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 25 № 6 (78) 2018

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. 25, №6. 2018

Семантика, спецификация и верификация программ

Анализ сигналов

Вычислительная геометрия

Модели процессов

Тезаурусы

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 17.12.2018. Дата выхода в свет 29.12.2018. Формат 60x84$^1/_8$. Усл. печ. л. 17,4. Уч.-изд. л. 15,4. Объем 149 с. Тираж 46 экз. Свободная цена. Заказ 118/018. 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 25 No 6 (78) 2018

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. 25, No 6. 2018

Program Semantics, Specification and Verification

Signal Analysis

Computational Geometry

Process Modeling

Thesauri

Моделирование и анализ информационных систем. Т. 25, № 6 (2018), с. 589–606
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 589–606

Семантика, спецификация и верификация программ
Program Semantics, Specification and Verification

DOI: 10.18255/1818-1015-2018-6-589-606

УДК 517.9

Даже простые процессы π-исчисления трудны для анализа

Аббас М. М.[1], **Захаров В.А.**[2]

Поступила в редакцию 15 сентября 2018
После доработки 30 октября 2018
Принята к публикации 10 ноября 2018

Аннотация. Математические модели распределенных вычислений, построенные на основе исчисления мобильных процессов (π-исчисления), широко используются для проверки свойств информационной безопасности криптографических протоколов. Поскольку π-исчисление является полной по Тьюрингу моделью вычислений, эта задача в общем случае алгоритмически неразрешима. Поэтому ее исследование проводится лишь для некоторых специальных классов процессов π-исчисления с ограниченными вычислительными возможностями, например, для нерекурсивных процессов, в которых все вычисления имеют ограниченную длину, для процессов с ограниченным числом параллельных компонентов и др. Однако и в этих случаях предложенные разрешающие алгоритмы оказываются весьма трудоемкими. Мы полагаем, что это обусловлено самой природой процессов π-исчисления. Цель данной работы — показать, что даже для наиболее слабой модели пассивного противника и для сравнительно простых протоколов, в которых используются лишь базовые операции π-исчисления, задача проверки свойств информационной безопасности этих протоколов является co-NP-полной.

Ключевые слова: π-исчисление, протокол, безопасность, пассивный противник, верификация, сложность, NP-полнота

Для цитирования: Аббас М. М., Захаров В.А., "Даже простые процессы π-исчисления трудны для анализа", *Моделирование и анализ информационных систем*, **25**:6 (2018), 589–606.

Об авторах:
Аббас Марат Мазен, orcid.org/0000-0001-8019-7090, студент,
Московский государственный университет им. М.В. Ломоносова,
Ленинские горы, 1, г. Москва, 119991 Россия, e-mail: unlock96@gmail.com

Захаров Владимир Анатольевич, orcid.org/0000-0002-3794-9565, доктор физ.-мат. наук, профессор,
Национальный исследовательский университет «Высшая школа экономики»,
ул. Мясницкая, 20, г. Москва, 101000 Россия,
Институт системного программирования им. В.П. Иванникова РАН, ул. А. Солженицына, 25, 109004, г. Москва,
e-mail: zakh@cs.msu.ru

Благодарности:
[1] Работа выполнена при финансовой поддержке гранта РФФИ N 18-01-00854
[2] Работа выполнена при финансовой поддержке гранта РФФИ N 16-01-00714

1. Введение

В стремлении расширить выразительные возможности исчисления взаимодействующих систем (CCS, Calculus of Communicating Systems) Робин Милнер и его коллеги в статье [25] ввели в рассмотрение новую математическую модель распределенных вычислительных систем — исчисление мобильных процессов, или коротко, π-исчисление. Оно имеет две отличительные особенности: 1) механизм порождения новых имен (ν-оператор) и 2) возможность передачи по каналам связи имен каналов связи. Благодаря этим качествам процессы π-исчисления способны изменять коммуникационную среду, вводя по ходу вычисления новые каналы связи, и моделировать, таким образом, миграцию процессов. В статье [26] Р. Милнеру удалось показать, что процессы π-исчисления способны воспроизводить вычисления термов λ-исчисления. Поэтому π-исчисление, в отличие от CCS, является полной по Тьюрингу моделью вычислений. Своей универсальностью π-исчисление обязано сочетанию двух указанных особенностей с оператором репликации, унаследованным от CCS; при отсутствии любого из этих трех факторов π-исчисление перестает быть универсальной вычислительной моделью.

Спустя сравнительно короткое время после публикации работы [25] было обнаружено, что помимо описания поведения систем мобильных процессов π-исчисление можно успешно использовать для формального описания механизмов работы с объектами в объектно-ориентированном программировании [33], моделирования бизнес-процессов [30] и биохимических реакций [28]. Но, вероятно, наибольший интерес вызвала обнаруженная авторами статьи [1] возможность построения на основе π-исчисления формальных моделей криптографических протоколов.

В основополагающей статье [19] Д. Долев и Э. Яо предложили разделить задачу проверки свойств безопасности криптографических протоколов на две подзадачи: 1) доказательство свойств безопасности (конфиденциальности, целостности и др.) для базовых функций (шифрование, хэширование и пр.), используемых в криптографических протоколах, и 2) проверка стойкости криптографических протоколов в предположении о том, что все используемые в них криптографические примитивы удовлетворяют необходимым требованиям стойкости. При этом модель Долева–Яо наделяет злоумышленника широкими возможностями, включая способность перехватывать, формировать и отправлять сообщения по открытым каналам связи. Предложенное в статье [1] исчисление криптографических протоколов — spi-исчисление — оказалось удачной моделью, в рамках которой проверка стойкости криптографических протоколов в модели Долева–Яо может быть сведена (или, если говорить более формально, определена посредством сведения) к задачам проверки различных видов эквивалентности процессов spi-исчисления или к задаче проверки достижимости особо выделенных «состояний уязвимости» процессов. В работе [2] было предложено еще одно расширение π-исчисления — так называемое прикладное π-исчисление, — в котором разрешается строить сложные термы и описывать их алгебраические свойства при помощи уравнений. Прикладное π-исчисление может быть также обогащено вспомогательной памятью с общедоступными и конфиденциальными ячейками [5, 9], при помощи которой удобно описывать протоколы аутентификации, распределения ключей и др.

Задачам анализа поведения процессов в исчислениях криптографических протоколов, построенных на основе π-исчисления, посвящено большое число работ. Поскольку π-исчисление является алгоритмически полной моделью вычислений, все указанные задачи в общем случае алгоритмически неразрешимы. Поэтому их исследование проводится лишь для некоторых специальных классов процессов с ограниченными вычислительными возможностями, например, для нерекурсивных процессов, в которых все вычисления имеют ограниченную длину, для процессов с ограниченным числом параллельных компонентов и др. Наиболее значительные достижения исследований задач анализа поведения процессов spi-исчисления таковы. Было показано, что задача проверки достижимости разрешима для нерекурсивных процессов spi-исчисления [3, 23] и является NP-полной [29]. Среди различных видов эквивалентности процессов наиболее полезными для криптографических приложений оказываются тестовая эквивалентность [1] и наблюдаемая эквивалентность [2]. Разрешимость задачи проверки тестовой эквивалентности нерекурсивных процессов была установлена в статье [20], однако предложенная разрешающая процедура имеет большую вычислительную трудоемкость. Алгоритмы проверки наблюдаемой эквивалентности для некоторых классов процессов spi-исчисления и прикладного π-исчисление были представлены и исследованы в статьях [10, 15, 16, 18]. В статье [16] было также показано, что для нерекурсивных процессов прикладного π-исчисления задача проверки наблюдаемой эквивалентности NP-полна. Наряду с наблюдаемой эквивалентностью процессов исследовались также и более простые отношения бисимуляции [8]. В статье [32] показано, что задача проверки отношения открытой бисимуляции нерекурсивных процессов spi-исчисления разрешима. Для процессов spi-исчисления исследовалась также возможность их верификации при помощи методов статического анализа [7]. Преимущество этого подхода состоит в том, что он применим к любым (в том числе рекурсивным) процессам. Предложенные формальные методы анализа криптографических протоколов на основе π-исчисление применялись на практике: с их помощью удалось, например, обнаружить и устранить уязвимость в сетевом протоколе безопасной маршрутизации ARAN [22].

Даже в тех случаях, когда нерекурсивный процесс spi-исчисления P имеет всего лишь конечное множество вычислений, задача анализа его поведения во взаимодействии с внешней средой оказывается непростой. Причина состоит в том, что в модели Долева–Яо внешняя среда (злоумышленник) представляется в виде бесконечного семейства процессов $\mathcal{A}$, и задача проверки свойств информационной безопасности состоит в анализе поведения всех процессов вида $P|R$, где $R \in \mathcal{A}$. Процессы семейства $\mathcal{A}$ способны, вообще говоря, формировать сколь угодно сложные сообщения, и поэтому система процессов $\{P|R \ : \ R \in \mathcal{A}\}$ может иметь бесконечно много состояний и вычислений. Методы, предложенные и развитые в работах [10, 15, 16, 18, 20, 32], позволяют выделить (иногда неявно) такое конечное подмножество процессов $\mathcal{A}'$, $\mathcal{A}' \subseteq \mathcal{A}$, описывающих внешнюю среду, что для проверки стойкости протокола P в модели Долева–Яо достаточно проанализировать поведение лишь конечного числа процессов вида $P|R$, где $R \in \mathcal{A}'$. Размер множества процессов $\mathcal{A}'$ оказывается, по меньшей мере, экспоненциально зависящим от размера процесса P, и поэтому алгоритмы проверки свойств безопасности процессов spi-исчисления или прикладного π-исчисления, предложенные в указанных работах, оказываются трудоемкими.

Впервые об NP-полноте задачи проверки свойств небезопасности криптографических протоколов с ограниченным числом сеансов (нерекурсивных протоколов) было объявлено в работах [3,4]. Однако в них рассматривались лишь модели криптографических протоколов с простыми (атомарными) ключами шифрования. Кроме того, авторы статьи [4] привели лишь общую схему обоснования сложности описанной ими разрешающей процедуры. Полное доказательство теоремы об NP-полноте задачи проверки свойств небезопасности нерекурсивных криптографических протоколов с атомарными ключами шифрования было опубликовано в статьях [21,31]. Этот результат был усилен в работе [29]; в ней было показано, что задача проверки свойства небезопасности остается NP-полной и для нерекурсивных криптографических протоколов, использующих составные ключи шифрования. Доказательство принадлежности этой задачи классу сложностей NP опирается на утверждение о том, что всякий нестойкий в модели Долева–Яо протокол может быть скомпрометирован активным противником, способным порождать сообщения, размер которых полиномиально зависит от размера протокола. Этот подход к построению и обоснованию трудоемкости процедур проверки свойств небезопасности был использован в последующих статьях [11–14] для протоколов с более сложными криптографическими примитивами. Однако, как было показано в статье [24], доказательство утверждения о существовании минимальной модели противника полиномиального размера, приведенное в работе [29], содержит ошибку. Она была исправлена в той же статье [24].

Во всех указанных выше работах авторы исследовали задачу проверки свойств небезопасности для криптографических протоколов со все более расширяющимся разнообразием вычислительных и коммуникационных действий, и основное внимание уделялось доказательству принадлежности этой задачи классу сложностей NP. Доказательство NP-трудности рассматриваемой задачи проводилось путем сведения к ней проблемы выполнимости 3-КНФ. Естественно, что это сведение осуществлялось все проще по мере усложнения модели криптографических протоколов. Для этих целей использовались различные средства, допускаемые в spi-исчислении, включая функции шифрования/расшифрования, функции пары, операторы ветвления и др. Мы полагаем, однако, что трудности анализа стойкости криптографических протоколов, моделируемых при помощи процессов различных расширений π-исчисления, обусловлены, в первую очередь, алгоритмическими трудностями, присущими самому π-исчислению как базовой модели систем распределенных мобильных вычислений. Мы рассматриваем задачу проверки стойкости протоколов, описанных при помощи базовых средств π-исчисления, в модели пассивного противника. Анализируемые протоколы представляют собой параллельную композицию процессов $P_1|P_2|\cdots|P_n$, каждый из которых является последовательной композицией действий отправления и приема сообщений $P_i = act_1.act_2.\cdots.act_m$. Среди имен, используемых в качестве сообщений, только одно является конфиденциальным, а все остальные считаются либо общеизвестными, либо случайными именами, использующимися лишь однократно. Пассивный противник имеет возможность перехватывать (подслушивать) сообщения, передаваемые по тем каналам связи, имена которых ему известны. Подслушанные имена расширяют знания противника и могут быть использованы при последующих перехватах. Протокол считается стойким,

если при любом его выполнении знания пассивного противника не будут включать в себя конфиденциальные имена.

В данной статье мы показываем, что для любой 3-КНФ φ можно построить, используя лишь базовые средства π-исчисления, такой процесс $Proc_\varphi = P_1|P_2|\cdots|P_n$ указанного выше вида, который является стойким относительно пассивного противника в том и только том случае, когда 3-КНФ φ невыполнима. При этом размер процесса $Proc_\varphi$ линейно зависит от размера формулы φ, и все вычисления этого процесса завершаются нормально, не попадая в тупики. В свете результатов об NP-полноте задачи проверки свойств небезопасности криптографических протоколов, полученных в статьях [3,4,11–14,21,24,29,31], основная теорема данной статьи показывает, что главным фактором, определяющим сложность рассматриваемой задачи, является ограниченность длины вычислений криптографических протоколов; влияние других факторов, таких как разнообразие используемых в протоколах криптографических примитивов и средств управления вычислением протоколов, структура и размер передаваемых сообщений и др., оказывается второстепенным.

Статья устроена следующим образом. Во втором разделе приводится описание синтаксиса и семантики π-исчисления. В разделе 3 определяется модель пассивного противника и формулируется задача проверки стойкости процессов π-исчисления в модели пассивного противника. В разделе 4 описано устройство процесса $Proc_\varphi$, соответствующего произвольной 3-КНФ φ, и показано, что этот процесс обладает свойством нормальной завершаемости — все его вычисления завершаются в одном и том же пустом процессе. Также установлено, что все вычисления процесса $Proc_\varphi$ безопасны в том и только том случае, если 3-КНФ φ невыполнима. Отсюда следует, что проблема выполнимости 3-КНФ сводима к задаче проверки свойства небезопасности процессов π-исчисления в модели пассивного противника. В заключении мы обсуждаем значение полученных результатов.

2. Синтаксис и семантика π-исчисления

Мы ограничиваемся рассмотрением базового нерекурсивного фрагмента синхронного монадического исчисления мобильных процессов. Пусть задано некоторое бесконечное множество объектов $\mathcal{N}$, которые будут называться *именами*. Имена служат для обозначения каналов связи, а также данных, передаваемых по этим каналам. Для записи имен будут использоваться строчные латинские буквы $a, b, \ldots, x, y, z$.

Элементарным действием синхронной коммуникации E называется всякое выражение вида $\overline{x}\langle y\rangle$ (отправление сообщения y по каналу связи x) или $x(y)$ (прием сообщения, связывающего имя y, по каналу связи x). *Процессом* π-исчисления называется всякое выражение, которое может быть построено по следующим правилам:

$$
\begin{array}{rcll}
P, Q & ::= & \mathbf{0} & \text{(завершить выполнение процесса)} \\
 & | & E.P & \text{(выполнить действие } E \text{ и перейти к выполнению процесса } P) \\
 & | & P|Q & \text{(выполнять параллельно процессы } P \text{ и } Q) \\
 & | & (\nu x)\ P & \text{(ввести новое имя } x \text{ и перейти к выполнению процесса } P).
\end{array}
$$

Множество всех определенных таким образом процессов обозначим символом $\mathcal{P}$.

Вхождения имен в процесс подразделяются на *свободные* и *связанные*. Множество $fn(P)$ свободных имен процесса P определяется в зависимости от устройства процесса по следующим правилам:

1. $fn(\mathbf{0}) = \emptyset$,
2. $fn(\overline{x}\langle y\rangle.P) = fn(P) \cup \{x, y\}$, $fn(x(y).P) = fn(P) \cup \{x\}) \setminus \{y\}$,
3. $fn((\nu x)\ P) = fn(P) \setminus \{x\}$,
4. $fn(P|Q) = fn(P) \cup fn(Q)$.

Вхождение имени x в процесс P считается свободным, если это вхождение не содержится ни в одном подпроцессе вида $(\nu x)\ P'$ или $y(x).P'$ процесса P. Запись вида $P\{x/y\}$ обозначает процесс, полученный из процесса P одновременной заменой всех свободных вхождений имени y именем x. Подстановка $\{x/y\}$ называется *правильной для процесса* P, если каждое свободное вхождение имени y не содержится ни в одном подпроцессе вида $(\nu x)\ P'$ процесса P.

Операционная семантика мобильных процессов определяется отношением структурной конгруэнтности $\equiv_\pi$ и отношением редукции $\to_\pi$. Отношение $\equiv_\pi$ — это наименьшее отношение конгруэнтности на множестве процессов $\mathcal{P}$, удовлетворяющее следующим тождествам:

1. $\mathbf{0}|P \equiv_\pi P$, $P|Q \equiv_\pi Q|P$, $P|(Q|R) \equiv_\pi (P|Q)|R$, т.е. система $(\mathcal{P}, |, \mathbf{0})$ является коммутативной полугруппой;
2. $(\nu y)\ P \equiv_\pi (\nu x)\ P\{x/y\}$ для всякого имени $x, x \notin fn(P)$, и правильной для процесса P подстановки $\{x/y\}$;
3. $((\nu x)\ P)|Q \equiv_\pi (\nu x)\ (P|Q)$ для всякого имени $x, x \notin fn(Q)$;
4. $(\nu x).\mathbf{0} \equiv_\pi \mathbf{0}$, $\quad (\nu x)\ ((\nu y)\ P) \equiv_\pi (\nu y)\ ((\nu x)\ P)$.

Нетрудно заметить, что если $P \equiv_\pi Q$, то $fn(P) = fn(Q)$.

Отношение редукции — это наименьшее четырехместное отношение $\to_\pi \subseteq \mathcal{P} \times \mathcal{N} \times \mathcal{N} \times \mathcal{P}$, удовлетворяющее для любых процессов P, Q, P', Q' и имен x, y, z следующим требованиям (выполнимость отношения $\to_\pi$ для четверки P, x, y, Q традиционно обозначается записью $P \xrightarrow{x(y)}_\pi Q$):

1. $(\overline{x}\langle y\rangle.P)|(x(z).Q) \xrightarrow{x(y)}_\pi P|Q\{y/z\}$;
2. если $P \xrightarrow{x(y)}_\pi P'$, то $P|Q \xrightarrow{x(y)}_\pi P'|Q$;
3. если $P \xrightarrow{x(y)}_\pi P'$, то $(\nu z).\ P \xrightarrow{x(y)}_\pi (\nu z).\ P'$;
4. если $P \equiv_\pi Q$, $P \xrightarrow{x(y)}_\pi P'$ и $P' \equiv_\pi Q'$, то $Q \xrightarrow{x(y)}_\pi Q'$.

Четверки $P \xrightarrow{x(y)}_{\pi} Q$, удовлетворяющие отношению редукции процессов, будем называть *редукциями* (или, иначе, коммуникациями по каналу связи x). Канал связи x, по которому может быть выполнена редукция процесса P, называется *активным в процессе* P.

Вычислением процесса P_0 назовем всякую последовательность редукций вида

$$P_0 \xrightarrow{x_1(y_1)}_{\pi} P_1 \xrightarrow{x_2(y_2)}_{\pi} \cdots \xrightarrow{x_{n-1}(y_{n-1})}_{\pi} P_{n-1} \xrightarrow{x_n(y_n)}_{\pi} P_n. \tag{1}$$

Процесс, не допускающий ни одной редукции, называется *тупиковым*. Вычисление (1) считается *завершенным*, если процесс P_n является тупиковым. Если $P_n \equiv_{\pi} \mathbf{0}$, то завершение вычисления является *нормальным*. Очевидно следующее утверждение, которым мы будем пользоваться при анализе вычислений процессов.

Утверждение 1. *Если в процессе P_0 активен канал связи x, то в любом завершенном вычислении (1) происходит коммуникация по каналу x.*

Иными словами, по каждому активному в процессе P_0 каналу связи рано или поздно обязательно будет передано сообщение.

3. Модель пассивного противника

По отношению к заданному процессу π-исчисления P сторонний наблюдатель (противник) мыслится как некоторый процесс R, который может вступать во взаимосвязь с процессом P. Это взаимодействие определяется параллельной композицией $P|R$. Все имена процесса P, связанные оператором ν, могут быть истолкованы как однократно используемые данные, порождаемые процессом. Поэтому вначале коммуникация процессов P и R может осуществляться только по каналам связи, принадлежащим множеству $fn(P)$. Но затем противник, получая (перехватывая) сообщения, переданные по этим каналам, узнает новые имена, которыми он сможет пользоваться для последующей коммуникации с P. Пассивный противник R способен лишь подслушивать сообщения, отправленные процессом P по тем каналам связи, которые известны противнику в момент выполнения действий отправления и приема сообщения. Такая способность пассивного противника моделируется парой последовательно выполняемых действий $x(y).\overline{x}\langle y\rangle$. Подслушанные таким образом сообщения не теряются и не подменяются. В роли пассивного противника может выступать любой процесс R, представляющий собой последовательную композицию пар действий приема и отправления одного и того же сообщения по одному и тому же каналу связи. В отличие от пассивного противника, активный противник способен не только подслушивать сообщения, но также конструировать новые сообщения и осуществлять подмену перехваченных сообщений; в роли активного противника может выступать любой процесс π-исчисления R.

