки в квазиупорядоченном множестве $(Subst(X, X, F), \preceq)$ отличаются лишь наименованием переменных в подставляемых термах; это обстоятельство не оказывает существенного влияния на алгоритм минимизации и обоснование его корректности, приведенные в статье [6].

Очевидный способ решения уравнений вида $\theta \circ X = \eta$ в алгебре подстановок — это простой перебор всех подстановок, в которых высота подставляемых термов не превосходит максимальной высоты термов в подстановке η. Однако можно заметить, что эта задача полиномиально сводится к проблеме проверки вхождения одного терма в другой в качестве подтерма и поэтому может быть решена за полиномиальное время.

Однако последнее требование не выполняется для произвольных подстановок. Рассмотрим подстановки $\theta = \{x_1/x_1, x_2/f(x_1)\}$, $\theta' = \{x_1/f(x_1), x_2/f(x_1)\}$ и $\theta'' = \{x_1/x_2, x_2/x_2\}$. Нетрудно заметить, что имеет место равенство $\theta \circ \theta' = \theta \circ \theta'' = \{x_1/f(x_1), x_2/f(x_1)\}$, однако, подстановки θ' и θ'' не являются эквивалентными в квазирешетке $(Subst(X, X, F), \preceq)$.

Коль скоро вся полугруппа подстановок не удовлетворяет требованию **Req1**, для применения метода минимизации автоматов-преобразователей, предложенного в статье [6], нужно выделить как можно более обширную подполугруппу подстановок, удовлетворяющих требованиям **Req1**–**Req3**. В качестве такой подполугруппы мы предлагаем использовать множество консервативных ортогональных подстановок, введенных в статье [27].

Подстановка $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ называется *консервативной*, если $Var_\theta = X$, и *ортогональной*, если ни один терм t_i, $1 \le i \le n$, не является подтермом никакого другого терма t_j, $j \ne i$. Например, подстановка $\theta_0 = \{x_1/f(x_2), x_2/x_1\}$ является консервативной и ортогональной, подстановка $\theta_1 = \{x_1/f(x_2), x_2/g(x_2)\}$ не является консервативной, а подстановка $\theta_2 = \{x_1/f(x_1), x_2/h(x_2, f(x_1))\}$ не является ортогональной. Класс всех консервативных и ортогональных подстановок обозначим записью $COSubst(X, X, F)$.

Исходя из приведенного определения, нетрудно заметить, что композиция консервативных ортогональных подстановок также является консервативной ортогональной подстановкой и, кроме того, любой пример и любой шаблон всякой консервативной ортогональной подстановки также является консервативной и ортогональной подстановкой. Поэтому класс $COSubst(X, X, F)$ образует подполугруппу подстановок, а квазиупорядоченное множество подстановок $(COSubst(X, X, F), \preceq)$ является фундированной квазирешеткой, удовлетворяющей требованию **Req1**.

Утверждение 7. *Полугруппа консервативных ортогональных подстановок удовлетворяет свойству левого сокращения.*

Доказательство. Если бы для некоторой пары различных термов t', t'' и ортогональной подстановки $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ выполнялось равенство $t'\theta = t''\theta$, то и для некоторой переменной x_i и отличного от нее терма t также выполнялось равенство $x_i\theta = t\theta$. Но это означало бы, терм t_i содержит в качестве подтермов некоторые термы из множества $\{t_1, \ldots, t_n\}$ вопреки определению свойства ортогональности подстановки. □

Таким образом, полугруппа консервативных ортогональных подстановок удовлетворяет требованиям **Req1**–**Req3**. Согласно результатам статьи [6] отсюда следует

Утверждение 8. *Задача минимизации конечных детерминированных автоматов-преобразователей над полугруппой консервативных ортогональных подстановок может быть решена за время, полиномиальное относительно размера минимизируемого автомата.*

Опираясь на утверждение 8 и принимая во внимание тот способ трансляции стандартных схем программ в автоматы-преобразователи над полугруппами подстановок, который описан в утверждении 5, мы можем описать один из классов схем программ, для которого задача минимизации решается за полиномиальное время.

Теорема 4. *Задача минимизации решается за полиномиальное время в классе стандартных схем программ, удовлетворяющих двум условиям:*

1) *все дуги схемы помечены консервативными ортогональными подстановками;*

2) *каждому узлу схемы приписан атом, содержащий вхождения всех переменных из множества X.*

6. Заключение

Нам удалось показать, что задачи минимизации стандартных схем программ и автоматов-преобразователей, работающих над полугруппой подстановок, тесно взаимосвязаны между собой — методы решения одной из этих задач могут быть адаптированы для решения другой. В частности, эффективный метод минимизации автоматов-преобразователей, предложенный в статье [6], можно использовать при посредничестве стандартных схем программ для решения задачи оптимизации последовательных императивных программ. В то же время разрешимость проблемы логико-термальной эквивалентности стандартных схем программ гарантирует разрешимость задач проверки эквивалентности и минимизации для детерминированных автоматов-преобразователей, работающих над полугруппами подстановок.

Вопрос о сложности задач минимизации стандартных схем программ и автоматов-преобразователей остается открытым и требует дальнейшего исследования. Метод минимизации автоматов-преобразователей, предложенный в статье [6], применим только в том случае, когда полугруппа состояний данных, над которой работают преобразователи, удовлетворяет требованиям **Req1**–**Req3**. Как было показано в упомянутой статье, для полугрупп такого вида каждый класс эквивалентности автоматов-преобразователей содержит единственный минимиальный преобразователь. Обычно в теории автоматов неоднозначность решения задачи минимизации

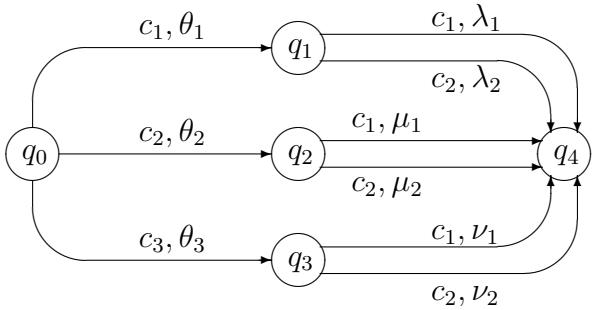


Рис. 1. Автомат-преобразователь П

Fig. 1. Transducer П

служит признаком того, что эта задача имеет большую вычислительную сложность. Так, например, однозначность решения задачи минимизации детерминированных конечных автоматов сопутствует быстрым алгоритмам минимизации автоматов этого вида, тогда как неоднозначность решения той же задачи для недетерминированных конечных автоматов сопровождается PSPACE-полнотой этой задачи в указанном классе автоматов. В заключение мы приведем простой пример, показывающий, что задача минимизации автоматов-преобразователей произвольного вида, работающих над полугруппой подстановок, может не иметь однозначного решения.

Рассмотрим автомат-преобразователь П, система переходов которого изображена на рис. 1. Его начальным состоянием является состояние q_0, а состоянием выхода — q_4. Действиями преобразователя служат подстановки над множеством переменных $X = \{x, y\}$ и функциональных символов $\mathcal{F} = \{f, g, h_1, h_2\}$. Действиями инициализации и финализации преобразователя служит тождественная подстановка $\varepsilon = \{x/x, y/y\}$. Прочие действия заданы следующими подстановками:

$$\theta_1 = \{x/g(f(x), g(x, x)), y = f(f(g(x, x)))\},$$
$$\theta_2 = \{x/f(f(x)), y/f(x)\},$$
$$\theta_3 = \{x/g(x, y), y/f(y)\},$$
$$\lambda_1 = \{x/h_1(x), y/h_1(y)\}, \qquad \lambda_2 = \{x/h_2(x), y/h_2(y)\},$$
$$\mu_1 = \{x/h_1(g(x, y)), y/h_1(f(f(y)))\}, \qquad \mu_2 = \{x/h_2(g(x, y)), y/h_2(f(f(y)))\},$$
$$\nu_1 = \{x/h_1(g(f(f(x))), f(x)), y = h_1(f(f(x)))\},$$
$$\nu_2 = \{x/h_2(g(f(f(x))), f(x)), y = h_2(f(f(x)))\}.$$

Непосредственной проверкой можно убедиться в том, что автомат-преобразователь П эквивалентен каждому из двух автоматов-преобразователей П′ и П″, которые заданы системами переходов, изображенными на рис. 2. В этих преобразователях используются действия, которые заданы подстановками

$$\xi_1' = \{x/x, y/g(x, x)\},\ \xi_2' = \{x/f(x), y/f(x)\},$$
$$\tau_i' = \{x/h_i(g(f(x), y)), y/h_i(f(f(y)))\}, \qquad i = 1, 2,$$
$$\xi_1'' = \{x/x, y/f(x)\},\ \xi_2'' = \{x/g(x, y), y/g(x, y)\},$$
$$\tau_i'' = \{x/h_i(g(f(f(x)), f(x))), y/h_i(f(f(y)))\}, \qquad i = 1, 2.$$

Автоматы-преобразователи П′ и П″ неизоморфны. Они являются минимальными преобразователями, поскольку никакой автомат-преобразователь с тремя состояни-

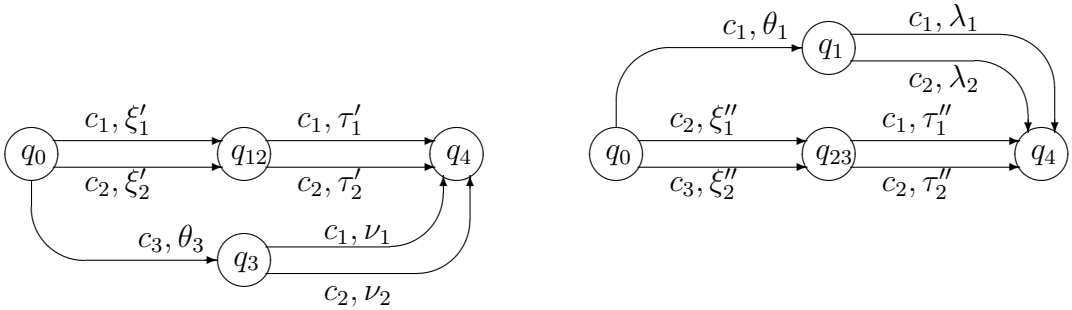


Рис. 2. Автоматы-преобразователи Π' и Π''

Fig. 2. Transducers Π' and Π''

ями им не эквивалентен. Таким образом, задача минимизации конечных детерминированных автоматов-преобразователей над полугруппой подстановок в общем случае может не иметь единственного решения. Поиск оценки вычислительной сложности этой задачи и разработка эффективных процедур ее решения составляют тему дальнейшего исследования.

И напоследок обратим внимание на то, что мы ограничились рассмотрением самой простой меры сложности стандартных схем программ — количеством узлов в графе, представляющем схему. Можно также оценивать сложность схемы суммарной сложностью всех термов в подстановках, приписанных дугам схемы. Задача минимизации схем относительно этой меры сложности также заслуживает внимания, и она, вероятно, может оказаться более трудной для решения.

Список литературы / References

[1] Глушков В. М., Летичевский А. А., “Теория дискретных преобразователей”, *Избранные вопросы алгебры и логики*, Новосибирск, 1973, 5–39; [Glushkov V. M., Letichevskij A. A., “Teoriya diskretnyh preobrazovatelej”, *Izbrannye voprosy algebry i logiki*, Novosibirsk, 1973, 5–39, (in Russian).]

[2] Ершов А. П., “Сведение задачи экономии памяти при составлении программ к задаче раскраски вершин графа”, *Доклады АН СССР*, **142**:4 (1962), 785–787; [Ershov A. P., “Svedenie zadachi ekonomii pamyati pri sostavlenii programm k zadache raskraski vershin grafa”, *Doklady AN SSSR*, **142**:4 (1962), 785–787, (in Russian).]

[3] Ершов А. П., “Современное состояние теории схем программ”, *Проблемы кибернетики*, **27**, 1973, 87–110; [Ershov A. P., “Sovremennoe sostoyanie teorii skhem program”, *Problemy kibernetiki*, **27** (1973), 87–110, (in Russian).]

[4] Захаров В. А., Новикова Т. А., “Полиномиальный по времени алгоритм проверки логико-термальной эквивалентности программ ”, *Труды Института системного программирования РАН*, **22** (2012), 435–455; [Zakharov V. A., Novikova T. A., “Polynomial time algorithm for checking strong equivalence of program”, *Proceedings of the Institute for System Programming*, **22** (2012), 435–455, (in Russian).]

[5] Захаров В. А., Подымов В. В., “Применение алгоритмов проверки эквивалентности для оптимизации программ”, *Труды Института системного программирования РАН*, **27**:4 (2015), 145–174; [Zakharov V. A., Podymov V. V., “On the application of equivalence checking algorithms for program minimization”, *Proceedings of the Institute for System Programming*, **27**:4 (2015), 145–174, (in Russian).]

[6] Захаров В. А., Темербекова Г. Г., “О минимизации конечных автоматов-преобразователей над полугруппами ”, *Моделирование и анализ информационных систем*, **23**:6 (2016), 741–753; [Zakharov V. A., Temerbekova G. G., “On the

Minimization of Finite State Transducers over Semigroups", *Modeling and Analysis of Information Systems*, **23**:6 (2016), 741–753, (in Russian).]

[7] Захаров В. А., *Проблема эквивалентности программ: модели, алгоритмы, сложность*, М., 2016, 304 с.; [Zakharov V. A., *Problema ekvivalentnosti programm: modeli, algoritmy, slozhnost*, Moskva, 2016, 304 pp., (in Russian).]

[8] Иткин В. Э., "Логико-термальная эквивалентность схем программ", *Кибернетика*, 1972, № 1, 5–27; [Itkin V. E., "Logiko-termalnaya ekvivalentnost skhem program", *Cybernetics*, 1972, № 1, 5–27, (in Russian).]

[9] Котов В. Е., Сабельфельд В. К., *Теория схем программ*, Наука, М., 1991, 248 с.; [Kotov V. E., Sabel'fel'd V. K., *Teoriya skhem program*, Nauka, Moskva, 1991, 248 pp., (in Russian).]

[10] Криницкий Н. А., *Равносильные преобразования алгоритмов и программирование*, М., 1970, 304 с.; [Krinitskiy N. A., *Ravnosilnye preobrazovaniya algoritmov i programmirovanie*, Moskva, 1970, 304 pp., (in Russian).]

[11] Лавров С. С., "Об экономии памяти в замкнутых операторных схемах", *Журнал вычислительной математики и математической физики*, **1**:4 (1961), 687–701; English transl.: Lavrov S. S., "Store economy in closed operator schemes", *U.S.S.R. Comput. Math. Math. Phys.*, **1**:3 (1962), 810–828.

[12] Ляпунов А. А., "О логических схемах программ", *Проблемы кибернетики*, **1**, 1958, 46–74; [Lyapunov A. A., "O logicheskikh skhemakh program", *Problemy kibernetiki*, **1**, 1958, 46–74, (in Russian).]

[13] Подловченко Р. И., "Модели последовательных программ, применяемые для изучения функциональной эквивалентности программ", *Кибернетика*, 1979, № 1, 20–28; English transl.: Podlovchenko R. I., "Models of sequential programs used to study functional equivalence of programs", *Cybern Syst Anal*, **15**:1 (1979), 22–31.

[14] Янов Ю. И., "О логических схемах алгоритмов", *Проблемы кибернетики*, **1**, 1958, 75–127; [Yanov Yu. I., "O logicheskikh skhemakh algoritmov", *Problemy kibernetiki*, **1**, 1958, 75–127, (in Russian).]

[15] Eder E., "Properties of substitutions and unifications", *Journal of Symbolic Computations*, **1**:1 (1985), 31–46.

[16] Ershov A. P., "Alpha — an automatic programming system of high efficiency", *Journal of the Association for Computing Machinary*, **13**:1 (1966), 17–24.

[17] Friedman E. P., "Equivalence problems for deterministic languages and monadic recursion schemes", *Journal of Computer System Science*, **14**:3 (1977), 362–399.

[18] Harju T., Karhumaki J., "The equivalence of multi-tape finite automata", *Theoretical Computer Science*, **78**:2 (1991), 347–355.

[19] Luckham D. C., Park D. M., Paterson M. S., "On formalized computer programs", *Journal of Computer and System Science*, **4**:3 (1970), 220–249.

[20] Mohri M., "Minimization algorithms for sequential transducers", *Theoretical Computer Science*, **234** (2000), 177–201.

[21] Muchnick S., *Advanced Compiler Design and Implementation*, 1997, 1–856.

[22] Rutledge J. D., "Program schemata as automata", *Proc. of the 11-th Annual Symposium on Switching and Automata Theory (SWAT 1970)*, 1970, 7–24.

[23] Sabelfeld V. K., "The logic-termal equivalence is polynomial-time decidable", *Information Processing Letters*, **10**:2 (1980), 57–62.

[24] Senizergues G., "The equivalence problem for deterministic pushdown automata is decidable", *Proc. of the 24-th International Colloquium on Automata, Languages, and Programming (ICALP-1997). Lecture Notes in Computer Science*, **1256** (1997), 671–681.

[25] Shofrutt C., "Minimizing subsequential transducers: a survey", *Theoretical Computer Science*, **292** (2003), 131–143.

[26] Watson B. W., "A taxonomy of finite automata minimization algorithm", *Computing Science Report. Eindhoven University of Technology*, **93/44** (2005), 1–32.

[27] Zakharov V. A., "On the decidability of the equivalence problem for orthogonal sequential programs", *Grammars*, **2**:3 (1999), 271–281.

[28] Zakharov V. A., "Equivalence checking problem for finite state transducers over semigroups", *Proc. of the 6-th International Conference on Algebraic Informatics (CAI-2015). Lecture Notes in Computer Science*, **9270** (2015), 208–221.

Zakharov V. A., Zhailauova S. R., "On the Minimization Problem for Sequential Programs", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 415–433.

DOI: 10.18255/1818-1015-2017-4-415-433

Abstract. First-order program schemata is one of the simplest models of sequential imperative programs intended for solving verification and optimization problems. We consider the decidable relation of logical-thermal equivalence of these schemata and the problem of their size minimization while preserving logical-thermal equivalence. We prove that this problem is decidable. Further we show that the first-order program schemata supplied with logical-thermal equivalence and finite state deterministic transducers operating over substitutions are mutually translated into each other. This relationship implies that the equivalence checking problem and the minimization problem for these transducers are also decidable. In addition, on the basis of the discovered relationship, we have found a subclass of first-order program schemata such that their minimization can be performed in polynomial time by means of known techniques for minimization of finite state transducers operating over semigroups. Finally, we demonstrate that in general case the minimization problem for finite state transducers over semigroups may have several non-isomorphic solutions.

Keywords: sequential program, transducer, minimization, substitution, semigroup, equivalence

On the authors:
Vladimir A. Zakharov, orcid.org/0000-0002-3794-9565, PhD, senior researcher
National Reserach University Higher School of Economics, Faculty of Computer Science
20 Myasnitskaya ul., Moscow 101000, Russia, e-mail: zakh@cs.msu.ru

Shynar R. Zhailauova, graduate student,
Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics,
Leninskiye Gory, GSP-1, 1-52, Moscow 119991, Russia, e-mail: gulgaisha93@mail.ru

Acknowledgments:
This work is supported by the by RFBR grants №16-01-00546 and №15-01-05742.

Моделирование и анализ информационных систем. Т. 24, № 4 (2017), с. 434–444
Modeling and Analysis of Information Systems. Vol. 24, No 4 (2017), pp. 434–444

DOI: 10.18255/1818-1015-2017-4-434-444

УДК 004.051

О скоростях передачи данных на шинах между кеш-памятью второго и третьего уровней и между процессором и оперативной памятью в современных компьютерах

Комар М. С.

получена 18 июля 2017

Аннотация. В данной работе рассматривается архитектура используемых в настоящее время центральных процессоров и ограничения их производительности в современном виде. Так как чаще всего для повышения производительности центральных процессоров предлагаются решения, связанные с изменением существующей архитектуры, необходимо иметь представление о скоростях передачи данных внутри процессора и на шинах, подходящих к нему. Это позволит оценить применимость предлагаемых решений и даст возможность их оптимизировать. В этой статье решается задача измерения реальных скоростей передачи данных на интерфейсе между кеш-памятью второго и третьего уровней внутри процессора и на интерфейсе между процессором и оперативной памятью, а также изучения зависимости численных результатов от количества активных ядер, тактовой частоты процессора и типа проводимого теста. В статье приводится методология проведения измерений с помощью программного инструмента Intel Performance Counter Monitor от компании Intel, а также приводятся формулы для получения итогового результата из полученных в ходе измерений значений. Приведено подробное описание тестов, имитирующих реальную нагрузку на центральный процессор, и синтетических тестов. Зависимости скоростей передачи данных от количества активных ядер и от тактовой частоты процессора представлены в виде графиков. Зависимости скоростей передачи данных от типа теста представлены в виде столбиковых диаграмм для трех различных значений тактовой частоты процессора.

Ключевые слова: многоядерные процессоры, оценка скоростей передачи данных, системы на кристалле, сети на кристалле, беспроводные системы на кристалле

Для цитирования: Комар М. С., "О скоростях передачи данных на шинах между кеш-памятью второго и третьего уровней и между процессором и оперативной памятью в современных компьютерах", *Моделирование и анализ информационных систем*, **24**:4 (2017), 434–444.

Об авторах:
Комар Мария Сергеевна, orcid.org/0000-0002-0995-7744, аспирант,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия
магистрант, Технологический Университет г. Тампере,
PO Box 527, FI-33101, Korkeakoulunkatu 10, Тампере, Финляндия
e-mail: maria.s.komar@gmail.com

1. Введение

В настоящее время все большее распространение получают персональные компьютеры, серверы, мобильные телефоны, ноутбуки, нетбуки, электронные книги, игровые консоли, “умные” часы и другие устройства, относящиеся к классу носимой электроники, и многое другое. Практически в каждом доме найдется не один представитель перечисленных выше типов устройств. Объединяет их друг с другом и другими представителями техники наличие центрального процессора. В настоящее время центральные процессоры (ЦП) используются практически повсеместно и являются неотъемлемой частью практически любого электронного устройства, соответственно прогресс в развитии этих устройств неразрывно связан с прогрессом существующих ЦП и созданием новых, улучшенных процессоров.

К сожалению, за последнее десятилетие прогресс в развитии производительности ЦП был не таким существенным, как хотелось бы пользователям и представителям индустрии. Если в 80-е и 90-е года прошлого века процессоры эволюционировали столь стремительно, что раз в 24 месяца количество транзисторов на чипе удваивалось (это эмпирическое наблюдение получило название закона Мура [1]), а раз в 18 месяцев в два раза увеличивалась производительность, то сейчас каждое новое поколение дает улучшение едва ли на 15 процентов [2]. Причина этого кроется в том, за счет чего происходил прогресс в прошлом веке и за счет чего он происходит сейчас. В прошлом веке и самом начале этого электроника бурно развивалась, соответственно была возможность относительно легко и быстро увеличивать количество транзисторов на чипе и повышать тактовую частоту процессора. Однако в настоящее время технологии дошли до той границы, когда ни увеличивать тактовую частоту, ни увеличивать количество транзисторов на чипе не представляется возможным [3]. В связи с этим сейчас ведется поиск новых способов увеличения производительности процессоров. Одно из перспективных направлений, над которым сейчас работают ученые, – это изменение существующей архитектуры центрального процессора на более эффективную. Основным кандидатом для замены являются конструкции из класса беспроводных систем на кристалле (Wireless Networks-on-Chip, WNoC), в которой предлагается часть проводных соединений заменить на беспроводные каналы связи [4], [5]. Однако для того, чтобы оценить, насколько емкими должны быть эти каналы, необходимо иметь представление о том, какова скорость передачи данных в существующих процессорах. Особенно интересна для исследователей загрузка шин данных между процессором и оперативной памятью и между кеш-памятью второго и третьего уровней, так как большинство решений предполагают построение сетей, в которых именно эти шины будут заменены на беспроводные каналы связи [5].

Исходя из всего вышеперечисленного, встает задача измерения реальных скоростей передачи данных на интерфейсе между кеш-памятью второго и третьего уровней внутри процессора и на интерфейсе между процессором и оперативной памятью, а также изучение зависимости численных результатов от количества активных ядер, тактовой частоты процессора и типа проводимого теста, решение которой описано в данной статье.

Статья имеет следующую структуру: в разделе 2 описана типичная архитектура современного центрального процессора и упрощенная схема работы кеш-памяти.

Раздел 3 посвящен описанию стенда, использовавшегося для проведения измерений, тестам, проведенным в ходе исследования, методологии их проведения и расчета скоростей передачи данных на интересующих интерфейсах по имеющимся результатам. В разделе 4 приведены численные результаты, полученные в ходе исследования и их анализ. Окончательные выводы сделаны в разделе 5.

2. Архитектура центрального процессора и модель передачи данных внутри него

В современных компьютерах чаще всего используются процессоры, имеющие кеш-память нескольких уровней. Это особый тип памяти, позволяющий получить очень быстрый доступ к хранящейся в ней информации, но имеющий ограниченный размер. Чаще всего используются процессоры с кеш-памятью двух уровней (особенно распространено в процессорах от AMD [6]) и трех уровней (в процессорах от Intel [7] с архитектурой x86 [8], принципиальная схема которого изображена на рис. 1).

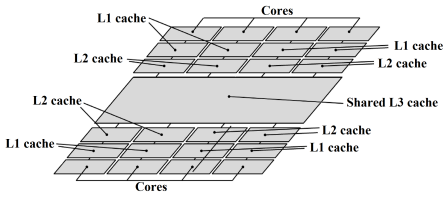


Рис. 1. Архитектура центрального процессора с тремя уровнями кеш-памяти и восемью ядрами

Fig. 1. CPU architecture with 3 cache layers and 8 cores

Первый уровень кеш-памяти является самым быстрым, дорогим и находится ближе всего к ядру процессора. Также он обеспечивает наименьшее время доступа к хранящимся в нем данным. Второй уровень кеш-памяти более медленный, более дешевый, однако тоже достаточно дорог в производстве. Кеш-память первого и второго уровней обычно являются отдельными для каждого ядра в случае многоядерных процессоров. Кеш-память первого уровня чаще всего разделяется на кеш инструкций и кеш данных. Кеш-память третьего уровня чаще всего является общей для всех ядер, имеет самый большой размер и самую большую из трех уровней задержку на доступ к хранимым данным. Сложность работы с кеш-памятью третьего уровня состоит в том, что к ней должны иметь доступ все ядра, например для организации совместных вычислений и своевременного обновления данных, которые находятся в общем доступе нескольких ядер. В такой системе данные, обработанные одним из ядер, могут быть оперативно получены другим. Это приводит к уменьшению времени простоя ядра, то есть повышает производительность системы. Упрощенная модель работы кеш-памяти, позволяющая, однако, реалистично оценить объемы данных, проходящие по шинам внутри процессора и между процессором и оперативной памятью, но лишенная ненужных для этой задачи подробностей, представлена ниже. Она включает в себя основные запросы, связанные с чтением и записью данных. Предположим, что одно из ядер ЦП пытается обратиться на чтение или на запись к определенным данным.

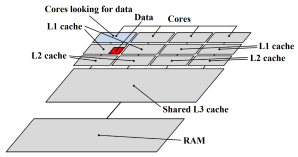


Рис. 2. Данные находятся в кеш-памяти первого уровня того ядра, которое делает запрос

Fig. 2. Piece of data is stored in L1 cache of the core that looks for it

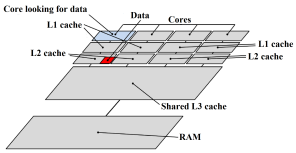


Рис. 3. Данные находятся в кеш-памяти второго уровня того ядра, которое делает запрос

Fig. 3. Piece of data is stored in L2 cache of the core that looks for it

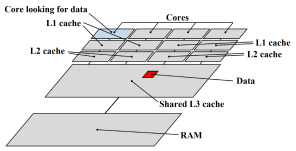


Рис. 4. Данные находятся в общей кеш-памяти третьего уровня

Fig. 4. Piece of data is stored in shared L3 cache

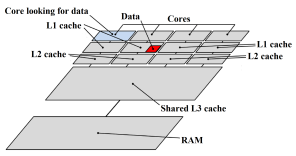


Рис. 5. Актуальная копия данных хранится в кеш-памяти первого уровня другого ядра

Fig. 5. Up-to-date copy of data is stored in the L1 cache of another core

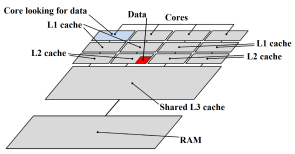


Рис. 6. Актуальная копия данных хранится в кеш-памяти второго уровня другого ядра

Fig. 6. Up-to-date copy of data is stored in the L2 cache of another core

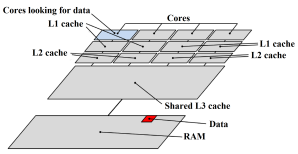


Рис. 7. Данные находятся в оперативной памяти

Fig. 7. Piece of data is stored in RAM

Тогда возможны следующие случаи, проиллюстрированные на рисунках 2 — 7:

1. Данные находятся в кеш-памяти первого уровня того ядра, которое делает запрос.
2. Данные находятся в кеш-памяти второго уровня того ядра, которое делает запрос.
3. Данные находятся в общей кеш-памяти третьего уровня.
4. Данные находятся в оперативной памяти.
5. Актуальная копия данных хранится в кеш-памяти первого или второго уровня другого ядра.

При попытке чтения данных их поиск проводится следующим образом: сначала информация ищется в кеш-памяти первого уровня, если совпадений не обнаружено, происходит так называемый «cache miss» (промах), и поиск продолжается в кеш-памяти второго уровня. Если данные найдены, то происходит «cache hit» (попадание), и результат возвращается ядру. В противном случае проводится поиск в кеш-памяти третьего уровня. Если данные найдены и копия, хранящаяся там,

актуальная, то результат возвращается ядру. Если данные найдены, но копия, хранящаяся в кеш-памяти третьего уровня, не актуальна, то контроллер кеша запрашивает данные у того ядра, которое хранит актуальную копию, и, получив, отдает их запрашивавшему ядру. Если данные не найдены в кеш-памяти третьего уровня, происходит обращение к оперативной памяти, и данные оттуда передаются ядру. Для запроса на запись ситуация практически аналогичная, за исключением случая, когда актуальная копия данных хранится в кеш-памяти другого ядра. В этом случае данные записываются в кеш-память третьего уровня, а данные в кеш-памяти другого ядра признаются неактуальными и обновляются при необходимости.

Также необходимо примерно оценить размеры пакетов, проходящих по интересующим нас шинам. С точки зрения самого вычислительного ядра его работа с кеш-памятью выглядит следующим образом: оно отсылает адрес, по которому хранятся интересующие его данные, и через какое-то время ему приходит блок информации. Размер адреса в современных ЦП различен, но его верхняя граница в настоящее время - 64 бита, то есть 8 байт [8]. Блок данных, возвращающийся к центральному процессору, всегда составляет 64 байта в случае памяти без корректировки ошибок [8]. Таким образом на один запрос всего по шинам данных проходит не больше 72 байт информации. Если данные не нашлись в кеш-памяти первого уровня, адрес передается в кеш-память второго уровня. Если данные действительно находятся там, то пакет с необходимой информацией уходит обратно к ядру по шине между кеш-памятью первого и второго уровней. Если данные не были найдены в кеш-памяти второго уровня, то запрос с адресом отправляется в кеш-память третьего уровня, а при отсутствии в ней таких данных – в оперативную память. Если информация нашлась в кеш-памяти третьего уровня, то в кеш-память второго уровня идет пакет с найденной информацией. Аналогично этому, если данные нашлись только в оперативной памяти, то найденная информация идет в кеш-память третьего уровня.

Отдельно следует рассмотреть случай, изображенный на рисунках 5 и 6, когда актуальная информация хранится в кеш-памяти первого или второго уровня другого ядра. В этом случае контроллером кеша генерируется сервисное сообщение-запрос на получение актуальной копии данных. Размер этого сообщения так же можно аппроксимировать 8 байтами. В ответ на это в кеш-память третьего уровня отправляется пакет данных стандартного размера – 64 байта. Таким образом, в данном случае по двум разным шинам между кеш-памятью второго и третьего уровней проходит по 72 байта. Для запросов на запись схема следующая: ядро отдает кеш-памяти первого уровня данные, которые необходимо записать, и адрес, по которому их необходимо записать. Кеш-память первого уровня записывает данные и по мере необходимости (или в других случаях, если это определено протоколом) передает данные в кеш-память верхних уровней и через них в оперативную память.