Свойства информационной безопасности процесса P проявляются в его взаимодействии с произвольным противником того или иного вида. Чтобы избежать необходимости рассматривать поведение всех возможных параллельных композиций вида $P|R$, обычно используется модель абстрактного противника, которая определяется состоянием его знаний K — множеством известных противнику имен. Это

множество имен может изменяться по ходу выполнения действий процессом P. В случае взаимодействия процесса P с пассивным противником, преобразование состояния (P, K) взаимодействующей системы в состояние (P', K') возможно только за счет выполнения некоторого перехода процесса P. Если противник является активным, то преобразование состояния (P, K) в состояние (P', K') может произойти как за счет выполнения некоторой редукции процесса P, так и в результате выполнения некоторого действия $x(y)$ или $\overline{x}\langle y\rangle$ процесса P при условии, что состояние знаний противника K позволяет ему сформировать парное действие вида $\overline{x}\langle z\rangle$ или $x(z)$ соответственно. Целью противника является достижение состояния знаний из некоторого выделенного семейства S, которое составляет угрозу безопасности.

В данной статье мы ограничиваемся рассмотрением взаимодействия процессов π-исчисления с пассивным противником. Пассивный противник характеризуется списком записей об известных ему именах. Для формирования записей введем новый оператор κ и будем использовать выражение (κx), где $x \in \mathcal{N}$, для обозначения записи имени x в базе данных противника. *Модель пассивного противника* представляет собой всякий список записей (базу данных) A, который может быть построен по следующим правилам:

$$\begin{array}{rcll} A & ::= & \mathbf{0} & \text{(пустая база данных)} \\ & | & (\kappa x).A & \text{(база данных, содержащая запись имени } x\text{)}. \end{array}$$

В отличие от оператора ν оператор записи κ не осуществляет связывание имен, и поэтому множество $fn(A)$ свободных имен противника A состоит из всех имен, входящих в состав выражения A.

Пассивный противник может наблюдать за вычислениями процесса π-исчисления; для формального описания этого явления будем использовать оператор параллельной композиции. *Мониторингом* процесса P противником A будем называть всякое выражение M, которое может быть построено по следующим правилам:

$$\begin{array}{rcll} M & ::= & P|A & \text{(противник } A \text{ наблюдает за процессом } P\text{)} \\ & | & (\nu x).M & \text{(мониторинг } M \text{ проводится в области определения} \\ & & & \text{однократно используемого имени } x\text{)}. \end{array}$$

Множество всех возможных мониторингов обозначим символом $\mathcal{M}$. Множество свободных имен мониторинга M обозначим $fn(M)$. В множестве $fn(M)$ свободных имен мониторинга M нас будут особо интересовать свободные имена, входящие в состав записей противника вида (κx). Подмножество всех свободных имен указанного вида обозначим выражением $open(M)$.

Операционная семантика мониторингов, так же как семантика процессов π-исчисления, определяется отношением структурной конгруэнтности $\equiv$ и отношением переходов $\rightarrow$. Отношение $\equiv$ — это наименьшее отношение конгруэнтности на множестве мониторингов $\mathcal{M}$, которое включает отношение структурной конгруэнтности $\equiv_\pi$ на множестве процессов и удовлетворяет следующим тождествам:

1. $(\nu y).\ M \equiv (\nu x).\ M\{x/y\}$ для всякого имени $x, x \notin fn(M)$, и правильной для мониторинга M подстановки $\{x/y\}$;

2. $(\kappa x).(\kappa y).A \equiv (\kappa y).(\kappa x).A$ для всякой пары имен x, y и противника A;

3. $(\kappa x).(\kappa x).A \equiv (\kappa x).A$ для всякого имени x и противника A

Первое из этих тождеств означает, что все сведения противника об однократно используемых именах действительны только в контексте наблюдаемого процесса, а два других тождества позволяют рассматривать базу знаний противника как неупорядоченное множество записей. Учитывая последнее обстоятельство, условимся для произвольного множества имен $X = \{x_1, x_2, \ldots, x_n\}$ обозначать выражением $A(X)$ противника $(\kappa x_1).(\kappa x_2).\ldots.(\kappa x_n).\mathbf{0}$.

Отношение переходов $\rightarrow$ — это наименьшее бинарное отношение на множестве мониторингов $\mathcal{M}$, удовлетворяющее следующим требованиям для любых процессов P, Q, P', Q', модели противника A, мониторингов M, N, M', N' и имен x, y:

1. если $P \xrightarrow{x(y)}_{\pi} P'$, то $P|A \rightarrow P'|A$;
2. если $P \xrightarrow{x(y)}_{\pi} P'$, то $P|(\kappa x).A \rightarrow P'|(\kappa y).(\kappa x).A$;
3. если $M \rightarrow M'$, то $(\nu x).\ M \rightarrow (\nu x).\ M'$;
4. если $M \equiv N$, $M \rightarrow M'$ и $M' \equiv N'$, то $N \rightarrow N'$.

Второе из приведенных здесь правил гласит: если наблюдаемый процесс выполняет коммуникацию некоторых данных по каналу связи, имя которого известно противнику, то передаваемые данные тоже становятся известны противнику. Рефлексивно-транзитивное замыкание отношения переходов обозначим $\rightarrow^*$. Мониторинг M' считается *достижимым* из мониторинга M, если выполняется отношение $M \rightarrow^* M'$.

Пример 1. Пусть имеется процесс

$$P = ((\nu x).(\nu y).\ \overline{ch}\langle x\rangle.x(y).\mathbf{0})|((\nu z).\ ch(z).\overline{z}\langle secret\rangle.\mathbf{0})$$

и модель пассивного противника $A = (\kappa\ ch).\mathbf{0}$. Тогда мониторинг $M = P|A$ порождает последовательность переходов

$$\begin{array}{c} M \\ \downarrow \\ (\nu x).(\nu z).\ \big(((\nu y).\ x(y).\mathbf{0})|(\overline{x}\langle secret\rangle.\mathbf{0})|(\kappa x).(\kappa\ ch).\mathbf{0}\big) \\ \downarrow \\ (\nu x).(\nu y).(\nu z).\ \big((\kappa\ secret).(\kappa x).(\kappa\ ch).\mathbf{0}\big)\ . \end{array}$$

Цель противника, располагающего определенными априорными сведениями о процессе, состоит в выведывании некоторых конфиденциальных данных, с которыми оперирует процесс. Поэтому и атакой, и угрозой, относительно которых формулируется требование стойкости процессов, являются конечные множества имен X и Y. Будем говорить, что процесс P является *стойким относительно угрозы* Y *при осуществлении атаки* X (коротко, (X, Y)-*стойким*), если для любого мониторинга M, достижимого из начального мониторинга $P|A(X)$, неверно, что $Y \subseteq \mathit{open}(M)$, т.е. ни в одном вычислении процесса P пассивный противник, знающий вначале лишь множество имен X, не может подслушивать коммуникации процесса так, чтобы сформировать из подслушанных имен множество Y. Задача проверки стойкости

процессов в модели пассивного противника состоит в том, чтобы для произвольного заданного процесса P, атаки X и угрозы Y выяснить, является ли (X, Y)-стойким процесс P. В приведенном выше примере процесс P не является стойким относительно угрозы $\{secret\}$ при осуществлении атаки $\{ch\}$.

4. Сложность задачи проверки стойкости процессов

Теорема 1. *Задача проверки стойкости процессов множества $\mathcal{P}$ в модели пассивного противника является co-NP-полной.*

Доказательство. Поскольку процессы множества $\mathcal{P}$ не содержат оператора репликации ! или иных средств рекурсивного описания вычислений, длина каждого вычисления процесса P из множества $\mathcal{P}$ не превосходит размера процесса P. Кроме того, как можно видеть из определения отношения переходов $\to$, у каждого мониторинга M число его попарно неконгруэнтных наследников (образов) M' по отношению $\to$ не превосходит квадрата от размера M. Отсюда следует, что задача проверки стойкости процессов множества $\mathcal{P}$ принадлежит классу сложностей co-NP.

Для обоснования co-NP-трудности рассматриваемой задачи покажем, что к ней *log-space* сводима задача проверки невыполнимости 3-КНФ. Для произвольной заданной 3-КНФ φ мы построим процесс $Proc_\varphi$, который способен моделировать вычисление значения формулы φ на всех наборах значений переменных и позволяет противнику подслушать конфиденциальное имя $secret$, переданное по открытому каналу связи ch, в том и только том случае, когда φ выполнима.

Пусть задана произвольная 3-КНФ $\varphi = D_1 \wedge D_2 \wedge \cdots \wedge D_N$, зависящая от переменных $x_1, x_2, \ldots, x_n$, в которой каждый дизъюнкт $D_i,\ 1 \le i \le N$, имеет вид $\ell_{1i} \vee \ell_{2i} \vee \ell_{3i}$, где $\ell_{1i}, \ell_{2i}, \ell_{3i}$ — литеры, являющиеся переменными или их отрицаниями. Мы будем различать имя литеры ℓ и имя той переменной, на основе которой определяется эта литера. Условимся для каждой литеры ℓ использовать запись x^σ, где $\sigma = 1$, если $\ell = x$, и $\sigma = 0$, если $\ell = \neg x$. Без ограничения общности можно считать, что каждая литера $x_i^\sigma,\ 1 \le i \le n,\ \sigma \in \{0, 1\}$, входит в состав 3-КНФ φ. Кроме того, для литеры ℓ мы будем обозначать записью ℓ^* контрарную литеру противоположной полярности, а записью m_ℓ общее число вхождений литеры ℓ в формулу φ.

Опишем устройство процесса $Proc_\varphi$. В нем есть только два свободных имени ch и $secret$. Имя ch обозначает открытый (доступный для подслушивания) канал связи; это имя составляет атаку. Имя $secret$ обозначает конфиденциальные данные, и оно составляет угрозу. Все остальные имена, фигурирующие в процессе $Proc_\varphi$, связаны операторами ν. Это множество имен состоит из имен переменных $x_1, \ldots, x_n$, имен положительных и отрицательных литер, соответствующих этим переменным, $\ell_1, \ldots, \ell_n, \ell_1^*, \ldots, \ell_n^*$, имен дизъюнктов $d_1, \ldots, d_N$, а также три особых имени g, h и r. Процесс $Proc_\varphi$ представляет собой композицию параллельных подпроцессов, которым отведены следующие роли.

1. Подпроцесс $Init$ призван активизировать для каждой переменной $x_i,\ 1 \le i \le n$, в точности одну из литер x_i или $\neg x_i$, отправив для каждой переменной x_i сообщение, которое может принять только один из двух процессов S_{ℓ_i} или $S_{\ell_i^*}$:

$$Init = \overline{x_1}\langle z\rangle.\overline{x_2}\langle z\rangle.\ldots.\overline{x_n}\langle z\rangle.\mathbf{0}\ .$$

2. Для каждой из $2n$ литер $\ell = x_i^\sigma$, $1 \le i \le n$, $\sigma \in \{0,1\}$, подпроцесс S_ℓ призван активизировать все каналы связи с именем ℓ, используемые в процессе $Proc_\varphi$:

$$S_\ell = x_i(z).\underbrace{\overline{\ell}\langle z\rangle.\overline{\ell}\langle z\rangle.\ldots.\overline{\ell}\langle z\rangle}_{m_\ell + m_{\ell^*}\ \text{раз}}.\mathbf{0}\ .$$

3. Подпроцесс $Check_{\varphi=0}$ предназначен для проверки того, что формула φ принимает значение 0 на том наборе значений переменных, который соответствует активизированным литерам:

$$Check_{\varphi=0} = Check_{D_1=0}|Check_{D_2=0}|\ldots|Check_{D_N=0}\ ,$$

где для каждого дизъюнкта $D_i = \ell_{1i} \vee \ell_{2i} \vee \ell_{3i}$ подпроцесс $Check_{D_i=0}$ имеет вид

$$Check_{D_i=0} = \ell^*_{1i}(z).\ell^*_{2i}(z).\ell^*_{3i}(z).\overline{r}\langle ch\rangle.\mathbf{0}\ .$$

Таким образом, каждый процесс $Check_{D_i=0}$ в том случае, если активизированы литеры, контрарные по отношению к литерам $\ell_{1i}, \ell_{2i}, \ell_{3i}$, отправляет по каналу связи r имя открытого канала ch.

4. Подпроцесс $Check_{\varphi=1}$ предназначен для проверки того, что формула φ принимает значение 1 на том наборе значений переменных, который соответствует активизированным литерам:

$$Check_{\varphi=1} = Check_{D_1=1}|Check_{D_2=1}|\ldots|Check_{D_N=1}|CheckAll\ ,$$

где для каждого дизъюнкта $D_i = \ell_{1i} \vee \ell_{2i} \vee \ell_{3i}$ подпроцесс $Check_{D_i=1}$ имеет вид

$$Check_{D_i=1} = (\ell_{1i}(z).\overline{d_i}\langle z\rangle.\mathbf{0})|(\ell_{2i}(z).\overline{d_i}\langle z\rangle.\mathbf{0})|(\ell_{3i}(z).\overline{d_i}\langle z\rangle.\mathbf{0})\ ,$$

а подпроцесс $CheckAll$ имеет вид

$$CheckAll = d_1(z).d_2(z).\ldots.d_N(z).\overline{r}\langle secret\rangle.\mathbf{0}\ .$$

Таким образом, процесс $Check_{\varphi=1}$ отправляет по каналу связи r секретное имя $secret$, если для каждого дизъюнкта $D_i = \ell_{1i} \vee \ell_{2i} \vee \ell_{3i}$, $1 \le i \le N$, была активизирована хотя бы одна входящая в него литера ℓ_{ji}, $1 \le j \le 3$.

5. Подпроцесс $OpenCh$ призван осуществить коммуникацию по открытому каналу связи ch после того, как было проверено значение формулы φ, и запустить подпроцесс «сборки мусора», который позволит выполнить все незавершенные действия коммуникации процесса $Proc_\varphi$:

$$OpenCh = (r(y).\overline{ch}\langle y\rangle.\overline{g}\langle z\rangle.\overline{h}\langle z\rangle.\mathbf{0})|(ch(x).\mathbf{0})\ .$$

Запуск подпроцесса «сборки мусора» осуществляют действия отправления сообщений $\overline{g}\langle z\rangle$ и $\overline{h}\langle z\rangle$.

6. Подпроцесс «сборки мусора» $Garbage$ предназначен для того, чтобы активизировать все те литеры, которые не были активизированы подпроцессом $Init$, и заставить выполниться все действия подпроцессов $Check_{\varphi=0}$ и $Check_{\varphi=1}$, оставшиеся невыполненными после первой активизации литер процессом $Init$:

$$Garbage = Final \mid Collect \ ,$$

где

$$Final = g(z).\overline{x_1}\langle z\rangle.\overline{x_2}\langle z\rangle.\ldots.\overline{x_n}\langle z\rangle.\mathbf{0} \ ,$$

$$Collect = h(z).d_1(z).d_1(z).d_2(z).d_2(z).\ldots.d_N(z).d_N(z).r(y_1).r(y_2).\ldots.r(y_N).\mathbf{0} \ .$$

Обозначим записью $names$ список всех имен, за исключением ch и $secret$, встречающихся в указанных выше подпроцессах. Процесс $Proc_\varphi$ представляет собой параллельную композицию всех этих подпроцессов, в которой все имена из списка $names$ связаны оператором ν:

$$Proc_\varphi = (\nu\ names).(Init|S_{\ell_1}|S_{\ell_1^*}|\ldots|S_{\ell_n}|S_{\ell_n^*}|Check_{\varphi=0}|Check_{\varphi=1}|OpenCh|Garbage) \ .$$

Легко видеть, что, располагая КНФ φ, процесс $Proc_\varphi$ можно построить, используя лишь логарифмическую память.

Вначале покажем, что любое завершенное вычисление процесса $Proc_\varphi$

$$Proc_\varphi = P_0 \longrightarrow_\pi P_1 \longrightarrow_\pi \cdots \longrightarrow_\pi P_{m-1} \longrightarrow_\pi P_m \tag{2}$$

оканчивается нормально, т.е. $P_m = \mathbf{0}$.

Прежде всего, заметим, что ни одно имя u, являющееся аргументом какого-либо действия приема сообщения $v(u)$, не является именем никакого канала связи. Это означает, что имена каналов связи по ходу вычисления (2) не изменяются (возможно лишь переименование, допустимое по отношению конгруэнтности $\equiv_\pi$). Кроме того, в процессе $Proc_\varphi$ для каждого имени канала связи u число действий отправления сообщений по каналу u равно числу действий приема сообщений по этому же каналу. Поэтому для доказательства нормального завершения вычисления (2) достаточно показать, что в нем выполнены все действия отправления сообщений процесса $Proc_\varphi$.

Заметим, что в процессе $Proc_\varphi$ активен канал x_1, а процесс P_m является тупиковым. Значит, согласно утверждению 1, по ходу вычисления (2) была выполнена коммуникация по каналу x_1. Действия приема сообщения по этому каналу связи есть только в подпроцессах активизации литер S_{x_1} и $S_{\neg x_1}$. Рассмотрим тот из этих двух подпроцессов, в котором коммуникация по каналу связи x_1 выполнилась раньше всего в вычислении (2); пусть этот подпроцесс предназначен для активизации литеры $\ell_1 = x_1^{\sigma_1}$. Тогда после выполнения первой коммуникации по каналу x_1 активным становится канал связи ℓ_1. Согласно описанию подпроцесса S_ℓ активность канала ℓ_1 будет поддерживаться по ходу вычисления до тех пор, пока не сработают все возможные действия приема сообщений по этому каналу.

После выполнения коммуникации по каналу x_1 (в любом из подпроцессов $Init$ или $Final$) активным становится также канал связи x_2, для которого будут справедливы рассуждения подобные тем, что были приведены выше для канала x_1. Проводя

эти рассуждения для всех каналов связи $x_1, x_2, \ldots, x_n$, приходим к заключению о том, что в вычислении (2) для некоторого двоичного набора $\alpha = (\sigma_1, \sigma_1, \ldots, \sigma_n)$ оказываются активизированными каналы связи $\ell_1 = x_1^{\sigma_1}, \ell_2 = x_2^{\sigma_2}, \ldots, \ell_n = x_n^{\sigma_n}$, и их активность будет поддерживаться по ходу вычисления до тех пор, пока не сработают все возможные действия приема сообщений по этим каналам.

Далее необходимо рассмотреть два случая, в зависимости от того, какое значение принимает формула φ на наборе α значений переменных.

Если $\varphi(\alpha) = 0$, то для некоторой тройки выделенных выше литер $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$ КНФ φ содержит дизъюнкт $D_j = \neg\ell_{i_1} \vee \neg\ell_{i_2} \vee \neg\ell_{i_3}$. Согласно описанию подпроцесса $Check_{\varphi=0}$, одним из компонентов его параллельной композиции является подпроцесс

$$Check_{D_j=0} = \ell_{i_1}(z).\ell_{i_2}(z).\ell_{i_3}(z).\overline{r}\langle ch\rangle.\mathbf{0}\ .$$

Поскольку, как было установлено выше, подпроцессы вида S_ℓ поддерживают активность каналов $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$, пока не выполнятся все возможные действия приема сообщений по этим каналам, в вычислении (2) осуществляются коммуникации по каналам $\ell_{i_1}, \ell_{i_2}, \ell_{i_3}$, в которых принимают участие действия приема сообщений подпроцесса $Check_{D_j=0}$. После выполнения последней из этих трех редукций активным становится канал связи r, поскольку действием приема сообщения по этому каналу начинается один из компонентов параллельной композиции подпроцесса $OpenCh$.

Если $\varphi(\alpha) = 1$, то каждый дизъюнкт D_j, $1 \le j \le N$, содержит одну из выделенных выше литер $\ell_1, \ell_2, \ldots, \ell_n$. Значит, в соответствии с описанием подпроцесса $Check_{\varphi=1}$ для каждого j, $1 \le j \le N$, параллельная композиция этого подпроцесса содержит компонент $\ell_i(z).\overline{d_j}\langle z\rangle.\mathbf{0}$, где ℓ_i — одна из выделенных литер, являющаяся именем активизированного канала связи. Ввиду того, что активность «литерных» каналов поддерживается, пока не сработают все возможные действия приема сообщений по этим каналам, в вычислении (2) осуществляются редукции, в которых принимают участие все действия приема сообщений, стоящие в начале указанных компонентов. После выполнения этих редукций поочередно активизируются каналы связи с именами $d_1, d_2, \ldots, d_N$. Активизация этих каналов обусловлена тем, что действия приема сообщений по этим каналам стоят в начале последовательной композиции действий, образующей подпроцесс $CheckAll$. Тогда, согласно утверждению 1, в вычислении (2) происходят коммуникации по каналам связи $d_1, d_2, \ldots, d_N$.

Если все упомянутые выше коммуникации по каналам связи $d_1, d_2, \ldots, d_N$ происходят с привлечением действий приема сообщений из подпроцесса $CheckAll$, то после их выполнения активизируется канал связи r. А в том случае, если хотя бы одна из упомянутых коммуникаций происходит с привлечением действия приема сообщения из подпроцесса $Collect$, то это оказывается возможным согласно описанию этого подпроцесса только после коммуникации по каналу связи h. Однако передача сообщения по этому каналу, как видно из описания подпроцесса $OpenCh$, возможна только после коммуникации по каналу связи r.

Таким образом, независимо от значения $\varphi(\alpha)$ в вычислении (2) активизируется канал r. Значит, согласно утверждению 1, в вычислении (2) происходит коммуникация по каналу связи r. Рассмотрим самую первую из таких редукций. Она не может проводиться с использованием действий приема сообщений из подпроцесса $Collect$: как было отмечено выше, эти действия приема сообщений могут быть использованы только после коммуникации по каналу связи h, а этот канал может быть активи-

зирован лишь после проведения хотя бы одной коммуникации по каналу связи r. Следовательно, первая коммуникация по каналу связи r в вычислении (2) проводится с привлечением действия приема сообщения по этому каналу из подпроцесса $OpenCh$.

Как видно из описания этого подпроцесса, после выполнения первого содержащегося в нем действия приема сообщения по каналу r последовательно активизируются каналы ch, g и h. Поэтому, согласно утверждению 1, в вычислении (2) происходят коммуникации по указанным каналам связи. После выполнения этих редукций оказываются вновь активизированы все каналы связи $x_1, x_2, \ldots, x_n$. После проведения коммуникаций по этим каналам с использованием действий приема сообщений из подпроцессов вида S_ℓ оказываются активизированы все каналы связи, соответствующие всем литерам КНФ φ. После проведения коммуникации по всем активизированным «литерным» каналам связи с привлечением соответствующих действий приема сообщений из подпроцессов $Check_{\varphi=0}$ и $Check_{\varphi=1}$ активными оказываются все каналы связи $d_1, d_2, \ldots, d_N$, а также канал связи r. Выполнение коммуникаций по этим каналам связи проводится с использованием действий приема сообщений из подпроцесса $Collect$, а также подпроцесса $CheckAll$ в случае $\varphi(\alpha) = 0$. В результате проведения этих редукций в вычислении (2) образуется процесс P_m, в котором нет ни одного действия, т.е. $P_m \equiv \mathbf{0}$.

Таким образом, всякое вычисление (2) процесса $Proc_\varphi$ завершается нормально.

Далее заметим, что процесс $Proc_\varphi$ допускает передачу по каналу связи r только имен ch и $secret$. Покажем, что в любом вычислении (2) процесса $Proc_\varphi$ выполнение редукции $P_i \overset{ch(secret)}{\longrightarrow_\pi} P_{i+1}$ возможно в том и только том случае, когда 3-КНФ φ выполнима.

Единственное действие передачи сообщения по открытому каналу связи ch содержится в подпроцессе $OpenCh$. Он устроен так, что редукции $P_i \overset{ch(secret)}{\longrightarrow_\pi} P_{i+1}$ в вычислении (2)

- должна предшествовать редукция $P_j \overset{r(secret)}{\longrightarrow_\pi} P_{j+1}$,
- не может предшествовать ни одна коммуникация с привлечением действий из подпроцесса $Garbage$.

Все действия отправления сообщений по каналу r содержатся только в подпроцессах $Check_{\varphi=0}$ и $Check_{\varphi=1}$. Однако в подпроцессе $Check_{\varphi=0}$ эти действия призваны отправлять по каналу r имя ch, и лишь в подпроцессе $CheckAll$ имеется единственное действие отправления имени $secret$ по каналу связи r. Этому действию в подпроцессе $CheckAll$ предшествуют действия приема сообщений по каналам связи $d_1, d_2, \ldots, d_N$. Значит, выполнение редукции $P_j \overset{r(secret)}{\longrightarrow_\pi} P_{j+1}$ в вычислении (2) возможно в том и только том случае, если этому выполнению предшествовали выполнения коммуникаций по каналам связи $d_1, d_2, \ldots, d_N$.

Действия отправления сообщений по указанным каналам связи содержатся только в подпроцессах $Check_{D_k=1}$, $1 \le k \le N$. Согласно описанию каждого из подпроцессов $Check_{D_k=1}$ активизации канала связи d_k предшествует выполнение коммуникации по одному из каналов $\ell_{1k}, \ell_{2k}, \ell_{3k}$. Обозначим записью ℓ_{i_k} имя одного из тех каналов $\ell_{1k}, \ell_{2k}, \ell_{3k}$, выполнение коммуникации по которому предшествовало в вычислении (2) активизации канала связи d_k.

Рассмотрим множество выделенных литер $L = \{\ell_{i_k} \ : \ 1 \le k \le N\}$. Согласно описанию подпроцессов $Check_{D_k=1}$, $1 \le k \le N$, в каждом дизъюнкте D_k КНФ φ содержится одна из литер рассматриваемого множества. Кроме того, можно заметить, что в множестве литер L нет контрарных пар. Действительно, если бы в L содержалась контрарная пара литер $\ell = x^0_{i_k}$ и $\ell^* = x^1_{i_k}$, то это означало бы, что в вычислении (2) оба канала связи ℓ и ℓ^* были активизированы ранее, чем была выполнена редукция $P_i \overset{ch(secret)}{\longrightarrow}_\pi P_{i+1}$. Согласно описанию подпроцессов S_ℓ и S_{ℓ^*} активизация обоих указанных каналов может осуществиться только после выполнения двух коммуникаций по каналу x_m. Действия отправления сообщения по каналу x_m содержатся в подпроцессах $Garbage$ и $Init$, однако, на отрезке вычисления (2), предшествующем выполнению редукции $P_i \overset{ch(secret)}{\longrightarrow}_\pi P_{i+1}$, все действия подпроцесса $Garbage$ еще заблокированы, а подпроцесс $Init$ содержит лишь одно действие отправления сообщения по каналу x_m. Следовательно, только один из двух каналов связи ℓ и ℓ^* может быть активизирован до выполненния редукции $P_i \overset{ch(secret)}{\longrightarrow}_\pi P_{i+1}$ в вычислении (2).

Существование непротиворечивого множества литер, которое имеет хотя бы одну общую литеру с каждым дизъюнктом КНФ $Proc_\varphi$, является признаком выполнимости КНФ φ. Таким образом, если в каком-либо вычислении процесса $Proc_\varphi$ проводится редукция $P_i \overset{ch(secret)}{\longrightarrow}_\pi P_{i+1}$, то КНФ φ выполнима. И, напротив, если КНФ φ выполнима, то нетрудно построить вычисление процесса $Proc_\varphi$, в котором по открытому каналу связи ch передается имя $secret$. Значит, процесс $Proc_\varphi$ является стойким относительно угрозы $\{secret\}$ при осуществлении атаки $\{ch\}$ в том и только том случае, если КНФ φ невыполнима. Следовательно, проблема невыполнимости 3-КНФ *log-space* сводима к задаче проверки стойкости процессов множества $\mathcal{P}$ в модели пассивного противника. □

5. Заключение

Авторы еще раз отмечают, что представленный в статье результат о co-NP-полноте задачи проверки стойкости моделей нерекурсивных криптографических протоколов не является принципиально новым. Новизна этого результата состоит в том, что он был получен для, вероятно, самой простой модели вычислений, в которой можно сформулировать содержательную задачу проверки свойств информационной безопасности. Теорема 1 свидетельствует о том, что данная задача является вычислительно трудной даже в самой простой постановке, когда в проверяемых протоколах отсутствуют какие-либо криптографические примитивы, а противник является пассивным.

Особого внимания заслуживает модель пассивного противника и новое понятие мониторинга, расширяющее выразительные возможности исчислений мобильных процессов как средства описания криптографических протоколов. Ранее, насколько нам известно, моделирование противника и его взаимодействия с протоколом осуществлялось за рамками строгой модели π-исчисления. Мы убеждены, что с привлечением понятия мониторинга удастся разработать и общую модель активного противника, соответствующую концепции Долева–Яо. Создание такой модели

и получение для нее результатов о сложности задачи проверки стойкости протоколов, подобных тем, которые были установлены в статьях [4,11,21,24,29,31], является целью наших дальнейших исследований.

Поскольку задача проверки стойкости нерекурсивных процессов π-исчисления оказалась вычислительно трудной, представляет интерес вопрос о том, для каких классов процессов эта задача может быть решена за полиномиальное время. Как показывает доказательство теоремы 1, эта задача тесно связана с задачей проверки нормальной завершаемости процессов π-исчисления, которая является актуальной задачей проверки правильности поведения систем взаимодействующих процессов. Установление эффективно проверяемых достаточных условий нормальной завершаемости процессов π-исчисления также рассматривается нами как одна из тем дальнейших исследований.

Список литературы / References

[1] Abadi M., Gordon A. D., “A Calculus for Cryptographic Protocols: The Spi Calculus”, *Information and Computation*, **148**:1 (1999), 1–70.

[2] Abadi M., Fournet C., “Mobile Values, New Names, and Secure Communication”, *Proceedings of the 28-th ACM Symposium on Principles of Programming Languages*, 2001, 104–115.

[3] Amadio M. R., Lugiez D., “On the Reachability Problem for Cryptographic Protocols”, *Proceedings of the 11-th International Conference on Concurrency Theory*, 2000, 380–394.

[4] Amadio M. R., Lugiez D., Vanackere V., “On the symbolic reduction of processes with cryptographic functions”, *Theoretical Computer Science*, **290**:1 (2003), 695–740.

[5] Arapinis M., Liu J., Ritter E., Ryan M., “Stateful Applied Pi Calculus”, *Proceedings of the Principles of Security and Trust—Third International Conference*, 2014, 22–41.

[6] Blanchet B., Smith B., “Automated reasoning for equivalences in the applied pi calculus with barriers”, *Proceedings of the 29-th IEEE Computer Security Foundations Symposium*, 2014, 310–324.

[7] Bodei C., Degano P., Nielson F., Nielson H. R., “Static Analysis for the pi-Calculus with Applications to Security”, *Information and Computation*, **168**:1 (2001), 68–92.

[8] Borgstrom J., Nestmann U., “On bisimulations for the spi calculus”, *Mathematical Structures in Computer Science*, **15**:3 (2005), 487–552.

[9] Bruni A., Modersheim S., Nielson F., Nielson H. R., “Set-Pi: Set Membership Pi-Calculus”, *Proceedings of the 28-th IEEE Computer Security Foundations Symposium*, 2015, 185–198.

[10] Chadha R., Cheval V., Ciobaca S., Kremer S., “Automated Verification of Equivalence Properties of Cryptographic Protocols”, *ACM Transactions on Computational Logic*, **17**:4 (2016), 1–32.

[11] Chevalier Y., Kusters R., Rusinowitch M., Turuani M., “Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents”, *Proceedings of the 23-rd Annual Conference on the Foundations of Software Technology and Theoretical Computer Science*, 2003, 124–135.

[12] Chevalier Y., Kusters R., Rusinowitch M., Turuani M., “An NP decision procedure for protocol insecurity with XOR”, *Theoretical Computer Science*, **338**:1–3 (2005), 247–274.

[13] Chevalier Y., Kusters R., Rusinowitch M., Turuani M., “Deciding the security of protocols with commuting public key encryption”, *Electronic Notes in Theoretical Computer Science*, **125**:1 (2005), 55–66.

[14] Chevalier Y., Kusters R., Rusinowitch M., Turuani M., “Complexity results for security protocols with Diffie-Hellman exponentiation and commuting public key encryption”, *ACM Transactions on Computational Logic*, **9**:4 (2008), 1–52.

[15] Chretien R., Cortier V., Delaune S., "Decidability of trace equivalence for protocols with nonces", *Proceedings of the 28-th IEEE Computer Security Foundations Symposium*, 2015, 170–184.

[16] Cortier V., Delaune S., "A method for proving observational equivalence", *Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium*, 2009, 266–276.

[17] Curti M., Degano P., Priami C., Balardi C. T., "Modelling biochemical pathways through enhanced pi-calculus", *Theoretical Computer Science*, **325**:1 (2004), 111–140.

[18] Delaune S., Ryan M., Smyth B., "Automatic verification of privacy properties in the applied pi calculus", *Trust Management II*, IFIP International Federation for Information Processing, **263**, Springer, Boston, 2008, 263–278.

[19] Dolev D., Yao A., "On the security of public key protocols", *IEEE Transactoions on Information Theory*, **29**:2 (1983), 198–208.

[20] Durante L., Sisto R., Valenzano A., "Automatic Testing Equivalence Verification of Spi Calculus Specifications", *ACM Transactions on Software Engineering and Methodology*, **12**:2 (2003), 222–284.

[21] Durgin N. A., Lincoln P., Mitchell J. C., "Multiset rewriting and the complexity of bounded security protocols", *Journal of Computer Security*, **12**:2 (2004), 247–311.

[22] Godskesen J. C., "Formal Verification of the ARAN Protocol Using the Applied Pi-calculus", *Proceedings of the Sixth International IFIP WG 1.7 Workshop on Issues in the Theory of Security*, 2006, 99–113.

[23] Huima A., "Efficient infinite state analysis of security protocols", *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[24] Liang Z., Verma R. M., "Correcting and Improving the NP Proof for Cryptographic Protocol Insecurity", *Proceedings of the 5-th International Conference on Information Systems Security*, 2009, 101–116.

[25] Milner R., Parrow J., Walker D., "A calculus of mobile processes, I and II", *Information and Computation*, **100**:1 (1992), 1–40 and 41–77.

[26] Milner R., "Functions as Processes", *Mathematical Structures in Computer Science*, **2** (1992), 119–141.

[27] Milner R., *Communicating and mobile systems — the Pi-calculus*, MIT Press, 1999.

[28] Regev A., "Representation and Simulation of Biochemical Processes Using the pi-Calculus Process Algebra", *Proceedings of the 6-th Pacific Symposium on Biocomputing*, 2001, 459–470.

[29] Rusinowitch M., Turuani M., "Protocol Insecurity with Finite Number of Sessions is NP-complete", *Theoretical Computer Science*, **299**:1–3 (2003), 451–475.

[30] Smith H., Fingar P., *Business Process Management: The Third Wave*, Meghan-Kiffer Press Tampa, 2003.

[31] Tiplea F. L., Enea C., Birjoveanu C. V., "Decidability and complexity results for security protocols", *Verification of Infinite-State Systems with Applications to Security*, IOS Press, Amsterdam, 2006, 185–211.

[32] Tiu A., Dawson J., "Automating Open Bisimulation Checking for the Spi Calculus", *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, 2010, 307–321.

[33] Walker D., "Objects in the π-calculus", *Information and Computation*, **116**:4 (1995), 253–271.

Abbas M. M., Zakharov V.A., "Even Simple Processes of π-calculus are Hard for Analysis", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 589–606.

DOI: 10.18255/1818-1015-2018-6-589-606

Abstract. Mathematical models of distributed computations, based on the calculus of mobile processes (π-calculus) are widely used for checking the information security properties of cryptographic protocols. Since π-calculus is Turing-complete, this problem is undecidable in general case. Therefore, the study is carried out only for some special classes of π-calculus processes with restricted computational capabilities, for example, for non-recursive processes, in which all runs have a bounded length, for processes with a bounded number of parallel components, etc. However, even in these cases, the proposed checking procedures are time consuming. We assume that this is due to the very nature of the π -calculus processes. The goal of this paper is to show that even for the weakest model of passive adversary and for relatively simple protocols that use only the basic π-calculus operations, the task of checking the information security properties of these protocols is co-NP-complete.

Keywords: π-calculus, protocol, security, passive adversary, verification, complexity, NP-completeness

On the authors:
Marat M. Abbas, orcid.org/0000-0001-8019-7090, student,
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia, e-mail: unlock96@gmail.com

Vladimir A. Zakharov, orcid.org/0000-0002-3794-9565, Dr. of Science, professor,
National Research University Higher School of Economics (HSE),
20 Myasnitskaya st., Moscow, 101000 Russia,
Institute for system programming of the Russian Academy of Science (ISP RAS),
25 Alexander Solzhenitsyn st., Moscow, 109004 Russia,
e-mail: zakh@cs.msu.ru

Acknowledgments:
[1] This work was supported by the Russian Foundation for Basic Research, Grant N 18-01-00854
[2] This work was supported by the Russian Foundation for Basic Research, Grant N 16-01-00714

Моделирование и анализ информационных систем. Т. 25, № 6 (2018), с. 607–622
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 607–622

DOI: 10.18255/1818-1015-2018-6-607-622

УДК 004.052, 519.179.2

Онтология процессов, ориентированная на верификацию

Гаранина Н. О.[1], **Ануреев И. С.**, **Боровикова О. И.**

Поступила в редакцию 20 июля 2018
После доработки 10 октября 2018
Принята к публикации 10 ноября 2018

Аннотация. В статье представлена онтология процессов, близких взаимодействующим последовательным процессам Хоара. Она является частью интеллектуальной системы поддержки верификации свойств поведения таких процессов. Наше онтологическое представление процессов ориентировано как на применение формальных методов верификации, так и на извлечение информации из технической документации (с помощью нашей ранее разработанной системы извлечения информации из текстов на естественном языке). Мы описываем классы и домены онтологии, которые определяют взаимодействующие процессы. Эти процессы характеризуются множествами локальных и разделяемых переменных, списком действий над этими переменными, которые изменяют их значения, списком каналов взаимодействия процессов (которые, в свою очередь, характеризуются типом чтения сообщений, емкостью, способами записи и считывания, а также надежностью), списком коммуникационных действий для отправки сообщений. Помимо формального математического определения классов и доменов онтологии, приведены примеры описаний некоторых онтологических классов, а также типовых свойств и аксиом для них в редакторе Protégé на языке OWL с использованием правил вывода на языке SWRL. Для онтологического представления взаимодействующих процессов определяется их формальная операционная семантика, которая задается с использованием помеченной системы переходов. В интерливинговой модели она сводится к локальной операционной семантике отдельных экземпляров процессов. Представлена специализация онтологии для некоторых типов процессов из предметной области систем автоматического управления, моделирующих типовые функциональные элементы системы автоматического управления (датчики, сравнивающие устройства и регулирующие устройства), а также их комбинации. Понятия специализированной онтологии иллюстрируются на примере управляющей части системы розлива бутылок.

Ключевые слова: онтология, верификация, проверка моделей, процессы

Для цитирования: Гаранина Н. О., Ануреев И. С., Боровикова О. И., "Онтология процессов, ориентированная на верификацию", *Моделирование и анализ информационных систем*, **25**:6 (2018), 607–622.

Об авторах: Гаранина Наталья Олеговна, orcid.org/0000-0001-9734-3808, канд. физ.-мат. наук, ст. науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: garanina@iis.nsk.su

Ануреев Игорь Сергеевич, orcid.org/0000-0001-9574-128X, канд. физ.-мат. наук, ст. науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН, e-mail: anureev@iis.nsk.su

Боровикова Олеся Игнатьевна, orcid.org/0000-0001-5456-8513, мл. науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН, e-mail: olesya@iis.nsk.su

Благодарности:
[1] Работа поддержана грантом РФФИ 17-07-01600.

Введение

Наша долгосрочная цель — комплексный подход к обеспечению качества систем взаимодействующих последовательных процессов (далее систем процессов) с помощью формальных методов, который включает извлечение формальных моделей и свойств систем из текстов технической документации, а также их верификацию. Наш подход ориентирован на использование проверки моделей (впоследствии и дедуктивных методов) в качестве средств формальной верификации. Недостатком существующих систем поддержки разработки и верификации требований является то, что они предлагают только ручную формулировку требований и описаний систем процессов [1, 15, 19, 20].

Разрабатываемое нами интеллектуальное средство поддержки формальной верификации систем процессов будет автоматизированным образом извлекать и порождать как формальное описание системы, так и требования к ней. В этой статье мы предлагаем онтологию систем процессов. Другим ключевым компонентом системы является онтология шаблонов требований, предложенная в [5]. Содержание этих онтологий, т. е. множества экземпляров их классов, является онтологическими описаниями, соответственно, некоторой системы процессов и требований к ней. Эти описания извлекаются из корпуса технической документации с помощью наших методов извлечения информации из текстов на естественном языке [2–4]. Набор описаний требований также может быть расширен описаниями типичных требований к корректности системы, автоматически порождаемых из ее онтологического описания.

Онтологические описания системы и требований являются основой для ее формальной верификации. Чтобы верифицировать систему процессов, необходимо сначала выбрать подходящий верификатор, основанный на проверке моделей, с учетом формальной семантики онтологического представления требований к онтологии. Если он существует, онтологическое описание системы транслируется в язык спецификации модели, а описание требований — в язык спецификации свойств (обычно это язык темпоральной логики), которые являются входными языками этого верификатора. Если подходящего верификатора не существует, разумно разработать специальный верификатор шаблонов требований на основе формальной семантики соответствующей онтологии, который использует специализированные упрощенные алгоритмы верификации. Если семантика онтологических представлений модели системы и требований к ней, а также входные языки средства верификации строго определены и обеспечена корректность трансляции этих представлений во входные данные верификатора, корректность проверки требований гарантируется.