3. Тестовый стенд и методология измерений

Основой тестового стенда был персональный компьютер со следующими характеристиками:

- Процессор Intel Core i7-5960X с архитектурой Haswell семейства Haswell-E с тактовой частотой (без включения режима Turbo Boost) от 1200 МГц до 3000 МГц.
- Материнская плата Asus X99-A LGA2011-3
- Оперативная память 4 модуля по 4 ГБ Kingston HyperX Predator DDR4
- Видеокарта NVIDIA 750GTX
- Накопитель Samsung 850 256ГБ SSD
- Операционная система Linux Kubuntu 14.10 [9]

Оценка загрузки шин данных была получена с помощью натурных измерений. Для этого использовался Intel Performance Counter Monitor (Intel PCM, [10]) – специальный инструмент для оценки производительности процессоров, выпущенный корпорацией Intel. Кроме вывода на экран показаний со счетчиков, он позволяет сохранять их в csv файл, что является большим преимуществом для решения поставленных задач. Стоит отметить, что ни один из известных нам инструментов для проведения натурных измерений не дает количественных значений скорости передачи данных в явном виде, а лишь предоставляет информацию о данных с некоторого набора счетчиков, таких, как количество cache hit (попаданий) и cache miss (промахов) в единицу времени. Однако по этим значениям можно оценить интересующие нас характеристики.

Для тестирования были выбраны как искусственные тесты, так и реальные сценарии использования компьютера. К искусственным тестам можно отнести две программы, написанные на языке C. Они получили условные названия ”хороший стиль программирования” и ”плохой стиль программирования”. В оперативной памяти создавался массив большого объема, который гарантированно не мог поместиться в кеш-памяти третьего уровня. В массиве находились элементы размером 1 байт каждый. Далее в бесконечном цикле производилось чтение данных из этого массива. В случае теста, имитирующего ”хороший стиль программирования” считалось, что при создании программы разработчики позаботились об оптимизации своего продукта и постарались минимизировать количество обращений ядра к оперативной памяти в ходе работы программы, при этом не допуская ее простоя. В ходе этого теста из массива, описанного выше, считывался каждый байт подряд. В таком случае в кеш-память уходили блоки по 64 элемента (байта), такие же блоки перемещались между уровнями кеш-памяти. Тогда ядро запрашивало новую порцию данных только через 63 операции, так как все данные для последующих 63 операций уже хранились в кеш-памяти после первого обращения за данными. Этот же эффект кеш-памяти, но для другой цели использовался в тесте, имитирующем плохой стиль программирования. Целью этого теста было максимально загрузить шины данных, для этого на каждом шаге брался каждый 64-й элемент массива, соответственно информация, переданная в каждом из блоков, не могла быть использована для следующей операции и на каждом шаге процессору приходилось запрашивать новую порцию данных из массива.

Кроме искусственных тестов, были проведены тесты, представляющие собой сценарии реального использования компьютера. Первым из них была оценка того, насколько сильно загружены шины, когда нет никакой нагрузки на процессор, кроме работы операционной системы. Вторым тестом было шифрование файла с помощью алгоритма AES-256 [11]. Третьим тестом была компьютерная игра. К сожалению, большинство современных игр достаточно сложно запустить в операционной системе Linux без использования дополнительного ПО. По этой причине пришлось ограничиться играми, требующими меньшее количество ресурсов, но запускаемыми в операционной системе Linux без использования дополнительных программ. Для тестирования была выбрана игра Total Annihilation [12], запущенная на максимальных возможных настройках. Четвертым и последним тестом было декодирование видео на центральном процессоре.

Все тесты проводились с частотой измерений 5 раз в секунду в течение 20 минут каждый. При проведении тестов была поставлена задача оценить, какова загрузка интересующих нас шин для каждого из тестов и как она меняется в зависимости от изменения количества активных ядер и тактовой частоты процессора. Если тестирование проводилось для нескольких активных ядер, запускалось количество тестовых программ, равное количеству активных ядер.

Запрос к кеш-памяти третьего уровня от ядра возможен только в случае, когда был выполнен поиск в кеш-памяти второго уровня и необходимые данные были не найдены. Соответственно, чтобы рассчитать количество данных, проходящее по шине между кеш-памятью второго и третьего уровней, необходимо количество промахов в кеш-памяти второго уровня умножить на размер пакета данных, проходящего по шине для одного запроса, которое мы приняли равным 72 байтам. Таким образом, скорость передачи данных в гигабитах в секунду исходя из полученных нами данных можно рассчитать по следующей формуле:

$$S_{\mathrm{L2-L3}} = \frac{L2_{\mathrm{miss,av}} \cdot 10^6 \cdot 8 \cdot 72}{10^9}, \tag{1}$$

где $L2_{\mathrm{miss,av}}$ — это среднее значение количества промахов в кеш-памяти второго уровня за секунду.

Аналогично для шины между кеш-памятью третьего уровня и оперативной памятью запрос к оперативной памяти возможен, только если данные не были найдены в кеш-памяти третьего уровня, поэтому скорость передачи данных можно рассчитать по формуле:

$$S_{\mathrm{L3-RAM}} = \frac{L3_{\mathrm{miss,av}} \cdot 10^6 \cdot 8 \cdot 72}{10^9}, \tag{2}$$

где $L3_{\mathrm{miss,av}}$ — это среднее значение количества промахов в кеш-памяти третьего уровня за секунду.

Хочется отметить, что для шины между кеш-памятью второго и третьего уровней результаты даны для всех ядер сразу, а не в пересчете на шину между кеш-памятью третьего уровня и ядром с кеш-памятью первого и второго уровней. Для шины данных между кеш–памятью третьего уровня и оперативной памятью дана не полная загрузка всей шины, а только ее части, связанной непосредственно с процессором и оперативной памятью, так как в настоящее время по этой шине идут

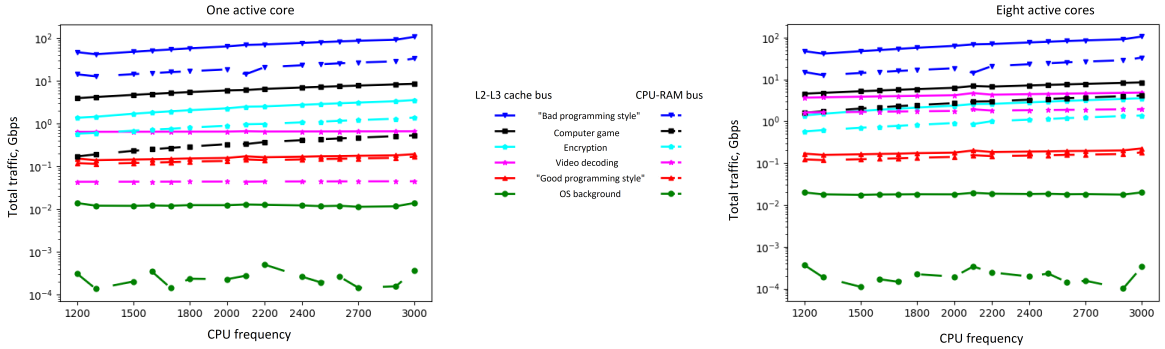


Рис. 8. Зависимость скорости передачи данных от тактовой частоты для одного и восьми активных ядер

Fig. 8. Dependence of total traffic on CPU frequency for 1 and 8 active cores

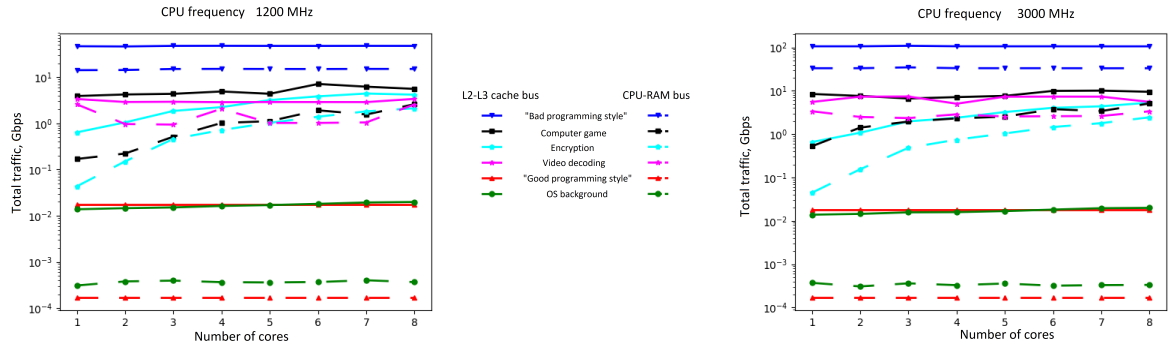


Рис. 9. Зависимость скорости передачи данных от количества активных ядер для минимальной и максимальной тактовой частоты

Fig. 9. Dependence of total traffic on number of active cores for minimal and maximal CPU frequencies

также данные от многих других устройств, но при разработке моделей процессоров на основе беспроводных систем на кристалле они обычно не берутся во внимание, и требуется только оценка скорости передачи данных именно между оперативной памятью и процессором.

В следующей главе рассмотрены полученные результаты и приведены зависимости скорости передачи данных на каждой из интересующих нас шин в зависимости от тактовой частоты процессора и количества активных ядер.

4. Численные результаты

Как видно из рисунков 8, 9 и 10, фоновая нагрузка операционной системы очень слабо влияет как на шину между кеш-памятью второго и третьего уровней, так и на шину между процессором и оперативной памятью. По сравнению с результатами других тестов фоновая нагрузка меньше на порядок. Следовательно, можно считать данные, полученные для всех тестов, достоверными. Также по графикам на рисунке

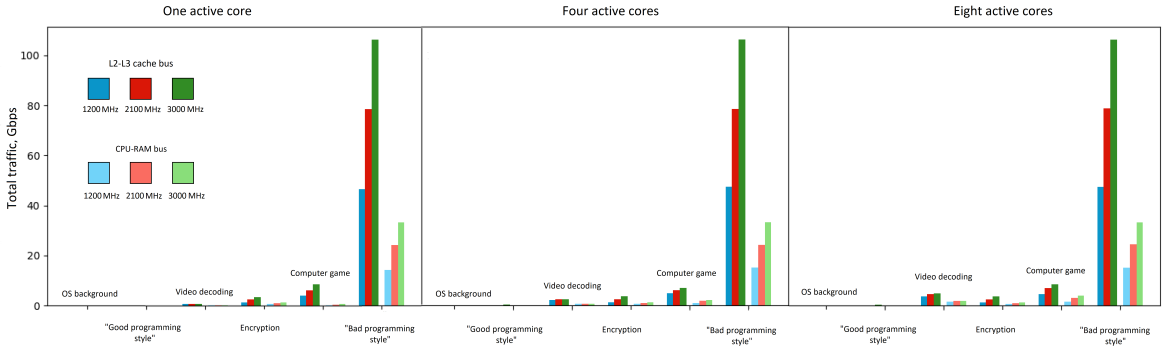


Рис. 10. Зависимость скорости передачи данных от типа теста и тактовой частоты процессора для одного, четырех и восьми активных ядер

Fig. 10. Dependence of total traffic on test type and CPU frequency for 1, 4 and 8 active cores

8 видно, что увеличение тактовой частоты процессора влияет на скорости передачи данных между кеш-памятью второго и третьего уровней, а также на скорости передачи данных между процессором и оперативной памятью, однако значения отличаются не очень значительно. Кроме того, стоит отметить, что характер изменения графиков одинаковый для любого количества активных ядер. Из всего вышеперечисленного можно сделать вывод о том, что тактовая частота процессора оказывает некоторое влияние на количество данных, передаваемых между кеш-памятью второго и третьего уровней и между процессором и оперативной памятью, однако это влияние не очень значительно. Так же мы видим, что скорость передачи данных на интерфейсе между кеш-памятью второго и третьего уровней может превышать 100 Гбит в секунду, а на интерфейсе между процессором и оперативной памятью достигать 40 Гбит в секунду. Графики на рис. 9 показывают зависимость скорости передачи данных между кеш-памятью второго и третьего уровней и между процессором и оперативной памятью от количества активных ядер. На них видно, что количество данных очень сильно зависит от того, какая программа использует процессор. Если для компьютерной игры или декодирования видео значения скорости оказываются в районе 10 Гбит в секунду для шины между вторым и третьим уровнем кеш-памяти и 5 Гбит в секунду для шины между оперативной памятью и процессором, то для теста, имитирующего плохой стиль программирования, эти значения превышают 100 Гбит в секунду для интерфейса между кеш-памятью второго и третьего уровней и приближаются к 40 Гбит в секунду для интерфейса между процессором и оперативной памятью, как было сказано выше. Кроме того, можно отметить, что при увеличении количества ядер количество данных, передаваемых на обоих интерфейсах, увеличивается для всех тестов кроме “плохого стиля программирования”. В случае с “плохим стилем программирования” количество данных, переданных за секунду, не изменяется. Это связано со спецификой теста – он достаточно сильно нагружает шины данных, и это вызывает дополнительные задержки при обращении к оперативной памяти, вследствие чего уменьшается количество операций в секунду, а значит, и промахов при поиске в кеш-памяти второго и третьего уровней. На рисунке 10 наглядно показано сравнение количества передаваемых за секунду данных для всех типов проводимых тестов для одного, четырех и восьми ядер. Дан-

ные представлены для трех различных тактовых частот – 1200 МГц, 2100 МГц и 3000 МГц. На графиках наглядно видно, что количество передаваемых за секунду данных растет по-разному для каждого теста, но зависимость примерно одинаковая для всех частот.

Кроме того, можно заметить, что синтетические тесты “хороший стиль программирования” и “плохой стиль программирования” представляют собой минимальный и максимальный случаи полезной загрузки для шины между кеш-памятью второго и третьего уровней и почти минимальный и максимальные случаи для ОЗУ, и таким образом их можно использовать для тестирования новых дизайнов.

Также стоит отметить, что загрузка шины между процессором и оперативной памятью практически всегда существенно меньше, чем загрузка шины между вторым и третьим уровнем кеш-памяти. Это связано с тем, что существующая архитектура уже крайне эффективна и также подчеркивает ограничения существующих ЦП.

5. Заключение

В данной статье были рассмотрены интерфейсы внутри и вне центрального процессора – шины между кеш-памятью второго и третьего уровней и между процессором и оперативной памятью. Была решена задача измерения реальных скоростей передачи данных на интерфейсе между кеш-памятью второго и третьего уровней внутри процессора и на интерфейсе между процессором и оперативной памятью, а также изучена зависимость численных результатов от количества активных ядер, тактовой частоты процессора и типа проводимого теста. На этих интерфейсах с помощью специальных инструментов были проведены натурные измерения на различных частотах и на различном количестве ядер. Эта информация может быть использована для разработки новых архитектур ЦП, в том числе имеющих в своем составе беспроводные каналы связи.

Список литературы / References

[1] Peter J. Denning, Ted G. Lewis, “Exponential Laws of Computing Growth”, *Communications of the ACM*, **60**:1 (2017), 54–65.

[2] Intel Corporation, “7th gen intel core and Intel Xeon processor briefing”, https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/01/7th-gen-intel-core-january-product-brief.pdf.

[3] Thomas Walther, “Scaling through more cores. From single to multi core”, https://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2015_2016/nthr-16-walther-scaling_through_more_cores-ausarbeitung.pdf.

[4] Li X., “Survey of Wireless Network-on-Chip Systems”, http://www.eng.auburn.edu/ agrawvd/THESIS/LI/report.pdf.

[5] Ganguly A., Deb S., Belzer B., “Scalable hybrid wireless network-on-chip architectures for multicore systems”, *IEEE Transactions on Computers*, **60**:10 (2011), 1485–1502.

[6] Advanced Micro Devices Inc., “AMD desktop processor solutions”, “AMD desktop processor solutions”, www.amd.com.

[7] Intel Corporation, www.intel.com.

[8] Intel Corporation, “Intel 64 and IA-32 Architectures Software Developer’s Manual”, https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf.

[9] Kubuntu devs, “Kubuntu 14.10”, www.kubuntu.org/news/kubuntu-14.10.

[10] Intel Corporation, “Intel Performance Counter Monitor”, www.intel.com/software/pcm.

[11] Daemen J., Rijmen V., “AES Proposal: Rijndael”, 1999.

[12] Total Annihilation Universe, “Total Annihilation Universe”, www.tauniverse.com.

Komar M. S., "Data Rates Assessment on L2–L3 CPU Bus and Bus between CPU and RAM in Modern CPUs", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 434–444.

DOI: 10.18255/1818-1015-2017-4-434-444

Abstract. In this paper, a modern CPU architecture with several different cache levels is described, and current CPU performance limitations such as silicone physical limitations or frequency increase bounds are mentioned. As usual, changes of the currently existing architecture are proposed as a way of increasing CPU performance, data rates on the internal and external CPU interfaces must be known. It would help to assess applicability of proposed solutions and allow to optimize them. This paper is aimed at getting real values of traffic on L2-L3 cache interface inside CPU and CPU-RAM bus load as well as show dependencies of total traffic on the interfaces of interest on the number of active cores, CPU frequency and test type. Measurements methodology using Intel Performance Counter Monitor by Intel is provided and equations that allow to get data rates from internal CPU counters are explained. Both real life and synthetic tests are described. Dependency of total traffic on the number of active cores and dependency of total traffic on CPU frequency are provided as plots. Dependency of total traffic on test type provided as bar plot for multiple CPU frequencies.

Keywords: multicore CPUs, data rates assessment, System-on-Chip, Network-on-Chip, Wireless Network-on-Chip, NoC, WNoC

On the authors:
Maria S. Komar, orcid.org/0000-0002-0995-7744, PhD student,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia,
MSc student, Tampere University of Technology,
PO Box 527, FI-33101, Korkeakoulunkatu 10, Tampere, Finland
e-mail: maria.s.komar@gmail.com

Моделирование и анализ информационных систем. Т. 24, № 4 (2017), с. 445–458
Modeling and Analysis of Information Systems. Vol. 24, No 4 (2017), pp. 445–458

DOI: 10.18255/1818-1015-2017-4-445-458

УДК 004.942, 004.056

Исследование одной марковской модели угроз безопасности компьютерных систем

Магазев А. А., Цырульник В. Ф.

получена 6 июля 2017

Аннотация. В настоящей работе исследуется модель угроз безопасности компьютерных систем, формулируемая на языке марковских процессов. В рамках данной модели функционирование компьютерной системы рассматривается как последовательность отказов и восстановлений, возникающих вследствие воздействия на систему угроз информационной безопасности. Приведено подробное описание модели: получены явные аналитические формулы для вероятностей состояний компьютерной системы в произвольный момент времени, обсуждаются некоторые предельные случаи и анализируется динамика системы на больших временах. Отдельно исследуется зависимость вероятности безопасного состояния (т.е. состояния, в котором угрозы отсутствуют) от вероятностей угроз. В частности, показано, что указанная зависимость качественно различается для четных и нечетных моментов времени. Например, в случае одной угрозы вероятность безопасного состояния в четные моменты времени зависит от вероятности угрозы не монотонно, имея, по крайней мере, один локальный минимум в своей области определения. Эта особенность представляется нам важной, так как ее учет позволяет выявить наиболее «опасные» области угроз, при которых вероятность обнаружения системы в безопасном состоянии может оказаться ниже допустимого уровня. В заключение вводится важная характеристика модели — время релаксации, и с ее помощью конструируется допустимая область значений параметров защиты системы.

Ключевые слова: компьютерная система, угроза безопасности, марковский процесс

Для цитирования: Магазев А. А., Цырульник В. Ф., "Исследование одной марковской модели угроз безопасности компьютерных систем", *Моделирование и анализ информационных систем*, **24**:4 (2017), 445–458.

Об авторах:
Магазев Алексей Анатольевич, orcid.org/0000-0002-8725-9183, канд. физ.-мат. наук, доцент,
Омский государственный технический университет,
пр. Мира, 11, г. Омск, 644050, Россия, e-mail: magazev@mail.ru

Цырульник Валерия Федоровна, orcid.org/0000-0002-6875-7216, студентка,
Омский государственный технический университет,
пр. Мира, 11, г. Омск, 644050, Россия, e-mail: lera.tsyrulnik@mail.ru

Введение

Роль моделирования в вопросах информационной безопасности и защиты информации трудно переоценить. После метода натурных испытаний, применение которого традиционно считается весьма дорогостоящим и трудоемким, метод моделирования остается практически единственной альтернативой существующим подходам к

исследованию современных защищенных компьютерных систем. Разработка и использование соответствующих моделей в этой области также необходимы для надлежащего теоретического обоснования механизмов и методов защиты, используемых при проектировании и эксплуатации реальных объектов.

Особое место среди существующих моделей информационной безопасности занимают модели, формулируемые на языке случайных марковских процессов. Спектр прикладных задач, решаемых с применением подобных моделей, необычайно широк: обнаружение кибер-атак в компьютерных сетях [1–3], моделирование процессов распространения компьютерных вирусов [4], обнаружение вторжений в компьютерных системах и вычислительных сетях [5,6], оптимизация и повышение надежности защищенных информационных систем [7, 8]. В последнее время также повысился интерес к скрытым марковским моделям, нашедшим свое применение в криптографии [9], а также в компьютерной вирусологии [10].

В работах [11–14] исследуется класс марковских моделей, в которых компьютерные системы, подвергающиеся угрозам информационной безопасности, рассматриваются как системы с отказами и восстановлениями. Подобный подход к исследованию безопасности компьютерных систем позволяет привлечь развитый математический аппарат теории надежности, хорошо зарекомендовавший себя при расчетах и проектировании сложных технических систем [15,16]. В частности, в статьях [11,12] была предложена марковская модель с конечным числом состояний, характеризующих степень влияния угроз информационной безопасности на компьютерную систему. Авторы процитированных работ провели предварительное исследование модели, обосновывали ее корректность, а также дали интерпретацию получаемых с ее помощью результатов.

В настоящей работе мы проводим более углубленный анализ марковской модели угроз информационной безопасности, предложенной в [11,12]. Помимо исследования некоторых нетривиальных особенностей моделей, не отмеченных ее авторами, мы также обсуждаем возможность приложения этой модели к задаче повышения надежности функционирования компьютерных систем, в частности, к задаче поиска оптимальных значений параметров защиты информации.

Структура настоящей статьи следующая.

В первом разделе дается подробное описание исследуемой марковской модели угроз информационной безопасности. Мы приводим явные аналитические выражения для вероятностей состояний компьютерной системы в произвольный момент времени и, в частности, показываем, что вероятность безопасного состояния (т.е. состояния, в котором отсутствуют угрозы) как функция времени представляется в виде аддитивной комбинации двух зависимостей — монотонно убывающей и осциллирующей. В этом же разделе показано, что на больших временах осциллирующей компонентой в динамике модели можно пренебречь.

Во втором разделе настоящей работы мы изучаем зависимость вероятности безопасного состояния от параметров, характеризующих вероятности угроз информационной безопасности. Показано, что эта зависимость качественно различается для четных и нечетных моментов времени; в частности, в случае одной угрозы вероятность безопасного состояния является монотонно убывающей функцией вероятности угрозы в нечетные моменты времени, тогда как в четные моменты времени она имеет, по крайней мере, один локальный минимум. Отмеченная особенность

представляется нам весьма важной, так как ее учет может позволить выявить наиболее «опасные» области угроз, при которых вероятность обнаружения системы в безопасном состоянии окажется ниже допустимого уровня.

Третий раздел посвящен применению рассматриваемой нами модели угроз информационной безопасности к задаче поиска допустимой области значений параметров защиты системы. В отличие от традиционного подхода, принятого в теории надежности и сводящегося к вычислению вероятности безотказной работы системы в течение заданного времени, мы предлагаем альтернативный подход, основанный на введении *времени релаксации* системы.

1. Описание модели

Рассмотрим компьютерную систему (в дальнейшем просто *систему*), на которую воздействует n независимых внешних угроз с вероятностями $q_1, q_2, \ldots, q_n$: $\sum_{i=1}^{n} q_i < 1$. Будем считать, что одновременное воздействие двух и более угроз невозможно и, кроме того, очередная угроза может проявиться только после успешного парирования предыдущей. В соответствии с этим в каждый момент времени $t = 0, 1, 2, \ldots$ система находится в одном из $n + 1$ возможных состояний: s_0, s_1, $\ldots$, s_n, s_{n+1}. В состоянии s_0, называемом *безопасным*, ни одна из угроз не реализуется. Состояние s_i, где $i = 1, \ldots, n$, характеризуется воздействием i-й угрозы. При этом в последующий момент времени имеется две альтернативы: либо данная угроза будет успешно отражена с вероятностью r_i и система вернется в состояние s_0, либо с вероятностью $\bar{r}_i = 1 - r_i$ эта угроза приведет к выводу системы из строя. В последнем случае мы будем считать, что система переходит в состояние s_{n+1}. Граф состояний системы приведен на рис. 1.

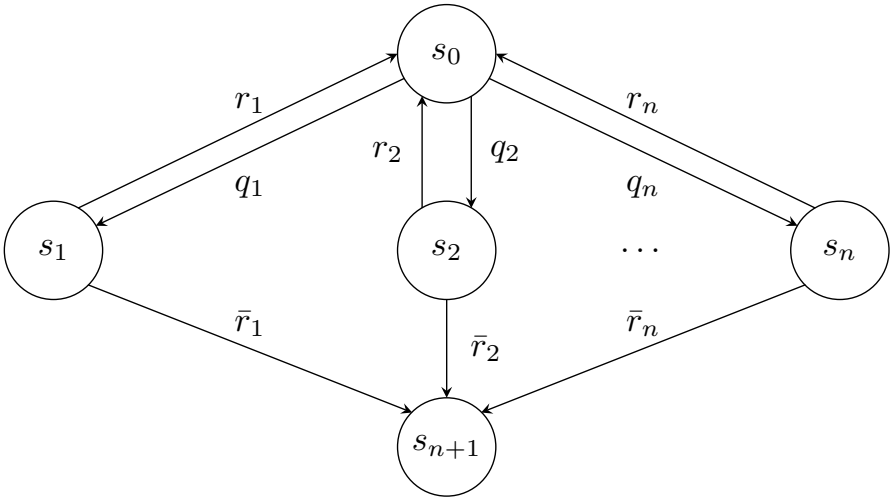


Рис. 1. Граф состояний модели

Fig. 1. The state graph of the model

Динамика рассматриваемой системы представляет собой простую марковскую цепь с матрицей переходных вероятностей

$$\Pi = \begin{pmatrix} q_0 & q_1 & q_2 & \dots & q_n & 0 \\ r_1 & 0 & 0 & \dots & 0 & \bar{r}_1 \\ r_2 & 0 & 0 & \dots & 0 & \bar{r}_2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ r_n & 0 & 0 & \dots & 0 & \bar{r}_n \\ 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}, \tag{1}$$

где $q_0 = 1 - \sum_{i=1}^{n} q_i$.

Обозначим через $p_i(t)$ вероятность того, что система находится в момент времени t в состоянии s_i. Эта вероятность определяется через вероятности состояний системы в момент времени $t-1$ согласно формуле

$$p_i(t) = \sum_{j=0}^{n+1} p_j(t-1)\,\Pi_{ji}, \tag{2}$$

или в матричной форме

$$\mathbf{p}(t) = \mathbf{p}(t-1) \cdot \Pi, \tag{3}$$

где $\mathbf{p}(t) = (p_0(t), p_1(t), \dots, p_{n+1}(t))$ — вектор вероятностей состояний системы в момент времени t. Используя (3) можно записать

$$\mathbf{p}(t) = \mathbf{p}(0) \cdot \Pi^t, \quad t = 0, 1, 2, \dots, \tag{4}$$

где через Π^t обозначена t-я степень матрицы Π.

Будем считать, что в начальный момент времени $t = 0$ система находилась в безопасном состоянии s_0, то есть $\mathbf{p}(0) = (1, 0, \dots, 0)$. Можно показать, что в этом случае вероятности состояний системы в произвольный момент времени t могут быть выражены следующими формулами:

$$p_0(t) = w^{-1}\left[\left(\frac{q_0 + w}{2}\right)^{t+1} - \left(\frac{q_0 - w}{2}\right)^{t+1}\right]; \tag{5}$$

$$p_i(t) = p_0(t-1)q_i, \quad i = 1, 2, \dots, n; \tag{6}$$

$$p_{n+1}(t) = 1 - p_0(t) - p_0(t-1)\sum_{i=1}^{n} q_i. \tag{7}$$

Здесь положительная величина w, которую мы далее будем называть *w-параметром* модели, определяется как

$$w^2 = q_0^2 + 4\sum_{i=1}^{n} r_i q_i > 0. \tag{8}$$

Докажем приведенные формулы индукцией по t. При $t = 1$ формулы (5) – (7) проверяются непосредственно. Допустим, что они верны для некоторого $t > 1$. Используя явный вид матрицы (1) для компонент вектора $\mathbf{p}(t+1) = \mathbf{p}(t) \cdot \Pi$ получаем:

$$p_0(t+1) = p_0(t)q_0 + \sum_{i=1}^{n} p_i(t) r_i; \tag{9}$$

$$p_i(t+1) = p_0(t)q_i, \quad i = 1, \ldots, n; \tag{10}$$

$$p_{n+1}(t+1) = \sum_{i=1}^{n} p_i(t)(1 - r_i) + p_{n+1}(t). \tag{11}$$

Равенство (9) с помощью формул (5), (6) и (8) может быть переписано как

$$p_0(t+1) = q_0 w^{-1} \left[\left(\frac{q_0 + w}{2} \right)^{t+1} - \left(\frac{q_0 - w}{2} \right)^{t+1} \right] +$$
$$+ w^{-1} \left[\left(\frac{q_0 + w}{2} \right)^{t} - \left(\frac{q_0 - w}{2} \right)^{t} \right] \sum_{i=1}^{n} q_i r_i = w^{-1} \left[\left(\frac{q_0 + w}{2} \right)^{t+2} - \left(\frac{q_0 - w}{2} \right)^{t+2} \right].$$

Далее, равенство (11) с учетом (9) принимает вид

$$p_{n+1}(t+1) = \sum_{i=1}^{n} p_i(t) - p_0(t+1) + p_0(t) \left(1 - \sum_{i=1}^{n} q_i \right) + p_{n+1}(t),$$

которое в силу формул (5) – (7) и тождества $\sum_{i=1}^{n} p_i(t) + p_0(t) + p_{n+1}(t) = 1$ может быть представлено в виде:

$$p_{n+1}(t+1) = 1 - p_0(t+1) - p_0(t) \sum_{i=1}^{n} q_i.$$

Таким образом, если равенства (5) – (7) верны для некоторого t, то они также будут верны и для $t+1$. Следовательно, по индукции эти равенства выполняются для любого t.

Отсутствие защиты в системе характеризуется условием $r_i = 0$, $i = 1, \ldots, n$. Тогда согласно (8) имеем $w = q_0$, то есть w-параметр в этом случае совпадает с вероятностью отсутствия реализации любой из угроз. Отсюда для вероятностей состояний системы будем иметь:

$$p_0(t) = q_0^t, \quad p_i(t) = q_0^{t-1} q_i, \quad i = 1, 2, \ldots, n; \quad p_{n+1}(t) = 1 - q_0^{t-1}. \tag{12}$$

Из этих равенств легко следуют предельные выражения для вероятностей состояний системы при $t \to \infty$:

$$\lim_{t\to\infty} p_0(t) = \cdots = \lim_{t\to\infty} p_n(t) = 0, \quad \lim_{t\to\infty} p_{n+1}(t) = 1. \tag{13}$$

Предельные соотношения (13) будут иметь место и в общем случае, когда не все параметры r_i равны нулю. Чтобы показать это, достаточно убедиться, что величины $(q_0 \pm w)/2$, стоящие в круглых скобках правой части равенства (5), по модулю всегда меньше единицы. Указанный факт, в свою очередь, является следствием того, что величины $(q_0 \pm w)/2$ представляют собой вещественные корни квадратного уравнения $f(x) = x^2 - q_0\, x - \sum_{i=1}^{n} q_i r_i = 0$, которые в силу неравенств $f(\pm 1) > 0$ и $f(0) < 0$ принадлежат интервалу $(-1, 1)$.

Из приведенных соображений видно, что по прошествии достаточно длительного времени система вероятнее всего будет обнаружена в состоянии s_{n+1}, в то время как

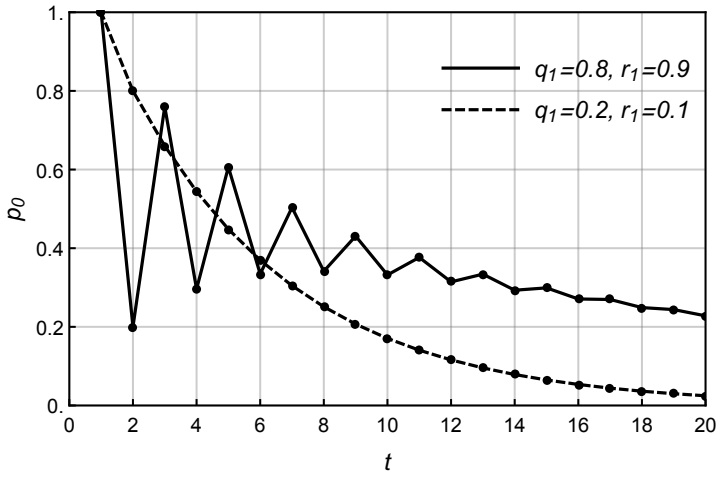


Рис. 2. Вероятность безопасного состояния как функция t при различных значениях параметров q_1 и r_1 в случае одной угрозы

Fig. 2. The security state probability as a function of t for different values of the parameters q_1 and r_1 in the case of one threat

вероятность ее обнаружения в остальных состояниях будет пренебрежимо мала. По этой причине состояние s_{n+1} будет являться *поглощающим состоянием* (что, впрочем, легко видно из графа состояний на рис. 1).

С практической точки зрения наибольший интерес представляет функция $p_0(t)$, являющаяся вероятностью обнаружения системы в безопасном состоянии в произвольный момент времени t. Из сказанного выше следует, что указанная вероятность будет уменьшаться с ростом t, однако в общем случае данная зависимость является не монотонной. Это легко видно из формулы (5): величина $p_0(t)$ представляется как разность двух стоящих в квадратных скобках слагаемых, первое из которых является монотонно убывающей функцией от t, а второе имеет осцилляционный характер (ввиду отрицательности величины $(q_0 - w)/2$). Скорость убывания функции $p_0(t)$ и выраженность ее осцилляций зависит, вообще говоря, от конкретных значений параметров модели q_i и r_i. На рис. 2 приведены графики зависимости $p_0(t)$ от t при двух различных значениях параметров q_1 и r_1 в случае $n = 1$.

В силу того, что

$$\lim_{t\to\infty} \left| \frac{q_0 - w}{q_0 + w} \right|^t = 0,$$

второе слагаемое, стоящее в квадратных скобках в выражении (5), является величиной бесконечно малой более высокого порядка, чем соответствующее первое слагаемое. Это означает, что «амплитуда» осцилляций функции $p_0(t)$ быстро убывает с ростом t, и на больших временах мы можем приближенно считать

$$p_0(t) \approx p_0^*(t) = w^{-1} \left(\frac{q_0 + w}{2} \right)^{t+1}. \tag{14}$$

Нетрудно оценить условие применимости приближения (14). Пусть $\varepsilon > 0$. Тогда

требование $|p_0(t) - p_0^*(t)| < \varepsilon$ эквивалентно неравенству

$$t > \log_{\frac{w-q_0}{2}} \varepsilon w - 1. \tag{15}$$

Таким образом, приближенное выражение (14) отличается от истинной вероятности $p_0(t)$ на величину, не большую ε, если система рассматривается на временах, удовлетворяющих условию (15).

2. Вероятность безопасного состояния как функция вероятности угроз

Изучим характер зависимости вероятности безопасного состояния $p_0(t)$ от вероятностей угроз $q_1, \ldots, q_n$. Для этого мы сначала разберем технически более простую ситуацию, когда на систему воздействует всего одна угроза: $n = 1$. Обозначим вероятность этой угрозы символом q, а соответствующий параметр отражения этой угрозы — r. Таким образом, матрица переходных вероятностей в этом случае имеет вид

$$\Pi = \begin{pmatrix} 1-q & q & 0 \\ r & 0 & 1-r \\ 0 & 0 & 1 \end{pmatrix}. \tag{16}$$

Всюду далее в этом параграфе мы предполагаем, что $0 \le q \le 1$ и $0 < r < 1$.

Из формул (4) и (16) следует, что зависимость $p_0(t)$ от q — полиномиальная, причем степень соответствующего полинома равна t:

$$p_0(t) = c_t q^t + c_{t-1} q^{t-1} + \cdots + c_1 q + c_0. \tag{17}$$

Используя для $p_0(t)$ равенство (5) нетрудно найти явный вид коэффициентов c_k:

$$c_k = (-1)^k \sum_{l=0}^{\min(t-k,k)} C_{t-l}^k C_k^l (-r)^l, \quad k = 0, 1, \ldots, t. \tag{18}$$

Здесь через C_n^k обозначен биномиальный коэффициент из n по k. В частности, для предельных значений параметра q с помощью (17) и (18) получаем:

$$p_0(t)|_{q=0} = 1, \quad p_0(t)|_{q=1} = \begin{cases} 0, & \text{если } t \text{ – нечетно,} \\ r^{t/2}, & \text{если } t \text{ – четно.} \end{cases} \tag{19}$$

Обратим внимание на различное поведение системы при $q = 1$ для четных и нечетных моментов времени: вероятность обнаружения системы в безопасном состоянии в нечетные моменты времени всегда равна нулю, в то время как аналогичная вероятность в четные моменты времени отлична от нуля и монотонно убывает с ростом t (напомним, что $0 < r < 1$). Отмеченная особенность связана с тем, что именно в нечетные моменты времени система «борется» с возникающей угрозой (т.е. находится в состоянии s_1), причем эффективность этой «борьбы» определяется значением параметра r.

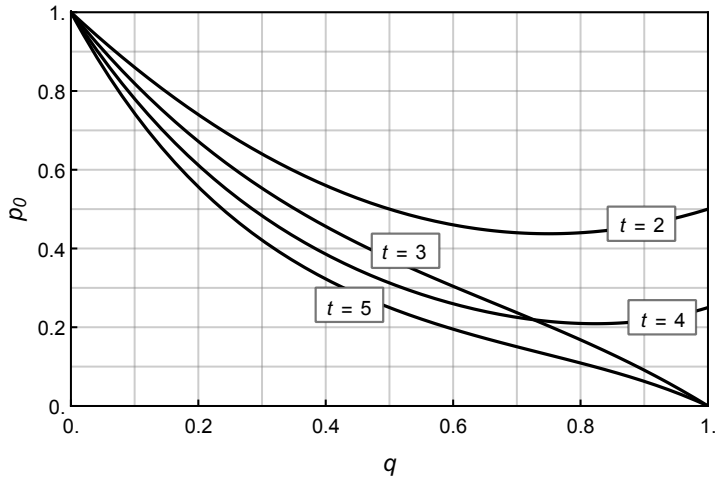


Рис. 3. Вероятность безопасного состояния как функция от q при $r = 0.5$ и $t = 2, 3, 4, 5$

Fig. 3. The security state probability as a function of q for $r = 0.5$ and $t = 2, 3, 4, 5$

Представление величины $p_0(t)$ в виде (17) позволяет предположить, что вероятность безопасного состояния системы не является, вообще говоря, *монотонной* функцией от вероятности угрозы q. В качестве примера на рис. 3 приведены графики зависимости $p_0(t)$ от q при $r = 0.5$ для четырех различных моментов времени t. Видно, что при четных значениях времени величина $p_0(t)$, рассматриваемая как функция от q, имеет выраженные минимумы, в то время как при нечетных значениях t эта функция является монотонно убывающей на всем промежутке $[0, 1]$. По-видимому, данная закономерность будет иметь место для любых значений t, хотя строгого доказательства этого нами пока не получено. Тем не менее, существование по крайней мере одного локального минимума для величины $p_0(t)$ (как функции q) при четных t можно легко доказать аналитически.

Вычисляя производную по q от выражения (17), получаем

$$\frac{dp_0(t)}{dq} = tc_t q^{t-1} + (t-1)c_{t-1}q^{t-2} + \cdots + 2c_2 q + c_1. \tag{20}$$

При *четных* значениях t данный полином будет иметь как минимум один вещественный корень $\bar{q}$ такой, что $0 < \bar{q} < 1$. Действительно, используя формулу (5), нетрудно найти, что

$$\frac{dp_0(t)}{dq}\Big|_{q=0} = t(r-1) - r < 0, \quad \frac{dp_0(t)}{dq}\Big|_{q=1} = \begin{cases} \frac{t}{2}\, r^{t/2} > 0, & \text{если } t \text{ – четное,} \\ -\frac{t+1}{2}\, r^{(t-1)/2} < 0, & \text{если } t \text{ – нечетное.} \end{cases}$$

Так как при четных t функция (20) имеет на концах отрезка $[0, 1]$ различные знаки, то в силу ее непрерывности заключаем, что по крайней мере в одной точке, принадлежащей интервалу $(0, 1)$, она равна нулю. При этом знаки производной $dp_0(t)/dq$ при $q = 0$ и $q = 1$ указывают на то, что хотя бы одна из указанных точек доставляет локальный минимум функции (17). Таким образом, *в четные моменты времени*

величина $p_0(t)$*, рассматриваемая как функция от вероятности угрозы* q*, имеет, по крайней мере, один локальный минимум на интервале* $(0, 1)$.

Возвращаясь к общему случаю произвольного числа угроз, отметим, что в определенном смысле он сводится к уже рассмотренному случаю одной угрозы. Действительно, согласно формулам (5) и (8) вероятность безопасного состояния системы, на которую воздействует n угроз с вероятностями $q_1, q_2, \ldots, q_n$, будет такой же, как и в случае системы с одной угрозой с вероятностью q и параметром защиты r, если положить, что

$$q = \sum_{i=1}^{n} q_i, \quad r = \frac{\sum\limits_{i=1}^{n} q_i r_i}{\sum\limits_{i=1}^{n} q_i}.$$

В частности, используя (17) и (18), нетрудно получить для вероятности $p_0(t)$ выражение, полиномиальное по переменным q_i и r_i:

$$p_0(t) = \sum_{k=1}^{t} \sum_{l=1}^{\min(k,t-k)} (-1)^{k+l} C_{t-l}^{k} C_{k}^{l} \left(\sum_{i=1}^{n} q_i r_i \right)^{l} \left(\sum_{j=1}^{n} q_j \right)^{k-l}.$$

Как функцию от вероятностей угроз $q_1, q_2, \ldots, q_n$ величину $p_0(t)$ следует рассматривать на выпуклой области $Q = \{(q_1, \ldots, q_n) \in \mathbb{R}_+^n \colon \sum_{i=1}^{n} q_i \le 1\}$. С помощью формул (19) нетрудно видеть, что имеют место следующие «граничные условия»:

$$p_0(t)|_{q_i=0} = 1, \quad p_0(t)|_{\sum q_i=1} = \begin{cases} 0, & \text{если } t \text{ – нечетно,} \\ \left(\sum\limits_{i=1}^{n} q_i r_i \right)^{t/2}, & \text{если } t \text{ – четно.} \end{cases}$$

Отметим также, что в отличие от случая одной угрозы, функция $p_0(t)$ при произвольном n в общем случае не имеет локальных минимумов внутри области Q. Тем не менее, она может иметь в этой области условные экстремумы. Их исследование — важная, хотя и технически сложная задача, решению которой будут посвящены наши последующие работы.

3. Время релаксации и построение допустимой области параметров защиты

Как было отмечено в первом разделе, по истечении длительного времени вероятность обнаружить систему в безопасном состоянии будет близкой к нулю. Принимая во внимание неизбежность подобного итога, мы, тем не менее, можем исследовать условия, при которых система будет находиться в безопасном состоянии как можно дольше. Естественно ожидать, что подобные условия будут сводиться к некоторым ограничениям, накладываемым на внутренние параметры системы $r_1, r_2, \ldots, r_n$.

Перейдем к строгим формулировкам. Будем рассматривать динамику системы на временах, удовлетворяющих неравенству (15); в этом случае с точностью до величин порядка ε мы можем приближенно считать

$$p_0(t) \approx w^{-1} \left(\frac{q_0 + w}{2} \right)^{t+1}. \tag{21}$$

В дальнейшем для нас будет важным то, что в рамках рассматриваемого приближения величина $p_0(t)$ является монотонно убывающей функцией t.

Временем релаксации τ назовем время, за которое вероятность безопасного состояния системы уменьшается в *два раза* (по сравнению с моментом времени $t = 0$). С помощью (21) из равенства $p_0(0)/p_0(\tau) = 2$ немедленно получаем

$$\tau = \log_{\frac{q_0+w}{2}} \frac{w}{2} - 1. \tag{22}$$

Пусть t_0 — фиксированный момент времени. Наша ближайшая задача будет заключаться в нахождении значений параметров защиты $r_1, \ldots, r_n$, при которых $\tau \geq t_0$. Другими словами, нас будут интересовать условия, при которых вероятность обнаружения системы в безопасном состоянии на временах, меньших или равных t_0, будет относительно большой.

Используя формулу (22), перепишем неравенство $\tau \geq t_0$ в виде

$$\frac{w}{2} \leq \left(\frac{q_0 + w}{2}\right)^{t_0+1}. \tag{23}$$

Будем рассматривать данное неравенство как ограничение на параметры защиты системы $r_1, \ldots, r_n$; так как последние входят только в выражение для w-параметра, нам необходимо решать неравенство (23) относительно переменной w. Нетрудно видеть, что соответствующее решение имеет вид

$$w \geq 2x^* - q_0, \tag{24}$$

где x^* — вещественный корень уравнения

$$x^{t_0+1} - x + \frac{q_0}{2} = 0, \tag{25}$$

принадлежащий отрезку $[q_0, 1]$.

Подставляя в (24) выражение для w-параметра (8), получаем ограничение на значения параметров защиты r_i:

$$\sum_{i=1}^{n} q_i r_i \geq x^*(x^* - q_0). \tag{26}$$

Вместе с неравенствами

$$r_i \leq 1, \quad i = 1, \ldots, n; \tag{27}$$

требование (26) определяет в пространстве значений параметров защиты выпуклую область $R_{t_0}(q_1, \ldots, q_n) \subset \mathbb{R}^n_+$, которую мы будем называть *допустимой*. Таким образом, только для значений параметров $r_1, \ldots, r_n$ из допустимой области $R_{t_0}(q_1, \ldots, q_n)$ время релаксации τ системы будет не меньшим, чем фиксированное значение t_0.

Приведем примеры построения допустимой области параметров защиты для случаев одной и двух угроз.

ПРИМЕР 1. В случае одной угрозы система характеризуется двумя параметрами: q и r. Как следует из неравенств (26) и (27), допустимая область $R_{t_0}(q)$ параметра защиты r представляет собой отрезок $[r^*, 1]$, где

$$r^* = \frac{x^*(x^* + q - 1)}{q}.$$

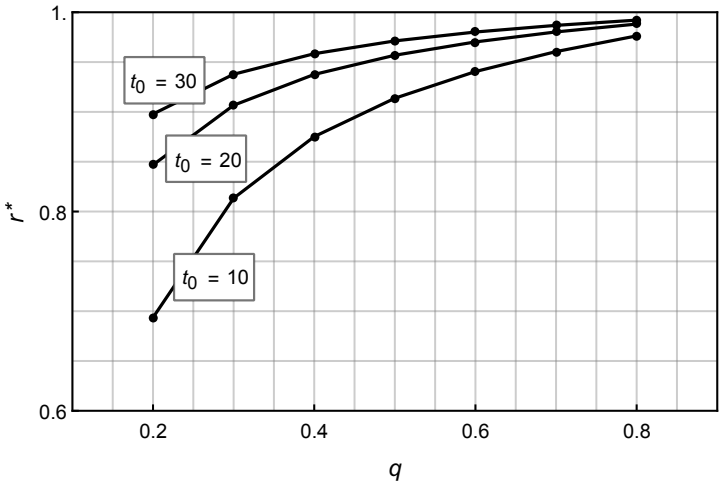


Рис. 4. Величина r^* как функция q при $t_0 = 10, 20, 30$

Fig. 4. The value r^* as a function of q for $t_0 = 10, 20, 30$

Здесь x^* — корень уравнения (25), принадлежащий отрезку $[1 - q, 1]$. На рис. 4 приведены результаты численного моделирования величины r^* как функции q для трех различных значений t_0. Видно, что с ростом q эта величина асимптотически стремится к единице, сужая, тем самым, допустимую область параметра защиты r.

ПРИМЕР 2. Рассмотрим систему с двумя угрозами, вероятности которых равны q_1 и q_2. Согласно (26) и (27) допустимая область параметров защиты — это выпуклая область

$$R_{t_0}(q_1, q_2) = \{(r_1, r_2) \in \mathbb{R}^2_+ \colon r_1 \le 1,\ r_2 \le 1,\ q_1 r_1 + q_2 r_2 \le x^*(x^* - 1 + q_1 + q_2)\},$$

где x^* — корень уравнения (25), принадлежащий отрезку $[q_1 + q_2, 1]$. В качестве примера на рис. 5 графически изображены допустимые области при $q_1 = 0.25$ и $q_2 = 0.45$ для моментов времени $t_0 = 10, 20, 30$.

В заключение обсудим возможное приложение полученных результатов к задаче выбора оптимального набора значений параметров защиты. Данная задача имеет важное прикладное значение, например, при проектировании и разработке систем защиты информации [17, 18].

С помощью описанного выше алгоритма мы можем сузить множество возможных значений параметров защиты до множества $R_{t_0}(q_1, \ldots, q_n)$, однако в пределах этой области выбор их более ничем не ограничен. Для дальнейшей конкретизации значений $r_1, \ldots, r_n$ нам необходимы дополнительные условия, сужающие область параметров защиты, например, до одной конкретной точки. Естественной постановкой задачи, гарантирующей единственность соответствующего решения, является задача поиска минимума (или максимума) некоторой целевой функции $I(r_1, \ldots, r_n)$, рассматриваемой на области $R_{t_0}(q_1, q_2, \ldots, q_n)$. В простейшем варианте целевая функция может быть выбрана линейной по переменным $r_1, \ldots, r_n$

$$I(r_1, \ldots, r_n) = \sum_{i=1}^{n} c_i r_i, \tag{28}$$

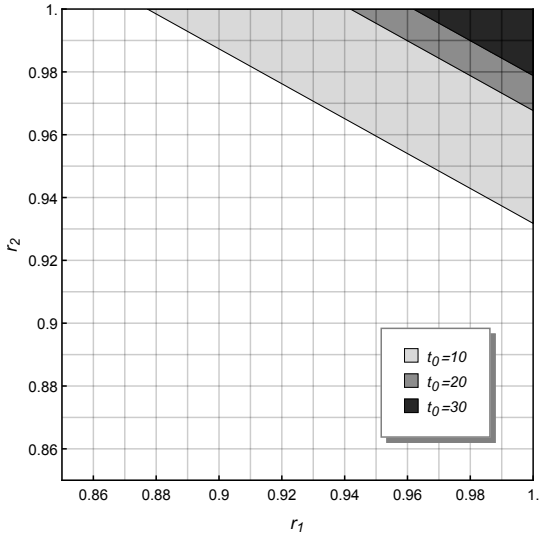


Рис. 5. Допустимая область $R_{t_0}(0.25, 0.45)$ при $t_0 = 10, 20, 30$

Fig. 5. The permitting domain $R_{t_0}(0.25, 0.45)$ for $t_0 = 10, 20, 30$

где c_i — некоторые заданные коэффициенты, интерпретация которых (как и целевой функции I) зависит от решаемой оптимизационной задачи. В такой постановке задача поиска минимума (или максимума) целевой функции (28) на выпуклой области $R_{t_0}(q_1, \ldots, q_n)$ представляет собой стандартную задачу линейного программирования, решению которой посвящена обширная литература (см., например, [19]).

Заключение

Рассмотрена модель угроз информационной безопасности, описываемая в терминах марковских процессов. Динамика модели представляется последовательностью сбоев и восстановлений компьютерной системы, происходящих в результате воздействия случайных внешних угроз. Получены явные аналитические формулы для вероятностей состояний системы, обсуждаются их некоторые предельные случаи и анализируется поведение системы на больших временах. Исследована зависимость вероятности безопасного состояния системы от вероятностей угроз; в частности, показано, что эта зависимость качественно различается для четных и нечетных моментов времени. Также введена важная характеристика модели — время релаксации, с помощью которой определяется допустимая область значений параметров защиты системы и приводится алгоритм ее построения.

Список литературы / References

[1] Ye N. et al., “Robustness of the Markov-Chain Model for Cyber-Attack Detection”, *IEEE Transactions on Reliability*, **53**:1 (2004), 116–123.

[2] Fava D. et al., "Projecting Cyberattacks through Variable-Length Markov Models", *IEEE Transactions on Information Forensics and Security*, **3**:3 (2008), 359–369.

[3] Piétre-Cambacédès L., Bouissou M., "Beyond Attack Trees: Dynamic Security Modeling with Boolean Logic Driven Markov Processes (BDMP)", *Proceedings of the 2010 European Dependable Computing Conference*, IEEE Computer Society, 2010, 199–208.

[4] Далингер Я. М. и др., "Математические модели распространения вирусов в компьютерных сетях различной структуры", *Информатика и системы управления*, 2011, № 4, 3–11; [Dalinger Ya. M. et al., "The mathematical models of the spreading of viruses in computer networks with the diferent structures", *Information Science and Control Systems*, 2011, № 4, 3–11, (in Russian).]

[5] Ye N., "A Markov Chain Model of Temporal Behavior for Anomaly Detection", *Proceeding on the 2000 IEEE Systems, Man, and Cybern. Information Assurance and Security Workshop*, IEEE Computer Society, 2000, 171–174.

[6] Kovalev S. M., Sukhanov A. V., "Anomaly detection based on Markov chain model with production rules", *Software and Systems*, **107**:3 (2014), 40–43.

[7] Богатырев В. А. и др., "Оптимизация интервалов проверки информационной безопасности систем", *Научно-технический вестник информационных технологий, механики и оптики*, 2014, № 5 (93), 119–125; [Bogatyrev V. A. et al., "Intervals optimization of systems information security inspection", *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2014, № 5 (93), 119–125, (in Russian).]

[8] Щеглов К. А. и др., "Математические модели эксплуатационной информационной безопасности", *Вопросы защиты информации*, 2014, № 3, 52–65; [Shcheglov K. A. et al., "Mathematical models of operational information security", *Information security questions*, 2014, № 3, 52–65, (in Russian).]

[9] Vobbilisetty R. et al., "Classic Cryptanalysis Using Hidden Markov Models", *Cryptologia*, **41**:1, 1–28.

[10] Austin T. H. et al., "Exploring Hidden Markov Models for Virus Analysis: a Semantic Approach", *Proceedings of the 2013 46th Hawaii International Conference on System Sciences*, IEEE Computer Society, 2013, 5039–5048.

[11] Клименко Е. С., Росенко А. П., "Марковская модель оценки влияния внутренних угроз на безопасность конфиденциальной информации", *Известия ЮФУ. Технические науки*, 2007, № 4(76), 123–126; [Klimenko E. S., Rosenko A. P., "Markovskaya model otsenki vliyaniya vnutrennikh ugroz na bezopasnost konfidentsialnoy informatsii", *Izvestiya SFedU. Engineering Sciences*, 2007, № 4(76), 123–126, (in Russian).]

[12] Росенко А. П., "Математическое моделирование влияния внутренних угроз на безопасность конфиденциальной информации, циркулирующей в автоматизированной информационной системе", *Известия ЮФУ. Технические науки*, 2008, № 8(85), 71–81; [Rosenko A. P., "Mathematical Modelling of Internal Threats on Safety of the Confidential Information Circulating in Automated Information System Availability", *Izvestiya SFedU. Engineering Sciences*, 2008, № 8(85), 71–81, (in Russian).]

[13] Щеглов К. А., Щеглов А. Ю., "Марковские модели угрозы безопасности информационной системы", *Известия высших учебных заведений. Приборостроение*, **58**:12 (2015), 957–965; [Shcheglov K. A., Shcheglov A. Yu., "Markov models for informational system security threat", *Izvestiya Vysshikh Uchebnykh Zavedeniy. Priborostroenie*, **58**:12 (2015), 957–965, (in Russian).]

[14] Щеглов К. А., Щеглов А. Ю., "Моделирование угрозы безопасности информационной системы с использованием аппроксимирующих функций", *Известия высших учебных заведений. Приборостроение*, **59**:1 (2016), 50–59; [Shcheglov K. A., Shcheglov A. Yu., "Modeling of information system security threat using approximating functions", *Izvestiya Vysshikh Uchebnykh Zavedeniy. Priborostroenie*, **59**:1 (2016), 50–59, (in Russian).]

[15] Беляев Ю. К. и др., *Математические методы в теории надежности*, Наука, 1963, 524 с.; English transl.: Gnedenko B. V. et al., *Mathematical Methods of Reliability Theory*, Academic Press, 1969, 503 pp.

[16] Rausand M, Hoyland A., *System Reliability Theory: Models, Statistical Methods, and Applications*, John Wiley & Sons, 2004, 664 pp.

[17] Овчинников А. И. и др., "Математическая модель оптимального выбора средств защиты от угроз безопасности вычислительной сети предприятия", *Вестник Московского государственного технического университета им. Н.Э. Баумана. Серия «Приборостроение»*, 2007, № 3, 115–121; [Ovchinnikov A. I. et al., "Mathematical Model of Optimal Selection of Aids of Protection Against Threats for Safety of Enterprise Computer Network", *Herald of the Bauman Moscow State Technical University. Series Instrument Engineering*, 2007, № 3, 115–121, (in Russian).]

[18] Завгородний В. И., "Системное управление информационными рисками. Выбор механизмов защиты", *Проблемы управления*, 2009, № 1, 53–58; [Zavgorodniy V. I., "System management of information risks: choice of mechanisms for protection against information risks", *Problemy Upravleniya*, 2009, № 1, 53–58, (in Russian).]

[19] Юдин Д. Б., Гольштейн Е. Г., *Линейное программирование. Теория, методы и приложения*, Наука, 1969, 424 с.; [Yudin D. B., Gol'shteyn E. G., *Lineynoe programmirovanie. Teoriya, metody i prilozheniya*, Nauka, 1969, 424 pp., (in Russian).]

Magazev A. A., Tsyrulnik V. F., "Investigation of a Markov Model for Computer System Security Threats", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 445–458.

DOI: 10.18255/1818-1015-2017-4-445-458

Abstract. In this work, a model for computer system security threats formulated in terms of Markov processes is investigated. In the framework of this model the functioning of the computer system is considered as a sequence of failures and recovery actions which appear as results of information security threats acting on the system. We provide a detailed description of the model: the explicit analytical formulas for the probabilities of computer system states at any arbitrary moment of time are derived, some limiting cases are discussed, and the long-run dynamics of the system is analysed. The dependence of the security state probability (i.e. the state for which threats are absent) on the probabilities of threats is separately investigated. In particular, it is shown that this dependence is qualitatively different for odd and even moments of time. For instance, in the case of one threat the security state probability demonstrates non-monotonic dependence on the probability of threat at even moments of time; this function admits at least one local minimum in its domain of definition. It is believed that the mentioned feature is important because it allows to locate the most dangerous areas of threats where the security state probability can be lower then the permissible level. Finally, we introduce an important characteristic of the model, called the relaxation time, by means of which we construct the permitting domain of the security parameters. Also the prospects of the received results application to the problem of finding the optimal values of the security parameters is discussed.

Keywords: computer system, security threat, Markov process

On the authors:
Alexey A. Magazev, orcid.org/0000-0002-8725-9183, PhD,
Omsk State Technical University, 11 Mira pr., Omsk, 644050, Russia,
e-mail: magazev@mail.ru

Valeria F. Tsyrulnik, orcid.org/0000-0002-6875-7216, student,
Omsk State Technical University, 11 Mira pr., Omsk, 644050, Russia,
e-mail: lera.tsyrulnik@mail.ru

Моделирование и анализ информационных систем. Т. 24, № 4 (2017), с. 459–480
Modeling and Analysis of Information Systems. Vol. 24, No 4 (2017), pp. 459–480

DOI: 10.18255/1818-1015-2017-4-459-480

УДК 519.682.6+004.021

Использование журналов событий для локальной корректировки моделей процессов

Мицюк А. А.[1], **Ломазова И. А.**[1], **ван дер Аалст В. М. П.**

получена 17 июля 2017

Аннотация. В ходе жизненного цикла информационной системы (ИС) ее реальное поведение может перестать соответствовать исходной модели системы. Между тем для поддержки системы очень важно иметь актуальную модель, отражающую текущее поведение системы. Для корректировки модели можно использовать информацию из журнала событий системы. Журналы событий процессно-ориентированных информационных систем содержат запись истории исполнения поддерживаемых процессов в виде более или менее детальных списков событий. Такие журналы, как правило, записываются всеми современным ИС. Эта информация может использоваться для анализа реального поведения ИС и ее усовершенствования. В работе рассматривается задача корректировки (исправления) модели процесса на основе информации из журнала событий. Исходными данными для этой задачи являются первоначальная модель процесса в виде сети Петри и журнал событий. Результатом корректировки должна быть новая модель процесса, лучше отображающая реальное поведение ИС, чем исходная модель. Актуальная модель может быть построена и полностью заново, например, с помощью одного из известных алгоритмов автоматического синтеза модели процесса по журналу событий. Однако структура исходной модели при этом может полностью измениться. Полученную модель будет трудно сопоставить с прежней моделью процесса, что затруднит ее понимание и анализ. Поэтому при корректировке модели важно по возможности сохранить ее прежнюю структуру. Предлагаемый в настоящей работе алгоритм корректировки модели основан на принципе «разделяй и властвуй». Исходная модель процесса декомпозируется на фрагменты. Для каждого из фрагментов проверяется, соответствует ли он актуальному журналу событий. Фрагменты, для которых выявлены несоответствия, заменяются на заново синтезированные. Новая модель собирается из фрагментов путем слияния переходов. Проведенные эксперименты показывают, что наш алгоритм корректировки дает хорошие результаты, если применяется для исправления локальных несоответствий. Работа содержит описание алгоритма, формальное обоснование его корректности, а также результаты экспериментального тестирования на искусственных примерах.

Ключевые слова: извлечение и анализ процессов, исправление моделей процессов, сети Петри, декомпозиция моделей процессов, разделяй и властвуй

Для цитирования: Мицюк А. А., Ломазова И. А., ван дер Аалст В. М. П., "Использование журналов событий для локальной корректировки моделей процессов", *Моделирование и анализ информационных систем*, **24**:4 (2017), 459–480.

Об авторах:
Мицюк Алексей Александрович, orcid.org/0000-0003-2352-3384, науч. сотр.,
Национальный исследовательский университет «Высшая школа экономики», лаборатория ПОИС
ул. Мясницкая, 20, г. Москва, 101000 Россия, e-mail: amitsyuk@hse.ru

Ломазова Ирина Александровна, orcid.org/0000-0002-9420-3751, д-р физ.-мат. наук, профессор
Национальный исследовательский университет «Высшая школа экономики», лаборатория ПОИС
ул. Мясницкая, 20, г. Москва, 101000 Россия, e-mail: ilomazova@hse.ru

Вил М.П. ван дер Аалст, orcid.org/0000-0002-0955-6940, профессор
Технический университет Эйндховена (TU/e), группа «Архитектура информационных систем»,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, e-mail: w.m.p.v.d.aalst@tue.nl

Благодарности:
[1]Исследование выполнено в рамках Программы фундаментальных исследований Национального исследовательского университета «Высшая школа экономики» (НИУ ВШЭ).

Введение

Журналы событий процессно-ориентированных информационных систем (ИС) содержат запись истории исполнения поддерживаемых процессов в виде более или менее детальных списков событий. Такие журналы, как правило, записываются всеми современным ИС. Эта информация может использоваться для анализа реального поведения ИС и ее усовершенствования [3].

В данной работе рассматривается задача корректировки (исправления) модели процесса на основе информации из журнала событий. Исходными данными для этой задачи являются (1) модель процесса для корректировки и (2) журнал (лог) событий. Необходимость в корректировке модели возникает в том случае, когда разработанная ранее модель процесса уже не соответствует его текущему исполнению. Такая ситуация возникает достаточно часто, так как в ходе жизненного цикла системы в нее могут вноситься изменения.