Работа с требованиями предполагает не только формальную семантику шаблонов требований, но и представление этих шаблонов как на естественном языке, так и в графической форме. Эти представления важны, поскольку из-за неоднозначности естественного языка извлеченные и порожденные требования могут не соответствовать ожиданиям инженера требований и может потребоваться их ручная корректировка. Онтологии систем процессов и шаблонов требований с их формальной

семантикой являются основой для развития системы поддержки формальной верификации. Наш подход позволяет моделировать и верифицировать широкий спектр систем: от программных распределенных систем до бизнес-процессов. Основным ограничением подхода является сложность и, временами, невозможность *точного* извлечения информации из технической документации, и извлеченные модели систем и требований могут потребовать ручной коррекции.

Наша онтология систем процессов задает набор процессов. Известные подходы к онтологическому описанию процессов обычно решают задачи моделирования. Онтологии PSL [16], Cyc [18], SBPM [6, 7] и GFO [9] фокусируются на задачах в области планирования производства, планирования процессов, управления рабочим процессом, реинжиниринга бизнес-процессов, симуляции, реализации процессов, моделирования процессов и управления проектами. Важным преимуществом этих подходов является то, что они учитывают концепции предметных областей при описании производственных, инженерных и бизнес-процессов. Указание конкретного времени и места событий и действий, связанных с этими процессами, в некоторых из этих подходов позволяет описать онтологии планирования, представляющие абстрактные и исполняемые планы организации процессов, что является еще одним достоинством. Однако эти преимущества делают очень сложной верификацию логики поведения процессов, лежащих в основе этих онтологий. Тесная взаимосвязь предметно-ориентированных понятий с логикой поведения процессов приводит к усложнению извлечения последней для использования в системах верификации, основанных на проверке моделей. Более того, извлеченная логика поведения по-прежнему должна транслироваться на язык системы верификации, основанный, как правило, на системах переходов, которые работают с понятиями состояния и перехода. В частности, для онтологий планирования конкретное время и место событий и действий нужно уметь выражать в терминах состояний. Кроме того, эти онтологии не имеют формальной семантики для логики поведения процессов, что еще более усложняет задачу.

В этой статье в разделе 1. мы определяем онтологические классы и домены, которые характеризуют взаимодействующие процессы системы в целом. В разделе 2. мы даем формальную семантику этих классов, включающую их представление на языке OWL (Web Ontology Language [13]) и операционную семантику действий процессов системы. Синтаксис языков представления онтологий, в частности языка OWL, является декларативным, но строго определенная операционная семантика нашей онтологии позволяет легко транслировать представление модели процессов во входные языки средств верификации, в частности, SPIN [11]. В разделе 3. излагается методология использования нашей онтологии систем процессов общего вида для спецификации некоторых компонентов систем автоматического управления (САУ). В разделе 4. – заключение и планы развития и использования онтологии процессов.

1. Онтология

Мы считаем *онтологией* структуру, включающую следующие элементы: (1) конечное непустое множество *классов*, (2) конечное непустое множество *атрибутов дан-*

ных и атрибутов отношений и (3) конечное непустое множество *доменов атрибутов данных*. Каждый класс определяется множеством атрибутов. Атрибуты данных принимают значения из доменов, а значениями атрибутов отношения являются экземпляры классов. *Экземпляр класса* определяется набором значений атрибутов для этого класса. *Информационный контент* онтологии представляет собой множество экземпляров ее классов. Задача пополнения онтологии заключается в извлечении информационного контента этой онтологии из входных данных. В нашем случае входными данными для пополнения онтологии систем процессов является техническая документация. Чтобы пополнить эту онтологию, мы используем наши методы извлечения семантической информации из текстов на естественном языке [3]. Заметим, что некоторые значения атрибутов извлеченных экземпляров онтологии могут быть не определены. Система процессов описывается в нашей онтологии с помощью набора экземпляров.

Мы рассматриваем систему последовательных взаимодействующих процессов в духе CSP [10]. Процессы (описываемые классом *Process*) характеризуются множествами локальных и разделяемых переменных, списком действий над этими переменными, которые изменяют их значения, списком каналов взаимодействия процессов, а также списком коммуникационных действий для отправки сообщений. Переменные процессов (класс *Variable*) принимают значения основных типов (булевские, конечные подмножества целых чисел или строк для перечислимых типов). Исходные условия для значений переменных определяются их сравнением с константами. Действия процессов (класс *Action*) включают базовые операции над основными типами. Условие выполнимости каждого действия зависит от охранных условий (класс *Condition*) для значений переменных и содержимого отправленных сообщений. Процессы могут отправлять сообщения через каналы (класс *Channel*) при выполнении охранных условий (класс *Condition*). Каналы характеризуются типом чтения сообщений, емкостью, способами записи и считывания, а также надежностью. На рисунке 1 классы представлены белыми овалами. Отношения между классами показаны стрелками с именами в серых овалах. Эти стрелки обозначены сплошными линиями, если отношение является “один ко многим”, и пунктирными, если отношение является взаимнооднозначным. Атрибуты данных класса, помещенные в штрих-пунктирные прямоугольники, связаны с их классами штрих-пунктирными стрелками.

В следующих разделах мы детально опишем классы и домены нашей онтологии, определим их формальную семантику. Также мы приведем примеры описаний некоторых онтологических классов, их свойств и аксиом, заданных в редакторе Protégé [14] средствами языка OWL с использованием правил вывода на языке SWRL (Semantic Web Rule Language) [12]. Эти свойства и аксиомы задают как правила согласованности значений атрибутов разных классов, так и ограничения на значения, количество и тип атрибутов отдельного класса. Мы приведем несколько типовых свойств и аксиом для некоторых классов. С помощью набора правил SWRL мы задаем условия, которые обеспечивают в Protégé вывод новых знаний и проверку корректности и совместности онтологического описания набора процессов с помощью машины логического вывода Hermit [8].

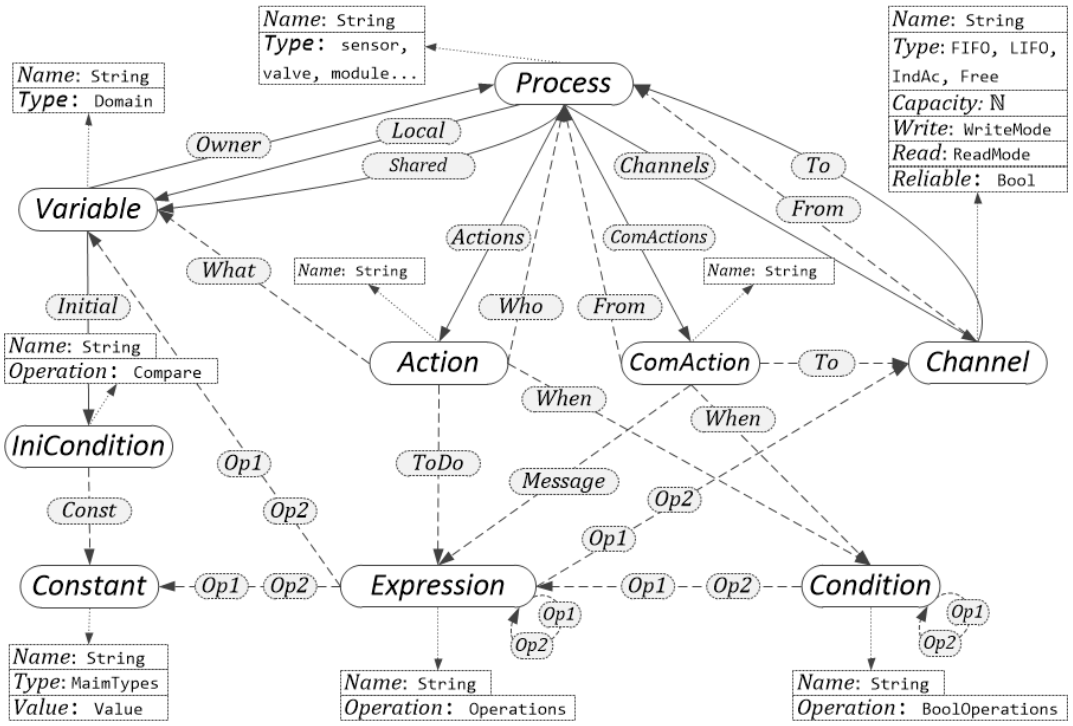


Рис. 1. Онтология систем процессов
Fig. 1. The Concurrent Process Ontology

2. Формальная семантика онтологии

Мы рассматриваем параллельную систему, которая является множеством последовательных процессов, взаимодействующих через разделяемые переменные и каналы ограниченной емкости. Значения разделяемых и локальных переменных, связанных с процессом, определяют его локальное состояние. В зависимости от предметной области задаются особенности взаимодействия процессов (синхронные, асинхронные) и характеристики каналов (FIFO, LIFO, надежные, ограниченные, и т.п.). Состояния канала задаются множеством сообщений, хранимым в нем.

Онтологическое описание O_S системы процессов S — это набор экземпляров онтологии, которые представляют процессы, каналы и другие элементы системы S. Мы определяем операционную семантику для O_S, используя стандартную модель помеченных систем переходов $M_S = (D_S, D_S^0, T_S, Act_S)$, где D_S — конечное непустое множество состояний, D_S^0 — непустое множество начальных состояний, T_S — отношение переходов между состояниями, Act_S — конечное множество меток переходов, описывающих действия системы. Семантика модели онтологического описания O_S — это частичная функция, которая отображает значения атрибутов экземпляров в элементы помеченной системы переходов $Sem : O_S \longrightarrow El(M_S)$.

Опишем множество $El(M_S)$. Пусть параллельная система S содержит конечное множество P_S процессов и конечное множество C_S ограниченных каналов. Пусть $\texttt{Domains} = \{\texttt{Integer}, \texttt{Bool}, \texttt{String}\}$ — конечная версия стандартных основных типов для значений переменных со стандартным приведением типов. Для процесса

$P \in P_S$ мы определим следующие множества: локальные переменные L_P, разделяемые переменные U_P, все переменные $V_P = L_P \cup U_P$, действия A_P, каналы C_P, коммуникационные действия S_P. Область определения переменной $v \in V_P$ есть $Val(T_P(v)) \subseteq \texttt{Values}$ для некоторого базового типа $T_P(v) \in \texttt{Domains}$. Пусть L_S — множество всех локальных переменных процессов из P_S, U_S — множество всех разделяемых переменных процессов из P_S, и функция означивания $V : L_S \cup U_S \longrightarrow 2^{\texttt{Values}}$ представляет значения переменных. Локальное состояние процесса P — это кортеж значений его переменных: $st_P = (V(v_1), ...V(v_{n_V}))$. Пусть $st_P(v) = V(v)$ для $v \in V_P$ — это значение переменной v в состоянии st_P. Начальные состояния процесса $st^0_P = (V^0(v_1), ...V^0(v_{n_V}))$ определяются начальными значениями его переменных $V^0 : L_S \cup U_S \longrightarrow 2^{\texttt{Values}}$. Локальное состояние канала c — это упорядоченный кортеж сообщений, хранимых в нем: $st_c = (m_1, ..., m_{n_c})$. Пусть $st_c(k) = m_k$ для $k \in [1..n_c]$ — содержимое k-го сообщения канала c в состоянии st_c. Начальное состояние всех каналов — пустой кортеж. Множество состояний D_S системы M_S — это декартово произведение значений локальных и разделяемых переменных процессов и состояний каналов. Множество начальных состояний D^0_S — это декартово произведение начальных состояний переменных процессов и состояний каналов. Мы считаем, что операционная семантика системы — это семантика чередования, т. е. ее состояние может изменяться только одним процессом системы. Следовательно, множество действий системы есть $Act_S = \cup_{P \in S} A_P \cup S_P$. В силу чередования отношение переходов T_S задается локальной операционной семантикой отдельных экземпляров процессов $P \in S_P$, которая определяется далее.

Опишем классы нашей онтологии. Далее для всех классов атрибут $\texttt{Name} \in \texttt{String}$ задает имя экземпляра класса. Имена атрибутов со множественными значениями заканчивается символом *.

Класс *Process* описывает процессы системы при помощи следующих атрибутов:
- $\texttt{Type}$: $\texttt{\{sensor, valve, ...\}} \in \texttt{String}$ содержит спецификатор процесса.
- $Local^*$: $Variable$ перечисляет переменные, видимые только в действиях процессов.
- $Shared^*$: $Variable$ перечисляет переменные, видимые процессам, их разделяющим.
- $Actions^*$: $Action$ перечисляет действия процесса над его переменными.
- $Channels^*$: $Channel$ перечисляет каналы, соединяющие процессы.
- $ComActions^*$: $ComAction$ перечисляет действия, отправляющие сообщения.

В этом классе не обязательно задание значений всех атрибутов. Однако, если $Local$ и $Shared$ не заданы, значение $Actions$ также не задано. Если значение $Channels$ не задано, значение $ComActions$ также не задано. Если задано значение $Channels$, то должны быть определены $Actions$ или $ComActions$. Если для процесса не заданы значения ни $Actions$, ни $ComActions$, ни $Channels$, то этот пассивный процесс считается *объектом*. Такие объекты удобно выделять для ясности описания системы и проверки его онтологической согласованности, но в процессе формальной верификации свойства поведения объектов не формулируются и не проверяются.

На рисунке 2 представлена иерархия классов онтологии процессов и фрагмент описания класса *Process*, описанный с помощью редактора *Protégé*. Слева, во вкладке “Class hierarchy” отображается иерархия заданных в онтологии классов. Вкладка “Usage: Process” для выбранного в иерархии класса описывает список его свойств и связей с другими классами и экземплярами онтологии, задаваемых атрибутами. Так, например, для класса *Process* указан фрагмент списка, содержащий имя про-

цесса `nameProcess`, связи с переменными и действиями (`localVar`, `actions`, `comActions`), а также отношения с классами *Channel*, *ComAction*, *Variable*, которые заданы в *Process* соответствующими атрибутами `fromChannel`, `fromComActions`, `owner`. Во вкладке "Description: Process" указываются дополнительные характеристики класса. Для экземпляров класса *Process* с помощью задания ограничений на атрибуты отношения определяются требования на количество понятий, связанных с этим классом: задание точно одного имени `nameProcess` и типа `typeProcess` процесса. Кроме того, с помощью следующих SWRL-правил можно задать требование определенности действий `a` или коммуникационных действий `ca` процесса `p` в случае определенности канала `ch` этого процесса:

```
Process(?p)^Channel(?ch)^channels(?p, ?ch)^differentFrom(?ch,NullChan)^
          Action(?a)^actions(?p,?a)^sameAs(?a,NullAct) ->
       ComAction(?ca)^comActions(?p,?ca)^differentFrom(?ca,NullComAct)

Process(?p)^Channel(?ch)^channels(?p,?ch)^differentFrom(?ch,NullChan)^
     ComAction(?ca)^comActions(?p,?ca)^sameAs(?ca,NullComAct) ->
                  Action(?a)^actions(?p,?a)^differentFrom(?a,NullAct)
```

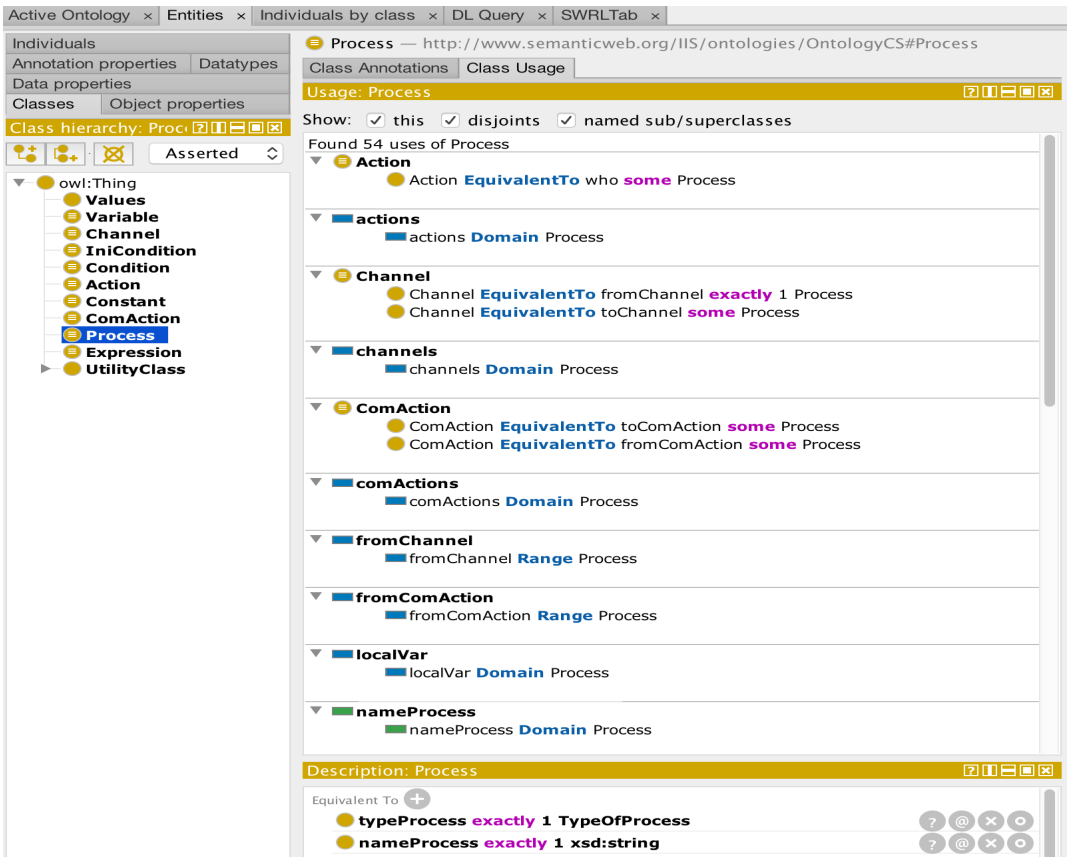


Рис. 2. Фрагмент описания онтологии процессов
Fig. 2. The Fragment of Description of the Concurrent Process Ontology

Для процесса P формальная операционная семантика его онтологического описания $Sem(P)$ определяется формальной семантикой онтологического описания его атрибутов: локальных переменных $Sem(P.Local)$, разделяемых переменных $Sem(P.Shared)$, действий $Sem(P.Actions)$, коммуникационных действий $Sem(P.ComActions)$, а также каналов $Sem(P.Channels)$. Процесс P может быть однозначно представлен в M_S посредством этих семантик. Для процесса P определим следующие множества: $L_P = \{l\}_{l \in Local}$, $U_P = \{u\}_{u \in Shared}$, $A_P = \{a\}_{a \in Actions}$, $C_P = \{c\}_{c \in Channels}$, $S_P = \{s\}_{s \in ComActions}$. Далее мы поочередно определяем семантику этих атрибутов.

Класс *Variable* описывает переменные процессов с помощью атрибутов:
- `Type`: `Domains` описывает тип переменной.
- $Initial^*$: $IniCondition$ описывает ограничения на начальное значение переменной.
- $Owner^*$: $Process$ ссылается на процессы, имеющие доступ к переменной.

Значения всех атрибутов этого класса, кроме $Initial^*$, должны быть заданы.

Ниже приведен фрагмент OWL-описания отношения `localVar` в формате *Turtle*. В данном примере для атрибута отношения `localVar` свойство обратной функциональности `InverseFunctionalProperty` гарантирует, что только для одного процесса *Process* переменная *Variable* является локальной:

```
:localVar rdf: type
          owl: ObjectProperty,
              owl: InverseFunctionalProperty;
          rdfs: domain: Process;
          rdfs: range: Variable.
```

Семантика переменной v процесса P задается таким образом: $Sem(v.Name) \in$ `String`, $Sem(v.Type) \in$ `Domains`, $Sem(v.Owner) \subseteq S_P$. Если $Sem(v.Owner) = \{P\}$, то $v \in L_P$. Семантика начального условия определяет множество начальных значений переменной $V^0(v)$:
– если $v.Initial = \emptyset$, то $V^0(v) = \{false\}$ для $v.Type =$ `Bool`, $V^0(v) = \{0\}$ для $v.Type \in$`Integer`, $V^0(v) = \{'' ''\}$ для $v.Type \in$ `String`;
– если $v.Initial \neq \emptyset$, то $V^0(v) = \cap_{ini \in v.Initial} Sem(ini)$.

Класс *IniCondition* задает начальные значения переменных с помощью следующих атрибутов:
- $IniConstant$: $\{Constant$, `Values`$\}$ определяет константу для сравнения.
- `Operation`: `Compare`$= \{<, >, =, \leq, \geq, \neq\}$ определяет сравнивающие действия.

Значения атрибутов $Const$ и `Operation` должны быть заданы.