Для моделирования процессов могут использоваться различные формализмы. В данной работе мы рассматриваем модели в виде классических сетей Петри. В свою очередь, журнал (лог) событий представляется как набор последовательностей записей о событиях, где запись о событии содержит информацию о выполняемом действии и включает отметку о времени его выполнения. Результатом корректировки должна быть новая модель процесса, лучше отображающая реальное поведение ИС, чем исходная модель.

В литературе описаны различные методы автоматического синтеза моделей процессов на основе журналов событий [6,7,10,20,23,25,27,41,42]. Однако синтез модели по журналу событий «с нуля» может дать модель, которую трудно сопоставить с исходной моделью — это затруднит ее понимание и анализ. Поэтому в задаче корректировки нужно не только исправить несоответствие между моделью и реальным поведением системы, но и по возможности сохранить прежнюю структуру модели.

Общая идея подхода, применяемого в данной работе, состоит в корректировке модели *по частям*. Предлагается сначала *декомпозировать* исходную модель на фрагменты, выявить фрагменты, не соответствующие поведению, представленному в логе, и заменить их на новые, исправленные фрагменты. Если изменения в поведении системы являются локальными, и их не много, то большая часть корректируемой модели остается неизменной. Общая схема такого подхода и критерии его применимости были представлены нами в [28].

Данная статья имеет следующую структуру. Общая постановка задачи исправления моделей процессов и иллюстрирующий пример приводятся в разделе 1. Раздел 3. содержит используемые в работе определения и предварительные сведения. В разделе 4. описывается базовый алгоритм корректировки модели путем разбиения ее на фрагменты. Усовершенствованная версия алгоритма представлена в разделе 5.

Раздел 6. содержит результаты экспериментов по исправлению моделей. Обзор других работ по исправлению моделей процессов дан в разделе 2.

1. Исправление моделей процессов

Описание постановки задачи по корректировке модели процесса начнем с мотивирующего примера.

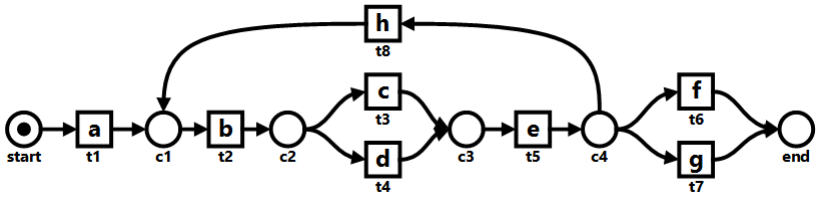


Рис. 1. Пример: обработка запросов на компенсацию затрат *(имена переходов: a = Зарегистрировать запрос; b = Проверка билета; c = Рядовая проверка; d = Экстраординарная проверка; e = Принятие решения; f = Отправка сообщения об отказе; g = Выплата компенсации затрат; h = Повторное рассмотрение запроса)*

Fig. 1. Example: the compensation requests processing *(transition labels: a = Register request; b = Check ticket; c = Examine casually; d = Examine thoroughly; e = Decide; f = Send rejection letter; g = Pay compensation; h = Re-initiate request)*

Рассмотрим модель процесса в виде сети Петри на Рис. 1. Данный пример модели процесса приводится в [3] и часто используется для иллюстрации методов синтеза моделей процессов. Сеть Петри на этом рисунке представляет процесс обработки запросов на компенсацию затрат в некоторой компании. Такая модель может быть, в частности, автоматически синтезирована по следующему логу событий: $[\langle a, b, c, d, e, f, g\rangle, \langle a, b, d, c, e, f, g\rangle, \langle a, b, c, d, e, h, b, d, c, g, f\rangle]$. Этот лог состоит из трех трасс, каждая из которых есть последовательность событий, представленных именами выполняемых действий. Время выполнения событий используется только для их упорядочения и может быть опущено.

Предположим, что данный лог событий был дополнен трассой $\langle a, c, b, e, f, g\rangle$, которая не может быть воспроизведена в модели. В этой новой трассе действие «Проверка билета» b выполняется после действия «Рядовая проверка» c, что не соответствует порядку их выполнения в исходной модели на Рис. 1, где любые проверки выполняются строго после проверки билета. Таким образом, исходная модель не соответствует *реальности*, представленной для нас логом событий.

Для того, чтобы получить модель, отражающую реальное исполнение процесса, можно применить два подхода: заново синтезировать модель по логу событий или выполнить корректировку имеющейся модели на основании информации о новом поведении процесса. Второй подход предпочтительней в тех случаях, когда исходная модель имеет определенную *ценность*, например ясную компактную структуру. Пример возможного исправления модели показан на Рис. 2. Большая часть графической структуры исходной модели осталась прежней. Далее мы представим алгоритм, который позволит выполнять подобную корректировку автоматически.

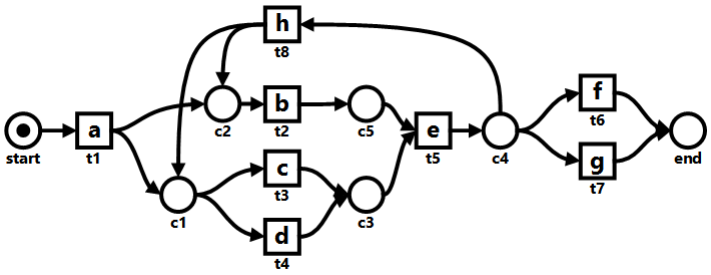


Рис. 2. Пример: модель после корректировки
Fig. 2. Example: the corrected model

Формулировка задачи исправления модели процесса. Даны сеть Петри N — это исходная модель процесса, и лог событий L, представляющий текущее поведение процесса. Нужно построить сеть Петри N^r такую, что

– модель N^r позволяет выполнить все трассы из L (идеальное соответствие модели и лога);

– модель N^r допускает не «слишком много» исполнений, не представленных в логе (высокая точность модели);

– модели N и N^r имеют похожую графовую структуру (сходство моделей).

2. Обзор работ по теме исследования

Идея исправления моделей процессов на основании использования информации из логов событий была предложена в [16] и развита в [17], где авторы предлагают способ повышения степени соответствия (*fitness*) модели процесса в виде сети потоков работ (подкласс сетей Петри) логу событий. Для этого все трассы лога событий разделяются на соответствующие и не соответствующие модели. Трассы, не соответствующие модели, рассматриваются как специальный вариант исполнения, для которого строится модель подпроцесса, встраиваемого затем в исходную модель.

Ещё один способ исправления моделей бизнес-процессов, называемый «автоматизированной коррекцией ошибок», был предложен в [19]. Авторы используют метод симуляции отжига для поиска оптимального набора операций исправления, что позволяет им находить одну или несколько сетей потоков работ, моделирующих заданный процесс без ошибок. Отметим, что авторы не используют при этом информации из логов событий.

А. Поливаный и др. предложили в [31] метод исправления моделей процессов, называемый «impact-driven repair». Метод основан на учете влияния, оказываемого различными операциями исправления на свойства итоговой модели. Этот метод может применяться, когда количество допустимых операций исправления ограничено, а различные операции имеют неодинаковое влияние на ценность итоговой модели. В такой постановке задача исправления сети Петри сводится к оптимизационной задаче. Авторы приводят результаты большого количества экспериментов, произведенных для выбора наиболее эффективного алгоритма.

В [12] предлагается способ усовершенствования *хорошо структурированных* (well-structured) моделей процессов на основе специального генетического алгоритма, в

котором наряду с четырьмя классическими для process mining метриками качества: соответствие (fitness), точность (precision), обобщенность (generalization), простота (simplicity) учитывается также и *сходство* (similarity) синтезируемой модели с исходной.

Основная идея применения принципа *разделяй и властвуй* для синтеза моделей процессов и проверки согласованности модели с логами событий описана в [2, 39]. Практические вопросы применения декомпозиции моделей процессов рассматриваются в [1, 38] и других работах. В них исследуется метод эффективного автоматического синтеза моделей процессов, использующий разбиение лога событий на части.

В работах [21, 22] рассматривается задача так называемой *ре-композиции* (recomposition) и исследуется выбор наиболее подходящего размера фрагмента для задачи автоматического синтеза модели процесса. Декомпозиция модели процесса и лога событий может применяться также для повышения производительности оценки согласованности модели и лога [29, 30, 35]. Применение модульных подходов при решении задач анализа процессов также рассматривалось в некоторых работах, в частности, в [5, 26, 33].

Задача автоматического *упрощения* модели родственна задаче исправления. Части модели, используемые в малом количестве трасс, могут удаляться из модели. Это упрощает модель при некотором снижении соответствия логу событий. Методы упрощения модели предложены, например, в [18, 32]. В [20] также рассматривается вопрос упрощения *гибкой* (fuzzy) модели процесса путём удаления несущественных частей модели с использованием частотных метрик.

Предлагаемый в данной работе метод отличается от подходов, описанных выше. В нашем подходе декомпозиция модели используется для последующей замены некоторых фрагментов. Если расхождения между логом и моделью локальны, то исправленная модель будет отличаться от исходной только замененными фрагментами. Это особенно важно в случаях, когда исходная модель была построена экспертом, а потому хорошо воспринимается человеком. Заново синтезированная модель может быть лишена такого достоинства. Другой особенностью нашего подхода является то, что набор действий процесса не изменяется при исправлении. Это может быть как минусом (если в реальности он изменялся), так и плюсом (не будут добавлены никакие искусственные действия).

3. Предварительные сведения

В этом разделе приводятся основные определения и понятия, используемые в работе.

Мультимножества, функции, последовательности. Через $\mathbb{N}$ обозначим множество натуральных чисел с нулем. Пусть S — некоторое множество. Мультимножеством над S называется функция $B : S \to \mathbb{N}$, которая сопоставляет каждому элементу из S число его вхождений в B. Через $\mathcal{B}(S)$ будем обозначать множество всех мультимножеств над S. Конечное мультимножество можно задать перечислением его элементов с указанием их кратности, например, $B = [e, e, e, c, d, d] = [e^3, c, d^2]$ обозначает мультимножество, содержащее три экземпляра элемента e, один элемент

c и два экземпляра элемента d. Обычные операции над множествами распространяются и на мультимножества естественным образом с сохранением стандартной нотации.

Для функции $f\colon X \to Y$ через $dom(f)$ будем обозначать область ее определения. Нотация $f\colon X \nrightarrow Y$ используется для частичных функций. Для функции $f\colon X \nrightarrow Y$ ее проекция на множество $Q \subseteq X$ определяется как функция $f\restriction_Q\colon Q \nrightarrow Y$ такая, что $\forall x \in Q\colon f\restriction_Q(x) = f(x)$. Определение проекции распространяется и на мультимножества: $[e^3, c, d^2]\restriction_{\{c,d\}} = [c, d^2]$.

Множество всех конечных последовательностей над множеством X будем обозначать через X^*. Для записи последовательностей будем использовать треугольные скобки: $\sigma = \langle x_1, x_2, ..., x_n\rangle$ — последовательность длины n. Конкатенацию двух последовательностей σ_1 и σ_2 будем обозначать как $\sigma_1 \cdot \sigma_2$, а проекцию последовательности σ на множество Q — как $\sigma\restriction_Q$.

Модель процесса. В данной работе для моделирования процессов используются классические сети Петри, а точнее, более узкий класс моделей — сети потоков работ (WF-сети).

Сеть Петри — это тройка (P, T, F), где P и T — непересекающиеся множества позиций и переходов, а $F\colon (P \times T) \cup (T \times P) \to \mathbb{N}$ — функция потока. Через $\bullet t = \{p | F(p, t) \neq 0\}$ будем обозначать множество позиций, входных для перехода $t \in T$, а через $t\bullet = \{p | F(t, p) \neq 0\}$ — множество выходных для t позиций.

Помеченная сеть Петри — это четверка (P, T, F, l), где $l\colon T \to \mathcal{U}_A \cup \{\tau\}$ — функция пометки, ставящая каждому переходу из сети Петри имя действия из некоторого заранее заданного множества $\mathcal{U}_A$. Если $l(t) = \tau$, переход t называют *невидимым*. В противном случае переход t называют *видимым*.

Текущее состояние сети Петри задается *разметкой* $M\colon P \to \mathbb{N}$. Переход $t \in T$ *активен* в разметке M тогда и только тогда, когда $\forall p \in P\colon M(p) \geq F(p, t)$. Активный переход t может *сработать*, изменив разметку сети на новую $M'(p) = M(p) - F(p, t) + F(t, p)$ для каждого $p \in P$. Такое срабатывание перехода будем обозначать как $M \xrightarrow{t} M'$.

Определение 1. ***Сеть потоков работ*** *(WF-сеть) — это помеченная сеть Петри $N = (P, T, F, l)$ с выделенными начальной разметкой M_i и конечной разметкой M_o, в которой*
(1) есть ровно одна позиция-исток $i \in P$ такая, что $\bullet i = \emptyset$ и $M_i = [i]$,
(2) есть ровно одна позиция-исток $o \in P$ такая, что $o\bullet = \emptyset$ и $M_o = [o]$,
(3) каждая позиция или переход $n \in P \cup T$ лежит на пути из i в o.

На Рис. 1 приводится пример сети потоков работ. Позиции изображаются кругами, переходы — квадратами, а начальная разметка — черным токеном в позиции-истоке.

Множество всех видимых переходов в сети N будем обозначать через $T_v(N)$. Будем говорить, что имя перехода *уникально*, если оно помечает в сети ровно один переход. Множество видимых переходов в сети N с уникальными именами будем обозначать через $T_v^u(N)$. Сеть $N^U = N^1 \cup N^2$ — это объединение двух сетей потоков работ, которая строится путем обычного объединения множеств позиций, переходов и отношения потока: $N^1 \cup N^2 = (P^1 \cup P^2, T^1 \cup T^2, F^1 \cup F^2, l^u)$, где $l^u \in (T^1 \cup T^2) \nrightarrow \mathcal{U}_A$

— объединение l^1 и l^2, $dom(l^u) = dom(l^1) \cup dom(l^2)$, а $l^u(t) = l^1(t)$, если $t \in dom(l^1)$, и $l^u(t) = l^2(t)$, если $t \in dom(l^2) \setminus dom(l^1)$. В данной работе мы рассматриваем только модели с уникальными именами переходов.

Лог событий. Множество всех действий процесса обозначим через $A \subseteq \mathcal{U}_A$. Поскольку для построения модели потока работ нам важны только имена действий и их последовательность, будем считать, что событие в логе — это имя действия, и далее будем использовать термины события и (имени) действия как синонимы.

Определим *трассу* σ как конечную последовательность событий из A, то есть $\sigma \in A^*$. Конечное мультимножество трасс L будем называть *логом событий* или простро *логом*, $L \in \mathcal{B}(A^*)$. Например, $L = [\langle a, b, c, d, e\rangle^3, \langle a, b, a, d, e\rangle^5, \langle a, d, c, b, e\rangle]$ — лог событий с тем же множеством действий, что и модель на Рис. 1. Проекция лога событий $L = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ на множество действий B — это лог событий $L\restriction_B = [\sigma_1\restriction_B, \sigma_2\restriction_B, \ldots, \sigma_n\restriction_B]$, где $\sigma_i\restriction_B$ — проекция трассы σ_i на B. Такой лог событий будем называть *суб-логом* лога L.

Проверка согласованности. При оценке качества модели обычно применяются четыре критерия: *соответствие* (fitness), *точность* (precision), *обобщенность* (generalization), *простота* (simplicity) [3]. Два первых критерия используются для измерения степени согласованности модели и лога. Мера соответствия определяет, в какой степени трассы лога могут быть исполнены в модели. Точность определяет, в какой мере модель может порождать поведение, не представленное в логе. Обобщенность и простота характеризуют качество собственно модели. Обобщенность определяет меру абстрактности модели, а простота — ее компактность. Обсуждение смысла и значения указанных критериев можно найти, например, в [3, 11].

При исправлении моделей процессов прежде всего важны критерии соответствия между моделью и логом. Важен также критерий простоты — модель не должна слишком усложняться после ее корректировки. Помимо этого будет оцениваться степень различия между исходной и скорректированной моделями.

Пусть N — сеть потоков работ с именами переходов из A, начальной разметкой $[i]$ и конечной разметкой $[o]$, а σ — трасса над A. Будем говорить, что трасса $\sigma = a_1, \ldots, a_k$ *идеально соответствует* (perfectly fits) модели N, если существует последовательность срабатываний переходов сети $[i] = m_0 \xrightarrow{t_1} \ldots \xrightarrow{t_k} m_{k+1} = [o]$ такая, что последовательность действий $\lambda(t_1), \lambda(t_2), \ldots, \lambda(t_k)$ после удаления всех невидимых переходов совпадает с σ. Лог событий L идеально соответствует сети N, если каждая трасса из L идеально соответствует N. Например, набор трасс $[\langle a, b, c, e, f\rangle, \langle a, b, c, e, h, b, d, e, g\rangle, \langle a, b, d, e, g\rangle]$ идеально соответствует модели, показанной на Рис. 1.

В литературе представлено достаточно большое количество методов проверки согласованности [4, 8, 9, 13, 15, 29, 30, 40]. В данной работе для оценки соответствия мы используем методику, основанную на выравниваниях [8]. *Выравнивание* — это специальное соответствие между трассой и возможной последовательностью срабатываний переходов сети Петри. Пример двух выравниваний показан на Рис. 3. Верхний ряд каждого выравнивания соответствует трассе из лога событий, тогда как нижний ряд соответствует исполнению модели. Действию в трассе должен соответствовать переход, помеченный этим действием. Если действию в трассе не

удается сопоставить соответствующий переход в исполнении сети, то либо в трассе, либо в исполнении сети добавляется $\gg$ («пропуск»).

log	a	b	c	d	e	f	g
model	a	b	c	d	e	f	g
	t_1	t_2	t_3	t_4	t_5	t_6	t_7

log	a	$\gg$	d	c	b	e	f	g
model	a	b	d	c	$\gg$	e	f	g
	t_1	t_2	t_4	t_3		t_5	t_6	t_7

Рис. 3. Выравнивания (слева: идеальное соответствие; справа: несоответствие)
Fig. 3. Alignments (left: perfectly fitting; right: unfitting)

Левое выравнивание на Рис. 3 представляет идеальное соответствие. В правом выравнивании добавлены два пропуска, так как трасса не может быть исполнена на модели.

Очевидно, что для одной и той же пары трасса — исполнение можно построить различные выравнивания. Хорошее (*оптимальное*) выравнивание — это выравнивание с минимальным числом пропусков. В [8] рассматриваются способы построения оптимальных выравниваний, которые используются для оценки соответствия модели и лога событий.

Автоматический синтез моделей процессов. Для исправления фрагментов модели, не соответствующих логу, мы применяем методы автоматического синтеза моделей процессов (process discovery). Достаточно большое число таких алгоритмов с разными характеристиками были описаны ранее, а именно [6,7,10,20,23–25,41,42] и другие.

В данной работе используется метод *индуктивного синтеза* (Inductive Miner) [24]. Этот алгоритм, запущенный с определенными параметрами (в данной работе мы используем настройку *inductive miner incompleteness*), гарантирует идеальное соответствие получающейся модели. Мы также используем для тестирования нашего подхода алгоритм автоматического синтеза, сводящий задачу построения модели к задаче целочисленного линейного программирования (ILP Miner) [41]. Этот алгоритм также гарантирует идеальное соответствие построенной модели логу событий, если запущен с правильными параметрами.

4. Корректировка моделей процессов с разбиением на фрагменты

В [28] нами была предложена обобщенная модульная схема исправления моделей процессов, в которой используется подход *разделяй и властвуй*. Исправление модели процесса делится на несколько шагов: (1) декомпозиция, (2) выбор, (3) починка, (4) композиция, (5) оценка. В данной работе мы описываем конкретную реализацию этой схемы, определяемую следующим образом.

Пусть $\mathcal{U}_A$ — множество действий процесса, а $\mathcal{U}_N$ — множество всех помеченных сетей потоков работ, пометки на переходах которых из $\mathcal{U}_A$.

Определим **алгоритм исправления** как процедуру, получающую на вход лог событий $L \in \mathcal{B}(\mathcal{U}_A^*)$ и помеченную сеть потоков работ $N \in \mathcal{U}_N$, и выдающую в каче-

стве результата своей работы сеть Петри $N^r = ModularRepair(L, N)$, которая идеально соответствует логу L, т.е. $fitness(L, N^r) = 1$.

В качестве процедурных параметров схемы починки используются:

- $eval \in (\mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \to [0; 1]$ — функция оценки, основанная на использовании выравниваний [8],
- $repair \in \mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \to \mathcal{U}_N$ — индуктивный алгоритм автоматического синтеза модели процесса [24], либо алгоритм, основанный на решении задачи линейного программирования [41],
- $split \in \mathcal{U}_N \to \mathcal{P}(\mathcal{U}_N)$ — алгоритм максимальной декомпозиции, описанный в [28],
- а $compose \in \mathcal{P}(\mathcal{U}_N) \to \mathcal{U}_N$ — парный ему алгоритм композиции, объединяющий фрагменты сети путем склейки через граничные переходы с одинаковыми пометками.

Неформально алгоритм исправления можно описать следующим образом. Исходная модель декомпозируется на несколько фрагментов. Для каждого фрагмента с помощью проекции получается соответствующий ему суб-лог событий. Каждая пара (подсеть; суб-лог) оценивается с точки зрения соответствия друг другу. Фрагменты, для которых уровень соответствия недостаточен, заменяются с использованием алгоритма автоматического синтеза модели процесса. Результирующая модель получается путём объединения фрагментов по граничным переходам.

В работе [28] описан алгоритм, строящий т. н. «максимальную» декомпозицию, предложенную в [2]. Максимальная декомпозиция предназначена для разбиения сети на подсети следующим образом. Все узлы фрагмента делятся на граничные и внутренние. Внутренними узлами могут быть позиции, невидимые переходы, а также переходы, имеющие в исходной сети одинаковые пометки. Граничными узлами могут быть только видимые переходы с уникальными (в исходной сети) пометками. Максимальной является такая декомпозиция, в которой для каждого из фрагментов выполняется следующее правило. Внутренний узел фрагмента может быть связан дугами либо с граничными узлами фрагмента, либо с аналогичным внутренним узлом, тогда как граничный узел может быть связан только с внутренними узлами одного фрагмента. Все видимые переходы с уникальными (в исходной сети) пометками делятся пополам между двумя или более фрагментами (по ним происходит разбиение). Начальная разметка сети разбивается очевидным образом. Позиция-исток попадает ровно в один из фрагментов разбиения вместе с содержащимся в ней токеном. Максимально декомпозированная сеть может быть собрана (композирована) путём слияния граничных переходов с одинаковыми именами из разных фрагментов. На Рис. 5 показана максимальная декомпозиция модели, изображенной на Рис. 1.

Самое важное свойство максимальной декомпозиции состоит в том, что это *валидная декомпозиция* (valid decomposition). Валидная декомпозиция для сетей Петри определяется в работе [2]. Мы используем версию определения для сетей потоков работ.

Определение 2. *Пусть $N \in \mathcal{U}_N$ — сеть потоков работ с функцией пометки l. Будем называть $D = \{N^1, N^2, \ldots, N^n\} \subseteq \mathcal{U}_N$ её* ***валидной декомпозицией****, если*

- $N^i = (P^i, T^i, F^i, l^i)$ *— помеченные сети Петри для каждого* $1 \leq i \leq n$,
- $l^i = l\restriction_{T^i}$ *для каждого* $1 \leq i \leq n$,
- $P^i \cap P^j = \emptyset$ *для* $1 \leq i < j \leq n$,
- $T^i \cap T^j \subseteq T^u_v(N)$ *для* $1 \leq i < j \leq n$, *а*
- $N = \bigcup_{1 \leq i \leq n} N^i$.

Множество всех валидных декомпозиций сети N будем обозначать $\mathcal{D}(N)$.

Заметим, что максимальная декомпозиция единственна для сети. Более того, можно показать, что это валидная декомпозиция с минимально возможным размером фрагментов (т.е. разбиение *максимально*).

В [2] показано, что процедура проверки соответствия (fitness checking) может быть декомпозирована с использованием любой валидной декомпозиции. Базируясь на этих результатах, мы сформулировали в [28] условия, при которых модульная схема исправления модели процесса обеспечивает идеальное соответствие исправленной модели логу событий.

Утверждение 1. *Модульная схема исправления ModularRepair обеспечивает идеальное соответствие, если*

1. *split — функция валидной декомпозиции;*
2. *repair — идеальная функция автоматического синтеза модели, т. е. такая функция, которая для любого лога событий возвращает идеально соответствующую ему сеть Петри;*
3. *compose — функция композиции фрагментов, в которой используется простое слияние переходов с одинаковыми именами.*

Доказательство данного утверждения содержится в [28]. Заметим, что алгоритм исправления полностью удовлетворяет условиям (1), (2) и (3). Действительно,
(1) используется максимальная декомпозиция, являющаяся валидной,
(2) используются алгоритмы автоматического синтеза, которые гарантируют идеальное соответствие конструируемой модели,
(3) для объединения подсетей используется простое слияние переходов.

Рассмотрим пример применения данного алгоритма. Допустим, некоторый лог событий содержит трассы $\langle a, d, b, e, f\rangle$ и $\langle a, b, c, e, g\rangle$, а трасс, в которых b предшествует d, в логе нет. Таким образом, d всегда предшествует b, если оба события есть в логе. Такой лог не соответствует модели, показанной на Рис. 1, в которой переход с именем b срабатывает строго раньше переходов с именами c и d.

В результате применения нашего модульного подхода будет получена модель, показанная на Рис. 4. Данная модель идеально соответствует логу, состоящему из

трасс $\langle a, d, b, e, f\rangle$ и $\langle a, b, c, e, g\rangle$. Результирующая модель уже не является сетью потоков работ, так как содержит дополнительные позиции *s-start* и *s-end*, добавленные алгоритмом автоматического синтеза при замене исправляемого фрагмента. Заметим, что такая модель может быть трансформирована в эквивалентную сеть потоков работ путём добавления искусственных начальной (конечной) позиций, соединенных с позициями *source* и *s-start* (*sink* и *s-end*) через невидимые переходы.

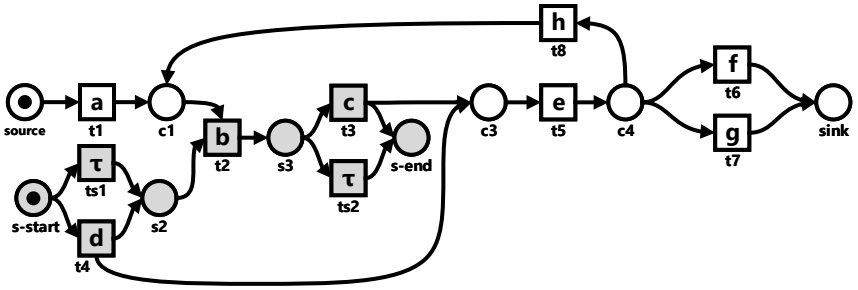


Рис. 4. Исправленная модель
Fig. 4. The repaired model

Позиции *s-start* и *s-end* существенным образом ограничивают поведение исправленной модели. В частности, формируется блокировка (deadlock), не позволяющая исполнять цикл через переход t_8 более одного раза. Впрочем, это не противоречит логу событий, по которому исправлялась модель, но, тем не менее, может понизить её ценность. Для решения данной проблемы мы удаляем позиции *s-start* и *s-end* как излишние. Данная операция не изменяет характеристики соответствия модели и лога событий, но сильно увеличивает количество возможных исполнений модели. Действительно, переходы t_4 и ts_1 теперь могут на любом шаге добавить токен в позиции s_2 и c_3, а переход ts_2 может удалить токен из позиции s_3. Таким образом, существенно понижается точность исправленной сети Петри по отношению к логу событий.

Для того, чтобы справиться с данным недостатком, мы предлагаем усовершенствовать наивный алгоритм. В следующем разделе будет описано предлагаемое усовершенствование.

5. Усовершенствованный алгоритм исправления модели процесса

Рассмотрим подробнее построение модели для исправляемого фрагмента. Первым делом к исходной модели, показанной на Рис. 1, применяется максимальная декомпозиция, в результате чего получаются шесть фрагментов (см. Рис. 5).

Затем подсчитывается соответствие каждого фрагмента и соответствующего фрагмента лога событий. Оказывается, что ровно один фрагмент $SN3$ не соответствует своему суб-логу. Данный фрагмент заменяется на модель, синтезированную с при-

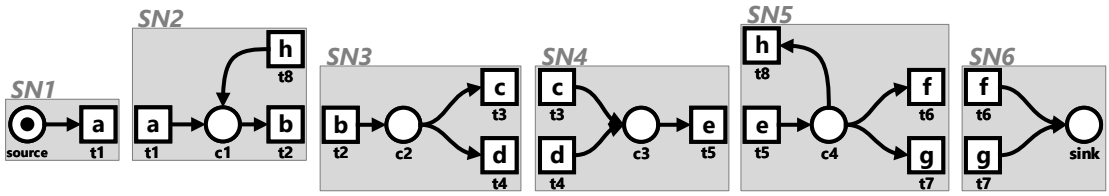


Рис. 5. Максимальная декомпозиция модели, изображенной на Рис. 1
Fig. 5. A maximal decomposition for the model shown in Fig. 1

менением индуктивного алгоритма (см. Рис. 6). Затем все фрагменты композируются в итоговую исправленную модель (см. Рис. 4).

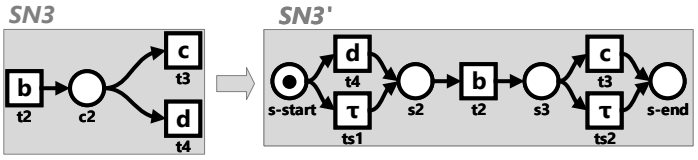


Рис. 6. Замена фрагмента $SN3$ (индуктивный алгоритм)
Fig. 6. The replacement of $SN3$ fragment (inductive miner)

При пристальном рассмотрении примера становится понятно, что в ходе процедуры исправления велась работа в том числе и с граничными переходами исправляемого фрагмента, по которым были разделены фрагменты. В результате причинно-следственные связи на границах фрагмента могут нарушаться, что снижает точность итоговой модели.

В качестве решения данной проблемы мы предлагаем *увеличивать* исправляемый фрагмент. Такое увеличение должно происходить по определенному правилу. К каждому фрагменту, требующему замены, предлагается присоединять все соседние фрагменты, идеально соответствующие своим суб-логам. В результате такого присоединения гарантируется, что внешняя граница изменяемого фрагмента не будет меняться при синтезе сети для замены фрагмента.

Действительно, пусть имеется последовательность срабатываний переходов в сети потоков работ, проходящая через большой фрагмент $m_i \xrightarrow{t_0} \dots \xrightarrow{t_{n-1}} m_n \xrightarrow{t_n} \dots \xrightarrow{t_{n+k}} m_{n+k+1} \xrightarrow{t_{n+k+1}} \dots \xrightarrow{t_{n+k+l}} m_o$, где $t_n, t_{n+1}, \dots, t_{n+k-1}, t_{n+k}$ — переходы, принадлежащие фрагменту, причем t_n и t_{n+k} — граничные для большого фрагмента, а остальные $t_{n+1}, \dots, t_{n+k-1}$ — внутренние. Так как к заменяемому фрагменту были присоединены все соседние фрагменты, то переходы t_n и t_{n+k} в первоначальном разбиении гарантированно принадлежали одному из соседних фрагментов с идеальным соответствием логу. Таким образом, фрагменты последовательности срабатываний $\dots \xrightarrow{t_{n-1}} m_n \xrightarrow{t_n} \dots$ и $\dots \xrightarrow{t_{n+k}} m_{n+k+1} \xrightarrow{t_{n+k+1}} \dots$ также соответствуют логу и не изменятся, если применяемый алгоритм автоматического синтеза гарантирует идеальное соответствие конструируемой модели.

Рис. 7 демонстрирует пример присоединения соседей к фрагменту $SN3$ из рассматриваемого нами примера. С этим фрагментом в декомпозиции граничат только фрагменты $SN2$ и $SN4$, которые и объединяются с ним.