Семантика начального условия ini переменной v — это подмножество множества ее значений: $Sem(ini.Name) \in$ `String`, и $Sem(ini) = \{val \in Val(v.Type) \mid val \circ Sem(Const)$ для $\circ \in \{<, >, =, \leq, \geq, \neq\}\}$, где $Sem(Const) = Sem(cs)$ для $cs \in Constant$, или $Sem(Const) = cs$ для $cs \in$ `Values`.

Класс *Constant* описывает константы и содержит следующие атрибуты:
- `Domain`: `Domains` описывает тип константы.
- `Value`: `Values` описывает значение константы.

Значения всех атрибутов должны быть определены.

Семантика константы cs — ее значение из множества возможных значений системы: $Sem(cs.Name) \in$ `String`, $Sem(cs.Domain) \in$ `Domains`, $Sem(cs.Value) \in$`Values`, и $Sem(cs) = cs.Value$.

Класс *Channel* описывает коммуникационные каналы с помощью атрибутов:
- $From$: $Process$ задает отправителя сообщений.
- To^*: $Process$ задает получателей сообщений из канала. Если получателей несколько, то канал — широковещательный.
- `Type`: `Order = {FIFO, LIFO, IndAc, Free}` определяет порядок чтения из канала, где `IndAc` означает доступ по индексу и `Free` означает случайный доступ.
- `Capacity`: `Integer` задает емкость (вместительность) канала.
- `Reliable`: `Bool` характеризует внешнюю надежность канала в том смысле, что сообщения не теряются по внешним причинам.
- `Write`∈`WriteMode = {NSent, Old, New, Some}` описывает режим записи при полном канале:
  - `NSent`: сообщение не записывается в канал, то есть теряется;
  - `New`: сообщение пишется вместо удаляемого сообщения с наибольшим индексом;
  - `Old`: сообщение записывается вместо удаляемого сообщения с индексом 1;
  - `Some`: сообщение записывается вместо некоторого удаляемого сообщения.
- `Read`∈`ReadMode = {Del, Keep, Tail, Head, Some}` задает режим чтения:
  - `Del`: сообщение удаляется из канала после прочтения;
  - `Keep`: сообщение не удаляется из канала после прочтения;
  - `Tail`: индекс сообщения после чтения становится наибольшим в канале;
  - `Head`: индекс сообщения после чтения становится равным 1;
  - `Some`: индекс сообщения после чтения становится произвольным.

Значения атрибутов $From$ и To должны быть определены. Мы задаем следующие значения атрибутов по умолчанию: `Type` = `FIFO`, `Capacity` = 1, `Write` = `NSent`, `Read` = `Del`, `Reliable` = $true$. Если у канала процесс P есть в атрибуте $From$ или To^*, то этот канал должен быть в атрибуте $Channels^*$ этого процесса.

Семантика канала c процесса P определяется следующим образом:
$Sem(c.Name) \in$ `String`, $Sem(c.From) = P$, и $Sem(s.To) \subseteq S_P$. Семантика локальных состояний канала и их изменений определена ниже в описаниях классов действий и коммуникационных действий.

Класс *Action* описывает действие, выполняемое процессом.
- Who: $Process$ указывает на процесс, который выполняет действие.
- $What$: $Variable$ ссылается на переменную, значение которой может измениться в результате действия.
- $ToDo$: $Expression$ описывает выражение, значение которого присваивается переменной из $What$.
- $When$: $Condition$ описывает булевскую формулу, которая задает условие выполнения действия в локальном состоянии.

Значения атрибутов Who, $What$ и $ToDo$ должны быть заданы. Если $When$ не задан, это означает, что действие может быть выполнено в любом локальном состоянии процесса. Значения атрибутов $ToDo$ и $When$ могут использовать только переменные процесса и сообщения из его каналов.

Следующее SWRL-правило задает требование того, что если атрибут `what` действия `a` процесса `p` включает переменную `v`, то ее атрибут `owner` включает `p`:

```
Action(?a) ^ Variable(?v) ^ Process(?p) ^ what(?a, ?v) ^ who(?a, ?p) ->
                                                        owner(?v, ?p)
```

Определим семантику действия a процесса P. Очевидно, $Sem(a.Name) \in$`String`, $Sem(a.Who) = P$, и $Sem(a.What) \in V_P$. Пусть $a.What = v$ — переменная, которая должна быть изменена действием a, и $C_a \subseteq C_P$ — каналы процесса, сообщения из которых читаются действием. Отношение перехода между локальными состояниями как процесса P, так и его каналов определяется следующим образом. Если локальные состояния процесса st_P и st_c каждого канала $c \in C_a$ таковы, что условие $a.When$ удовлетворено ($Sem(a.When) = true$), и выражение действия $a.ToDo$ имеет значение ($Sem(a.ToDo) \neq null$), то после выполнения действия a, новые локальные состояния st'_P процесса и его каналов таковы, что $st'_P(v) = Sem(a.ToDo) \wedge \forall x \in V_P \setminus \{v\} : st'_P(x) = st_P(x) \wedge \forall y \in C_P \setminus C_a : st'_y = st_y$, и новое состояние st'_c каждого канала c из C_a соответствует семантике операции чтения для каналов (см. следующий раздел). Заметим, что в процессе вычисления $Sem(a.When)$ и $Sem(a.ToDo)$, состояния каналов не меняются. Если $Sem(a.ToDo) = null$ или $Sem(a.When) \neq true$, все состояния остаются неизменными: $st'_P = st_P$, и $\forall c \in C_a : st'_c = st_c$.

Класс *Expression* описывает выражения, значения которых присваиваются переменным процесса в результате действий, и содержит следующие атрибуты.
- Атрибуты $Op1, Op2$: $\{Expression, Variable, Constant, Channel,$ `Values`$\}$ задают левый и правый операнды выражения.
- `Operation`: `Operations`$= \{+, -, ^*, \%, >, <, \leq, \geq, \neq, \neg, \wedge, \vee, \rightarrow\}$ определяет стандартные целочисленные и булевские операции над операндами.

Значения $Op1$ и $Op2$ должны использовать переменные и сообщения канала, доступные процессу. Значения атрибутов $Op1$ и `Operation` должны быть определены. Если $Op2$ не определен, то значение `Operation` должно быть унарной операцией.

Следующий набор SWRL-правил ограничивает значения операндов $Op1$ и $Op2$, входящих в выражение атрибута `toDo` действия `a` процесса `p`, переменными `v` и сообщениями канала `ch`, доступными для процесса:

```
who(?a, ?p) ^ Action(?a) ^ Process(?p) ^ toDo(?a, ?e1) ^
        Expression(?e1) ^ Expression(?e2) ^ subExp(?e1, ?e2)
                 op1Expression(?e2, ?v) ^ Variable(?v) -> owner(?v, ?p)

who(?a, ?p) ^ Action(?a) ^ Process(?p) ^ toDo(?a, ?e1) ^
        Expression(?e1) ^ Expression(?e2) ^ subExp(?e1, ?e2)
            op1Expression(?e2, ?ch) ^ Channel(?ch) -> channels(?ch, ?p)
```

Здесь `SubExp(?e1, ?e2)` является транзитивным отношением “быть подвыражением” на классе *Expression*.

Пусть для действия a процесса P в состоянии st_P выражение действия имеет вид $a.ToDo = e$. Семантикой этого выражения является значение базового типа или $null$: $Sem(e) \in$ `Values` $\cup\, null$.
– Для бинарных операций $\circ \in \{+, -, ^*, \%, >, <, \leq, \geq, \neq, \wedge, \vee, \rightarrow\}$:

$Sem(e) = Sem(e.Op1) \circ Sem(e.Op2)$;
– Для унарных операций $\circ \in \{-, \neg\}$: $Sem(e) = \circ Sem(e.Op1)$;
– $Sem(e) = null$ при $Sem(e.Op1) = null$, или $Sem(e.Op2) = null$.
Пусть $Op \in Op1, Op2$. $Sem(e.Op)$ определяется следующим образом:
– $e.Op = e' \in \{Expression, Constant, \texttt{Values}\}$: $Sem(e.Op) = Sem(e')$;
– $e.Op = w \in Variable$, $w \in V_P$: $Sem(e.Op) = st_P(w)$;
– $e.Op = c \in Channel$, $c \in C_a$: $Sem(e.Op) = Val$, где
  – $st_c = \emptyset$, то $Val = null$, и $st'_c = st_c$;
  – Если c.Type = `FIFO`, то $Val = st_c(1)$, и если c.`Read` равно
    – `Del`, то $st'_c = (m_2, ..., m_{n_c})$;
    – `Keep`, то $st'_c = st_c$;
    – `Tail`, то $st'_c = (m_2, ..., m_{n_c}, m_1)$;
    – `Head`, то $st'_c = st_c$;
    – `Free`, то $st'_c = (m_2, ..., m_1, ..., m_{n_c})$.
Аналогичным образом определяется семантика для других типов чтения.

Класс *Condition* описывает охранное условие для действий и коммуникационных действий процессов. Его описание и семантика почти такие же, как для класса *Expression*, за исключением того, что *Condition* использует только булевские операции и значения.

Класс *ComAction* описывает коммуникационные действия отправки сообщений и содержит следующие атрибуты:
- $From$: $Process$ задает процесс, который отправляет сообщение.
- To: $Channel$ задает канал, который доставляет сообщения получателю.
- $Message$: $Expression$ описывает отправленное сообщение.
- $When$: $Conditions$ описывает охранное условие для отправки сообщения.
Заметим, что наша модель взаимодействия процессов реализует получение сообщений по необходимости, т. е. чтение из канала выполняется, только когда процесс вычисляет охранное условие или выражение действия. Значения атрибутов $Message$, $From$ и To должны быть определены. Если $When$ не задан, это означает, что коммуникационное действие может выполняться в любом локальном состоянии процесса. Значения атрибутов $Message$ и $When$ могут использовать только переменные процесса и сообщения из его каналов.

Семантика действия s процесса P, отправляющего сообщение в состоянии st_P, определяется следующим образом: $Sem(s.Name) \in \texttt{String}$, $Sem(s.From) = P$, и $Sem(s.To) \in C_P$. Пусть $s.To = c$ — канал процесса для отправки сообщения, $st_c = (m_1, ..., m_{n_c})$ — его локальное состояние, и $C_s \subseteq C_P$ — каналы процесса, из которых сообщение читается. Пусть $Sem(s.Message) = Mes$, и $Sem(s.When) = Cnd$. Локальная операционная семантика отправки сообщений определяется следующим образом. Локальные состояния процесса P и неиспользуемые каналы не меняются: $st'_P = st_P$, и $\forall cl \in C_P \setminus (C_s \cup \{c\}) : st'_{cl} = st_{cl}$. Если локальные состояния P и каналов $cs \in C_s$ таковы, что $Mes = null$, или $Cnd \neq true$, то $\forall c \in C_P : st'_c = st_c$. Если условие $s.When$ выполнено ($Cnd = true$) и значение сообщения можно вычислить ($Mes \neq null$), то после выполнения коммуникационного действия s новое состояние каналов st'_{cs} для $cs \in C_s$ соответствует семантике операции чтения, определенной выше. Новое локальное состояние st'_c канала c зависит от емкости канала c.`Capacity`

и метода записи в канал c.`Write` следующим образом:
– Если $n_c < c$.`Capacity`, то $st'_c = (m_1, ..., m_{n_c}, Mes)$.
– Когда канал полон, если c.`Write` равно
  – `NSent`, то $\forall c \in C_P : st'_c = st_c$;
  – `Old`, то $st'_c = (Mes, m_2, ..., m_{n_c})$;
  – `New`, то $st'_c = (m_1, ..., m_{n_c-1}, Mes)$;
  – `Some`, то $st'_c = (m_1, ..., m_{i-1}, Mes, m_{i+1}, ..., m_{n_c})$, $i \in [1..n_c]$.
Если известна локальная семантика действий и коммуникационных действий каждого процесса, то может быть однозначно оперделено отношение перехода T_S.

3. Онтологическое моделирование функциональных элементов системы автоматического управления

В этом разделе мы моделируем в нашей онтологии некоторые типовые функциональные элементы САУ (датчики, сравнивающие устройства и регулирующие устройства) и представляем иллюстративный пример онтологического моделирования САУ. Состояние объекта управления, которым управляет САУ, характеризуется переменными состояния — количественными характеристиками, меняющимися во времени.

Датчик отслеживает изменение некоторой переменной состояния и посылает информацию об этом другим элементам САУ. Датчик моделируется в онтологии процессом со следующими свойствами. Атрибут `Type` этого процесса имеет значение `sensor`. Атрибут $Shared^*$ содержит переменную mv, моделирующую переменную состояния, отслеживаемую датчиком (monitored variable). Атрибут $ComActions^*$ содержит действия по передаче значения переменной mv по каналам, содержащимся в атрибуте $Channels^*$.

Сравнивающее устройство сравнивает значение своего входного параметра с некоторым эталонным значением. Сравнивающее устройство представляется в онтологии процессом со следующими свойствами. Атрибут `Type` этого процесса имеет значение `comparator`. Атрибут $Channels^*$ содержит каналы передачи результата сравнения значения входного параметра с эталонным значением, а также каналы, по которым процесс получает значения входного параметра. Атрибут $ComActions^*$ содержит действия по вычислению результата сравнения и его передачи по каналам.

Датчик с функцией сравнения, комбинирующий функции датчика и сравнивающего устройства, моделируется в онтологии процессом, свойства которого являются комбинацией свойств процессов, представляющих датчики и сравнивающие устройства. Атрибут `Type` этого процесса имеет значение `sensor-comparator`. Поскольку такой датчик сам отслеживает значение переменной состояния, нет необходимости в специальных каналах передачи этого значения.

Регулирующее устройство поддерживает некоторую заданную характеристику объекта управления. Оно непосредственно влияет на состояние объекта управления, изменяя его переменные в соответствии со значениями входных параметров регулирующего устройства. Моделирование регулирующего устройства в онтологии зависит от количества режимов его работы (например, “вкл.” и “выкл.”). Онтологическое представление регулирующего устройства с количеством режимов работы n

задается процессом со следующими свойствами. Атрибут `Type` этого процесса имеет значение `regulator`. Атрибут $Local^*$ содержит переменную $mode$, которая моделирует режимы работы и принимает значения перечислимого типа, состоящего из n элементов. Атрибут $Shared^*$ содержит переменные, моделирующие переменные объекта управления, на которые влияет регулирующее устройство. Атрибут $Channels^*$ содержит каналы, по которым процесс получает значения входных параметров регулирующего устройства. Атрибут $Actions^*$ содержит действия, которые изменяют разделяемые переменные. Выражения, задаваемые атрибутом $ToDo$ этих действий и вычисляющие новые значения указанных переменных, зависят от значения локальной переменной $mode$ и сообщений, содержащих значения входных параметров регулирующего устройства.

Чтобы проиллюстрировать, как САУ моделируется в нашей онтологии, мы представим подсистему системы розлива бутылок из [17], описанную на естественном языке. Мы рассмотрим следующую упрощенную подсистему: датчики низкого и высокого уровня жидкости задаются процессами P_l и P_u, а впускной и выпускной клапаны — процессами P_i и P_b. Мы считаем, что все каналы имеют тип FIFO, емкость 1 и сообщения удаляются после прочтения.

Датчик низкого уровня моделируется процессом P_l=(`Type`: `sensor-comparator`, $Shared^*$: $\{mv\}$, $Channels^*$: $\{li, lb\}$, $ComActions^*$: $\{lo, lc\}$). Переменная mv задает информацию о текущем уровне жидкости. Она принимает целочисленные значения от $min.Value$ до $max.Value$, где константы min и max определяют уровни, когда бак почти пуст и когда он переполнен. Каналы li и lb соединяют датчик P_l с клапанами P_i и P_b, соответственно. Коммуникационное действие lo посылает сообщение “toOpen” впускному клапану P_i, когда бак почти пуст: $lo = (From$: P_l, To: li, $Message$: “toOpen”, $When$: $mv \leq min)$. Коммуникационное действие lc определяется симметричным образом.

Впускной клапан моделируется процессом P_i=(`Type`: `regulator`, $Local^*$: $\{mode\}$, $Shared^*$: $\{ev\}$, $Actions^*$: $\{inc, io, ic\}$, $Channels^*$: $\{li, ui\}$). Переменная $mode$ имеет перечислимый тип {“open”, “closed”}. Действие inc увеличивает уровень жидкости на 1: $inc = (Who$: P_i, $What$: mv, $ToDo$: $ev + 1$, $When$: $mode =$ “open”). Действие io открывает впускной клапан: $io = (Who$: P_i, $What$: $mode$, $ToDo$: “open”, $When$: $li =$ “toOpen”). Действие по закрытию клапана ic определяется симметричным образом. Процессы P_u и P_b определяются аналогичным образом.

4. Заключение

В данной статье мы представляем онтологию систем процессов, которая является одним из базовых компонентов интеллектуальной системы поддержки верификации. Наши методы извлечения информации из текстов технической документации позволяют пополнять эту онтологию экземплярами процессов, информация о которых обнаружена в этих текстах. Затем модуль трансляции должен транслировать онтологическое представление системы во входной язык некоторого верификатора. Чтобы обеспечить корректность трансляции и последующей верификации, в этой статье мы определяем формальную семантику нашей онтологии, связывая формальными правилами помеченную систему переходов со множеством экземпля-

ров процессов из онтологии. Формальная семантика онтологического представления процессов, ориентированная на формальные методы верификации, отличает наш подход от современных подходов к определениям онтологий процессов.

В дальнейшем мы планируем разработать системы онтологических шаблонов процессов для различных предметных областей: систем автоматического управления, бизнес-процессов и т. д. В рамках построения нашей интеллектуальной системы поддержки верификации мы также разработаем метод извлечения типичных требований к системе из онтологического представления процессов для пополнения онтологии требований.

Список литературы / References

[1] Autili M. et al., “Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar”, *IEEE Transactions on Software Engineering*, **41**:7 (2015), 620–638, https://doi.org/10.1109/TSE.2015.2398877.

[2] Garanina N., Sidorova E., “Context-dependent Lexical and Syntactic Disambiguation in Ontology Population”, *Concurrency, Specification and Programming (CS&P)*, Proc. of the 25th Int. Workshop (Humboldt-Universitat zu Berlin), 2016, 101–112.

[3] Garanina N., Sidorova E., Bodin E., “A Multi-agent Text Analysis Based on Ontology of Subject Domain”, *Perspectives of System Informatics. PSI 2014*, Proc. Int. Conference (St. Petersburg, Russia, June 24–27), Lecture Notes in Computer Science, **8974**, eds. Voronkov A., Virbitskaite I., Springer, Berlin, Heidelberg, 2015, 102–110, https://doi.org/10.1007/978-3-662-46823-4_9.

[4] Garanina N. et al., “Using Multiple Semantic Measures For Coreference Resolution In Ontology Population”, *International Journal of Computing*, **16**:3 (2017), 166–176.

[5] Garanina N., Zyubin V., Liakh T., “Ontological Approach to Organizing Specification Patterns in the Framework of Support System for Formal Verification of Distributed Program Systems”, *System Informatics*, **9** (2017), 111–132, http://dx.doi.org/10.31144/si.2307-6410.2017.n9.p111-132.

[6] Hepp M., Dumitru R., “An Ontology Framework for Semantic Business Process Management”, *Wirtschaftsinformatik Proc.*, (Karlsruhe, Germany, February 28 – March 2), 2007, 423–440, https://aisel.aisnet.org/wi2007/27/.

[7] Hepp M. et al., “Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management”, *e-Business Engineering (ICEBE 2005)*, Proc. Int. Conf. (Beijing, China, October 12–18), IEEE, 2005, 535–540, https://doi.org/10.1109/ICEBE.2005.110.

[8] *HermiT OWL Reasoner*, http://www.hermit-reasoner.com/.

[9] Herre H., “General Formal Ontology (GFO): A Foundational Ontology for Conceptual Modelling”, *Theory and Applications of Ontology: Computer Applications*, eds. Poli R., Healy M., Kameas A., Springer, Dordrecht, 2010, 297–345, https://doi.org/10.1007/978-90-481-8847-5_14.

[10] Hoare C.A.R., *Communicating sequential processes*, Prentice-Hall, 1985, 256 pp.

[11] Holzmann G.J., *The Spin Model Checker: Primer and Reference Manual*, Addison-Wesley Professional, 2003, 608 pp.

[12] Horrocks I. et al., *SWRL: A Semantic Web Rule Language combining OWL and RuleML*, http://www.w3.org/Submission/SWRL.

[13] *OWL Web Ontology Language Overview: W3C Recommendation 10 February 2004*, https://www.w3.org/TR/owl-features/, eds. D.L. McGuinness, F. Harmelen van..

[14] *Protégé. A free, open-source ontology editor and framework for building intelligent systems*, http://protege.stanford.edu/.

[15] Salamah S., Gates A.Q., Kreinovich V., "Validated patterns for specification of complex LTL formulas", *Journal of Systems and Software*, **85**:8 (2012), 1915–1929, https://doi.org/10.1016/j.jss.2012.02.041.

[16] Schlenoff C. et al., *The Process Specification Language (PSL): Overview and Version 1.0 Specification*, NIST Interagency/Internal Report (NISTIR), **6459**, 1999, https://www.nist.gov/publications/process-specification-language-psl-overview-and-version-10-specification.