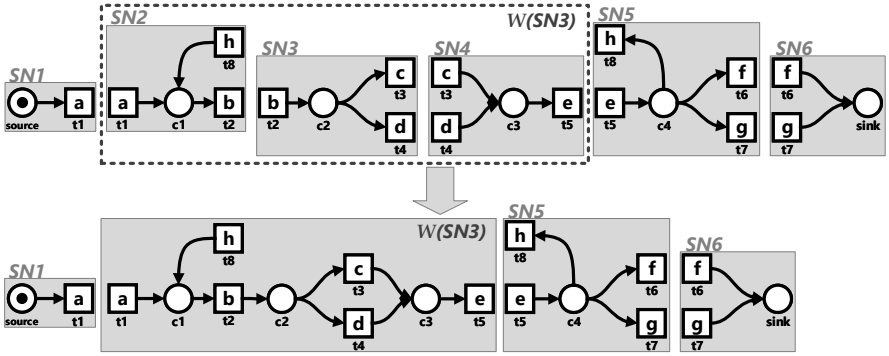


Рис. 7. Присоединение соседей к фрагменту
Fig. 7. Neighbours are connected to the fragment

Определение 3. *Пусть $N^i \in D_N$ — это фрагмент декомпозиции D сети потоков работ N. Соседними фрагментами для D будем называть фрагменты, в которых имеются переходы, общие с D. Множество соседей для фрагмента N^i определим так $\mathcal{N}(N^i) = \{N^j \mid N^j \in D_N \wedge T_i \cap T_j \neq \emptyset\}$.*

Укрупненный фрагмент для фрагмента N^i, к которому присоединены все соседние фрагменты, может быть определен так $\mathcal{W}(N^i) = N^i \cup \mathcal{N}(N^i)$.

Определим процедуру укрупнения фрагментов для случая, когда в декомпозиции модели больше одного фрагмента, который требуется заменять при исправлении.

Определение 4. *Пусть D — декомпозиция сети потоков работ N, и пусть $D = D_b \cup D_c$, где D_b — множество фрагментов сети, не соответствующих своим сублогам, а D_c — множество остальных фрагментов. Очевидно, что $D_b \cap D_c = \emptyset$.* ***Процедура укрупнения фрагментов*** *состоит из следующих шагов:*

1. *Каждой фрагмент из D_b объединяется со своими соседями $\forall N^k \in D_b : D^i_w = \mathcal{W}(N^k)$, $D^i_n = \mathcal{N}(N^k)$; Множество всех укрупненных фрагментов обозначим $D_w = \bigcup\limits_i D^i_w$, а множество всех фрагментов, затронутых процедурой, обозначим $D_n = \bigcup\limits_i D^i_n$; заметим, что множества соседей для каждого фрагмента могут пересекаться;*

2. *Все фрагменты, затронутые процедурой, удаляются из исходной декомпозиции $D_r = D_c \setminus D_n$;*

3. *Собирается новая декомпозиция, содержащая бо́льшие фрагменты: $D^w_N = D_w \cup D_r$.*

Можно показать, что после применения процедуры укрупнения фрагментов получается валидная декомпозиция.

Утверждение 2. *Пусть* $D_N = \{N^1, \ldots, N^n\} \in \mathcal{D}(N)$ *— любая валидная декомпозиция сети* N*. Пусть* D_N^w *— укрупненная декомпозиция, в которой объединены два фрагмента, то есть* $\exists i, j$ *такие, что* $1 \leq i < j \leq n$ *и* $D_N^w = \{N^i \cup N^j\} \cup D_N \setminus \{N^i, N^j\}$*. Тогда* D_N^w *— валидная декомпозиция, т. е.* $D_N^w \in \mathcal{D}(N)$*.*

Доказательство. Необходимо показать, что для $D_N^w = \{N_w^1, \ldots, N_w^{n-1}\}$ выполняются все пять свойств валидной декомпозиции.

(1) Все подсети $N^1, \ldots, N^n$ являются помеченными сетями Петри, $N^i \cup N^j$ — также помеченная сеть Петри по определению операции объединения сетей. Один фрагмент итоговой декомпозиции получается объединением, т. е. $\exists k$ такое, что $1 \leq k \leq n-1$ и $N_w^k = N^i \cup N^j$. Остальные фрагменты не изменяются, т.е. $\forall f$ такого, что $1 \leq f \leq n-1$ и $f \neq k$: $\exists h$ такое, что $1 \leq h \leq n$ и $N_w^f = N^h$.

(2) Для всех подсетей, кроме двух объединяемых, ничего не меняется. При объединении подсетей N^i и N^j по определению $dom(l^{N^i \cup N^j}) = dom(l^i) \cup dom(l^j)$, а $l^{N^i \cup N^j}(t) = l^i(t)$, если $t \in dom(l^i)$, и $l^{N^i \cup N^j}(t) = l^j(t)$, если $t \in dom(l^j) \setminus dom(l^i)$. Так что $l^k = l\restriction_{T_w^k}$ для каждого $1 \leq k \leq n-1$.

(3) Каждая позиция остается в своем фрагменте $P^k \cap P^f = \emptyset$ для $1 \leq k < f \leq n-1$, так как в исходной декомпозиции каждая позиция находилась ровно в одном фрагменте, а множество позиций объединяемых фрагментов — $P^i \cup P^j$, $i < j$.

(4) На границах фрагментов остаются только видимые переходы с уникальными именами. Действительно, при объединении набор граничных переходов уменьшается, $\left\{t \mid t \in T_w^k \cap T_w^f, 1 \leq k < f \leq n-1\right\} \subset \left\{t \mid t \in T^h \cap T^g, 1 \leq h < g \leq n\right\} \subseteq T_v^u(N)$.

(5) $N = \bigcup_{1 \leq h \leq n} N^h = \bigcup_{1 \leq h < i} N^h \cup \bigcup_{i < h < j} N^h \cup \bigcup_{j < h \leq n} N^h \cup N^i \cup N^j = \bigcup_{1 \leq f < k} N_w^f \cup \bigcup_{k < f \leq n-1} N_w^f \cup N_w^k$, что следует из того, каким образом объединялись фрагменты. □

Мы показали, что объединение двух фрагментов не портит валидности декомпозиции. Очевидно, что объединение любого количества фрагментов сводится к последовательному объединению пар фрагментов, причем на каждом шаге валидность получающейся декомпозиции сохраняется. В ходе процедуры укрупнения фрагментов выполняется только объединение некоторых фрагментов. Следовательно, декомпозиция, конструируемая процедурой укрупнения фрагментов, валидна.

Таким образом, базовый (*наивный*) алгоритм исправления может быть усовершенствован путем добавления одного промежуточного шага. После декомпозиции исходной модели и выявления фрагментов, которые подлежат замене, выполняется процедура построения оберток. В результате получается новая декомпозиция, содержащая более крупные фрагменты. Замена фрагментов с использованием алгоритма автоматического синтеза выполняется для этой модифицированной декомпозиции. Такой алгоритм будем называть **усовершенствованным алгоритмом исправления**.

Рассмотрим пример работы усовершенствованного алгоритма. Допустим, что необходимо привести модель, показанную на Рис. 1, в идеальное соответствие логу событий, состоящему из таких трасс: $\langle a, d, b, e, f\rangle$, $\langle a, b, c, e, g\rangle$, $\langle a, b, c, e, h, d, b, e, g\rangle$, $\langle a, c, b, e, f\rangle$. Необходимо заменить фрагмент $SN3$ из максимальной декомпозиции, показанной на Рис. 6. Процедура присоединения соседей к данному фрагменту показана на Рис. 7. Затем с помощью простого проецирования получается суб-лог,

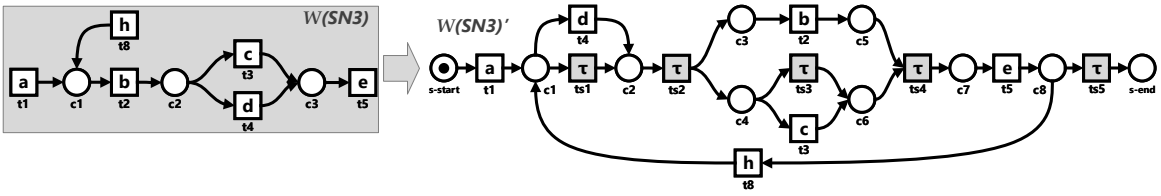


Рис. 8. Замена укрупненного фрагмента $\mathcal{W}(SN3)$ (индуктивный алгоритм)
Fig. 8. The replacement of enlarged fragment $\mathcal{W}(SN3)$ (inductive miner)

соответствующий укрупненному фрагменту: $\langle a, d, b, e\rangle$, $\langle a, b, c, e\rangle$, $\langle a, b, c, e, h, d, b, e\rangle$, $\langle a, c, b, e\rangle$. Для полученного суб-лога с помощью индуктивного алгоритма автоматически строится модель (см. Рис. 8).

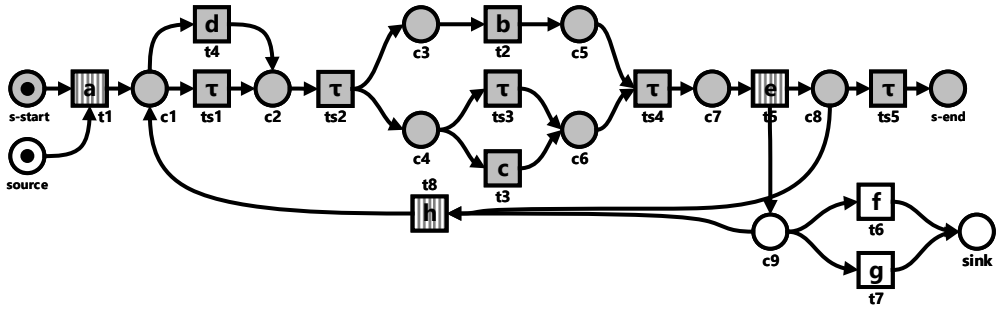


Рис. 9. Исправленная модель (усовершенствованный подход)
Fig. 9. The repaired model (improved approach)

В результате объединения всех фрагментов получается модель, показанная на Рис. 9. Нетрудно проверить, что эта модель идеально соответствует логу событий, который использовался при исправлении. Более того, она точнее отражает лог событий, чем модель, которая могла бы быть получена без укрупнения фрагментов.

Заметим, что позиции *s-start* и *s-end*, добавленные индуктивным алгоритмом автоматического синтеза модели, также могут быть удалены из модели. Более того, отметим, что весь фрагмент, состоящий из позиций c_8, *s-end* и невидимого перехода ts_5 был добавлен во фрагмент при исправлении только по той причине, что цикл через переход t_8 оказался разорван между несколькими фрагментами.

6. Экспериментальное тестирование

Описанный подход был протестирован не некотором количестве искусственных примеров. Тестовая установка была реализована в виде расширения для среды *ProM 6 Framework* [37]. Программная архитектура этого расширения и другие детали реализации описаны нами в [34].

С использованием генератора тестовых логов событий Gena [36] автоматически подготовлены идеально соответствующие логи для тестовых сетей потоков работ.

Таблица 1. Некоторые экспериментальные результаты *(Соотв. — соответствие, Точн. — точность, Сходство — сходство моделей, измеренное с помощью расширения* `Calculate Graph Edit Distance Similarity` *для среды ProM [21], N-f — количество фрагментов в декомпозиции, N-bf — количество замененных фрагментов, Размер — количество узлов в сети, Изм. — количество узлов, затронутых изменениями, Время — время работы программы в миллисекундах, тестовая конфигурация:* `Intel Core i7-3630QM, 2.40 ГГц, 4 Гб RAM, Windows 7 x64`*)*

Table 1. Some experimental results *(Similarity is measured with ProM plug-in* `Calculate Graph Edit Distance Similarity` *[21], N-f — the number of fragments in decomposition, N-bf — the number of replaced fragments, Size — the number of nodes in the model, Ch. — the number of nodes in changed fragments, Time — the working time in Milliseconds, test configuration:* `Intel Core i7-3630QM, 2.40 GHz, 4 Gb RAM, Windows 7 x64`*)*

Модель Model	Метод Method	Алгоритм синтеза Discovery Algorithm	Соотв. Fitness	Точн. Precision	Сходство Similarity	N-f	N-bf	Размер Size	Изм. Ch.	Время Time
SM1	Исходная модель Initial model		1	0,91	-	-	-	40	-	-
	Синтез новой модели New Model Discovery	Ind	1	0,91	0,51	-	-	47	47	829
		ILP	1	0,91	0,57	-	-	38	38	10069
SM1-BL-1	Измененная модель Changed model		0,94	0,86	-	-	-	40	-	-
	Наивный Naive	Ind	1	0,57	0,97	19	1	40	3	2531
		ILP	1	0,57	0,97	19	1	40	3	2531
	Улучшенный Improved	Ind	1	0,91	0,95	19	1	40	7	2365
		ILP	1	0,91	0,95	19	1	40	7	2842
SM1-BL-2	Измененная модель Changed model		0,89	0,89	-	-	-	40	-	-
	Наивный Naive	Ind	1	f	0,89	19	3	45	9	2983
		ILP	1	0,5	0,94	19	3	39	9	3215
	Улучшенный Improved	Ind	1	0,91	0,82	19	1	47	15	2426
		ILP	1	0,91	0,88	19	1	41	15	4030
LM2	Исходная модель Initial model		1	0,59	-	-	-	133	-	-
	Синтез новой модели New Model Discovery	Ind	1	f	f	-	-	166	166	2920
		ILP	1	0,53	f	-	-	137	137	1816898
LM2-BL-1	Измененная модель Changed model		0,98	0,59	-	-	-	133	-	-
	Наивный Naive	Ind	1	0,28	f	63	2	133	7	10745
		ILP	1	0,28	f	63	2	133	7	9588
	Улучшенный Improved	Ind	1	0,59	f	63	1	133	12	9205
		ILP	1	0,59	f	63	1	133	12	10339
LM2-BL-2	Измененная модель Changed model		0,99	0,59	-	-	-	133	-	-
	Наивный Naive	Ind	1	0,38	f	63	1	133	3	8048
		ILP	1	0,38	f	63	1	133	3	12847
	Улучшенный Improved	Ind	1	0,59	1	63	1	133	8	12960
		ILP	1	0,59	1	63	1	133	8	8619
LM2-BL-3	Измененная модель Changed model		0,97	0,59	-	-	-	133	-	-
	Наивный Naive	Ind	1	0,23	f	63	3	133	10	8097
		ILP	1	0,23	f	63	3	133	10	8855
	Улучшенный Improved	Ind	1	0,59	f	63	2	134	21	9149
		ILP	1	0,59	f	63	2	134	21	12410

Затем исходные тестовые модели были немного изменены так, чтобы испортить идеальное соответствие соответствующим логам. Алгоритм исправления тестировался на этих примерах. Некоторые результаты приводятся в Таблице 1.

Модели `SM1` и `LM2` — две тестовые сети потоков работ, состоящие из 40 и 133 узлов соответственно — внесены изменения. В каждом случае переставлены одна или больше пар переходов в исходной сети. Таким образом получены модели `-BL1` и т. д.

Отметим, что в модели специально вносились не слишком существенные изменения, ведь именно для таких случаев и предназначается описываемых подход.

Таблица 1 содержит результаты исправления модели процесса тремя основными способами: синтез новой модели по логу (без использования информации об исходной модели), наивный и улучшенный методы корректировки. Каждый способ опробован для двух алгоритмов автоматического синтеза модели, гарантирующих, что результирующая модель сможет исполнять все трассы из лога — индуктивного `Ind` и использующего целочисленное линейное программирование `ILP`. Обратим внимание, что плагин оценки сходства моделей не всегда может успешно справиться с задачей, по причине высокой ресурсоемкости. Имеющееся количество оперативной памяти (4 Гб) в некоторых случаях недостаточно.

Заметим, что результаты, содержащиеся в Таблице 1, получены с использованием процедуры удаления начальной и конечной позиции для заменяемых фрагментов, описанной в разделе 4.

В данной работе подробно разбиралось исправление нашим методом модели небольшого размера, изображенной на Рис. 1. Это удобно для иллюстративных соображений, но заменяемый фрагмент в таком случае включает большую часть модели. Заметим, что эффективность предлагаемого метода тем выше, чем меньший фрагмент от исходной модели изменяется. Рис. 10 демонстрирует результат работы подхода тестовой модели (`LM2-BL3`). Затронутый процедурой фрагмент выделен цветом. Модель, автоматически синтезированная индуктивным алгоритмом по логу событий, показана на Рис. 11.

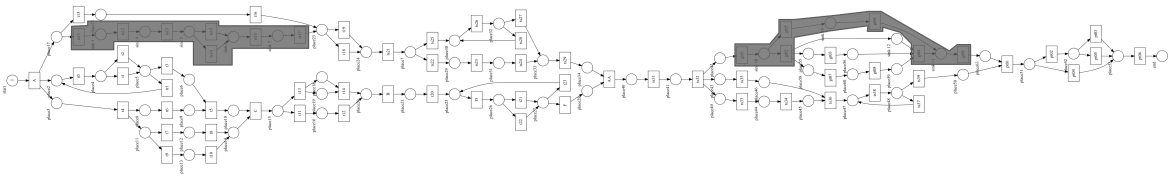


Рис. 10. Исправленная модель `LM2-BL3` (усовершенствованный подход)
Fig. 10. The repaired model `LM2-BL3` (improved approach)

Для всех демонстрационных примеров, результаты работы с которыми представлены в Таблице 1, наш алгоритм демонстрирует ожидаемые результаты. Соответствие *исправленных* моделей логу улучшается до идеального во всех примерах. Точность модели падает при исправлении наивным способом. Более того, в некоторых случаях поведение модели получается настолько разнообразным, что имеющихся 4 Гб оперативной памяти недостаточно для успешной работы реализации алгоритма подсчета точности модели. При применении усовершенствованного способа в рассмотренных случаях достигается точность исходной модели.

Изменяемый фрагмент для всех примеров составил небольшую часть модели. В частности, для самого масштабного из представленных примеров, изменялись 2 фрагмента из 63, содержащие вместе 21 переход и позицию из 133 имеющихся в модели.

Отметим, что для рассматриваемых примеров исправление с использованием алгоритма `ILP` выполняется существенно быстрее, чем построение полностью новой модели даже с учетом затрат времени на декомпозицию и дополнительную провер-

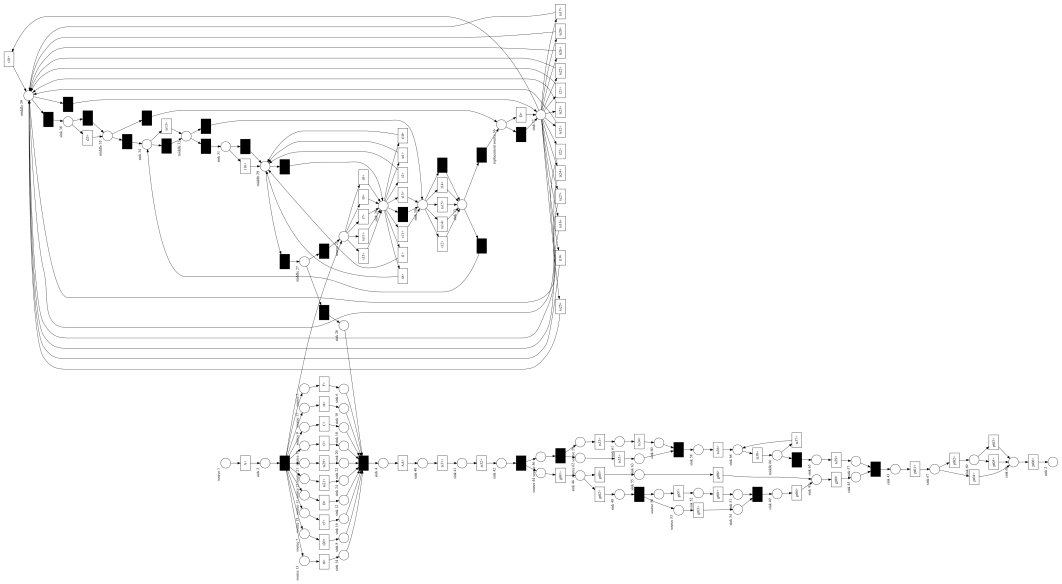


Рис. 11. Новая модель, синтезированная индуктивным алгоритмом по логу событий для модели `LM2`

Fig. 11. The new model discovered using inductive miner from the event log of `LM2` model

ку соответствия. Это ожидаемый результат для такого рода плохо масштабируемых алгоритмов. При использовании индуктивного алгоритма, устроенного иначе, исправление ожидаемо выполняется дольше. Впрочем, оптимизация эффективности не была целью данной работы.

Заметим, что конкретные настройки используемых алгоритмов автоматического синтеза и проверки соответствия модели и лога могут изменить вид получаемой модели. Например, алгоритм проверки соответствия, основанный на выравниваниях, позволяет задавать различную цену штрафа за разные виды несоответствий. Таким образом, некоторые из несоответствий можно игнорировать, если настроить алгоритм соответствующим образом.

Описанный в работе метод корректировки модели не является универсальным. В частности, открытым остается вопрос об исправлении моделей с нелокальными изменениями. При существенном изменении модели автоматический синтез новой модели на основании лога может быть более полезен, чем применение того или иного алгоритма исправления. Как и для поношенной одежды, заплатки работают только до определенного момента.

Заключение

В данной работе предлагается метод исправления (корректировки) модели процесса, использующий информацию из журнала событий. Метод использует принцип «разделяй и властвуй» для того, чтобы поделить модель на фрагменты с ясными границами. Фрагменты, которые не соответствуют данному журналу событий, заменяются новыми, построенными с помощью алгоритма автоматического синтеза. Затем собирается итоговая исправленная модель, которая имеет сходство с исходной, но соответствует журналу событий.

Кроме формального обоснования корректности метода, работа содержит результаты тестирования метода на нескольких искусственных примерах. Результаты тестирования демонстрируют применимость метода. Особенно полезным он может быть в случаях, когда исходная модель была разработана экспертом и, в результате, легко воспринимается человеком. Применение алгоритма автоматического синтеза может привести к построению корректной, но плохо воспринимаемой модели. Предлагаемый метод исправления меняет только часть модели, в результате чего читаемость сохраняется.

Тем не менее, предложенный метод не является универсальным. Метод хорошо зарекомендовал себя в случае, когда несоответствия локальны, то есть, расхождения с логом могут быть устранены путем замены отдельных фрагментов. В настоящее время авторы работают над усовершенствованным методом исправления, который бы позволял *подстраивать* размер фрагмента разбиения под конкретное несоответствие. Однако не стоит забывать, что если поведение системы изменилось кардинально, то может быть проще построить модель системы полностью заново.

Данная работа продолжает и конкретизирует работу [28], в которой описана общая модульная схема исправления моделей процессов, а именно, представлена одна из возможных реализаций такой схемы. В будущем планируется рассмотреть другие возможные реализации общей схемы на основе других способов декомпозиции модели процесса. В [17] описан метод исправления процессов, позволяющий добавлять в новую модель действия, которые не встречались в исходной модели. Мы планируем исследовать возможности комбинации этого метода с методом, предложенным в данной статье.

Список литературы / References

[1] van der Aalst W. M. P., “Decomposing Process Mining Problems Using Passages”, Lecture Notes in Computer Science, **7347**, Springer-Verlag, 2012, 72–91.

[2] van der Aalst W. M. P., “Decomposing Petri Nets for Process Mining: A Generic Approach”, *Distributed and Parallel Databases*, **31**:4 (2013), 471–507.

[3] van der Aalst W. M. P., *Process Mining — Data Science in Action, Second Edition*, Springer, 2016.

[4] van der Aalst W. M. P., Adriansyah A., van Dongen B. F., “Replaying History on Process Models for Conformance Checking and Performance Analysis”, *WIREs Data Mining and Knowledge Discovery*, **2** (2012), 182–192.

[5] van der Aalst W. M. P., Bolt A., van Zelst S. J., “RapidProM: Mine Your Processes and Not Just Your Data”, *CoRR*, **abs/1703.03740** (2017).

[6] van der Aalst W. M. P., Kalenkova A. A., Rubin V. A., Verbeek H. M. W., “Process Discovery Using Localized Events”, LNCS, **9115**, Springer, 2015, 287–308.

[7] van der Aalst W. M. P, Weijters A. J. M. M., Maruster L., “Workflow Mining: Discovering Process Models from Event Logs”, *IEEE Transactions on Knowledge and Data Engineering*, **16** (2004), 1128–1142.

[8] Adriansyah A., *Aligning observed and modeled behavior*, Ph.D. Thesis, Technische Universiteit Eindhoven, 2014.

[9] Begicheva A. K., Lomazova I. A., Does Your Event Log Fit the High-level Process Model? *Modeling and Analysis of Information Systems*, **22**:3 (2015), 392–403.

[10] Begicheva A. K., Lomazova I. A., “Discovering High-Level Process Models from Event Logs”, *Modeling and Analysis of Information Systems*, **24**:2 (2017), 125–140.

[11] Buijs J. C. A. M., van Dongen B. F., van der Aalst W. M. P., "On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery", *On the Move to Meaningful Internet Systems: OTM 2012*, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012 (Rome, Italy, September 10-14, 2012), Proceedings, Part I, 2012, 305–322.

[12] Buijs J. C. A. M., La Rosa M., Reijers H. A., van Dongen B. F., van der Aalst W. M. P., "Improving Business Process Models Using Observed Behavior", LNBIP, **162**, Springer-Verlag, Berlin, 2013, 44–59.

[13] Burattin A., Maggi F. M., Sperduti A, "Conformance checking based on multi-perspective declarative process models", *Expert Syst. Appl.*, **65** (2016), 194–211.

[14] Dijkman R. M., Dumas M., Garcia-Banuelos L., "Graph matching algorithms for business process model similarity search", Lecture Notes in Computer Science, **5701**, 2009, 48–63.

[15] van Dongen B. F., Carmona J., Chatain T., "A Unified Approach for Measuring Precision and Generalization Based on Anti-alignments", Lecture Notes in Computer Science, **9850**, Springer, 2016, 39–56.

[16] Fahland D., van der Aalst W. M. P., "Repairing Process Models to Reflect Reality", Lecture Notes in Computer Science, **7481**, Springer-Verlag, 2012, 229–245.

[17] Fahland D., van der Aalst W. M. P., "Model repair - Aligning process models to reality", *Inf. Syst.*, **47** (2015), 220–243.

[18] Fahland D., van der Aalst W. M. P., "Simplifying discovered process models in a controlled manner", *Inf. Syst.*, **38**:4 (2013), 585–605.

[19] Gambini M., La Rosa M., Migliorini S., Ter Hofstede A. H. M., "Automated Error Correction of Business Process Models", Lecture Notes in Computer Science, **6896**, Springer, 2011, 148–165.

[20] Günther C. W., van der Aalst W. M. P., "Fuzzy mining: Adaptive process simplification based on multi-perspective metrics", *BPM, Lecture Notes in Computer Science*, **4714** (2007), 328–343.

[21] Hompes B. F. A., *On Decomposed Process Discovery: How to Solve a Jigsaw Puzzle with Friends*, Master's Thesis, Technische Universiteit Eindhoven, 2014.

[22] Hompes B. F. A., Verbeek H. M. W., van der Aalst W. M. P., "Finding suitable activity clusters for decomposed process discovery", *SIMPDA 2014*, **1293** (2014), 16–30.

[23] Kalenkova A. A., Lomazova I. A., van der Aalst W. M. P., "Process Model Discovery: A Method Based on Transition System Decomposition", *Application and Theory of Petri Nets and Concurrency*, Proceedings of 35th International Conference, PETRI NETS 2014, Lecture Notes in Computer Science, **8489**, Springer, 2014, 71–90.

[24] Leemans S. J. J., Fahland D., van der Aalst W. M. P., "Discovering Block-Structured Process Models from Incomplete Event Logs", Lecture Notes in Computer Science, **8489**, Springer, 2014, 91–110.

[25] Leemans S. J. J., Fahland D., van der Aalst W. M. P., "Scalable Process Discovery with Guarantees", LNBIP, **214**, Springer, 2015, 85–101.

[26] Mans R., van der Aalst W. M. P., Verbeek H. M. W., "Supporting Process Mining Workflows with RapidProM", *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014)*, CEUR-WS.org, 2014, 56–60.

[27] Мицюк А. А., Шугуров И. С., "Синтез моделей процессов по журналам событий с шумом", *Моделирование и анализ информационных систем*, **21**:4 (2014), 181–198; English transl.: Mitsyuk A. A., Shugurov I. S., "On Process Model Synthesis Based on Event Logs with Noise", *Automatic Control and Computer Sciences*, **50**:7 (2016), 460–470.

[28] Mitsyuk A. A., Lomazova I. A., Shugurov I. S., van der Aalst W. M. P., "Process Model Repair by Detecting Unfitting Fragments", *Supplementary Proceedings of AIST 2017, CEUR-WS.org*, 2017.

[29] Muñoz-Gama J., "Conformance Checking and Diagnosis in Process Mining — Comparing Observed and Modeled Processes", *LNBIP*, **270** (2016).

[30] Muñoz-Gama J., Carmona J., van der Aalst W. M. P., "Single-Entry Single-Exit Decomposed Conformance Checking", *Inf. Syst.*, **46** (2014), 102–122.

[31] Polyvyanyy A., van der Aalst W. M. P., ter Hofstede A. H. M., Wynn M. T., "Impact-driven process model repair", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **25**:4 (2017), 28:1–28:60.

[32] de San Pedro J., Carmona J., Cortadella J., "Log-based simplification of process models", *BPM, Lecture Notes in Computer Science*, **9253** (2015), 457–474.

[33] Шершаков С. А., "Графический язык DPMine для автоматизации экспериментов в области автоматического построения и анализа процессов", *Моделирование и анализ информационных систем*, **21**:5 (2014), 102–115; English transl.: Shershakov S. A., "DPMine graphical language for automation of experiments in process mining", *Automatic Control and Computer Sciences*, **50**:7 (2016), 477–485.

[34] Shugurov I. S., Mitsyuk A. A., "Iskra: A Tool for Process Model Repair", *Proceedings of the Institute for System Programming of the Russian Academy of Sciences*, **27**:3 (2015), 237–254.

[35] Shugurov I. S., Mitsyuk A. A., "Applying MapReduce to Conformance Checking", *Proceedings of the Institute for System Programming of the Russian Academy of Sciences*, **28**:3 (2016), 103–122.

[36] Shugurov I. S., Mitsyuk A. A., "Generation of a Set of Event Logs with Noise", *Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014)*, 2014, 88–95.

[37] Verbeek H. M. W., Buijs J. C. A. M., van Dongen B. F., van der Aalst W. M. P., "ProM 6: The Process Mining Toolkit", *Proc. of BPM Demonstration Track 2010*, **615**, CEUR-WS.org, 2010, 34–39.

[38] Verbeek H. M. W., "Decomposed Process Mining with Divide And Conquer", *BPM 2014 Demos*, **1295** (2014), 86–90.

[39] Verbeek H. M. W., van der Aalst W. M. P., Muñoz-Gama J., "Divide and Conquer: A Tool Framework for Supporting Decomposed Discovery in Process Mining", *The Computer Journal*, 08.05.2017. https://doi.org/10.1093/comjnl/bxx040.

[40] Weber I., Rogge-Solti A., Li C., Mendling J., "CCaaS: Online Conformance Checking as a Service", *BPM 2015 Demos*, **1418** (2015), 45–49.

[41] van der Werf J. M. E. M., van Dongen B. F., Hurkens C. A. J., Serebrenik A., "Process Discovery using Integer Linear Programming", *Fundam. Inform.*, **94** (2009), 387–412.

[42] Weijters A. J. M. M., Ribeiro J., Chawla N., "Flexible Heuristics Miner", *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, 2011, 310–317.

Mitsyuk A. A., Lomazova I. A., van der Aalst W. M. P., "Using Event Logs for Local Correction of Process Models", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 459–480.

DOI: 10.18255/1818-1015-2017-4-459-480

Abstract. During the life-cycle of an Information System (IS) its actual behaviour may not correspond to the original system model. However, to the IS support it is very important to have the latest model that reflects the current system behaviour. To correct the model, the information from the event log of the system may be used. In this paper, we consider the problem of process model adjustment (correction) using the information from an event log. The input data for this task are the initial process model (a Petri net) and the event log. The result of correction should be a new process model, better reflecting the real IS behavior than the initial model. The new model could be also built from scratch, for example, with the help of one of the known algorithms for automatic synthesis of the process model from an event log. However, this may lead to crucial changes in the structure of the original model, and it will be difficult to compare the new model with the initial one, hindering its understanding and

analysis. It is important to keep the initial structure of the model as much as possible. In this paper, we propose a method for process model correction based on the principle of "divide and conquer". The initial model is decomposed in several fragments. For each fragment its conformance to the event log is checked. Fragments which do not match the log are replaced by newly synthesized ones. The new model is then assembled from the fragments via transition fusion. The experiments demonstrate that our correction algorithm gives good results when it is used for correcting local discrepancies. The paper presents the description of the algorithm, the formal justification for its correctness, as well as the results of experimental testing by some artificial examples.

Keywords: process mining, process model repair, Petri nets, process model decomposition, divide and conquer

On the authors:
Alexey A. Mitsyuk, orcid.org/0000-0003-2352-3384, research fellow,
National Research University Higher School of Economics, Laboratory of Process-Aware Information Systems,
20 Myasnitskaya St., Moscow 101000, Russia, e-mail: amitsyuk@hse.ru

Irina A. Lomazova, orcid.org/0000-0002-9420-3751, doctor of sciences, professor
National Research University Higher School of Economics, Laboratory of Process-Aware Information Systems,
20 Myasnitskaya St., Moscow 101000, Russia, e-mail: ilomazova@hse.ru

Wil M.P. van der Aalst, orcid.org/0000-0002-0955-6940, PhD, dr.ir., professor
Eindhoven University of Technology (TU/e), Architecture of Information Systems,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, e-mail: w.m.p.v.d.aalst@tue.nl

Acknowledgments:
This work is an output of a research project implemented as part of the Basic Research Program at the National Research University Higher School of Economics (HSE).

Моделирование и анализ информационных систем. Т. 24, № 4 (2017), с. 481–495
Modeling and Analysis of Information Systems. Vol. 24, No 4 (2017), pp. 481–495

DOI: 10.18255/1818-1015-2017-4-481-495

UDC 510.52:372.851

Automated System for Teaching Computational Complexity of Algorithms Course

Rublev V. S., Yusufov M. T.

Received April 14, 2017

Abstract. This article describes problems of designing automated teaching system for "Computational complexity of algorithms" course. This system should provide students with means to familiarize themselves with complex mathematical apparatus and improve their mathematical thinking in the respective area. The article introduces the technique of algorithms symbol scroll table that allows estimating lower and upper bounds of computational complexity. Further, we introduce a set of theorems that facilitate the analysis in cases when the integer rounding of algorithm parameters is involved and when analyzing the complexity of a sum. At the end, the article introduces a normal system of symbol transformations that allows one both to perform any symbol transformations and simplifies the automated validation of such transformations. The article is published in the authors' wording.

Keywords: automated learning, algorithm complexity analysis, algorithm's symbol scroll table, normal system of symbol transformations

For citation: Rublev V. S., Yusufov M. T., "Automated System for Teaching Computational Complexity of Algorithms Course", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 481–495.

About the authors:
Vadim S. Roublev, orcid.org/0000-0002-0252-9958, PhD,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003, Russia, e-mail: roublev@mail.ru

Murad T. Yusufov, orcid.org/0000-0002-0362-2951, graduate student,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003, Russia, e-mail: flood4life@gmail.com

Introduction

One of the most important aspects of educating a Computer Science specialist is teaching them how to analyze computational complexity of algorithms. This enables them to choose the algorithm that is most appropriate for a certain set of conditions and predict the time necessary for the program execution. Success in this aspect requires a lot of individual learning. However, a large amount of individual assignments forces the professor to spend a lot of personal time. A possible solution for this problem is a certain set of computer programs that allows to manage individual learning for each student, which is also known as automated teaching system. Remote education is an alternative education method that does not require the student's personal contact with his professor. Ability to study in comfortable hours and places provides current popularity growth of various remote education methods.

1. Approaches to designing an automated teaching system

Existing automated teaching systems are fine for teaching areas of knowledge focused on definitions and shallow connections between them. But algorithm analysis requires development of logical-mathematical thinking; there is, therefore, a need for smart teaching systems that are focused on development of this kind of thinking and acquiring erudition and skills in a complex area of knowledge [1].

There is a large amount of program systems called teaching ones (e.g., Moodle, Claroline, Dokeos, ATutor), but most of them do not support full cycles of teaching (methods), because they are just applications that provide access to texts, allow to take part in some tests and check up whether the user has passed those tests [2]. A more advanced solution is to use various techniques that modify the system behavior towards each user depending on their individual traits. Adaptive teaching systems (ATS), whose primary paradigm is adapting to every user, is a suggested solution to this issue. Adaptive technologies for teaching is a relatively recent development but they have already become popular with teaching systems developers. The following techniques are the base adaptive teaching paradigm:

- building the course teaching sequence;
- smart analysis of user's solutions;
- interactive support during problem solving;
- problem solving support using examples;
- adaptive support for navigation;
- adaptive presentation;
- adaptive support for users collaboration.

Applying these techniques secures the system flexibility in interacting with its users and in presenting the material for studying. An addition to the concept of adaptive teaching systems is the following proposition:

Proposition 1. *Learning can be reduced to an aggregate of following pieces:*

- *the information for studying;*
- *control events that allow to check knowledge of that information;*
- *a method for assessing the quality of knowledge;*
- *the following management, the most important and complex component that makes the system actually teaching.*

Therefore, we have to provide answers for the following questions: how to perform partitioning, how to check up knowledge and how flexible should the system be in interaction with its users.

Adaptive teaching method aligns well with these postulates about the learning process:

- any information that is too hard to understand can be divided to a sequence of such smaller chunks so that each chunk can be understood when all previous chunks are understood too;
- any chunk of information has a certain finite amount of such tests so that if the student has successfully passed all of them then it is fairly certain that they have absorbed the chunk.

Therefore, it is possible to partition both information that has to be taught and control tests in such sections so that each section is a couple of a chunk of information and a set of control tests related to this chunk.

The whole information in "Computational Complexity of Algorithms" course of study can be divided into two large categories. The first category includes sections on basic theoretical knowledge and definition of algorithm, computational complexity and asymptotic estimation of complexity. The main goal of these sections is development and training student's memory using memorization techniques. Tests are the most common control method for these sections.

However, the course of study also covers teaching of informal usage of mathematical apparatus and this presents additional difficulties. Those are technical difficulties related to formula input and transformation and to insufficient level of logical thinking of the student who has to reach some result by building a transformation sequence. So, the second section is focused on mathematical methods for estimating algorithm computational complexity and logical-mathematical thinking development.

The following traits distinguish the second category from the first one:

1. Tests cannot be the only control method because they cannot test students' ability to apply various mathematical transformations;
2. The system has to teach the student to combine single transformations into a directed process by building a sequence of previously learned transformations;
3. The system has to be able to control students' ability to connect multiple processes when solving the final problem on estimating algorithm computational complexity.

One of the tools used in the control events of sections from the second category is the algorithm for verifying symbol transformations.

The most important component of the system is the one that carries out the information and control events interaction. It determines the volume of information for a single session, set and amount of control events and other parameters of the system. It is this component that adds flexibility to the system and distinguishes adaptive teaching system from the basic ones. The next few paragraphs describe a suggested use case scenario of the system and the user interaction.

The user logs in into the system and sees a list of the course sections that may be available or unavailable depending on the user's progress in the course. The user has

to pass the sections sequentially (a principle of linear learning). After the user enters a section they have chosen they see the information it contains. Then the control event starts. To pass it the user has to complete a certain amount of multiple choice tests defined for each section. The number of answers for each question is determined by the developers of the specific tasks. That is why this number may be different for different questions and their choice depends on the program itself. The presented answers may contain either only correct answers or only incorrect ones or both.

The user has to tick all the correct answers missing incorrect ones. If the user has not chosen all the correct answers or has chosen some of the incorrect ones, the system will display a warning text ("Not all correct answers are selected", "Some selected answers are incorrect" or a combination of both) and a chunk of material related to the question. Now the user has an opportunity to fix the answer. If the answer is incorrect again, the system removes this question and adds two new questions. So, the amount of questions required to complete the section may increase. If this amount grows too much, the current session ends. If this happens too many times in a row, the system temporarily disables the user's account, which means a visit to the professor.

If the user has a lot of penalty questions but has a long streak of correct answers, the system begins to remove them according to some progression. Such an approach makes the user study the section information carefully. Thus, it impacts both the control events and the learning process. The user can move on to the next section only after completing all the questions in the current section.

However, there are some sections whose information relies on that from the previous ones, hence it is impossible for the students to absorb it unless they have mastered the previous material. Such sections require to additionally control the level of knowledge of information from previous sections. In this case it could be enough to answer only one control question.

2. Computational complexity of algorithms and techniques for its estimation

This section deals with the subject area closely related to the material above. It starts with the definition of algorithm computational complexity. Generally, it is the time (amount of steps) necessary for the algorithm to finish; it usually depends on some input parameters. Despite that in some cases also require memory usage analysis, this section considers only time-wise complexity.

The O-notation is a common tool that allows to estimate the amount of steps $T(n)$, where n is an input parameter, growth speed estimation: $T(n) = O(f(n))$. This expression means that the upper bound of the $T(n)$ growth can be expressed through $f(n)$, i.e.:

$$\exists C > 0, n_0 \; \forall n \geq n_0 : \; T(n) \leq C \cdot f(n),$$

while Ω-notation: $T(n) = \Omega(g(n))$ expresses the lower bound of the $T(n)$ growth through $g(n)$, i.e:

$$\exists C > 0, n_0 \; \forall n \geq n_0 : \; T(n) \geq C \cdot g(n).$$

These bounds may be both too high and too low. E.g., for $T(n) = n^2 - n + 3$ estimations $T(n) = \Omega(n), T(n) = O(n^3)$ are correspondingly too low and too high. So, the most accurate bounds estimations are done using the Θ-notation [2]: $T(n) = \Theta(f(n))$ that expresses both the upper and the lower bounds with the same $f(n)$ function, i.e.:

$$\exists C_1 > 0, C_2 > 0, n_0 \; \forall n \geq n_0 : \; C_1 \cdot f(n) \leq T(n) \leq C_2 \cdot f(n).$$

Note: for the example above, $T(n) = \Theta(n^2)$.

It is very important to choose a proper set of $f(n)$ functions used in computational complexity estimation. The following set of functions is commonly accepted in the theory of algorithm complexity:

$$\log n, \; n^{m/k}, \; 2^n,$$

where n is an algorithm parameter, which is usually an integer, and m, k are some arbitrary integer constants. Any possible superposition of these functions is also a valid estimation of algorithm complexity. E.g.,

$$\log \log n, \; \log^{1/2} n, \; n^{3/2}, \; n^{\log n}, \; 2^{2^n}.$$

If the algorithm has several input parameters, then the Θ-notation is defined via the superposition of functions of each parameter, e.g. $T(n, m, k) = \Theta(2^n m \log k)$.

The next several paragraphs describe the method for estimating the bounds of algorithm computational complexity. Its goal is to get Θ-notation expressed bounds if possible, or O-notation and Ω-notation ones. Since this estimation is strongly related to loop complexity estimation, it makes sense to describe the estimation process for a single loop.

The basis for algorithm complexity estimation is a symbol scroll table that helps defining the complexity by the amount of loop executions. Each variable of the algorithm has a corresponding column in the table. The table also contains several special columns:

- for each loop there is a column with the index of this loop and a symbol for the last time when the loop is executed;
- column *loop condition* that shows symbol condition of the loop. It has comment *last* for the loop last execution condition and comment *exit* for the loop exit condition.

These conditions use the symbol value of the loop index for its last execution. These conditions provide two estimations for the loop execution amount defined by a symbol: the upper and the lower bounds. The analysis of these estimations allows to determine algorithm computational complexity if condition expressions are not too complex. Most of the time [2] the upper and the lower bounds differ only by a constant factor; therefore, the estimation for the algorithm runtime growth is a theta-estimation $T(n) = \Theta(f(n))$. Analyze the following example 1 of an algorithm with a single loop:

```
void f1 (unsigned long n) {
    float x = n;
    while (x > 2)
        x = sqrt(x);
}
```

It is possible to build an algorithm symbol scroll table by including a single column for the loop index, a single column for the single variable and the column for the loop condition. Note that p in the i column means the index of the loop last execution.

Table 1. Symbol scroll table for the Example 1

i	x	loop condition	
	n		
1	$n^{1/2}$	$n > 2$	
2	$n^{(1/2)^2}$	$n^{1/2} > 2$	
3	$n^{(1/2)^3}$	$n^{(1/2)^2} > 2$	
...	...	...	
i	$n^{(1/2)^i}$	$n^{(1/2)^{i-1}} > 2$	
...	...	...	
p	$n^{(1/2)^p}$	$n^{(1/2)^{p-1}} > 2$	last
$p+1$		$n^{(1/2)^p} \not> 2$	exit

Analysis of the loop last execution provides inequality $n^{(1/2)^{p-1}} > 2$, which is transformed into $\log_2 n > 2^{p-1}$ and then into $p < \log_2 \log_2 n + 1$. The loop exit condition $n^{(1/2)^p} \leq 2$ is transformed into $p \geq \log_2 \log_2 n$ and, since p defines the algorithm computational complexity, the result is

$$T_{f1}(n) = \Theta(\log \log n).$$

If there are several loops which are not nested and not dependent (variables affected in one loop do not affect the other loops), then it is sufficient to estimate the algorithm complexity as the maximum of its loops complexities.

If there are loops that are nested but not dependent (amount of executions for the inner loop is not dependent on the amount of executions for the outer loop), then the product of both loops complexities equals to the algorithm complexity. E.g., if the outer loop A complexity is $T_A(n_A) = \Theta(f(n_A))$ and the inner loop B complexity is $T_B(n_B) = \Theta(g(n_B))$, then their combined complexity is

$$T_{AB}(n_A, n_B) = \Theta(f(n_A) \cdot g(n_B)).$$

If there are loops that are nested and dependent (inner loop parameter is determined in the outer loop) but the complexity of the inner loop is estimated with the same function $\Theta(g(n))$ for any iteration of the outer loop. The complexity of the outer loop is $\Theta(f(n))$. We will call such case *weak dependency*. Combined complexity of weakly dependent loops is the same as for the case of independent loops.

If the loops are not nested, then overall complexity equals to the maximum of the loop complexities:

$$T_{A+B} = T_A + T_B = \max\{T_A, T_B\}.$$

If there are non-nested dependent loops, then it is imperative to determine the value of the variable used in the second loop. Example 2:

```
void f2 (unsigned long n) {
    float x = n, z = n;
    while (x > 2) {
        x = sqrt(x);
        z = z * z;
    }
    while (z /= 2 > 1);
}
```

The algorithm symbol scroll table includes a N_l column with loop numbers 1 and 2, columns i_1, i_2 with corresponding loop indexes, columns x, z with values of these variables and the loop condition column. Here, p_1 is the index for the last execution of the loop 1 (outer) and p_2 means the index for the last execution of the loop 2 (inner).

Table 2. Symbol scroll table for Example 2

N_l	i_1	i_2	x	z	loop condition	
			n	n		
1	1		$n^{1/2}$	n^2	$n > 2$	
	2		$n^{(1/2)^2}$	n^{2^2}	$n^{1/2} > 2$	
	...		...	...	...	
	i		$n^{(1/2)^i}$	n^{2^i}	$n^{(1/2)^{i-1}} > 2$	
	...		...	...	...	
	p_1		$n^{(1/2)^{p_1}}$	$n^{2^{p_1}}$	$n^{(1/2)^{p_1-1}} > 2$	last
	$p_1 + 1$				$n^{(1/2)^{p_1}} \not> 2$	exit
2		1		$n^{2^{p_1}}/2$	$n^{2^{p_1}} > 1$	
		2		$n^{2^{p_1}}/2^2$	$n^{2^{p_1}}/2 > 1$	
		...		...	...	
		p_2		$n^{2^{p_1}}/2^{p_2}$	$n^{2^{p_1}}/2^{p_2-1} > 1$	last
		$p_2 + 1$			$n^{2^{p_1}}/2_2^p \not> 1$	exit

The analysis for the loop 1 is the same as in Example 1 and gives the following bounds estimation: $\log_2 \log_2 n \ \leq p_1 < \log_2 \log_2 n + 1$ which is transformed to $p_1 = \log_2 \log_2 n$. Therefore, loop 1 complexity is $T_1(n) = \Theta(\log \log n)$.

The analysis for the loop 2 last execution condition: $n^{2^{p_1}}/2^{p_2} > 1$, using p_1 value gives inequality $2^{p_2} < n^{2^{\log_2 \log_2 n + 1}} = n^{2 \log_2 n}$. Applying logarithm to the both sides gives $p_2 < 2log_2^2 n$.

The analysis for the loop 2 exit condition: $n^{2^{p_1}}/2^{p_2+1} \leq 1$ using value of p_1 is transformed to $2^{p_2+1} \geq n^{2^{\log_2 \log_2 n}} = n^{\log_2 n}$. Applying logarithm to the both sides gives $p_2 \geq log_2^2 n - 1$. Consider that $1 \leq \frac{1}{4} \log_2^2 n$ when $n \geq 4$. Perform the substitution in the previous inequality: $p_2 \geq \frac{3}{4} \log_2^2 n$. Coupled with the loop last execution inequality, it results in $\frac{3}{4} \log_2^2 n \leq p_2 \leq 2 \log_2^2 n$. Therefore, loop 2 complexity is $T_2(n) = \Theta(log_2^2 n)$.

The algorithm complexity is the maximum of loops complexities:

$$T_{f2}(n) = \Theta(log^2 n).$$

If there are nested dependent loops, then algorithm complexity is determined as the total amount of inner loop executions throughout the total amount of outer loop executions. Example 3:

```
void f3 (unsigned long n) {
    float x = n, y, z = n;
    while (x > 2) {
        x = sqrt(x);
        z = z * z;
        y = z;
        while (y /= 2 > 1);
    }
}
```

The algorithm symbol scroll table includes the column N_l with loop numbers 1 and 2, columns i_1, i_2 with corresponding loop indexes, columns x, y, z with values of the corresponding variables and the column with the loop condition. Here, p_1 means the index for the last execution of the loop 1 (outer) and p_2 means the index for the last execution of the loop 2 (inner).

The analysis for the loop 1 is the same as for examples 1 and 2 and gives following estimations: $\log_2 \log_2 n \ \leq p_1 < \log_2 \log_2 n + 1$, which is transformed to $p_1 = \log_2 \log_2 n$.

The total amount of the loop 2 executions is defined as $T_2(n) = \sum_1^{p_1} p_2(i)$. In order to solve it, we will define common term $p_2(i)$ as the amount of executions of the loop 2 at i-th execution of the loop 1. At the last execution of the loop 2 the following inequality is truthy: $n^{2^i}/2^{p_2(i)} > 1$, which is transformed into $2^{p_2(i)} < n^{2^i}$. Applying logarithm to the both sides gives $p_2(i) < 2^i \log_2 n$. At the loop exit the following statement is truthy: $n^{2^{p_1}}/2^{p_2(p_1)+1} \leq 1$, which is transformed into $2^{p_2(i)+1} \geq n^{2^i}$. Applying logarithm to the both sides and moving the 1 on the other side gives $p_2(i) \geq 2^i \log_2 n - 1$. Placing these inequalities into the sum gives the upper and the lower bounds for the total amount of the loop 2 executions: $\sum_1^{p_1}(2^i \log_2 n - 1) \leq \sum_1^{p_1} p_2(i) < \sum_1^{p_1} 2^i \log_2 n$. Transforming the sum in the upper bound: $\sum_1^{p_1} 2^i \log_2 n = \log_2 n \sum_1^{p_1} 2^i = 2(2^{p_1} - 1) \log_2 n = 2(2^{\log_2 \log_2 n} - 1) \log_2 n = 2(\log_2 n - 1) \log_2 n < 2 \log_2^2 n$.

Transforming the sum in the lower bound: $\sum_1^{p_1}(2^i \log_2 n - 1) = \sum_1^{p_1} 2^i \log_2 n - p_1 = 2(\log_2 n - 1) \log_2 n - p_1 = 2 \log_2^2 n - \log_2 n - \log_2 \log_2 n > 2 \log_2^2 n - \log_2^2 n - 1/2 \cdot \log_2^2 n = 1/2 \cdot \log_2^2 n$. Placing both bounds gives $1/2 \cdot \log_2^2 n < \sum_1^{p_1} p_2(i) < 2 \log_2^2 n$, which gives algorithm complexity:

$$T_{f3}(n) = \Theta(log^2 n).$$

Note that for all the previous examples loop exit condition is a simple inequality for the single algorithm input parameter. Generally, the loop exit condition may be a complex boolean function. However, any boolean function can be represented using the disjunctive normal form (DNF). It is possible to analyze all the equalities and inequalities for each elementary conjunction in order to estimate the lower bound of loop index that makes all the conditions in the conjunction truthy. The minimum of these estimations across all elementary conjunctions in the DNF is the lower bound estimation for the loop. It is possible to estimate the upper bound in the same way. Example 4:

Table 3. Symbol scroll table for Example 3

N_l	i_1	i_2	x	z	y	loop condition	
			n	n			
1	1		$n^{1/2}$	n^2	n^2	$n > 2$	
2		1			$n^2/2$	$n^2/2 > 1$	
		2			$n^2/2^2$	$n^2/2^2 > 1$	
		...			...	...	
		$p_2(1)$			$n^2/2^{p_2(1)}$	$n^2/2^{p_2(1)} > 1$	last
		$p_2(1)+1$			$n^2/2^{p_2(1)+1}$	$n^2/2^{p_2(1)+1} \not> 1$	exit
1	2		$n^{(1/2)^2}$	n^{2^2}	n^{2^2}	$n^{1/2} > 2$	
2		1			$n^{2^2}/2$	$n^{2^2}/2 > 0$	
		2			$n^{2^2}/2^2$	$n^{2^2}/2^2 > 0$	
		...			...	...	
		$p_2(2)$			$n^{2^2}/2^{p_2(2)}$	$n^{2^2}/2^{p_2(2)} > 1$	last
		$p_2(2)+1$			$n^{2^2}/2^{p_2(2)+1}$	$n^{2^2}/2^{p_2(2)+1} \not> 1$	exit
...	...	...	...	...	...	...	
1	i		$n^{(1/2)^i}$	n^{2^i}	n^{2^i}	$n^{(1/2)^{i-1}} > 2$	
2		1			$n^{2^i}/2$	$n^{2^i}/2 > 1$	
		2			$n^{2^i}/2^2$	$n^{2^i}/2^2 > 1$	
		...			...	...	
		$p_2(i)$			$n^{2^i}/2^{p_2(i)}$	$n^{2^i}/2^{p_2(i)} > 1$	last
		$p_2(i)+1$			$n^{2^i}/2^{p_2(i)+1}$	$n^{2^i}/2^{p_2(i)+1} \not> 1$	exit
...	...	...	...	...	...	...	
1	p_1		$n^{(1/2)^{p_1}}$	$n^{2^{p_1}}$	$n^{2^{p_1}}$	$n^{(1/2)^{p_1-1}} > 1$	last
2		1			$n^{2^{p_1}}/2$	$n^{2^{p_1}}/2 > 1$	
		2			$n^{2^{p_1}}/2^2$	$n^{2^{p_1}}/2^2 > 1$	
		...			...	...	
		$p_2(p_1)$			$n^{2^{p_1}}/2^{p_2(p_1)}$	$n^{2^{p_1}}/2^{p_2(p_1)} > 1$	last
		$p_2(p_1)+1$			$n^{2^{p_1}}/2^{p_2(p_1)+1}$	$n^{2^{p_1}}/2^{p_2(p_1)+1} \not> 1$	exit
1	p_1+1					$n^{(1/2)^{p_1}} \not> 2$	exit

The symbol scroll table for this case includes one loop condition for each elementary conjunction in the DNF.

Condition 1 analysis gives bounds $\log_2 n - 1 \le p < \log_2 n$ which may give $\Theta(\log n)$ estimation. The analysis for condition 2 gives different bounds: $n/128 - 17 \le p < n/128 - 16$ which may give $\Theta(n)$ estimation. Therefore, an algorithm may have different estimations in different intervals of the n parameter. In order to get the precise bounds of these intervals, it makes sense to write down equalities for both the upper and the lower bounds: $n/128 - 16 = \log_2 n, n/128 - 17 = \log_2 n - 1$; Equalities are the same, so we can focus only on one of them. Its roots are numbers $n_1 = 3558, n_2 \approx 0.00001$.

It is clear that the loop will not be executed if $n \in [0, 2048]$ hence n_2 is irrelevant. If

```
void f4(unsigned long n) {
    float x, y;
    x = y = n;
    while (x > 1 || y < 2048) {
        x = x / 2;
        y = y - 128;
    }
}
```

Table 4. Symbol scroll table for Example 4

i	x	y	condition 1	condition 2	
	n	n			
1	$n/2$	$n-128$	$n/2>1$	$n-128>2048$	
2	$n/2^2$	$n-128\cdot 2$	$n/2^2>1$	$n-128\cdot 2>2028$	
...	...	...	...	...	
i	$n/2^i$	$n-128\cdot i$	$n/2^i>1$	$n-128\cdot i>2048$	
...	...	...	...	...	
p	$n/2^p$	$n-128\cdot p$	$n/2^p>1$	$n-128\cdot p>2048$	last
$p+1$	$n/2^{p+1}$	$n-128\cdot(p+1)$	$n/2^{p+1}>1$	$n-128\cdot(p+1)>2048$	exit

$n \in [2049, 3558]$, then the estimation is linear, and if $n \in [3559, +\infty)$, then the estimation is logarithmic. Therefore, the algorithm complexity is $\Theta(\log n)$.

3. Loop conditions analysis and its simplification for certain cases

The following relation is truthy for all the previously mentioned loop conditions:

$$f(p,n)\text{<relation sign><expression that does not contain } n \text{ or } p\text{>},$$

where n is a natural input parameter, p is the loop execution number parameter. In simple cases of $f(p,n)$ it is relatively easy to obtain the $\Theta(n)$ estimation. However, in more advanced cases it is sufficient to obtain the estimation for a simpler function $g(p,n)$, which allows to conduct the further analysis of the following equality:

$$f(p,n) = \Theta(g(p,n))$$

that means

$$\exists C_1 > 0, C_2 > 0, n_0, \forall p > 0, n \geq n_0 : C_1 \cdot g(p,n) \leq f(p,n) \leq C_2 \cdot g(p,n).$$

If the algorithm is an integer one (parameters can be only integers because of some transformation), then it may be too hard to analyze the necessary expressions. Finding

sums for the sequences that are not arithmetic or geometric may also be too hard. The following theorems allow to get the Θ-notation estimation for multiple operations on integer parts of linear expressions for the n parameter, or for integrals of integer sums.

Let $\mu_1(a \cdot n + b) = a \cdot n + b$ is a linear form, n is a natural number and $a > 0$. Let $\mu_p(a \cdot n + b) = a \cdot (a \cdot \ldots (a \cdot n + b) + \cdots + b) + b$ is a μ_1 form applied p times, and let $\mu_p([a \cdot n + b]) = [a \cdot [a \cdot \ldots [a \cdot n + b] + \cdots + b] + b]$, where square brackets are the integer part of the expression. It is the μ_1 form applied p times to the integer part of the expression. If the amount of loop executions is expressed via such a function, then we need to be able to estimate it. Theorem 1 allows to remove the integer operation for the sake of getting Θ-notation estimation.

Theorem 1. *Let $a, b \in \mathbb{R}$ are the coefficients of the linear form $a \cdot n + b$, where $a > 0$, $a \neq 1, n \in \mathbb{N}$. Then the following equality is truthy:*

$$\mu_p([a \cdot n + b]) = \Theta(\mu_p(a \cdot n + b)) = \Theta(a^p n).$$

An example for using theorem 1 is the analysis of the following algorithm A:

```
void A (unsigned long N) {
    for (unsigned long k = N; k > 1; k = k/2);
}
```

The loop is executed p times and at the last execution the variable k is 1. So, the condition $[1/2 \cdot [1/2 \cdot \ldots [1/2 \cdot n] \ldots]] = 1$ is transformed to $1 \leq (1/2)^p \cdot N < 2$. Applying logarithm to the both sides gives $\log_2 N - 1 < p \leq \log_2 N$, hence, algorithm complexity is estimated as $\Theta(\log_2 N)$.

Theorem 2 allows to estimate a finite sum by estimating an integral using Θ-notation.

Theorem 2. *Let $f(x), x \geq 0$ is a non-negative monotonic growing function. Let $f(x) \leq C \cdot f(x-1), C \geq 1$. Then the following equality is truthy:*

$$\sum_{x=m}^{n} f(x) = \Theta\left(\int_{m-1}^{n} f(x)dx\right) (\forall m > 0).$$

An example for using theorem 2 is the analysis of the following algorithm B:

```
void B (unsigned long n) {
    unsigned long m = 0;
    for (unsigned long i = 1, j = 2; i < n; i++, j <<= 1)
        m += i * j;
    while (m--);
}
```

The first loop complexity is $\Theta(n)$, and the second loop complexity is $\Theta\left(\sum_{i=1}^{n} i \cdot 2^i\right) = \Theta\left(\int_{x=0}^{n} x \cdot 2^x dx\right) = \Theta(n \cdot 2^n)$. So, B complexity is $\Theta(n \cdot 2^n)$.

Theorem condition is substantial because without it the integral may be a non-elementary function. However, it is still possible to estimate algorithm complexity as Θ-notation for a function with an argument value that is equal to the sum upper bound.

Theorem 3. *Let* $f(x), x \geq 0$ *is a positive monotonic growing function. Let* $g(x) \equiv \frac{f(x)}{f(x-1)}, x \geq 1$ *is a non-negative monotonic growing function with no upper bound. Then the following equality is truthy:*

$$\sum_{x=m}^{n} f(x) = \Theta\left(f(n)\right), (\forall n > m > 0).$$

An example for using theorem 3 is the analysis of the following algorithm C:

```
void C (unsigned long n) {
    unsigned long k = 0, m = 1;
    for (unsigned long i = 1; i <= n; i++) {
        m *= i;
        k += m;
        while (k--);
    }
}
```

The first loop complexity is $\Theta(n)$, and the second loop complexity is $k = \sum_{i=1}^{n} i!$. Since $g(i) = \frac{i!}{(i-1)!} = i$ and $g(i) > 2$ when $i > n_0 = 2$, using theorem 3 $\Theta\left(\sum_{i=1}^{n} i!\right) = \Theta(n!)$. Using Stirling's approximation we can find C complexity as $\Theta(n^n)$.

4. Normal system of symbol transformations

The previously mentioned theorems allow to speed up the algorithm complexity analysis for the algorithms of certain classes. But since it is not possible in every case, it is necessary to transform the symbol expressions and the equalities and inequalities containing the symbol expressions. Hence the issue to control correctness of such transformations done by the students. Using complex mathematical packages like *Mathcad* is not correct since they do not help in teaching students to perform such transformations. We propose the following solution: 1) select the parts of the analysis process that require such transformations, 2) select a limited set of allowed transformations which can be used to express any transformation. Let this set be called *normal transformations*.

The following use cases require symbol transformations:

- algorithm symbol scroll with transforming expressions in the table;
- writing inequalities for loop parameter;
- symbol transformation of the equalities and inequalities for both the variables and loops executions amount;
- algorithm complexity estimation through loop complexity.

The first allowed transformation is the order change of two immediate additive terms or factors. The rest of allowed transformations do not change the order of the affected

terms or factors. It is easier to control correctness of such transformations, but they still allow to express any necessary transformation.

The list of the allowed symbol transformations:

- changing order of terms or factors;
- moving a term from one side of equality to the other with changing its sign;
- reducing a fraction by a single factor;
- factoring a single factor out;
- factorial expansion by a single factor;
- using fundamental equalities (a set of pre-defined functions);
- symbol parentheses removing;
- symbol grouping by factoring a single factor out;
- symbol factorization;
- symbol factorization for powers;

There is a similar system of the normal symbol transformations for inequalities, which requires a few additional transformations:

- moving lead additive term (maximum growth speed) to the first place on one of sides;
- strengthening the upper bound estimation by discarding negative additive terms;
- strengthening the lower bound estimation by discarding positive additive terms;
- estimating a non-lead positive additive term in the upper bound via lead term;
- estimating a non-lead negative additive term in the lower bound via lead term;

This system of normal symbol transformations requires additional sections in the adaptive teaching system to teach it to the students. So, we propose the following order of the sections in the teaching system:

1. Algorithm complexity characteristics
2. Determining algorithm computational complexity
3. System of normal symbol transformations for equalities
4. Algorithm symbol scroll table
5. System of normal symbol transformations for inequalities
6. Estimating complexity of an algorithm with a single loop

7. Estimating complexity of an algorithm with a nested independent loops
8. Estimating complexity of an algorithm with a non-nested dependent loops
9. Estimating complexity of an algorithm with a nested dependent loops
10. Estimating complexity of an algorithm with an integer transformations
11. Estimating complexity of an algorithm with a sequence sum
12. Final exam

It is possible to further divide the sections that may prove to be too hard, e.g., the sections about the system of the normal symbol transformations.