[17] Shanmugham S.G., Roberts C.A., "Application of graphical specification methodologies to manufacturing control logic development: A classification and comparison", *International Journal of Computer Integrated Manufacturing*, **11**:2 (1998), 142–152, http://dx.doi.org/10.1080/095119298130886.

[18] Stuart A., Curtis J., "A Process Ontology", *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web. EKAW 2002*, Proc. of the 13th Int. Conference (Sigüenza, Spain, October 01–04), Lecture Notes in Computer Science, **2473**, eds. Gómez-Pérez A., Benjamins V.R., Springer, Berlin, Heidelberg, 2002, 108–113, https://doi.org/10.1007/3-540-45810-7_13.

[19] Wong P.Y.H., Gibbons J., "Property Specifications for Workflow Modelling", *Integrated Formal Methods. IFM 2009*, Proc. Int. Conf. (Düsseldorf, Germany, February 16–19), Lecture Notes in Computer Science, **5423**, eds. Leuschel M., Wehrheim H., Springer, Berlin, Heidelberg, 2009, 166–180, https://doi.org/10.1007/978-3-642-00255-7_5.

[20] Yu J. et al., "Pattern based property specification and verification for service composition", *Web Information Systems. WISE 2006*, Proc. of 7th Int. Conf. (Wuhan, China, October 23–26), Lecture Notes in Computer Science, **4255**, eds. Aberer K. et al., Springer, Berlin, Heidelberg, 2006, 156–168, https://doi.org/10.1007/11912873_18.

Garanina N. O., Anureev I. S., Borovikova O. I., "Verification Oriented Process Ontology", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 607–622.

DOI: 10.18255/1818-1015-2018-6-607-622

Abstract. This paper presents the ontology of the concurrent processes close to Hoare communicating sequential processes. It is the part of the intellectual system for supporting verification of behavioural properties of these processes. Our ontological representation of the processes is oriented both to the application of formal verification methods and to the extraction of information from technical documentation (by our previously developed system of information extraction from a natural language text). We describe the ontology classes and domains that define communicating concurrent processes. These processes are characterized by sets of local and shared variables, a list of actions on these variables which change their values, a list of channels for the process communication (which, in turn, are characterized by the type of reading messages, capacity, ways of writing and reading, and reliability), and also a list of communication actions for sending messages. In addition to the formal mathematical definition of classes and domains of the ontology, examples of descriptions of some ontological classes as well as typical properties and axioms for them are specified in the editor Protégé in the OWL language with the use of the inference rules in the SWRL language. The formal operational semantics of communicating processes is determined on their ontological representation and is given as a labelled transition system. It is reduced to the local operational semantics of separate process instances in the interleaving model. We specialize several types of processes from the subject domain of automatic control systems that model the typical functional elements of the automatic control system (sensors, comparators and regulators) as well as their combinations. The concepts of the specialized ontology are illustrated by the example of a control part for a bottle-filling system.

Keywords: ontology, verification, model checking, concurrent processes

On the authors: Natalia O. Garanina, orcid.org/0000-0001-9734-3808, PhD, senior researcher
A.P. Ershov Institute of Informatics Systems,
6 Acad. Lavrentjev pr., Novosibirsk 630060, Russia, e-mail: garanina@iis.nsk.su

Igor S. Anureev, orcid.org/0000-0001-9574-128X, PhD, senior researcher
A.P. Ershov Institute of Informatics Systems,
6 Acad. Lavrentjev pr., Novosibirsk 630060, Russia, e-mail: anureev@iis.nsk.su

Olesya I. Borovikova, orcid.org/0000-0001-5456-8513, junior researcher
A.P. Ershov Institute of Informatics Systems,
6 Acad. Lavrentjev pr., Novosibirsk 630060, Russia, e-mail: olesya@iis.nsk.su

Acknowledgments:
[1] The research has been supported by Russian Foundation for Basic Research (grant 17-07-01600).

Моделирование и анализ информационных систем. Т. 25, № 6 (2018), с. 623–636
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 623–636

DOI: 10.18255/1818-1015-2018-6-623-636

UDC 519.987

Translation from Event-B into Eiffel

Reznikova S., Rivera V., Lee J. Y., Mazzara M.

Received September 10, 2018
Revised October 10, 2018
Accepted November 1, 2018

Abstract. Formal modelling languages play a key role in the development of software: they enable users to specify functional requirements that serve as documentation as well; they enable users to prove the correctness of system properties, especially for critical systems. However, there is still an open question on how to map formal models to a specific programming language. In order to propose a solution, this paper presents a source-to-source mapping between Event-B models, a formal modelling language for reactive systems, and Eiffel programs, an Object Oriented (O-O) programming language. The mapping not only generates an actual Eiffel code of the Event-B model, but also translates model properties as contracts. The contracts follow the Design by Contract principle and are natively supported by the programming language. The mapping is implemented in the freely available Rodin plug-in EB2Eiffel. Thus, users can develop systems (i) starting with the modelling of functional requirements (properties) in Event-B, then (ii) formally proving the correctness of such properties in Rodin and finally (iii) by using EB2Eiffel to translate the model into Eiffel. In Eiffel, users can extend/customise the implementation of the model and formally prove it against the initial model. This paper also presents different Event-B models from the literature to test EB2Eiffel and its limitations. The article is published in the authors' wording.

Keywords: stepwise refinement, Design-by-Contract, formal modelling, reactive systems, Event-B, Eiffel

For citation: Reznikova S., Rivera V., Lee J. Y., Mazzara M., "Translation from Event-B into Eiffel", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 623–636.

On the authors:
Sofia Reznikova, orcid.org/0000-0003-4616-2729, undergraduate student
Innopolis University,
1 Universitetskaya St., Innopolis 420500, Russia, e-mail: s.reznikova@innopolis.ru

Victor Rivera, orcid.org/0000-0002-1946-8979, Assistant Professor,
Innopolis University,
1 Universitetskaya St., Innopolis 420500, Russia, e-mail: v.rivera@innopolis.ru

JooYoung Lee, orcid.org/0000-0001-5421-730X, Assistant Professor,
Innopolis University,
1 Universitetskaya St., Innopolis 420500, Russia, e-mail: j.lee@innopolis.ru

Manuel Mazzara, orcid.org/0000-0002-3860-4948, Associate Professor,
Innopolis University,
1 Universitetskaya St., Innopolis 420500, Russia, e-mail: m.mazzara@innopolis.ru

Introduction

Modelling methodologies are used in software development to foresee how the resulting system will behave prior to building it. This stage might reveal hidden errors that can be handled at a smaller cost. This is of paramount importance since final users of systems are not aware of the consequences that malfunctioning systems might carry. There are different approaches to use these modelling methodologies (some are described in [6]), e.g. top-down and bottom-up approaches: using a top-down approach, one could think to start developing the system from a very abstract view point towards more concrete ones; in a bottom-up approach, on the other hand, one might think to start from a more concrete state of the system then add more functionality to it. The key point on both approaches is to always prove that properties of the systems hold.

Event-B is a formal modelling language for reactive systems, introduced by Abrial [1], which allows the modelling of complete systems. It follows the top-down approach by means of refinements. Event-B allows the creation of abstract systems and the expression of their properties. One can prove that the system indeed meets the properties then create a refinement of the system: same system with more details. It has been applied with success in both research and industrial projects, and in integrated EU projects aiming at putting together the two dimensions.

On the other side of the spectrum, following a bottom-up approach, one can work with the Eiffel programming language [8]. In Eiffel, one can create classes that implement any system. The behaviour of such classes is specified in Eiffel using contracts: pre- and post-conditions and class invariants. These mechanisms are natively supported by the language (as opposed to other programming languages). Having contracts, one can then verify that the implementation is indeed the intended (this can done statically by using a static verifier like Autoproof), and also one can track the specifications against the implementation [10]. This paper presents a series of rules to produce Eiffel programs from Event-B models, bridging both top-down and bottom-up approaches. An excerpt of the rules was presented elsewhere [15]. We also present a plug-in for Rodin, an IDE for Event-B, that implements the rules presented, the plug-in receives an Event-B model as input and produces the corresponding Eiffel classes equipped with contracts.

Several translations have been proposed and implemented that go in the same direction as the work presented on this paper. In [7], Mèry and Singh present the `EB2ALL` tool-set that includes a translation from Event-B models to C, C++ and Java. Unlike our translation, `EB2ALL` provides support for a small part of Event-B's syntax, and users are required to write a final Event-B implementation refinement in the syntax supported by the tool. The Code Generation tool [4] generates concurrent Java and Ada programs for a tasking extension of Event-B. Unlike these tools, the work presented here does not require user's intervention, while it works on the proper syntax of the Event-B model. In addition, these tools do not take full advantage of the elements present in the source language, e.g. invariants. The work presented in this paper, in addition to generating source code, it generates contracts from the source language, making use of the Design-by-Contract approach. In [14, 13, 2], authors present a translation from Event-B to Java, annotating the code with JML (Java Modelling Language) specifications, and [12] shows its application. The main difference with the work presented here is the target language. We are translating to Eiffel which natively supports Design-by-Contract. In addition, Eif-

fel comes with different tools to prove Eiffel code statically (e.g. Autoproof [16]) that fully supports the language. Another difference is the translation of carrier sets. EventB2Java translates them as set of integers, hence it does not capture the essence of carrier sets. We translate carrier sets as user defined datatype.

The paper is structured as follows. Section 1. presents the needed background. Section 2. defines the translation rules whilst Section 3. describes their implementation as a Rodin plug-in. Section 4. shows some evaluations of the plug-in. Finally, Section 5. is devoted for conclusions, outlining potential improvements and directions for future work.

1. Preliminaries

1.1. Event-B

Event-B is a formal modelling language for reactive systems, introduced by Abrial [1], which allows the modelling of complete systems. Figure 1 shows the general view of an Event-B machine and context. Event-B models are composed of contexts and machines. Contexts define constants (written after constant in context C), uninterpreted sets (written after set in context C) and their properties (written after axioms in context C). Machines define variables (written after variables in machine M) and their properties (expressed as invariants after invariant in machine M), and state transitions expressed as events (written between events and the last end). The initialisation event gives initial values to variables.

machine M sees C
variables v
invariants $label\_inv: \ I(s,c,v)$
events
  event $initialisation$
   then $A(s,c,v)$ end
  event evt
   any x
  where
   $label\_guard: \ G(s,c,v,x)$
  then
   $label\_action: \ A(s,c,v,x)$
  end
end

Context C
constant c
set S
axioms $X(s,c)$
end

Figure 1. General view of an Event-B machine and its context

An event is composed of guards and actions. The guard (written between keywords where and then) represents conditions that must hold for the event to be triggered. The action (written between keywords then and end) gives new values to variables

In Event-B, systems are modelled via a sequence of refinements. First, an abstract machine is developed and verified to satisfy whatever correctness and safety properties are desired. Refinement machines are used to add more detail to the abstract machine until

the model is sufficiently concrete for hand or automated translation to code. Refinement proof obligations are discharged to ensure that each refinement is a faithful model of the previous machine, so that all machines satisfy the correctness properties of the original.

1.2. Eiffel

Eiffel is an Object-Oriented programming language that natively supports the Design-by-Contract methodology. The behaviour of classes is specified by equipping them with contracts. Each routine of the class contains a pre- and post-condition: a client of a routine needs to guarantee the pre-condition on routine call. In return, the post-condition of the procedure, on routine exit, holds. The class is also equipped with class invariants. Invariants maintain the consistency of objects. Contracts in Eiffel follow a similar semantics of Hoare Triples.

Figure 2 depicts an Eiffel class that implements part of a Bank Account. The name of the class is `ACCOUNT` and it appears right after the keyword **class**. In Eiffel, implementers need to list creation procedures after the keyword **create**.

```
class ACCOUNT create make
feature -- Initalisation
  make -- Initialise an empty account.
    do
      balance := 0
    ensure
      balance_set: balance = 0
    end
feature -- Access
  balance: INTEGER -- Balance of this account.
feature -- Element change
  withdraw (amount: INTEGER) -- Withdraw 'amount' from this account.
    require
      amount_not_negative: amount >= 0
      amount_available: amount <= balance
    do
      balance := balance - amount
    ensure
      balance_set: balance = old balance - amount
    end
invariant balance_not_negative: balance >= 0
end
```

Figure 2. Eiffel class

In Figure 2, `make` is a procedure of the class that can be used as a creation procedure. Class `ACCOUNT` structures its procedures in `Initialisation`, `Access` and `Element change`, by using the keyword **feature**. This structure can be use for information hiding (not discussed here). `balance` is a class attribute that contains the actual balance of

the account. It is defined as an integer. Procedures in Eiffel are defined by given them a name (e.g. `withdraw`) and its respective arguments. It is followed by a head comment (which is optional). Procedures are equipped with pre- and post-conditions predicates. In Eiffel, a predicate is composed of a tag (optional) and a boolean expression. For instance, the pre-condition for `withdraw` (after the key work **require**) imposes the restriction on callers to provide and argument that is greater than or equal zero and less than or equal the balance of the account (amount_not_negative and amount_available are tags, identifiers, and are optionals). If the pre-condition of the procedure is met, the post-condition (after the key work **ensure**) holds on procedure exit. In a post-condition, the aid **old** refers to the value of an expression on procedure entry. The actions of the procedure are listed in between the key words **do** and **ensure**. The only action of `withdraw` procedure is to increase the value of `balance` by `amount`. Finally, The invariant is restricting the possible values for variables.

2. Translation

The translation is done by the aid δ : `Event-B` $\rightarrow$ `Eiffel`. δ takes an Event-B model and produces Eiffel classes. It is defined as a total function (i.e. $\rightarrow$) since any Event-B model can be translated to Eiffel. It uses two helpers: ξ translates Event-B Expressions or Predicates to Eiffel, and τ translates the type of Event-B variable to the corresponding type in Eiffel.

$$\frac{\tau(v) = \texttt{Type} \quad \xi(I(s,c,v)) = \texttt{Inv} \quad \delta(\mathsf{events}\ e) = \texttt{E} \qquad \delta(\mathsf{event}\ \mathit{initialisation}\ \mathsf{then}\ A(s,c,v)\ \mathsf{end}) = \texttt{Init}}{\begin{array}{l}\delta(\mathsf{machine}\ M\ \mathsf{sees}\ C \\ \quad\quad \mathsf{variables}\ v \\ \quad\quad \mathsf{invariants}\ \mathit{label\_inv}:\ I(s,c,v) \\ \quad\quad \mathsf{event}\ \mathit{initialisation}\ \mathsf{then}\ A(s,c,v)\ \mathsf{end} \\ \quad\quad \mathsf{events}\ e \\ \quad \mathsf{end}) = \\ \textbf{class}\ \texttt{M}\ \textbf{create}\ \texttt{initialisation} \\ \textbf{feature}\ --\ \mathsf{Initialisation} \\ \quad \texttt{Init} \\ \textbf{feature}\ --\ \mathsf{Events} \\ \quad \texttt{E} \\ \textbf{feature}\ --\ \mathsf{Access} \\ \quad \texttt{ctx : CONSTANTS} \\ \quad \texttt{v : Type} \\ \textbf{invariant} \\ \quad \texttt{label\_inv: Inv} \\ \textbf{end}\end{array}}\ \texttt{(machine)}$$

Figure 3. `machine` rule

Rule `machine` in Figure 3 is a high level translation. It takes an Event-B machine M and produces an Eiffel class `M`.

Variables are translated as class attributes in class `M`. Event-B invariants are translated to Eiffel invariants. Both, Event-B and Eiffel, have similar semantics for invariants. Rule `context` generate an Eiffel class `CONSTANT` that contains the translation of Event-B constants and carrier sets defined by the user. Axioms, which restrict the possible values for constants are translated to invariants of this class. Constants in Event-B are entities that cannot change their values. They are naturally translated to Eiffel as **once** variables.

$$\frac{\delta(\text{axioms } X(s,c)) = \texttt{X} \qquad \tau(c) = \texttt{Type}}{\delta(\text{Context } C \quad \text{constant } c \quad \text{set } S \quad \text{Axioms } X(s,c) \quad \text{end}) =} \ \texttt{(context)}$$

```
class CONSTANTS
feature -- Constants
   c : Type
         -- 'c' comment
      once
         create Type Result
      end
invariant
   X
end
```

Carrier sets represent a new type defined by the user. Each carrier set is translated as an afresh Eiffel class so users are able to use them as types. Rule `cset` shows the translation. Parts of the class are omitted due to space. Class `EBSET [T]` gives an implementation to sets of type `T`. Class `S` inherits `EBSET [T]` due to the nature of carrier sets in Event-B.

$$\frac{\tau(s) = \texttt{Type}}{\delta(\text{Context } C \quad \text{constant } c \quad \text{set } S \quad \text{Axioms } X(s,c) \quad \text{end}) =} \ \text{(cset)}$$

```
class S
inherit
   EBSET [Type]
...
end
```

Rule `event` produces an Eiffel feature given an Event-B *event*. Parameters of the event are translated as arguments of the respective feature in Eiffel with its respective type. In Event-B, an event might be executed only if the guard is true. In Eiffel, the guard is translated as the precondition of the feature. Hence, the client is now in charge of meeting the specification before calling the feature. The semantics of the execution is handle now by the client who wants to execute the feature rather than the system deciding. The actual execution of the actions still preserve its semantics: execution of the actions is only possible if the guard is true. In Eiffel, for a client to execute a feature he needs to meet the guard otherwise a runtime exception will be raised: Contract violation.

Event-B event actions are translated directly to Eiffel statements. In Event-B, the before-after predicate contains primed and unprimed variables representing the before and after value of the variables. We translated the primed variable with the Eiffel key word **old**. Representing old value of the variable. For simplicity. the rule only takes into account a single parameter, a single guard and a single action. However, this can be easily extended.

$$\frac{\xi(G(s,c,v,x)) = \texttt{G} \quad \xi(A(s,c,v,x)) = \texttt{A} \qquad \tau(x) = \texttt{Type}}{\delta(\text{event } evt \text{ any x where } label\_guard: G(s,c,v,x) \text{ then } label\_action: A(s,c,v,x) \text{ end}) =} \text{(event)}$$

```
evt(x : Type)
      -- 'evt' comment
   require
      label_guard: G
   do
      v.assigns(A)
   ensure
      label_action: v.equals(old A)
   end
```

Rule `init` below shows the translation of Event-B event *initialisation* to a creation procedure in Eiffel. The creation procedure initialises the object containing the constants definition. It also assigns initial values to variables taken from the initialisation in the *initialisation* event. In Eiffel, creation procedures are listed under the keyword **create**, as shown in rule `machine`. The **ensure** clause shows the translation of the before-after predicate of the assignment in Event-B.

$$\frac{\xi(A(s,c,v)) = \texttt{A}}{\delta(\textsf{event } initialisation \quad \textsf{then} \quad label: A(s,c,v) \quad \textsf{end}) =} \texttt{(init)}$$

```
initialisation
      -- evt comment
  do
    create ctx
    v.assigns(A)
  ensure
    label: v.is_equal(old A)
  end
```

3. Implementation

Rules presented in previous sections are implemented as a Rodin plug-in. Machines are translated to Eiffel classes. Event-B Events and variables are features and variables of the translated class, respectively. Guards and actions are translated as preconditions and body of the features. Contexts are being translated as class constants. The translation takes advantage of the embedded Design-by-Contract mechanism defined in Eiffel, for instance, Machine invariants are naturally translated to class invariants.

Another part of the implementation required translation of the mathematical language of Event-B. There are in total 90 symbols denoting different mathematical formulas. Some of them exist natively in Eiffel while others had to be implemented.

The implementation was done in several steps: first the structure for the Rodin IDE plug-in was set-up, then the information about the model was retrieved from the database and, finally, the resulting elements were translated into Eiffel.

3.1. Event-B and Rodin Structure

The plug-in's functionality is realized based on extensions and extension points that define the point of contact between different programs. The package responsible for Rodin extensions is `org.rodinp.core`. It provides an interface via which different extensions can communicate and provide extension points.

Event-B package – `org.eventb.core.ast` – provides an Abstract Syntax Tree (AST) of the system modeled in Rodin. This is a tree representation of the syntactic structure of an Event-B model. A visitor has been implemented to traverse it. It translates the mathematical notation into Eiffel code step by step.

3.2. The General Structure of the Tool

The packages included in the tool are `plug-in` and `rodinDB` (implementation of the tool can be found in [11]). The `plug-in` package contains `GenCodeEiffel.java` that is the entry point (defines the order of the translation), and `Translator.java` that implements a Visitor (`ISimpleVisitor2`) that parses the formulas and traverses the AST. The `rodinDB` package contains `RodinDBElements.java` that deals with retrieving information from the Rodin Database.

Each Rodin project has a set of children of class `IRodinElement` which also have `IInternalElements` in them. Depending on what type the internal elements is, it is possible to retrieve the information regarding machines and contexts from Rodin database. The package includes 14 methods that handle this task.

3.3. Traversing the Abstract Syntax Tree

Rodin provides the Abstract Syntax Tree (AST) of an Event-B model. It is then necessary to implement a Visitor to traverse the AST to translate the model parts into Eiffel.