5. Conclusion

This method of teaching Computational Complexity of Algorithms course of study allows to

- develop the methodical programs that extract information and control events for each section;
- develop the software that allows to automate all the stages of the learning process.

We hope that we will manage to implement the described method and that it will show its effectiveness in teaching this course to the students and developing their logical-mathematical thinking.

References

[1] Ermilova A. V., Rublev V. S., "Problemy razvitiya matematicheskogo myshleniya uchashchikhsya na primere obuchayushchey sistemy po kursu "Algoritmy i analiz slozhnosti"", *Sovremennye informatsionnye tekhnologii i IT-obrazovanie*, Sbornik izbrannykh trudov IX Mezhdunarodnoy nauchno-prakticheskoy konferentsii, INTUIT.RU, Moskva, 2014, 297–304, (in Russian).

[2] Cormen T. H., *Introduction to algorithms*, 3rd ed., MIT press, 2009.

Рублев В. С., Юсуфов М. Т., "Автоматизированная Обучающая Система для обучения курсу анализа сложности алгоритмов", *Моделирование и анализ информационных систем*, **24**:4 (2017), 481–495.

DOI: 10.18255/1818-1015-2017-4-481-495

Аннотация. В данной работе исследуются вопросы построения автоматизированной обучающей системы “Анализ сложности алгоритмов”, которая позволит учащемуся освоить сложный математический аппарат и развить логико-математическое мышление в этом направлении. Вводится технология символьной прокрутки алгоритма, позволяющая получать верхние и нижние оценки вычислительной сложности. Приводятся утверждения, облегчающие анализ в случае целочисленного округления параметров алгоритма, а также при оценке сложности сумм. Вводится

нормальная система символьных преобразований, позволяющая, с одной стороны, делать учащемуся любые символьные преобразования, а с другой стороны – упростить автоматический контроль корректности таких преобразований. Статья публикуется в авторской редакции.

Ключевые слова: автоматизированное обучение, анализ сложности алгоритмов, таблица символьной прокрутки алгоритма, нормальная система символьных преобразований

Об авторах:
Рублев Вадим Сергеевич, orcid.org/0000-0002-0252-9958, канд. физ.-мат. наук, профессор,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003, Россия, e-mail: roublev@mail.ru

Юсуфов Мурад Теймурович, orcid.org/0000-0002-0362-2951, аспирант,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003, Россия, e-mail: flood4life@gmail.com

Моделирование и анализ информационных систем. Т. 24, №4 (2017), с. 496–507
Modeling and Analysis of Information Systems. Vol. 24, No 4 (2017), pp. 496–507

DOI: 10.18255/1818-1015-2017-4-496-507

УДК 519.7

Синтез тестов с гарантированной полнотой для недетерминированных временных автоматов

Твардовский А. С.[1], **Эль-Факи К.**, **Громов М. Л.**[1], **Евтушенко Н. В.**[1]

получена 22 декабря 2016

Аннотация. В настоящее время при описании поведения дискретных систем достаточно часто необходимо принимать во внимание временные аспекты, и соответственно появляется необходимость в распространении автоматных методов синтеза тестов с гарантированной полнотой на временные автоматы. В данной статье мы предлагаем метод построения проверяющих тестов с гарантированной полнотой для полностью определенного, возможно, недетерминированного автомата с одной временной переменной. Такие временные автоматы используются при описании поведения программного обеспечения и цифровых устройств. Область неисправности содержит все полностью определенные автоматы с заданным числом состояний и известной верхней оценкой на интервалы, описывающие временные ограничения. Предлагаемый метод опирается на построение по заданному временному автомату соответствующей конечно автоматной абстракции (абстрактного автомата). По абстрактному автомату строится проверяющий тест, последовательности которого суть временные входные последовательности. Более короткие тесты можно построить, если ввести дополнительные ограничения на область неисправности, например, для случая, когда известна наименьшая продолжительность каждого временного интервала в тестируемой реализации, и её величина больше двух. Кроме того, тест можно сократить с сохранением его полноты в случае, когда все интервалы для временных ограничений закрыты справа (или все интервалы закрыты слева). Приводятся результаты проведенных компьютерных экспериментов по сравнению длин тестов, построенных по временному автомату различными методами.

Ключевые слова: временные автоматы, недетерминированные автоматы, синтез тестов, область неисправности

Для цитирования: Твардовский А. С., Эль-Факи К., Громов М. Л., Евтушенко Н. В., "Синтез тестов с гарантированной полнотой для недетерминированных временных автоматов", *Моделирование и анализ информационных систем*, **24**:4 (2017), 496–507.

Об авторах:
Твардовский Александр Сергеевич, orcid.org/0000-0001-7705-7214, магистрант,
Национальный исследовательский Томский государственный университет,
пр. Ленина, 36, г. Томск, 634050 Россия, e-mail: tvardal@mail.ru

Калед Эль-Факи, д-р компьютерных наук, доцент,
Американский университет Шарджа, Университетский город, г. Шарджа, 26666 Объединенные Арабские Эмираты,
e-mail: kelfakih@aus.edu

Громов Максим Леонидович, orcid.org/0000-0002-2990-8245, канд. физ.-мат. наук, доцент,
Национальный исследовательский Томский государственный университет, пр. Ленина, 36, г. Томск, 634050 Россия,
e-mail: maxim.leo.gromov@gmail.ru

Евтушенко Нина Владимировна, orcid.org/0000-0002-4006-1161, д-р техн. наук, профессор,
Национальный исследовательский Томский государственный университет, пр. Ленина, 36, г. Томск, 634050 Россия,
e-mail: nyevtush@gmail.com

Благодарности:
[1]Работа выполнена при поддержке гранта РНФ No. 16-49-03012.

Введение

Автоматные методы предполагают построение тестовых последовательностей по заданному автомату-спецификации, для того чтобы определить, соответствует ли тестируемая реализация, поведение которой также описано автоматом, спецификации. При этом достаточно часто реализация рассматривается как «чёрный ящик». Для проверки соответствия на тестируемую реализацию подаются входные последовательности, и генерируемые при этом выходные последовательности сравниваются с ожидаемыми выходными реакциями (согласно спецификации). Если наблюдаемые реакции не соответствуют спецификации, то в тестируемой реализации имеется ошибка, т. е. реализация не является *конформной* спецификации. Для конечных автоматов существуют методы построения полных проверяющих тестов относительно определенных моделей неисправности [1–3] без явного перечисления неконформных автоматов. Достаточно часто рассматривается случай, когда поведение автомата-спецификации является недетерминированным, однако поведение тестируемой системы описывается детерминированным автоматом, и автомат-реализация считается конформным спецификации, если его поведение содержится в поведении спецификации. Другими словами, в этом случае предполагается, что недетерминизм в спецификации является следствием опциональности в неформальных описаниях, и поведение конформной реализации «не выходит за рамки», предписанные спецификацией. При условии, что поведение исследуемой системы описано временным автоматом (см., например, [4–8]), появляется необходимость в распространении автоматных методов синтеза тестов с гарантированной полнотой на временные автоматы.

В настоящей статье мы рассматриваем полностью определенные, возможно, недетерминированные автоматы с входными временными ограничениями и выходными задержками на переходах [5, 8] и используем конечно автоматный метод построения проверяющих тестов относительно редукции в предположении, что известны максимальное число состояний проверяемого автомата и максимальная конечная граница для входных временных интервалов. Построенный тест можно существенно сократить, если все входные интервалы автомата-спецификации и автомата-реализации закрыты слева (или все интервалы закрыты справа). Дальнейшее сокращение проверяющего теста с использованием подходящей конечно автоматной абстракции можно получить для случая, когда известно, что минимальная длина временных интервалов в автомате-спецификации и проверяемом автомате больше двух.

Структура статьи следующая. Раздел 1 содержит основные определения и обозначения. Алгоритм построения полного проверяющего теста относительно редукции по конечно автоматной абстракции представлен в разделе 2. Раздел 3 содержит экспериментальные результаты.

1. Основные определения и обозначения

В данном разделе мы вводим основные определения и обозначения, взятые преимущественно из [3, 6]. Под *конечным автоматом* S понимается пятёрка $(S, I, O, \lambda_{\mathsf{S}}, s_0)$, где S, I, и O – конечные непустые множества состояний, входных и выходных символов соответственно, s_0 – начальное состояние, $\lambda_{\mathsf{S}} \subseteq S \times I \times O \times S$ – отношение

переходов. Временной, возможно, недетерминированный и частичный автомат (далее – временной автомат) есть конечный автомат с временной переменной и временными ограничениями на переходах. Таким образом, под *временным автоматом* S понимается пятёрка $(S, I, O, \lambda_{\mathsf{S}}, s_0)$, где S, I, и O – конечные непустые множества состояний, входных и выходных символов соответственно, s_0 – начальное состояние, $\lambda_{\mathsf{S}} \subseteq S \times I \times O \times S \times \Pi \times Z$ – отношение переходов. Для λ_{S} через Π обозначено множество входных временных интервалов и через Z – множество выходных задержек. Временной входной интервал g_i описывает промежуток времени, когда переход может быть совершён и задаётся в виде $\lceil min, max \rceil$, где $\lceil \in \{(, [\}, \rceil \in \{),]\}$. В этом обозначении min и max суть неотрицательные целые числа, $min \leq max$, кроме того, max может быть равно ∞. В случае, когда $min = max$, допускается единственный интервал $[min;\ min]$. Выходная задержка представляет собой целое неотрицательное число, которое описывает число тактов времени, затраченное на обработку входного символа, т.е. время между подачей входного и получением выходного символа. Пусть $(s, i, o, s', g_i, d) \in S \times I \times O \times S \times \Pi \times Z$, и на автомат S, находящийся в состоянии s, поступает входной символ i в момент времени $t \in g_i$, измеряемый с момента перехода автомата в состояние s. Тогда временная переменная устанавливается в 0, автомат S выдаёт выходной символ o в момент времени d, и автомат переходит в состояние s' вновь с обнулением временной переменной.

Для временного автомата $\mathsf{S} = (S, I, O, \lambda_{\mathsf{S}}, s_0)$ пара (i, t), где $i \in I$ и t неотрицательное действительное число, называется *временным входным символом*, который означает, что входной символ i подается в момент времени t после выдачи автоматом последнего выходного символа. Аналогично определяется *временной выходной символ* для целого числа t. Последовательность временных входных (выходных) символов называется *временной входной* (*выходной*) *последовательностью*. Временная выходная последовательность, соответствующая временной входной последовательности α, поступившей на автомат в состоянии s, называется (*выходной*) *реакцией* автомата в состоянии s на последовательность α. Последовательность $(i_1,\ t_1)/(o_1,\ d_1) \ldots (i_m,\ t_m)/(o_m,\ d_m)$ пар временных входных и выходных символов есть временная *входо-выходная* последовательность автомата S в состоянии s, если во временном автомате существует последовательность переходов $(s_j, i_j, o_j, s_{j+1}, g_j, d_j)$, такая что $s_1 = s$ и для любого $j = 1, \ldots, m$, справедливо, что $t_j \in g_j$.

На рисунке 1 представлен автомат S с временными ограничениями, начальным состоянием которого является состояние 1. Под действием входного символа $(i,\ 0)$ автомат остаётся в состоянии 1 и выдает выходную реакцию o либо через один такт времени (после подачи входного символа), либо через 2 такта времени. Если входной символ i подать при значении временной переменной 1, то автомат выполнит переход в состояние 3 и выдаст выходную реакцию o через один такт времени, либо перейдёт в состояние 2 и выдаст выходную реакцию o через три такта.

Автомат с временными ограничениями называется *полностью определённым*, если поведение автомата определено в каждом состоянии для каждой временной входной последовательности. Если для любых двух кортежей $(s,\ i,\ o_1,\ s_1,\ g_1,\ d_1)$, $(s,\ i,\ o_2,\ s_2,\ g_2,\ d_2) \in \lambda_{\mathsf{S}}$ справедливо соотношение $g_1 \bigcap g_2 = \varnothing$, то временной автомат называется *детерминированным*, иначе – *недетерминированным*. Недетерминированный автомат называется *наблюдаемым*, если для каждой тройки «состояние, временной входной символ, временной выходной символ» следующее состояние

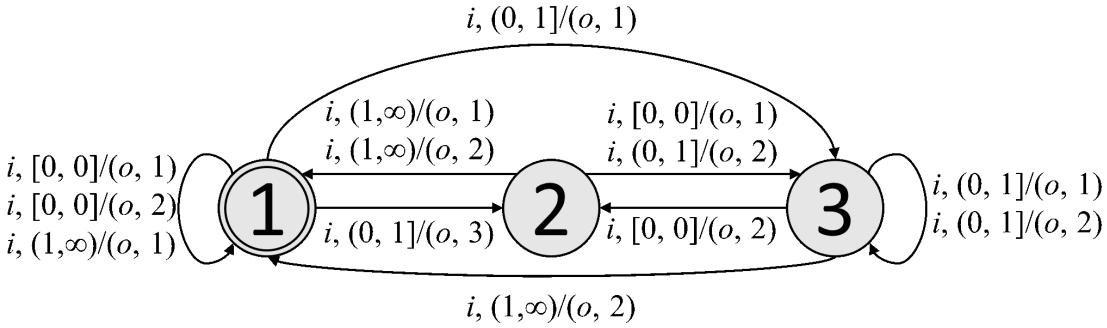


Рис. 1. Временной автомат S
Fig. 1. Timed Finite State Machine S

определяется единственным образом. Для любого недетерминированного автомата существует эквивалентный наблюдаемый автомат. В нашей работе все временные автоматы полагаются полностью определенными, автомат-реализация является детерминированным временным автоматом, в то время как автомат-спецификация может быть недетерминированным, но наблюдаемым.

Состояние p временного автомата P является *редукцией* состояния s временного автомата S ($p \leq s$), если множество входо-выходных последовательностей автомата P в состоянии p содержится в множестве входо-выходных последовательностей автомата S в состоянии s. Автомат P называется редукцией автомата S, если отношение редукции выполняется для начальных состояний этих автоматов.

Проверяющий тест для конечного автомата представляет собой множество входных последовательностей, позволяющее определить, конформна (соответствует) ли спецификации тестируемая реализация. В настоящей работе реализация *конформна* спецификации, если реализация есть редукция спецификации, т.е. реализация конформна спецификации, если и только если для каждой временной входной последовательности выходная реакция реализации содержится в множестве выходных реакций спецификации. В соответствии с [8] полный проверяющий тест для временного автомата относительно редукции можно построить по его конечно автоматной абстракции.

Пусть S – временной автомат и B – целое число, не меньшее чем наибольшая конечная граница B_{S} для временных входных интервалов, в то время как D_{S} определяет наибольшую выходную задержку. Построим *конечно автоматную абстракцию* временного автомата, которая является полностью определённым конечным автоматом $\mathsf{A}_{\mathsf{S}}(B) = (S, I_A, O_A, \lambda_{\mathsf{AS}}, s_0)$, где $I_A = \{(i,0),(i,(0,1)),\ldots,(i,B),(i,(B,\infty)) : i \in I\}$, а $O_A = \{(o,0),(o,1),\ldots,(o,D_{\mathsf{S}}) : o \in O\}$. Для состояния s автомата $\mathsf{A}_{\mathsf{S}}(B)$ и входного символа $(i,t_j), t_j = 0,\ldots,B$, множество λ_{AS} содержит переход $(s,(i,t_j),(o,d),s')$, если и только если существует переход $(s,i,o,s',g_i,d) \in \lambda_{\mathsf{S}}$, такой что $t_j \in g_i$. Для входного символа $(i,g), g = (0,1),\ldots,(B-1,B),(B,\infty)$, множество λ_{AS} содержит переход $(s,(i,g),(o,d),s')$, если и только если существует переход $(s,i,o,s',g_i,d) \in \lambda_{\mathsf{S}}$, такой что $g \subseteq g_i$. Если временной автомат S недетерминированный, то конечный автомат $\mathsf{A}_{\mathsf{S}}(B)$ также будет недетерминированным. Для временного автомата на рисунке 1 конечно автоматная абстракция изображена на рисунке 2.

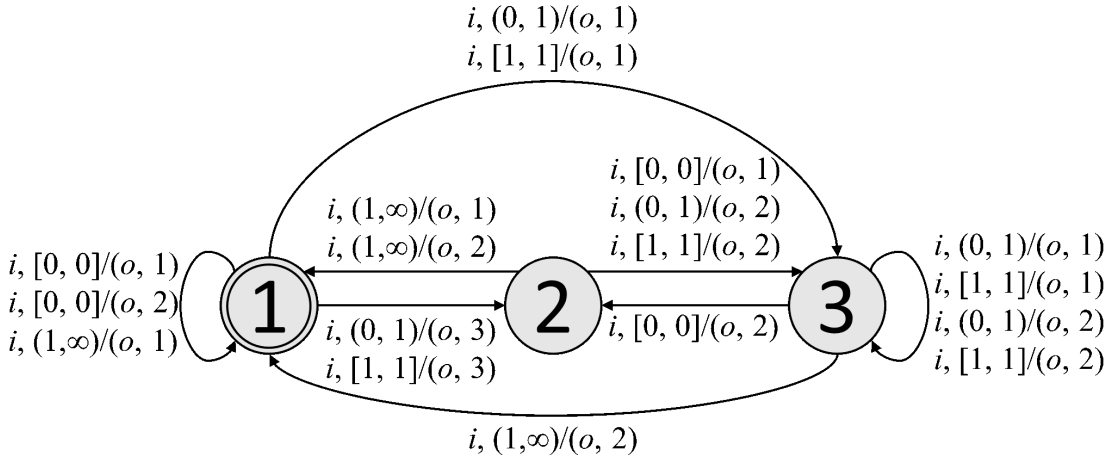


Рис. 2. Конечный автомат $\mathsf{A_S}(1)$
Fig. 2. Finite State Machine $\mathsf{A_S}(1)$

Аналогично [9] можно сформулировать и доказать следующее утверждение.

Утверждение 1. *Для двух полностью определённых наблюдаемых, возможно, недетерминированных временных автоматов* P *и* S *с наибольшей входной границей* B*, автомат* P *есть редукция автомата* S*, если и только если отношение редукции выполняется для их конечно автоматных абстракций, т.е. если и только если* $\mathsf{A_P}(B)$ *есть редукция* $\mathsf{A_S}(B)$*.*

Действительно, в начальном состоянии конечно автоматной абстракции $\mathsf{A_S}(B)$ возможен переход по временному входному символу (i, t), если и только если в начальном состоянии исходного автомата для входного символа i существует интервал g, где $t \in g$. Далее утверждение доказывается по индукции.

Мы рассматриваем *модель неисправности* $< \mathsf{S}, \leq, \mathfrak{J}_m(B) >$ [9], где S – временной автомат-спецификация, который является полностью определённым и наблюдаемым, $\leq$ – отношение редукции и $\mathfrak{J}_m(B)$ – область неисправности, содержащая каждый полностью определённый детерминированный временной автомат с таким же входным алфавитом, как и у спецификации, не более чем с m состояниями и наибольшей конечной границей B для входных временных интервалов.

Проверяющий тест есть конечное множество конечных временных входных последовательностей спецификации. Тест является *полным* относительно $< \mathsf{S}, \leq, \mathfrak{J}_m(B) >$, если для каждого временного автомата $\mathsf{P} \in \mathfrak{J}_m(B)$, такого что P есть редукция S, выходная реакция на каждую последовательность теста содержится в множестве выходных реакций S на эту последовательность, в то время как для каждого временного автомата $\mathsf{P} \in \mathfrak{J}_m(B)$, такого что P не является редукцией S, в тесте есть последовательность, выходная реакция на которую не принадлежит множеству выходных реакций S на эту последовательность. Согласно утверждению 1, полный проверяющий тест относительно $< \mathsf{S}, \leq, \mathfrak{J}_m(B) >$ может быть построен с использованием классических конечно автоматных методов относительно модели неисправности $< \mathsf{A_S}(B), \leq, \mathfrak{J}_m(B) >$. Здесь $\mathsf{A_S}(B)$ – конечно автоматная абстракция

временного автомата S и $\mathfrak{J}_m(B)$ – множество всех полностью определённых детерминированных временных автоматов не более чем с m состояниями и таким же входным алфавитом, как у $\mathsf{A}_\mathsf{S}(B)$. Мы далее вводим ряд понятий, которые используются в следующем разделе при построении полного проверяющего теста.

Состояния s_1 и s_2 автомата $\mathsf{A}_\mathsf{S}(B)$ *разделимы*, если существует входная последовательность α, такая что множества выходных реакций на α в состояниях s_1 и s_2 не пересекаются; последовательность α называется *разделяющей* для состояний s_1 и s_2.

Входная последовательность α называется *адаптивной*, если следующий входной символ зависит от реакции автомата на предыдущие входные символы. Адаптивную входную последовательность удобно представлять в виде специального автомата, который во многих работах называется *тестовым примером* (test case) [10]. В тестовом примере в каждом состоянии определен переход либо только по одному входному символу со всеми выходными символами, либо не определён ни один переход (это состояние назовём *терминальным*). Диаграмма переходов тестового примера есть ациклический граф (рис. 3).Тестовый пример представляет *адаптивную различающую последовательность* для состояний s_1 и s_2 автомата $\mathsf{A}_\mathsf{S}(B)$, если каждая входо-выходная последовательность из начального в терминальное состояние тестового примера возможна не более чем в одном из состояний s_1 или s_2. В первом случае распознаётся состояние s_1, во втором – s_2. Состояния s_1 и s_2 автомата $\mathsf{A}_\mathsf{S}(B)$ *адаптивно различимы*, если существует тестовый пример, который представляет адаптивную различающую последовательность для состояний s_1 и s_2. Если адаптивная последовательность различает каждую пару состояний автомата S, то такая последовательность есть адаптивная различающая последовательность для автомата S. Тестовый пример на рис. 3 представляет адаптивную различающую последовательность для автомата $\mathsf{A}_\mathsf{S}(1)$ на рис. 2. В вершинах графа переходов тестового примера содержатся подмножества состояний, достижимые по соответствующей входо-выходной последовательности. Заметим, что для этого автомата не существует разделяющей последовательности, существует только адаптивная различающая последовательность. В терминальных состояниях также указано состояние (`in_st`), в котором находился автомат перед поступлением адаптивной различающей последовательности.

Состояние s *детерминировано* (*∂-*) *достижимо* из начального состояния, если существует входная последовательность α такая, что при любой выходной реакции β на последовательность α автомат $\mathsf{A}_\mathsf{S}(B)$ переходит из начального состояния в состояние s. В этом случае α называется (*∂-*) передаточной последовательностью для состояния s.

Тестовый пример представляет *адаптивную передаточную последовательность* из начального состояния автомата $\mathsf{A}_\mathsf{S}(B)$ в состояние s, если каждая входо-выходная последовательность из начального в терминальное состояние тестового примера заканчивается в состоянии s [10]. В этом случае состояние s *адаптивно достижимо* из начального состояния. Тестовые примеры на рис. 4 представляют адаптивные передаточные последовательности из состояния 1 в состояния 2 и 3 для автомата на рис. 2.

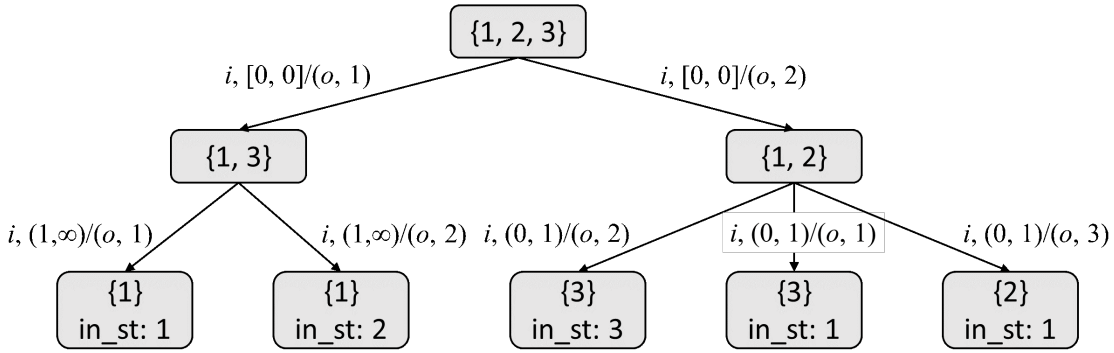


Рис. 3. Тестовый пример, представляющий адаптивную различающую последовательность для автомата $\mathsf{A_S}(1)$

Fig. 3. A distinguishing test case for $\mathsf{A_S}(1)$

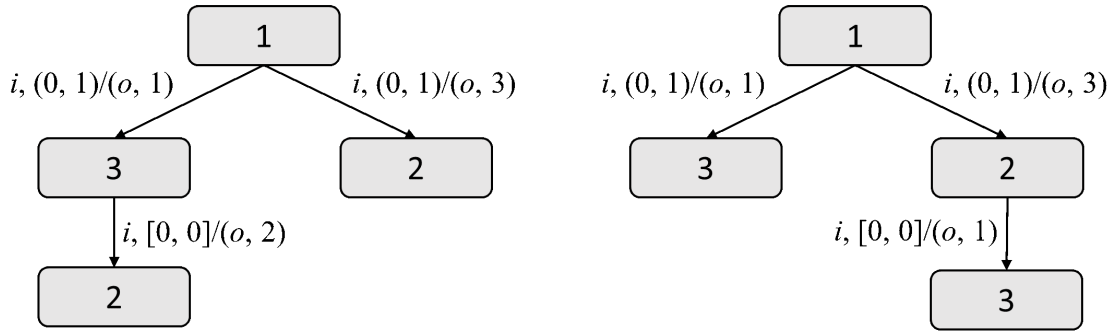


Рис. 4. Тестовые примеры, представляющие адаптивные передаточные последовательности для автомата $\mathsf{A_S}(1)$

Fig. 4. Transfer test cases for $\mathsf{A_S}(1)$

2. Синтез полного проверяющего теста

Общий алгоритм синтеза полного проверяющего теста относительно редукции для недетерминированного конечного автомата описан в [7]. Метод основан на использовании ∂-передаточных и различающих последовательностей, которые, как известно, не всегда существуют. В настоящей работе мы предлагаем использовать адаптивные передаточные и различающие последовательности, поскольку, во-первых, такие последовательности существуют значительно чаще, а во-вторых, в таком случае, длина полного проверяющего теста может быть значительно меньше [11].

Если все состояния конечно автоматной абстракции $\mathsf{A_S}(B)$ автомата S ∂-достижимы (адаптивно достижимы), автомат $\mathsf{A_S}(B)$ обладает разделяющей последовательностью (адаптивной различающей последовательностью), и число состояний реализации не превышает числа состояний спецификации, то алгоритм построения полного проверяющего теста относительно модели неисправности $< \mathsf{A_S}(B), \leq, \mathfrak{J}_m(B) >$, где m — число состояний автомата $\mathsf{A_S}(B)$, включает следующие шаги.

1. Строится *множество д-достижимости* (адаптивной достижимости) для автомата $\mathsf{A}_{\mathsf{S}}(B)$, которое содержит $д$-передаточную (адаптивную передаточную) последовательность для каждого состояния автомата $\mathsf{A}_{\mathsf{S}}(B)$.

2. Каждая последовательность из множества $д$-достижимости конкатенируется с разделяющей (адаптивной различающей) последовательностью и каждым входным символом; после каждого входного символа добавляется разделяющая (адаптивная различающая) последовательность.

Таким образом, в соответствии с утверждением 1, аналогичный подход может быть использован для построения полного проверяющего теста для спецификации, описанной временным автоматом.

Для того чтобы построить полный проверяющий тест для временного недетерминированного автомата, мы используем его конечно автоматную абстракцию. Полный тест строится для конечно автоматной абстракции; построенные входные последовательности (входо-выходные последовательности при построении адаптивного теста) являются временными входными (временными входо-выходными) последовательностями для исходного временного автомата. Построенный тест обладает полиномиальной длиной относительно числа состояний временного автомата-спецификации, когда длины разделяющей последовательности и $д$-передаточных последовательностей (адаптивных различающей и передаточных последовательностей) полиномиальны относительно числа состояний временного автомата-спецификации. Известно, что построенный тест обнаруживает все неконформные реализации с не более чем m состояниями и наибольшей конечной границей B для входных временных интервалов, где m есть число состояний временного автомата-спецификации.

Утверждение 2. *Если автомат $\mathsf{A}_{\mathsf{S}}(B)$ обладает разделяющей (адаптивной различающей) последовательностью и каждое состояние $\mathsf{A}_{\mathsf{S}}(B)$ д-достижимо (адаптивно достижимо) из начального состояния, то выше описанный алгоритм возвращает полный проверяющий тест относительно $< \mathsf{S}, \leq, \mathfrak{J}_m(B) >$.*

Действительно, по построению, автомат $\mathsf{A}_{\mathsf{S}}(B)$ имеет m состояний. Если каждая последовательность множества достижимости конкатенируется с разделяющей (адаптивной различающей) последовательностью, и проверяемый автомат реагирует на эту часть проверяющего теста согласно спецификации, то проверяемый автомат имеет ровно m состояний. Кроме того, можно установить взаимно однозначное соответствие между состояниями спецификации и тестируемого автомата по реакции в каждом состоянии на разделяющую (адаптивную различающую) последовательность. На следующем шаге с использованием этих реакций и по реакции тестируемого автомата на каждый входной символ и последующую разделяющую (адаптивную различающую) последовательность устанавливается взаимно однозначное соответствие между переходами автомата-спецификации и тестируемого автомата. После этого утверждение следует из утверждения 1.

Если спецификация не имеет разделяющей (адаптивной различающей) последовательности или не каждое состояние является детерминированно достижимым (адаптивно достижимым) из начального состояния, то проверяющий тест будет значительно длиннее [3]. В работе [12] показано, что можно сократить спецификацию,

чтобы получить тест разумной длины. В нашем примере на рисунке 1 предположим, что исходная спецификация имеет дополнительный переход $(1, i, o, 3, [0, 0], 1)$ из состояния 1 в состояние 3. Для такой спецификации не существует разделяющей (адаптивной различающей) последовательности, поскольку для любого входного символа существуют два состояния, из которых есть переход в одно и то же состояние с одним и тем же выходным символом. Однако после удаления этого перехода автомат-спецификация обладает необходимыми свойствами. Известно, что необходимое сокращение спецификации не всегда возможно, в частности, для детерминированного автомата-спецификации, и вопросы существования и оптимизации такого сокращения требуют дальнейших исследований.

Аналогично методу синтеза тестов по детерминированному автомату-спецификации [5], длину теста можно уменьшить с сохранением его полноты, если все временные входные интервалы спецификации закрыты слева (или все временные входные интервалы закрыты справа), или продолжительность входных временных интервалов в реализации и спецификации больше двух. В первом случае временные входные символы теста достаточно подавать только в целочисленные моменты времени, т.е. при построении теста из конечно автоматной абстракции спецификации удаляются все входные символы вида (i, g), где g – интервал ненулевой длины. Во втором случае временные входные символы достаточно подавать таким образом, чтобы на каждый входной временной интервал приходился как минимум один временной входной символ. В таком случае, входной алфавит конечно автоматной абстракции имеет вид $I_A = \{(i, 0), (i, (0, 1)), (i, w), (i, (w, w + 1)), (i, 2w), (i, (2w, 2w + 1)), \ldots, (i, n), (i, (n, \infty))) : i \in I, n < B\}$, где $w > 2$ – наименьшая длина временного интервала в спецификации, т.е. входные символы подаются в целочисленные моменты времени, кратные w и один раз в каждом интервале вида $(kw, kw + 1)$.

3. Экспериментальные результаты

В настоящем разделе мы кратко описываем полученные нами экспериментальные результаты. В частности, мы рассматриваем

- проверяющие тесты TS, полученные с использованием выше описанного алгоритма, т.е. тесты без использования какого-либо метода оптимизации;

- тесты $TS1$, построенные для спецификаций, в которых все временные интервалы закрыты слева (или все временные интервалы закрыты справа), и таким образом, достаточно рассматривать только целочисленные моменты времени при подаче входных символов;

- тесты $TS2$, когда все временные интервалы в спецификации и реализации имеют длительность больше, чем два, и для построения тестовых последовательностей используются только моменты времени, кратные минимальной длине временных интервалов и один из моментов времени из интервала $(kw, kw + 1)$;

- тесты $TS3$, когда оба выше упомянутых сокращения использованы в процессе синтеза теста.

Эксперименты проводились над временными автоматами с 20 состояниями и с не более чем десятью временными интервалами для каждой пары «состояние, входной

символ», для каждого автомата были построены тесты TS, $TS1$, $TS2$, и $TS3$ с использованием разделяющей и ∂-передаточных последовательностей. Приведённые экспериментальные данные иллюстрируют существенное различие между длинами тестов TS и $TS3$, которое практически не зависит от размеров автомата и числа входных временных интервалов. На рис. 5 представлено усредненное процентное соотношение (по 1000 экспериментов).

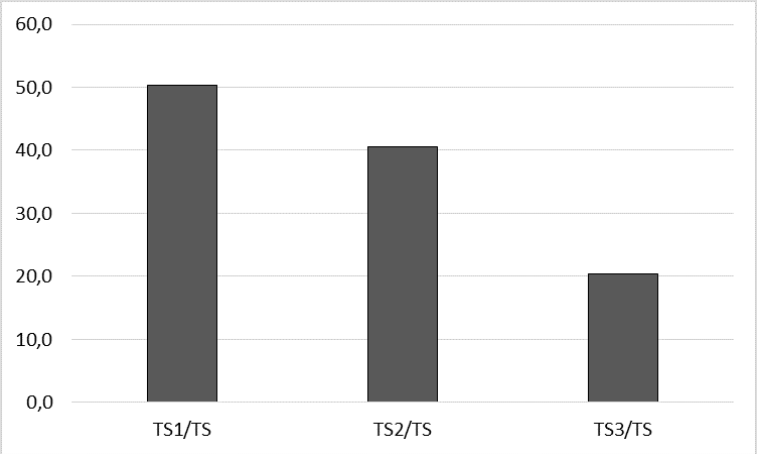


Рис. 5. Зависимость между длинами тестов, построенных различными методами
Fig. 5. Dependence of lengths of tests derived by different methods

При построении тестов $TS1$, $TS2$, $TS3$ нет гарантии, что проверяющий тест будет полным относительно модели неисправности $< \mathsf{S}, \leq, \mathfrak{J}_m(B) >$. Тем не менее, для большого количества случайно сгенерированных реализаций из множества $\mathfrak{J}_m(B)$, для которых $|I| = |O| = 4, |S| = 20$ и каждая из которых отличается от автомата-спецификации только на одном переходе, тест $TS3$ обнаружил каждую неконформную реализацию.

Следует также отметить, что адаптивные различающие и передаточные последовательности всегда существуют при наличии разделяющей и ∂-передаточных последовательностей. Более того, длины адаптивных последовательностей обычно существенно короче, и поэтому полученные экспериментальные результаты справедливы для временных автоматов при использовании адаптивных тестов.

4. Заключение

В настоящей статье предложен подход к синтезу полных проверяющих тестов для недетерминированных временных автоматов относительно редукции. Предложенный подход можно использовать для построения тестов для временных автоматов, когда автомат-спецификация обладает разделяющей (адаптивной различающей) последовательностью и любое состояние ∂-достижимо (адаптивно достижимо) из начального состояния. Кроме того, построенные тесты можно оптимизировать, и несмотря на то, что укороченные тесты теоретически не являются полными, все случайно сгенерированные временные автоматы, неконформные спецификации и

отличающиеся от нее только на одном переходе такими тестами были обнаружены. В дальнейшем для анализа полноты тестов, построенных оптимизированными методами, мы предполагаем исследовать специальные мутации спецификации, а также оценить процент необнаружимых неконформных реализаций для автоматов-спецификаций небольшого размера посредством генерации всех возможных реализаций. Заметим, что если автомат-спецификация не имеет разделяющей (адаптивной различающей) последовательности или не каждое состояние является ∂-(адаптивно) достижимым, то проверяющий тест будет значительно длиннее. Одной из возможностей сокращения проверяющих тестов является сокращение автомата-спецификации, но условия существования и оптимизация такого сокращения требуют дальнейших исследований.

Список литературы / References

[1] Chow T. S., "Test design modeled by finite-state machines", *IEEE Transactions on Software Engineering*, **4**:3 (1978), 178–187.

[2] Dorofeeva R., El-Fakih K., Maag S., Cavalli A., Yevtushenko N., "FSM-based conformance testing methods: a survey annotated with experimental evaluation", *Information and Software Technology*, **52** (2010), 1286–1297.

[3] Petrenko A., Yevtushenko N., "Conformance Tests as checking experiments for partial nondeterministic FSM", *Proc. of the 5th International Workshop on Formal Approaches to Testing of Software (FATES 2005)*, LNCS, **3997**, 2005, 118–133.

[4] Springintveld J., Vaandrager F., D'Argenio P., "Testing timed automata", *Theoretical Computer Science*, **254**:1–2 (2001), 225–257.

[5] El-Fakih K., Yevtushenko N., Fouchal H., "Testing timed finite state machines with guaranteed fault coverage", *Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop*, 2009, 66–80.

[6] Krichen M., Tripakis S., "Conformance testing for real-time systems", *Formal Methods in System Design*, **34**:3 (2009), 238–304.

[7] Merayo M.G., Nunez M., Rodriguez I., "Formal testing from timed finite state machines", *Computer Networks: The International Journal of Computer and Telecommunications Networking*, **52**:2 (2008), 432–460.

[8] Bresolin D., El-Fakih K., Villa T., Yevtushenko N., "Deterministic timed finite state machines: Equivalence checking and expressive power", International conference GANDALF, 2014, 203–216.

[9] Petrenko A., Yevtushenko N., Bochmann G.v., "Fault Models for Testing in Context", *FORTE*, 1996, 163–178.

[10] Yevtushenko N., El-Fakih K., Ermakov A., "On-the-fly construction of adaptive checking sequences for testing deterministic implementations of nondeterministic specifications", *LNCS*, **9976** (2016), 139–152.

[11] Petrenko A., Yevtushenko N., "Adaptive Testing of Deterministic Implementations Specified by Nondeterministic FSMs", *Proceedings of ICTSS*, 2011, 162–178.

[12] Tvardovskii A., Yevtushenko N., "Synthesis complete test suite for nondeterministic finite state machines with time guards", *Russian Physics Journal*, **58**:11/2 (2015), 107–110.

Tvardovskii A. S., El-Fakih K., Gromov M. L., Yevtushenko N. V., "Testing Timed Nondeterministic Finite State Machines with the Guaranteed Fault Coverage", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 496–507.

DOI: 10.18255/1818-1015-2017-4-496-507

Abstract. Nowadays, the behaviour of many systems can be properly described by taking into account time constraints, and this motivates the adaptation of existing Finite State Machine (FSM)-based test derivation methods to timed models. In this paper, we propose a method for deriving conformance tests with the guaranteed fault coverage for a complete possibly nondeterministic FSM with a single clock; such Timed FSMs (TFSMs) are widely used when describing the behaviour of software and digital devices. The fault domain contains every complete TFSM with the known upper bounds on the number of states and finite boundary of input time guards. The proposed method is carried out by using an appropriate FSM abstraction of the given TFSM; the test is derived against an FSM abstraction and contains timed input sequences. Shorter test suites can be derived for a restricted fault domain, for instance, for the case when the smallest duration of an input time guard is larger than two. Moreover, the obtained test suites can be reduced, while preserving the completeness, when all input time guards of the specification and an implementation are right closed (or all intervals are left closed). Experiments are conducted to study the length of test suites constructed by different methods.

Keywords: timed finite state machines, nondeterministic finite state machines, test derivation, fault coverage

On the authors:
Aleksandr S. Tvardovskii, orcid.org/0000-0001-7705-7214, Master student,
National Research Tomsk State University,
36 Lenin Ave., Tomsk, 634050, Russia, e-mail: tvardal@mail.ru

Khaled El-Fakih, PhD in computer science, professor,
American University of Sharjah,
University City, Sharjah, 26666, United Arab Emirates, e-mail: kelfakih@aus.edu

Maksim L. Gromov, orcid.org/0000-0002-2990-8245, PhD in computer science, associate professor,
National Research Tomsk State University,
36 Lenin Ave., Tomsk, 634050, Russia, e-mail: maxim.leo.gromov@gmail.ru

Nina V. Yevtushenko, orcid.org/0000-0002-4006-1161, doctor of technical sciences, professor,
National Research Tomsk State University,
36 Lenin Ave., Tomsk, 634050, Russia, e-mail: nyevtush@gmail.com

Acknowledgments:
This work is partly supported by RSF Project No. 16-49-03012.

Моделирование и анализ информационных систем. Т. 24, №4 (2017), с. 508–515
Modeling and Analysis of Information Systems. Vol. 24, No 4 (2017), pp. 508–515

DOI: 10.18255/1818-1015-2017-4-508-515

УДК 519.17

Разложение самоподобных функций в системе Фабера–Шаудера

Тимофеев Е. А.

получена 6 июля 2017

Аннотация.

Пусть $\Omega = \mathcal{A}^{\mathbb{N}}$ – пространство правосторонних бесконечных последовательностей символов алфавита $\mathcal{A} = \{0, 1\}$, $\mathbb{N} = \{1, 2, \dots\}$. Пусть

$$\rho(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^{\infty} |x_k - y_k| 2^{-k}$$

– метрика на $\Omega = \mathcal{A}^{\mathbb{N}}$, и μ – мера Бернулли на Ω с вероятностями $p_0, p_1 > 0$, $p_0 + p_1 = 1$. Обозначим через $B(\boldsymbol{x}, \omega)$ открытый шар радиуса r с центром в точке $\boldsymbol{\omega}$. Основной результат работы

$$\mu\left(B(\boldsymbol{\omega}, r)\right) = r + \sum_{n=0}^{\infty} \sum_{j=0}^{2^n-1} \mu_{n,j}(\boldsymbol{\omega}) \tau(2^n r - j),$$

где $\tau(x) = 2\min\{x, 1-x\}$, $0 \le x \le 1$, ($\tau(x) = 0$, if $x < 0$ or $x > 1$),

$$\mu_{n,j}(\boldsymbol{\omega}) = \left(1 - p_{\omega_{n+1}}\right) \prod_{k=1}^{n} p_{\omega_k \oplus j_k}, \quad j = j_1 2^{n-1} + j_2 2^{n-2} + \dots + j_n.$$

Семейство функций $1, x, \tau(2^n x - j)$, $j = 0, 1, \dots, 2^n - 1$, $n = 0, 1, \dots$ является системой Фабера – Шаудера в пространстве $C([0, 1])$ непрерывных функций на $[0, 1]$. Также получены разложения в системе Фабера – Шаудера для сингулярной функции Лебега, кривых Чезаро и кривых Коха – Пеано.

Ключевые слова: система Фабера–Шаудера, вейвлета Хаара, самоподобие, функция Лебега

Для цитирования: Тимофеев Е. А., "Разложение самоподобных функций в системе Фабера–Шаудера", *Моделирование и анализ информационных систем*, **24**:4 (2017), 508–515.

Об авторах:
Тимофеев Евгений Александрович, orcid.org/0000-0002-0980-2507, д-р физ.-мат. наук, профессор
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: timofeevEA@gmail.com

Изучению сингулярных функций в последнее время посвящено большое число работ. Одним из основных изученных классов являются самоподобные функции. Самоподобные функции задаются простыми рекуррентными уравнениями, но обычно не допускают явного задания. В настоящей работе будут найдены разложения некоторых непрерывных самоподобных функций в системе Фабера–Шаудера. Коэффициенты разложения найдены в явном виде.

Система Фабера–Шаудера. Семейство функций

$$1, x, \tau(2^n x - j), \quad j = 0, 1, \dots, 2^n - 1, \ n = 0, 1, \dots,$$

где $\tau(x) = 2\min\{x, 1-x\}$, $0 \le x \le 1$, ($\tau(x) = 0$, при $x < 0$ или $x > 1$), называется [1] системой Фабера–Шаудера в пространстве $C([0,1])$ всех непрерывных функций на отрезке $[0,1]$ с нормой $||f|| = \max_{0\le x\le 1} |f(x)|$.

Любая непрерывная функция $f(x)$ представляется в виде

$$f(x) = A_0 + A_1 x + \sum_{n=0}^{\infty} \sum_{j=0}^{2^n-1} A_{n,j} \tau(2^n x - j),$$

где $A_0 = f(0)$, $A_1 = f(1) - f(0)$ и

$$A_{n,j} = f((2j+1)2^{-n-1}) - 0.5 f(j2^{-n}) - 0.5 f((j+1)2^{-n}) \tag{1}$$

см. [1, гл.6 (4)].

Базис Хаара. Семейство функций

$$1, 2^{n/2}\psi(2^n x - j), \quad j = 0, 1, \dots, 2^n - 1, \ n = 0, 1, \dots,$$

где $\psi(t) = 0.5\tau'(t)$ называется вейвлетой Хаара, образует ортонормированный базис в пространстве $L^2(0,1)$ [1].

1. Мера шара в пространстве последовательностей

Рассмотрим пространство правосторонних бинарных последовательностей $\Omega = \mathcal{A}^{\mathbb{N}}$, где $\mathcal{A} = \{0,1\}$, $\mathbb{N} = \{1, 2, \dots\}$ с метрикой

$$\rho(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^{\infty} |x_k - y_k| 2^{-k}. \tag{2}$$

Через μ обозначим меру Бернулли на Ω с вероятностями $p_0, p_1 > 0$, $p_0 + p_1 = 1$.

Через $B(\boldsymbol{\omega}, r)$ обозначим открытый шар радиуса r с центром в точке $\boldsymbol{\omega}$.

Нетрудно видеть, что функция $\mu(\boldsymbol{\omega}, x) = \mu(B(\boldsymbol{\omega}, x))$ удовлетворяет рекуррентному уравнению

$$\mu(a\boldsymbol{\omega}, x) = \begin{cases} p_a \mu(\boldsymbol{\omega}, 2x), & 0 \le x \le 0.5, \\ p_a + p_{1-a}\mu(\boldsymbol{\omega}, 2x-1), & 0.5 \le x \le 1, \end{cases} \tag{3}$$

где $a \in \{0,1\}$ и через $a\boldsymbol{\omega}$ обозначается точка $a, \omega_1, \omega_2, \dots$

Теорема 1. *Функция $\mu(\boldsymbol{\omega}, x)$ имеет следующее разложение по системе Фабера–Шаудера*

$$\mu\left(\boldsymbol{\omega}, x\right) = x + \sum_{n=0}^{\infty} \sum_{j=0}^{2^n-1} \left(p_{\omega_{n+1}} - 0.5\right) p_0^{s_n(\boldsymbol{\omega},j)} p_1^{n-s_n(\boldsymbol{\omega},j)} \tau(2^n x - j), \tag{4}$$

где

$$s_n(\boldsymbol{\omega}, j) = \left|\{i \ : \ \omega_i = j_i, \ j = j_1 2^{n-1} + \cdots + j_n, \ i = 1, 2, \ldots, n\}\right|. \tag{5}$$

Доказательство. Покажем, что функция $\mu(\boldsymbol{\omega}, x)$, заданная в (4), удовлетворяет уравнению

$$\mu(a\boldsymbol{\omega}, x/2) = p_a \mu(\boldsymbol{\omega}, x). \tag{6}$$

Из (4) имеем

$$\mu(a\boldsymbol{\omega}, x/2) = \frac{x}{2} + (p_a - 0.5)\, x + \\ + \sum_{n=1}^{\infty} \sum_{j=0}^{2^{n-1}-1} (p_{\omega_n} - 0.5)\, p_0^{s_n(a\boldsymbol{\omega},j)} p_1^{n-s_n(a\boldsymbol{\omega},j)} \tau(2^{n-1} x - j).$$

Поскольку при $j < 2^{n-1}$

$$s_n(0\boldsymbol{\omega}, j) = s_{n-1}(\boldsymbol{\omega}, j) + 1, \\ s_n(1\boldsymbol{\omega}, j) = s_{n-1}(\boldsymbol{\omega}, j),$$

имеем

$$p_0^{s_n(a\boldsymbol{\omega},j)} p_1^{n-s_n(a\boldsymbol{\omega},j)} = p_a p_0^{s_{n-1}(\boldsymbol{\omega},j)} p_1^{n-1-s_{n-1}(\boldsymbol{\omega},j)}.$$

Подставляя, получим

$$\mu(a\boldsymbol{\omega}, x/2) = p_a x + \\ + p_a \sum_{n=1}^{\infty} \sum_{j=0}^{2^{n-1}-1} (p_{\omega_n} - 0.5)\, p_0^{s_{n-1}(\boldsymbol{\omega},j)} p_1^{n-1-s_{n-1}(\boldsymbol{\omega},j)} \tau(2^{n-1} x - j) = \\ = p_a x + p_a \sum_{n=0}^{\infty} \sum_{j=0}^{2^n-1} \left(p_{\omega_{n+1}} - 0.5\right) p_0^{s_n(\boldsymbol{\omega},j)} p_1^{n-s_n(\boldsymbol{\omega},j)} \tau(2^n x - j) = \\ = p_a \mu(\boldsymbol{\omega}, x).$$

Покажем, что функция $\mu(\boldsymbol{\omega}, x)$, заданная в (4), удовлетворяет уравнению

$$\mu\left(a\boldsymbol{\omega}, \frac{1}{2} + \frac{x}{2}\right) = p_a + p_{1-a} \mu(\boldsymbol{\omega}, x). \tag{7}$$

Из (4) имеем

$$\mu\left(a\boldsymbol{\omega}, \frac{1}{2} + \frac{x}{2}\right) = \frac{1}{2} + \frac{x}{2} + (p_a - 0.5)\,(1 - x) + \\ + \sum_{n=1}^{\infty} \sum_{j=2^{n-1}}^{2^n-1} (p_{\omega_n} - 0.5)\, p_0^{s_n(a\boldsymbol{\omega},j)} p_1^{n-s_n(a\boldsymbol{\omega},j)} \tau(2^{n-1} x + 2^{n-1} - j).$$

Поскольку при $2^{n-1} \le j < 2^n$

$$\begin{aligned} s_n(0\boldsymbol{\omega}, j) &= s_{n-1}(\boldsymbol{\omega}, j - 2^{n-1}), \\ s_n(1\boldsymbol{\omega}, j) &= s_{n-1}(\boldsymbol{\omega}, j - 2^{n-1}) + 1, \end{aligned}$$

имеем

$$p_0^{s_n(a\boldsymbol{\omega},j)} p_1^{n-s_n(a\boldsymbol{\omega},j)} = p_{1-a} p_0^{s_{n-1}(\boldsymbol{\omega},j-2^{n-1})} p_1^{n-1-s_{n-1}(\boldsymbol{\omega},j-2^{n-1})}.$$

Подставляя, получим

$$\begin{aligned} \mu\left(a\boldsymbol{\omega}, \frac{1}{2} + \frac{x}{2}\right) &= p_a + p_{1-a}x + \\ &+ p_{1-a} \sum_{n=1}^{\infty} \sum_{j=2^{n-1}}^{2^n-1} (p_{\omega_n} - 0.5)\, p_0^{s_{n-1}(\boldsymbol{\omega},j-2^{n-1})} p_1^{n-1-s_{n-1}(\boldsymbol{\omega},j)} \tau(2^{n-1}x - j) = \\ &= p_a x + p_{1-a} \sum_{n=0}^{\infty} \sum_{j=0}^{2^n-1} \left(p_{\omega_{n+1}} - 0.5\right) p_0^{s_n(\boldsymbol{\omega},j)} p_1^{n-s_n(\boldsymbol{\omega},j)} \tau(2^n x - j) = \\ &= p_a + p_{1-a}\mu(\boldsymbol{\omega}, x). \end{aligned}$$

Доказанные равенства (6), (7) эквивалентны определению (3) функции $\mu(\boldsymbol{\omega}, x)$. □

2. Кривые де Рама с двумя линейными сжатиями

Напомним конструкцию де Рама [3], которая позволяет построить большое число известных фрактальных кривых, например функции Кантора, Минковского, Такаги, кривые Пеано и Коха.

Пусть в метрическом пространстве M с метрикой d заданы два сжимающих отображения

$$\begin{aligned} f_0 &: \ M \to M; \\ f_1 &: \ M \to M. \end{aligned} \tag{8}$$

Кривая де Рама $p(x) \in M$, $x \in [0, 1]$ задается следующим образом. Пусть $x \in [0, 1]$ имеет бинарное разложение

$$x = \sum_{k=1}^{\infty} b_k 2^{-k}, \ b_k \in \{0, 1\},$$

тогда $p(x) \in M$ задается как та единственная точка, в которую M переходит при отображении

$$f_{b_1} \circ f_{b_2} \circ \cdots \circ f_{b_k} \circ \cdots$$

Пусть отображения f_0, f_1 имеют неподвижные точки m_0 и m_1. Тогда кривая $p(x)$ является непрерывной, если

$$f_0(m_1) = f_1(m_0).$$

Кривые де Рама по построению являются самоподобными, поскольку

$$\begin{aligned} p(x) &= f_0(p(2x)), && x \in [0, 0.5]; \\ p(x) &= f_1(p(2x - 1)), && x \in [0.5, 1]. \end{aligned}$$

2.1. Функция Лебега

Кривая де Рама $L(x)$, полученная при $M = [0,1]$, $f_0(x) = p_0 x$, $f_1(x) = p_0 + p_1 x$, называется функцией Лебега, где $p_0, p_1 > 0$, $p_0 + p_1 = 1$.

Функция Лебега является самой простой самоподобной функцией и удовлетворяет рекуррентным уравнениям

$$L(x) = \begin{cases} p_0 L(2x), & 0 \le x \le 0.5, \\ p_0 + p_1 L(2x-1), & 0.5 \le x \le 1. \end{cases} \tag{9}$$

Название дано Ломницким и Уламом [2] в 1934 г. В их работе также показано, что при заданном бинарном разложении числа

$$x = \sum_{k=0}^{\infty} 2^{-\gamma_k}, \quad \gamma_0 < \gamma_1 < \dots$$

значение $L(x)$ вычисляется по формуле

$$L(x) = \sum_{k=0}^{\infty} p_0^{\gamma_k - k} p_1^k. \tag{10}$$

Теорема 2. *Функция $L(x)$ имеет следующее разложение по системе Фабера–Шаудера*

$$L(x) = x + (p_0 - 0.5) \sum_{n=0}^{\infty} \sum_{j=0}^{2^n-1} p_0^{n-s(j)} p_1^{s(j)} \tau(2^n x - j), \tag{11}$$

где $s(j)$ – число единиц в бинарном разложении j.

Для сравнения приведем разложение $L(x)$ в базисе Хаара, полученное в [2]

$$L(x) = p_0 - p_0 p_1 \sum_{n=0}^{\infty} \sum_{j=0}^{2^n-1} 2^n p_0^{n-s(j)} p_1^{s(j)} \psi(2^n x - j),$$

где $\psi(t) = 0.5\tau'(t)$.

Доказательство. Доказательство этой теоремы можно провести аналогично доказательству теоремы 1. Однако здесь будет приведено доказательство, основанное на представлении (10) и формулах для коэффициентов (1).

По формуле (1) коэффициенты $l_{n,j}$ разложения функции $L(x)$ в системе Фабера–Шаудера задаются как

$$l_{n,j} = L((2j+1)2^{-n-1}) - 0.5L(j2^{-n}) - 0.5L((j+1)2^{-n}). \tag{12}$$

Пусть j – четное, тогда

$$j2^{-n} = \sum_{i=0}^{s(j)-1} 2^{-\gamma_i}, \quad \gamma_0 < \gamma_1 < \dots < \gamma_{s(j)-1} < n,$$

$$(j+1)2^{-n} = \sum_{i=0}^{s(j)-1} 2^{-\gamma_i} + 2^{-n},$$

$$(2j+1)2^{-n-1} = \sum_{i=0}^{s(j)-1} 2^{-\gamma_i} + 2^{-n-1}.$$

Подставляя (10) в (12), получим

$$l_{n,j} = (p_0 - 0.5)p_0^{n-s(j)}p_1^{s(j)}.$$

Пусть j – нечетное, тогда для некоторого m

$$j2^{-n} = \sum_{i=0}^{m-1} 2^{-\gamma_i} + \sum_{i=m}^{s(j)-1} 2^{-n+s(j)-i-1}, \quad \gamma_0 < \gamma_1 < \cdots < \gamma_{m-1} < m+n-s(j),$$

$$(j+1)2^{-n} = \sum_{i=0}^{m-1} 2^{-\gamma_i} + 2^{-n+s(j)-m},$$

$$(2j+1)2^{-n-1} = \sum_{i=0}^{m-1} 2^{-\gamma_i} + \sum_{i=m}^{s(j)} 2^{-n+s(j)-i-1}.$$

Подставляя (10) в (12), получим

$$\begin{aligned} l_{n,j} &= \sum_{i=m}^{s(j)} p_0^{n-s(j)+1}p_1^i - 0.5p_0^{n-s(j)}p_1^m - 0.5\sum_{i=m}^{s(j)-1} p_0^{n-s(j)+1}p_1^i = \\ &= p_0^{n-s(j)}p_1^m\left(1 - p_1^{s(j)-m+1}\right) - 0.5p_0^{n-s(j)}p_1^m - 0.5p_0^{n-s(j)}p_1^m\left(1 - p_1^{s(j)-m}\right) = \\ &= p_0^{n-s(j)}p_1^m\left(-p_1^{s(j)-m+1} + 0.5p_1^{s(j)-m}\right) = \\ &= (p_0 - 0.5)p_0^{n-s(j)}p_1^{s(j)}. \end{aligned}$$

□

2.2. Кривые Чезаро–Леви

Кривая де Рама $C(x)$, полученная при $M = \mathbb{C}$ – комплексная плоскость, $f_0(z) = az$, $f_1(x) = a + bz$, $b = 1 - a$, называется кривой Чезаро–Леви. На любом интервале у кривой Чезаро–Леви есть точки самопересечения. Эта кривая открыта Е.Чезаро в 1906 г. и исследована П. Леви [4].

Из теоремы 2 получаем

Следствие 1. *Функция $C(x)$ имеет следующее разложение по системе Фабера–Шаудера*

$$C(x) = x + (a - 0.5)\sum_{n=0}^{\infty}\sum_{j=0}^{2^n-1} a^{n-s(j)}b^{s(j)}\tau(2^n x - j), \tag{13}$$

где $s(j)$ – число единиц в бинарном разложении j.

2.3. Кривые Коха–Пеано

Кривая де Рама $p(x)$, полученная при $M = \mathbb{C}$ – комплексная плоскость, $f_0(z) = a\overline{z}$, $f_1(x) = a + b\overline{z}$, $b = 1 - a$, называется кривой Коха–Пеано, поскольку при $a = \frac{1}{2} + i\frac{\sqrt{3}}{6}$ получаем классическую снежинку Коха, а при $a = \frac{1+i}{2}$ — кривую Пеано, заполняющую треугольник с вершинами $0, 1, a$.

Из теоремы 2 получаем

Следствие 2. *Функция $p(x)$ имеет следующее разложение по системе Фабера–Шаудера*

$$p(x) = x + \sum_{n=0}^{\infty} \sum_{j=0}^{2^n-1} p_{n,j}\tau(2^n x - j), \tag{14}$$

$$\begin{aligned} p_{2k,j} &= a^{k-s(j)+s_0(j)}\overline{a}^{k-s_0(j)}b^{s(j)-s_0(j)}\overline{b}^{s_0(j)}\,(a-0.5)\,, \\ p_{2k+1,j} &= a^{k-s_0(j)+1}\overline{a}^{k-s(j)+s_0(j)}b^{s_0(j)}\overline{b}^{s(j)-s_0(j)}\,(a-0.5)\,, \end{aligned} \tag{15}$$

где $s(j)$ – число единиц в бинарном разложении j, $s_0(j)$ – число единиц на четных местах в бинарном разложении j.

Доказательство. Поскольку $p(0) = 0$, $p(1) = 1$ имеем разложение (14).

Подставляя разложение (14) в первое уравнение

$$p\left(\frac{t}{2}\right) = a\overline{p(t)},$$

получим $p_{0,0} = a - 0.5$ и

$$p_{n+1,j} = a\overline{p_{n,j}}, \quad 0 \le j < 2^n. \tag{16}$$

Подставляя разложение (14) во второе уравнение

$$p\left(\frac{1}{2} + \frac{t}{2}\right) = b\overline{p(t)} + a,$$

получим

$$p_{n+1,j+2^n} = b\overline{p_{n,j}}, \quad 0 \le j < 2^n. \tag{17}$$

Нетрудно проверить, что решение уравнений (16), (17) задается формулами (15). □

Список литературы / References

[1] Кашин Б.С., Саакян А.А., *Ортогональные ряды*, 2-е изд., доп., Изд-во АФЦ, М., 1999; English transl.: Kashin B.S., Saakyan A. A., *Orthogonal series*, 2nd ed., Izd. Nauchno-Issled. Aktuarno-Finans. Tsentra (AFTs), Moscow, 1999.

[2] Lomnicki Z., Ulam S. E., "Sur la theorie de la mesure dans les espaces combinatoires et son application au calcul des probabilites. I. Variables independantes", *Fundamenta Mathematicae*, **23**:1 (1934), 237–278.

[3] De Rham G., "On Some Curves Defined by Functional Equations", *Classics on Fractals*, ed. Gerald A. Edgar, Addison-Wesley, 1993, 285–298.

[4] Levy P., "Plane or Space Curves and Surfaces Consisting of Parts Similar to the Whole", *Classics on Fractals*, ed. Gerald A. Edgar, Addison-Wesley, 1993, 180–239.

Timofeev E. A., "The Expansion of Self-similar Functions in the Faber–Schauder System", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 508–515.

DOI: 10.18255/1818-1015-2017-4-508-515

Abstract. Let $\Omega = \mathcal{A}^{\mathbb{N}}$ be a space of right-sided infinite sequences drawn from a finite alphabet $\mathcal{A} = \{0, 1\}$, $\mathbb{N} = \{1, 2, \dots\}$. Let

$$\rho(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^{\infty} |x_k - y_k| 2^{-k}$$

be a metric on $\Omega = \mathcal{A}^{\mathbb{N}}$, and μ the Bernoulli measure on Ω with probabilities $p_0, p_1 > 0$, $p_0 + p_1 = 1$. Denote by $B(\boldsymbol{x}, \omega)$ an open ball of radius r centered at $\boldsymbol{\omega}$. The main result of this paper is

$$\mu\left(B(\boldsymbol{\omega}, r)\right) = r + \sum_{n=0}^{\infty} \sum_{j=0}^{2^n-1} \mu_{n,j}(\boldsymbol{\omega}) \tau(2^n r - j),$$

where $\tau(x) = 2\min\{x, 1-x\}$, $0 \le x \le 1$, ($\tau(x) = 0$, if $x < 0$ or $x > 1$),

$$\mu_{n,j}(\boldsymbol{\omega}) = \left(1 - p_{\omega_{n+1}}\right) \prod_{k=1}^{n} p_{\omega_k \oplus j_k}, \quad j = j_1 2^{n-1} + j_2 2^{n-2} + \dots + j_n.$$

The family of functions $1, x, \tau(2^n r - j)$, $j = 0, 1, \dots, 2^n - 1$, $n = 0, 1, \dots$, is the Faber–Schauder system for the space $C([0, 1])$ of continuous functions on $[0, 1]$. We also obtain the Faber–Schauder expansion for the Lebesgue's singular function, Cezaro curves, and Koch–Peano curves.

Keywords: Faber–Schauder system, Haar wavelet, self-similar, Lebesgue's function

On the authors:
Evgeniy A. Timofeev, orcid.org/0000-0002-0980-2507, ScD, professor
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003, Russia, e-mail: timofeevEA@gmail.com