The next step after retrieving information from the Rodin database is to parse the received formulas and to organize them in a way suitable for Eiffel translation. The wrapper methods dealing with parsing are created for almost each method from the `rodinDB` package. The visitor implements *parsePredicate()*, that parses an Event-B predicate, *parseExpression()* that parses an Event-B expression and *parseAssignment()* that parses assignments.

As parameters they take a String-representation of a formula and launch the pass through the AST. The methods in `ISimpleVisitor2` are overridden to handle visits of different branches of the tree such as Atomic Expressions (covers standalone integers, natural numbers, empty sets, booleans and others), Binary Predicates (implication and equality), Becomes Equal To (assignment to a variable or a parameter), Associative Expressions (unions, intersections, backward and forward compositions, addition, multiplication) and many others. The full set of types is described in [1].

3.4. Translating into Eiffel

As each AST branch for a specific formula is visited, Eiffel code is generated and added to the `eiffelCode` Array List that aggregates the translation and then returns it to the code generator (`GenCodeEiffel.java`).

An excerpt of the visitor is shown in Figure 4. A free identifier is any variable from a machine or an event. First the method checks whether the translation is done to retrieve types or to translate event or machine parameters. Depending on the result, code is added to different places.

```
Override
public void visitFreeIdentifier (FreeIdentifier identifierExpression){
 if (eiffelType != null){
  eiffelType.add (identifierExpression.getName());
 }
  eiffelCode.add (identifierExpression.getName());
 }
}
```

Figure 4. Free Identifier Translation

4. Evaluation

Several Event-B models were used in the testing phase of the plug-in. This phase also captures how much and accurate (as without compilation errors in Eiffel) of the Event-B mathematical language gets translated. Table 1 shows the results for all the tested models: first column is the Event-B model taken from the literature; the second column indicates how much code is translated in terms of the language syntax. For instance, 94% of the `MIO` model is being translated, meaning the model is using Event-B syntax (6%) that cannot be translated due to the limitations of the tool (discussed later); and the third column is how much code results in compilation errors in Eiffel.

Table 1. Summary of the results

Model	Translated	Compiled
Social Event Planner	100%	98%
MIO	94%	88%
Binary Search	100%	92%
Linear Search	100%	100%
Reversing the Array	78%	78%
Sorting the Array	100%	96%
Finding a minimum	94%	94%
Square root	100%	100%

Social Event Planner: This model was used for testing during the implementation stage. This model is described in more detail in [14]. It is a model for planing social events. The functionality includes creating events, inviting people to them and setting up permissions for inviting other people. This model defines its own sets ($PERSON$ and $CONTENTS$). They are translated as Eiffel user-defined classes.

An example Event-B event `create_account` is shown on Figure 5. ANY declares parameters of the event, WHERE denotes guards (necessaries conditions to hold for the

event to be triggered) of the event, THEN are the actions of the event. Figure 6 depicts the output of the plug-in. The output is a translation of Event-B event (in Figure 5) into Eiffel. There are two **require** statements corresponding to two guards (from Event-B) that ensure that the variables belong to the sets they are supposed to. The **do** statements assign translated expressions to the variables `contents, persons, owner and pages`, initializing them with an appropriate type. In Eiffel, **create** s a keyword used to create instances a classes, similar to **new** in Java or C++.

Class `EBSET` is a class created specifically for the translation of sets from Event-B. It inherits most of Eiffel set's functionality but allows more flexibility. It is part of `eb_math_lang` package that also includes natural numbers (`EBNAT`), integers, ranges and relations.

ANY
$c1$
$p1$
WHERE
grd1 : $p1 \in PERSON \backslash persons$
grd2 : $c1 \in CONTENTS \backslash contents$
THEN
act1 : $contents := contents \cup \{c1\}$
act2 : $persons := persons \cup \{p1\}$
act3 : $owner := owner \cup \{c1 \mapsto p1\}$
act4 : $pages := pages \cup \{c1 \mapsto p1\}$
END

Figure 5. *create_ account* event

```
create_account (p1: PERSON; c1: CONTENTS)
   require
     grd1: PERSON.difference (persons).has (p1)
     grd2: CONTENTS.difference (contents).has (c1)
   do
    contents.assigns ((contents).union (create {EBSET[CONTENTS]}.singleton
(c1)))
     persons.assigns ((persons).union (create { EBSET[PERSON]}.singleton
(p1)))
      owner.assigns ((owner).union (create  {EBREL[CONTENTS,PERSON]}.vals
(<<(create { EBPAIR[CONTENTS,PERSON]}. make (c1, p1))>>))
      pages.assigns ((pages).union (create  {EBREL[CONTENTS,PERSON]}.vals
(<<(create {EBPAIR[CONTENTS,PERSON]}. make(c1, p1))>>))
   end
```

Figure 6. Eiffel Code for *create_ account* event

MIO – Bus Transportation System: This model includes several entities: buses, bus stations, people, doors and sensors. It regulates bus transportation and is described in [3]. There are six refinements with each adding new entities and concepts. It is an extensive model with 21 actions in the initialisation event and 15 invariants for the last refinement.

Binary and Linear Search: These two models represented *binary* and *linear* search algorithms. The former one looks for a number, dividing each subsection into two, and the latter goes through a set of numbers in a linear way.

The binary search included three events apart from the initialisation (`inc`[rement], `dec`[rement] and `found`). The model for linear search included a similar `found` event as well as `progress` event that continued to search for the number if the correct one is not found.

In general, the tool is able to translate 86% of the 90 symbols that can be used with Event-B. Although the remaining 14% is not implemented yet, it is noted that they occur rarely in the models.

A common problem for most models (e.g. Social Event Planner, MIO, Binary Search and Sorting the Array) that results in compilation errors is due to the type translation of variables. For instance, the Event-B assignment $owner := owner \cup \{cmt \mapsto owner(rc)\}$ with a set extension ({}) including more than two expressions poses a problem as the types of the sets should be included into the Eiffel code before visiting the variable in the tree. Therefore, instead of a type, Java's `null` keyword is returned (as the tool is written in Java and for Eiffel this is an undefined keyword). This problem occurs for set extensions (declaring a set between two curly brackets), backward and forward composition of functions.

If the correct types are included into the Eiffel code manually, the compilation returns no errors. As Table 1 shows, these types of errors are not very common, e.g. for Social Event Planner there are only three cases of such statements, for MIO, Binary Search and Sorting the Array there is only one.

Other models (namely, reversing an array and finding a minimum) require those parts of the mathematical language that have not been implemented (Bound Declarations and Bound Identifiers that are used as variables in Quantified Predicates – $\forall$ and $\exists$). This is where most of their errors occur.

The list of Event-B models used in this phase and their translation to Eiffel using the plug-in can be found in [11].

5. Conclusion

We presented a series of rules to transform an Event-B model to an Eiffel program. The translation takes full advantage of all elements in the source by translating them as contracts in the target language. Thus, no information on the behaviour of the system is lost. These rules shows a methodology for software construction that makes use of two different approaches. We also presented a Rodin plug-in that implements the translation. The plug-in enables users to take advantages of Event-B (e.g. refinement) to then translation to Eiffel, which provides an actual implementation of the model, to take advantages of

the language (e.g. Design by Contract). The main limitation of the plug-in is that no proof of soundness has been carried out. This paper shows a proof-of-concept and opens up a direction to carry out with the proof.

In order to be able to fully automate the translation, type retrieval for variables that are further down the AST need to be implemented. This corresponds to 14% of the Event-B mathematical language. One of the challenges is to translate choice from set $(x :\in S)$ that arbitrarily chooses a value from the set S and choice by predicate $(z :| P)$ that arbitrarily chooses values for the variable in z that satisfy the predicate P. For this, we plan to make use of ProB or Constraint Programming to assign values that satisfy a predicate. Another direction is related to the translation of Proof Obligations (POs), mathematical formulas, automatically generated by Rodin, to Eiffel. POs need to be proven in order to ensure that a machine is correct [5]. They can be proved either automatically or interactively. By translating Proof Obligations into Specification Drivers [9] it will be possible to formally verify the translated Eiffel code against its contracts.

Formal modelling belongs to the technical domain to specify functional requirements. In order to prove correctness of system properties users need to be properly trained and understand technical aspects of formal models or, at least, of programming languages. In order to bridge the gap between business perspective and technical perspective, it would be necessary to provide a more comprehensive modelling framework that is left for future investigation [17].

References

[1] Abrial J.-R., *Modeling in Event-B: System and Software Engineering*, Cambridge University Press, New York, 2010.

[2] Cataño N., Rivera V., "EventB2Java: A Code Generator for Event-B", *NASA Formal Methods. NFM 2016*, Lecture Notes in Computer Science, **9690**, Springer, Cham, 2016, 166–171.

[3] Cataño N., Rueda C., "Teaching formal methods for the unconquered territory", *Teaching Formal Methods. TFM 2009*, Lecture Notes in Computer Science, **5846**, Springer, Berlin, Heidelberg, 2009, 2–19.

[4] Edmunds A., Butler M., "Tool support for Event-B code generation", *Workshop on Tool Building in Formal Methods*, Wiley and Sons, Quebec, Canada, 2010.

[5] Hallerstede S., "On the purpose of Event-B proof obligations", *Abstract State Machines, B and Z*, Lecture Notes in Computer Science, **5238**, Springer, Berlin, Heidelberg, 2008, 125–138.

[6] Mazzara M., "Deriving specifications of dependable systems: toward a method", *12th European Workshop on Dependable Computing (EWDC)*, 2009.

[7] Méry D., Singh N. K., "Automatic code generation from Event-B models", *Proceedings of the Second Symposium on Information and Communication Technology, SoICT '11*, ACM, New York, 2011, 179–188.

[8] Meyer B., "Applying "design by contract"", *Computer*, **25**:10 (1992), 40–51.

[9] Naumchev A., Meyer B., "Complete contracts through specification drivers", *CoRR*, 2016, abs/1602.04007.

[10] Naumchev A., Meyer B., Rivera V., "Unifying requirements and code: An example", *Perspectives of System Informatics – 10th International Andrei Ershov Informatics Conference, PSI 2015*, Revised Selected Papers, Kazan and Innopolis, Russia, 2015, 233–244.

[11] Reznikova S., *Innopolis thesis*, 2018, https://github.com/sonyareznikova/InnopolisThesis.

[12] Rivera V., Bhattacharya S., Cataño N., "Undertaking the tokeneer challenge in Event-B", *2016 IEEE/ACM 4th FME Workshop on Formal Methods in Software Engineering*, 2016, 8–14.

[13] Rivera V., Cataño N., "Translating Event-B to JML-specified Java programs", *Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC'14*, ACM, New York, 2014, 1264–1271.

[14] Rivera V., Cataño N., Wahls T., Rueda C., "Code generation for Event-B", *Int. J. Softw. Tools Technol. Transf.*, **19**:1 (2017), 31–52.

[15] Rivera V., Lee J. Y., Mazzara M., "Mapping Event-B machines into Eiffel programming language", *Proceedings of 6th International Conference in Software Engineering for Defence Applications – SEDA 2018*, Rome, Italy, 2018.

[16] Tschannen J. et al., "AutoProof: Autoactive functional verification of object-oriented programs", *Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2015*, Lecture Notes in Computer Science, **9035**, Springer, Berlin, Heidelberg, 2015, 566–580.

[17] Yan Z. et al., "BPMO: semantic business process modeling and WSMO extension", *IEEE International Conference on Web Services (ICWS 2007)*, 2007, 1185–1186.

Резникова С., Ривера В., Ли Д. Й., Маццара М., "Перевод моделей Event-B в Eiffel", *Моделирование и анализ информационных систем*, **25**:6 (2018), 623–636.

DOI: 10.18255/1818-1015-2018-6-623-636

Аннотация. Формальные языки моделирования играют важную роль в разработке программного обеспечения, так как позволяют пользователям, во-первых, определять функциональные требования, которые также служат документацией для проекта, а во-вторых, доказывать корректность свойств систем, что особенно важно для критических систем. Однако не существует четкого понимания того, как сопоставить формальную модель и определенный язык программирования. В качестве решения данной проблемы авторы статьи предлагают использовать возможность source-to-source соответствия между моделями, описанными на языке Event-B (языке моделирования для реактивных приложений и систем), и программами на объектно-ориентированном языке программирования Eiffel. Предложенное решение не только автоматически генерирует соответствующий модели на Event-B код на Eiffel, но также переводит свойства модели в виде контрактов. Контракты соответствуют принципу Design-by-Contract и нативно поддерживаются в Eiffel. Реализация решения доступна как плагин EB2Eiffel в Rodin (среде разработки для Event-B). Таким образом, пользователи могут разрабатывать различные системы, начиная с моделирования функциональных требований (свойств) в Event-B, затем формально доказывая корректность этих свойств в Rodin и, наконец, используя EB2Eiffel для перевода модели на язык программирования. Используя Eiffel, пользователи могут расширять и модифицировать реализацию модели и доказывать корректность измененной модели относительно ее оригинальной, изначально переведенной версии. Также в статье описан процесс тестирования EB2Eiffel разными моделями, написанными на Event-B, и представлены ограничения плагина. Статья публикуется в авторской редакции.

Ключевые слова: пошаговое улучшение систем, Design-by-Contract, формальное моделирование, реактивные приложения, Event-B, Eiffel

Об авторах:
Резникова Софья, orcid.org/0000-0003-4616-2729, студент-бакалавр, Университет Иннополис, ул. Университетская, 1, Иннополис 420500, Россия, email: s.reznikova@innopolis.ru

Ривера Виктор, orcid.org/0000-0002-1946-8979, доцент, Университет Иннополис, ул. Университетская, 1, Иннополис 420500, Россия, email: v.rivera@innopolis.ru

Ли Джу Йонг, orcid.org/0000-0001-5421-730X, доцент, Университет Иннополис, ул. Университетская, 1, Иннополис 420500, Россия, email: j.lee@innopolis.ru

Маццара Мануэль, orcid.org/0000-0002-3860-4948, профессор, Университет Иннополис, ул. Университетская, 1, Иннополис 420500, Россия, email: m.mazzara@innopolis.ru

Моделирование и анализ информационных систем. Т. 25, № 6 (2018), с. 637–666
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 637–666

DOI: 10.18255/1818-1015-2018-6-637-666

УДК 004.052

Платформенно-независимая спецификация и верификация стандартной математической функции квадратного корня

Шилов Н. В.[1], **Кондратьев Д. А., Ануреев И. С., Бодин Е. В., Промский А. В.**

Поступила в редакцию 10 сентября 2018
После доработки 15 октября 2018
Принята к публикации 10 ноября 2018

Аннотация. Цель проекта "Платформенно-независимый подход к формальной спецификации и верификации стандартных математических функций" — инкрементальный комбинированный подход к спецификации и верификации стандартных математических функций, таких как `sqrt`, `cos`, `sin` и так далее. Платформенно-независимый подход предполагает простую аксиоматизацию машинной арифметики в терминах вещественной арифметики (то есть арифметики поля $\mathbb{R}$ вещественных чисел), не фиксируя ни основание системы счисления, ни формат машинного слова. Инкрементальность означает, что спецификация и верификация начинается с рассмотрения наиболее "простого" случая – элементарной спецификации и верификации простого алгоритма, работающего с вещественными числами, а заканчивается модификацией элементарной спецификации и алгоритма для машинной арифметики и верификацией алгоритма, работающего в машинной арифметике. А комбинированность подхода означает, что мы начинаем с рассмотрения "базового случая" — "ручной" верификации (с ручкой и бумагой) для алгоритма, работающего в вещественной арифметике, затем выполняем ручную верификацию алгоритма, работающего в машинной арифметике, используя верификацию для базового случая в качестве "конспекта" (proof-outlines), а заканчиваем — верификацией с использованием автоматизированной системы построения/поиска доказательства для того, чтобы исключить апелляцию к "очевидности" в ручной верификации. В статье платформенно-независимый инкрементальный комбинированный подход применяется для спецификации и верификации стандартной математической функции квадратного корня. В настоящий момент автоматизированная верификация разработанных алгоритмов выполнена только частично: с использованием системы ACL2 доказана реализуемость (существование) чисел с фиксированной запятой и таблицы начальных приближений квадратного корня.

Ключевые слова: числа с фиксированной запятой, числа с плавающей запятой, машинная арифметика, формальная верификация, частичная и тотальная корректность, тройки Хоара, метод Флойда, точная функция, квадратный корень, метод Ньютона, справочная таблица

Для цитирования: Шилов Н. В., Кондратьев Д. А., Ануреев И. С., Бодин Е. В., Промский А. В., "Платформенно-независимая спецификация и верификация стандартной математической функции квадратного корня", *Моделирование и анализ информационных систем*, **25**:6 (2018), 637–666.

Об авторах:

Шилов Николай Вячеславович, orcid.org/0000-0001-7515-9647, канд. физ.-мат. наук, доцент
Автономная некоммерческая организация высшего образования "Университет Иннополис",
ул. Университетская, 1, г. Иннополис, Республика Татарстан, 420500, Россия, e-mail: shiloviis@mail.ru

Кондратьев Дмитрий Александрович, orcid.org/0000-0002-9387-6735, аспирант
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: apple-66@mail.ru

Ануреев Игорь Сергеевич, orcid.org/0000-0001-9574-128X, канд. физ.-мат. наук, ст. науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: anureev@iis.nsk.su

Бодин Евгений Викторович, orcid.org/0000-0002-5882-0365, науч. сотр.
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: bodin@iis.nsk.su

Промский Алексей Владимирович, orcid.org/0000-0002-5963-2390, канд. физ.-мат. наук, ученый секретарь
Институт систем информатики имени А.П. Ершова СО РАН,
пр. акад. Лаврентьева, 6, г. Новосибирск, 630090 Россия, e-mail: promsky@iis.nsk.su

Благодарности:
[1] Работа поддержана грантом РФФИ 17-01-00789.

Введение

В нашей статье речь пойдёт о, так сказать, “верификации в малом”, то есть о верификации отдельных небольших программ и функций (в нашем случае, стандартных математических функций вообще и функции квадратного корня в частности). Но наряду с “верификацией в малом” существует и “верификация в большом” — верификация сложных и больших комплексов программ, критических с точки зрения безопасности или миссии, как, например, программ управления космическим полётом, когда цена однократной программной ошибки может достигать миллиардов рублей.

Так, в декабре 2017 г. “Роскосмос” опубликовал [32] официальные результаты расследования неудачного пуска с космодрома Восточный 28 ноября того же года ракеты-носителя “Союз-2.1б” с космическим аппаратом “Метеор-М” и ещё 18 космическими аппаратами, в результате которого все 19 аппаратов были потеряны. Риски при запуске были застрахованы на сумму 2,6 млрд руб. В официальном расследовании значится следующее:

> *Сложилось такое сочетание параметров стартового стола космодрома, азимутов полета ракеты-носителя и разгонного блока, которое не встречалось ранее. Соответственно, оно не было выявлено при проведённой наземной отработке баллистической траектории согласно действующим методикам.*
>
> *Проведя всесторонний анализ, члены комиссии считают, что проявление этой некорректности алгоритма могло и не произойти при запуске с космодрома Восточный этой же полезной нагрузки с этим же разгонным блоком на этой же ракете. Пуск прошёл бы штатно, например, летом, либо в случае, если бы районы падения отделяемых частей РН лежали в стороне от выбранных.*

Однако верификация в малом так же имеет важное научное, прикладное и экономическое значение, так как “маленькие” ошибки в “маленьких”, но “массовых” (часто используемых) программах могут приводить к тем же миллиардным потерям из-за частого и повсеместного использования таких программ (стандартных функций, в частности) [11].

В настоящей статье платформенно-независимый инкрементальный комбинированный подход применяется для спецификации и верификации стандартной математической функции квадратного корня. Для вычислений в арифметике с фиксированной запятой мы выбрали основанный на методе Ньютона адаптивный алгоритм со справочной (look-up) таблицей начальных приближений квадратного корня; для спецификации этого алгоритма мы используем утверждения тотальной корректности Хоара; для аргументов, больших 1, мы доказываем методом Флойда его завершаемость в машинной арифметике с округлением к ближайшему представимому числу с ошибкой округления не более $\frac{1}{2}ULP$, где ULP — единица наименьшей точности (*Unit of Least Precision* или *Unit in the Last Place*), и с общей ошибкой вычислений не более $\varepsilon + 2ULP$, где $\varepsilon > 0$ - желаемая точность, задаваемая пользователем. Для вычислений квадратного корня в арифметике с плавающей запятой мы используем верифицированный алгоритм для вычисления квадратного корня из "приведённой мантиссы" (к значению большему 1) и деление на 2 (для чётных показателей экспоненты). Корректность алгоритма для арифметики с плавающей запятой следует из корректности алгоритма для арифметики с фиксированной запятой. В качестве автоматизированной системы формализации и поиска доказательства в нашем проекте на данном этапе используется ACL2. В настоящий момент автоматизированная верификация разработанных алгоритмов выполнена только частично: с использованием системы ACL2 доказана реализуемость (существование) чисел с фиксированной запятой и таблицы начальных приближений квадратного корня.

Прежде чем переходить к основному содержанию статьи, становимся на её структуре. В разделе 1.1. будут описаны метод Ньютона применительно к вычислению аппроксимаций квадратного корня, его алгоритмическая реализация в арифметике вещественных чисел, его спецификация условиями тотальной корректности (тройкой Хоара) и ручная верификация частичной корректности (методом Флойда для чисел, больших 1). Затем в разделе 1.2. будет доказана завершаемость нашего алгоритма в арифметике вещественных чисел и установлена квадратичная скорость сходимости этого алгоритма (при некоторых ограничениях на начальные приближения). В разделе 1.3. конкретизируется метод вычисления начальных приближений квадратного корня в арифметике вещественных чисел посредством справочной таблицы (look-up table), удовлетворяющей всем ограничениям для начальных приближений (выявленным в предыдущем разделе 1.2.), доказывается существование такой таблицы и корректность алгоритма с такой таблицей.

Вторая часть посвящена разработке (прототипированию) алгоритма аппроксимации квадратного корня и его спецификации в арифметике с фиксированной запятой и состоит из 5 разделов. В разделе 2.1. мы описываем (аксиоматизируем) нашу версию платформенно-независимой арифметики с фиксированной запятой в терминах вещественной арифметики. В разделе 2.2. мы модифицируем для арифметики с фиксированной запятой алгоритм (со справочной таблицей) аппроксимации квадратного корня и его спецификацию из раздела 1.3. первой части, доказываем существование справочной таблицы начальных приближений квадратного корня в арифметике с фиксированной запятой (при условии, что $ULP < \frac{1}{12}$). Раздел 2.3. посвящён анализу ошибок за одну итерацию цикла алгоритма аппроксимации квадратного корня в арифметике с фиксированной запятой и за несколько итераций; в этом разделе, в частности, доказывается, что при некоторых предположениях

"суммарная" (накопленная за несколько итераций) ошибка округлений не превосходит $2ULP$. В разделе 2.4. мы анализируем условия, гарантирующие завершаемость нашего алгоритма в арифметике с фиксированной запятой, а в разделе 2.5. — варианты модификации прототипа постусловия, первоначально предложенные в разделе 2.1.

Третья часть статьи посвящена спецификации и верификации алгоритмов аппроксимации квадратного корня в машинной арифметике как с фиксированной запятой, так и плавающей запятой. В разделе 3.1. мы (на основе прототипов и анализа, выполненных во второй части) приводим окончательный вариант алгоритма аппроксимации квадратного корня в арифметике с фиксированной запятой, его окончательную спецификацию, затем аннотируем этот алгоритм инвариантом цикла, а в разделе 3.2. — доказываем тотальную корректность этого полностью аннотированного алгоритма. Раздел 3.3. начинается с описания некоторых минимальных предположений об арифметике с плавающей запятой (в терминах арифметики с фиксированной запятой и вещественной арифметики) и посвящён разработке, спецификации (условиями тотальной корректности) и верификации алгоритма аппроксимации квадратного корня в арифметике с плавающей запятой.

В заключительной, четвертой части мы даём краткую сводку полученных результатов и обсуждаем их место среди исследований по верификации программного обеспечения (раздел 4.1.), даём достаточно подробный обзор литературы по верификации стандартных математических функций вообще и квадратного корня в частности, по формализации машинной арифметики (раздел 4.2.), а также намечаем планы дальнейших исследований в рамках нашего проекта "*Платформенно-независимый подход к формальной спецификации и верификации стандартных математических функций*" (раздел 4.3.).

1. Алгоритмы аппроксимации квадратного корня в вещественной арифметике, их спецификация и верификация

1.1. Метод Ньютона, его алгоритмическая реализация, спецификация и частичная корректность

Для вычисления приближённых значений квадратного корня из аргумента $y \geq 0$ широко используется *метод Ньютона* [2,7,31]. Математическое описание этого метода дано на Рис. 1.

Разумеется, что нас не интересует бесконечный процесс вычисления методом Ньютона новых приближений для $\sqrt{y}$, поэтому введем ещё один параметр $\varepsilon > 0$ для контроля точности вычислений и прерывания процесса при достижении желаемой точности. Понятие желаемой точности может быть математически формализовано двумя способами:

- x_i отличается от $\sqrt{y}$ не более чем на ε, то есть *ошибка* $|\sqrt{y} - x_i| \leq \varepsilon$;
- квадрат x_i отличается от y не более чем на ε, то есть *невязка* $|y - x_i^2| \leq \varepsilon$.

Пусть $x_1 = ini(y) > 0$ — произвольное начальное приближение для $\sqrt{y}$;
для любого $i > 0$ если i-ое приближение x_i для $\sqrt{y}$ уже определено,
то пусть $d_i = \frac{y-x_i^2}{2x_i}$ и $x_{i+1} = x_i + d_i$ (то есть $x_{i+1} = \frac{y+x_i^2}{2x_i}$).

Let $x_1 = ini(y) > 0$ be an arbitrary initial approximation for $\sqrt{y}$;
for all $i > 0$ if i-th approximation x_i for $\sqrt{y}$ has been already defined,
then let $d_i = \frac{y-x_i^2}{2x_i}$, and $x_{i+1} = x_i + d_i$ (i.e. $x_{i+1} = \frac{y+x_i^2}{2x_i}$).

Рис. 1. Математическое описание метода Ньютона
Fig. 1. Mathematical description of Newton (Newton-Raphson) method

К сожалению, контроль ошибки невозможен во время вычислений по методу Ньютона (если мы не знаем точного значения квадратного корня). Наоборот, контроль невязки легко осуществим во время вычислений по методу Ньютона. Однако, так как нас интересует вычисление квадратного корня, то по завершении вычислений мы хотели бы гарантировать именно то, что ошибка не превосходит ε. Поэтому возникает естественная идея контролировать абсолютное значение слагаемого d_i и попытаться доказать, что как только абсолютное значение мало, то ошибка не превосходит ε.

В результате мы приходим к представленной на Рис. 2 алгоритмической реализации $ANIR$ (*Adaptive Newton In Reals*) метода Ньютона для вычисления приближённого значения квадратного корня в идеальной арифметике (все операции и значения — в поле $\mathbb{R}$).

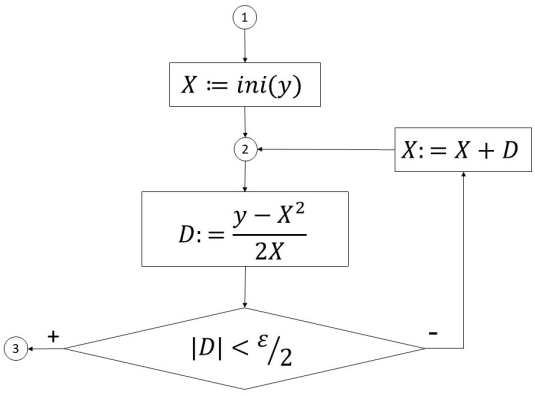


Рис. 2. Блок-схема алгоритма $ANIR$
Fig. 2. The flowchart of the algorithm $ANIR$

Этот и последующие алгоритмы вычисления приближённых значений квадратного корня мы будем применять только к аргументам $y > 1$. Поэтому специфици-

руем алгоритм следующим условием тотальной корректности Хоара:

$$[1 < y \ \& \ 0 < \varepsilon \ \& \ \sqrt{y} \leq ini(y)] \ ANIR \ [|\sqrt{y} - X| \leq \varepsilon]. \tag{1}$$

Доказательство условия (1) разобьем на доказательство утверждения частичной корректности этого же алгоритма с теми же предусловием и постусловием и доказательство завершаемости этого алгоритма, если выполнено предусловие. Доказательство частичной корректности рассмотрим в этом разделе, а доказательство завершаемости — в разделе 1.2.

Для доказательства частичной корректности выберем контрольные точки 1, 2 и 3 так, как показано на Рис. 2, аннотируем точки 1 и 3 предусловием и постусловием из (1), а в качестве инварианта (аннотации) точки 2 возьмём предусловие, усиленное условием $\sqrt{y} \leq X$. Используя метод Флойда доказательства частичной корректности, надо доказать, что

- если в контрольной точке 1 выполнена её аннотация (то есть предусловие) и путь (1..2) завершается, то после выполнения этого пути в контрольной точке 2 выполнена её аннотация (то есть инвариант);
- если в контрольной точке 2 выполнена её аннотация (то есть инвариант) и путь $(2-2)$ завершается, то после выполнения этого пути в контрольной точке 2 выполнена её аннотация (то есть тот же инвариант, но при новых значениях переменных);
- если в контрольной точке 2 выполнена её аннотация (то есть инвариант) и путь $(2+3)$ завершается, то после выполнения этого пути в контрольной точке 3 выполнена её аннотация (то есть постусловие).

Здесь и далее мы используем следующую нотацию для пути по блок-схеме алгоритма между парой контрольных точек i и j: путь заключается в круглые скобки ( ), начинается с указания начальной контрольной точки i, заканчивается указанием конечной контрольной точки j, а между ними указываются знаки рёбер, исходящих из условных операторов на этом пути, по которым прошёл путь, или две точки "..", если путь не содержит условных операторов.

Доказательство всех трёх перечисленных путей носит довольно-таки рутинный характер и было ранее представлено в [26].

1.2. Тотальная корректность алгоритма метода Ньютона

В предыдущем разделе 1.1. мы доказали частичную корректность, остаётся доказать завершаемость в случае, когда выполнено предусловие. Первый участок алгоритма (1..2) выполняется только один раз и поэтому всегда завершается. Пусть, как и в описании метода Ньютона, x_1, x_2, ... x_n, $x_{(n+1)}$, ... — значения переменной X непосредственно перед первой, второй, ... n-й, $(n+1)$-й и так далее итерациями цикла $(2-2)$, и, кроме того, d_1, d_2, ... d_n, $d_{(n+1)}$, ... — значения переменной D непосредственно сразу после её вычисления на первой, второй, ... n-й, $(n+1)$-й и так далее итерациях этого цикла. В частности, $x_1 = ini(y)$ и $d_n = \frac{y - x_n^2}{2x_n}$, $x_{(n+1)} = x_n + d_n$ для всех $n > 0$.

Выразим $d_{(n+1)}$ через d_n:

$$d_{(n+1)} = \frac{y-x_{(n+1)}^2}{2x_{(n+1)}} = \frac{y-(x_n+d_n)^2}{2(x_n+d_n)} = \frac{y-(\frac{y+x_n^2}{2x_n})^2}{2\frac{y+x_n^2}{2x_n}} = -\frac{(y-x_n^2)^2 x_n}{4x_n^2(y+x_n^2)} = -\frac{d_n^2 x_n}{y+x_n^2} = -\frac{d_n^2}{2x_{(n+1)}}.$$

Заметим, что, в силу инварианта цикла, все значения d_1, d_2, … d_n, $d_{(n+1)}$, … не положительные. Следовательно,

$$\frac{|d_{(n+1)}|}{|d_n|} = \frac{d_{(n+1)}}{d_n} = -\frac{d_n}{2x_{(n+1)}} = \frac{1}{2} \times \frac{x_n^2 - y}{x_n^2 + y} < \frac{1}{2},$$

то есть $|d_{(n+1)}| < \frac{d_1}{2^n}$. Так как $|D| < \frac{\varepsilon}{2}$ — условие завершения алгоритма $ANIR$, то этот алгоритм завершается, выполнив не более $2 + \log_2 \frac{d_1}{\varepsilon}$ итераций цикла $(2-2)$. Кроме того, мы имеем

$$|d_1| = \frac{x_1^2 - y}{2x_1} = (x_1 - \sqrt{y}) \times \frac{x_1 + \sqrt{y}}{2x_1} < x_1 - \sqrt{y} = ini(y) - \sqrt{y}.$$

Поэтому алгоритм $ANIR$ завершается, выполнив не более

$$2 + \log_2 \frac{ini(y) - \sqrt{y}}{\varepsilon} \tag{2}$$

итераций этого цикла. В частности, если начальное приближение “с точностью до целых”, то $(ini(y) - \sqrt{y}) \leq 1$, и поэтому число итераций цикла не более $(2 - \log_2 \varepsilon)$.

Таким образом, утверждение тотальной корректности (1) полностью доказано.

Можно, однако, обратить внимание на то, что в инварианте алгоритма $ANIR$ речь идет об оценке изменения значения переменной, а в постусловии алгоритма — об ошибке, то есть о значении разности $(X - \sqrt{y})$ после завершения алгоритма. Поэтому возникает идея перейти в инварианте от оценки этого изменения значения переменной к оценке значения самой разности $(X - \sqrt{y})$ перед очередной итерацией.

Для того чтобы найти подходящее условие для инварианта, использующее оценку разности $(X - \sqrt{y})$, проанализируем, как эта оценка меняется за одну итерацию тела цикла $(2-2)$ в алгоритме $ANIR$: пусть a — значение переменной X в начале этого пути, а b — значение этой переменной после завершения этого пути; тогда $b - \sqrt{y} = \frac{y+a^2}{2a} - \sqrt{y} = \frac{(a-\sqrt{y})^2}{2a}$. Так как начальная ошибка (то есть перед первой итерацией тела цикла) — это $(ini(y) - \sqrt{y})$, то, следовательно, в качестве оценки сверху ошибки $(X - \sqrt{y})$ после $m \geq 0$ итераций тела этого цикла можно принять

$$\frac{(ini(y) - \sqrt{y})^{2^m}}{(2\sqrt{y})^m} \tag{3}$$

(которую можно доказать индукцией по m).

Если в предусловие добавить дополнительное ограничение, что $(ini(y) - \sqrt{y}) \leq \frac{1}{2}$, и памятуя, что $1 < \sqrt{y}$, оценку сверху (3) ошибки $(X - \sqrt{y})$ после $m \geq 0$ итераций тела цикла $(2-2)$ можно упростить:

$$\frac{(ini(y) - \sqrt{y})^{2^m}}{(2\sqrt{y})^m} \leq \frac{1}{2^{(2^m+m)}}. \tag{4}$$

Так как при разработке алгоритма $ANIR$ мы воспользовались дополнительным предположением, что $(ini(y) - \sqrt{y}) \leq \frac{1}{2}$, то спецификация этого алгоритма отличается предусловием от спецификации алгоритма $ANIR$ (1) и может быть представлена следующим утверждением тотальной корректности:

$$[1 < y \ \&\ \&\ 0 < \varepsilon \ \&\ \sqrt{y} \leq ini(y) \leq \sqrt{y} + \frac{1}{2}]\ ANIR\ [|\sqrt{y} - X| \leq \varepsilon]. \tag{5}$$

Оценка (4) говорит, что достаточно "очень небольшого" числа итераций цикла $(2-2)$ для достижения желаемой точности ε приближённого значения квадратного корня $\sqrt{y}$. Действительно, пусть, как и выше, x_1, x_2, $\ldots$ x_n, x_{n+1}, $\ldots$ — значения переменной X непосредственно перед первой, второй, $\ldots$ n-й, $(n+1)$-й и так далее итерациями цикла $(2-2)$, а d_1, d_2, $\ldots$ d_n, d_{n+1}, $\ldots$ — значения переменной D непосредственно сразу после её вычисления на первой, второй, $\ldots$ n-й, , $(n+1)$-й и так далее итерациях этого цикла. Выберем любое $n > 0$ такое, что $\frac{1}{2^{(2^n+n)}} \leq \frac{\varepsilon}{4}$. В силу оценки сверху (4) ошибки $|X - \sqrt{y}|$, верной для любого $m \geq 0$ числа итераций тела цикла $(2-2)$, имеем

$$|d_n| = |x_{n+1} - x_n| = \left|(x_{n+1} - \sqrt{y}) \ + \ (x_n - \sqrt{y})\right| \leq$$
$$\leq |x_{n+1} - \sqrt{y}| \ + \ |x_n - \sqrt{y}| \leq \frac{\varepsilon}{4} + \frac{\varepsilon}{4} = \frac{\varepsilon}{2}.$$

Но, как показано выше, при верификации пути $(2+3)$, как только $|D| < \frac{\varepsilon}{2}$, то ошибка $|X - \sqrt{y}| < \varepsilon$, то есть $|x_n - \sqrt{y}| < \varepsilon$. Поэтому при том же предположении, что $(ini(y) - \sqrt{y}) \leq \frac{1}{2}$, алгоритм $ANIR$ выполнит не более $\log_2\left(1 + |\log_2 \varepsilon|\right)$ итераций цикла $(2-2)$. В качестве иллюстрации скорости $ANIR$ приведём простой пример: если для вычисления $\sqrt{2}$ в качестве начального приближения выбрать $1,5$ (обратите внимание, что $y = 2 > 1$ и $\sqrt{y} < ini(y) = 1,5 \leq \sqrt{y} + \frac{1}{2}$), то

- после $n = 0$ итераций тела цикла $(2-2)$ (то есть перед началом цикла) мы точно знаем $m = (2^n + n) - 1 = 0$ двоичных цифр $\sqrt{2}$ после запятой,
- после $n = 1$ итераций тела этого цикла мы точно знаем $m = (2^n + n) - 1 = 2$ двоичных цифр $\sqrt{2}$ после запятой,
- после $n = 2$ итераций тела этого цикла мы точно знаем $m = (2^n + n) - 1 = 5$ двоичных цифр $\sqrt{2}$ после запятой,
- после $n = 3$ итераций тела этого цикла мы точно знаем $m = (2^n + n) - 1 = 10$ двоичных цифр $\sqrt{2}$ после запятой,
- и так далее.

Заметим, что именно поэтому в нескольких алгоритмах вычисления квадратного корня для компьютерных игр [7] использована явная развертка цикла $(2-2)$ на несколько итераций (на 2–4).

1.3. Адаптивный алгоритм со справочными таблицами, его спецификация и верификация

Конкретизируем способ (метод) $X := ini(y)$ реализации вычисления начального приближения для квадратного корня, использованный в алгоритме $ANIR$, в виде присваивания

$$X := UpRoot\left[SelroundUp(y)\right],$$

где

- $SelRoundUp : \mathbb{R}^+ \to Sel \subseteq \mathbb{R}^+$ — функция из положительных вещественных чисел в некоторое фиксированное множество "избранных" положительных вещественных чисел Sel, которая для каждого вещественного числа $x > 0$ возвращает некоторое число из Sel, которое не меньше, чем x;
- $UpRoot : Sel \to \mathbb{R}$ — предвычисленная справочная таблица (look-up table, "массив") приближений с избытком квадратных корней из избранных вещественных чисел.

В результате алгоритм $ANIR$ (Рис. 2) заменяется алгоритмом $LANIR$ (*Look-up table Adaptive Newton In Reals*), представленным на Рис. 3.

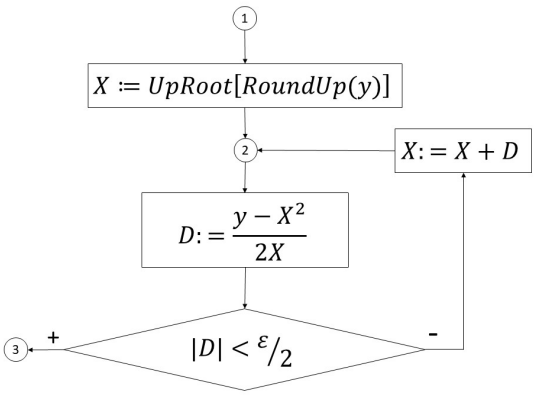


Рис. 3. Блок-схема алгоритма $LANIR$
Fig. 3. The flowchart of the algorithm $LANIR$

Специфицируем этот алгоритм следующим образом:

$$\left[1 < y \;\&\; 0 < \varepsilon \;\&\; \sqrt{y} \le UpRoot\left[SelRoundUp(y)\right] \le \sqrt{y} + \tfrac{1}{2}\right] \; LANIR \; \left[|\sqrt{y} - X| \le \varepsilon\right]. \tag{6}$$

Заметим, что корректность спецификации (6) следует из корректности спецификации (5). При этом, однако, необходимо показать, что существуют (могут быть реализованы) функция округления $SelRoundUp$ и справочная таблица $UpRoot$, которые "делают" предусловие в (6) истинным, и, следовательно, алгоритм $LANIR$ завершает работу, после чего выполнено постусловие спецификации (6).

Так как в данной части мы "работаем" с вещественной арифметикой, то функция округления и справочная таблица могут быть реализованы, например, следующим образом[1]:

1. $SelRoundUp : \mathbb{R} \to \mathbb{R}$ — посредством операции $\lambda\ z \in \mathbb{R}.\lceil z \rceil$ округления до ближайшего сверху целого числа (то есть мы принимаем $Sel = \mathbb{N}$);
2. $UpRoot : \mathbb{N}^+ \to \mathbb{R}$ — таблично-заданная функция $\lambda\ z \in \mathbb{R}^+.\sqrt{z}$, составляющая каждому положительному натуральному числу (вещественный) квадратный корень из этого числа.

Покажем, что $\sqrt{y} \le UpRoot\,[SelRoundUp(y)] \le \sqrt{y} + \frac{1}{2}$ для любого $y > 1$. Действительно, так как $UpRoot[SelRoundUp(y)] = \sqrt{\lceil y \rceil}$, а $\lceil y \rceil \le (y+1)$, то

$$0 \le UpRoot[SelRoundUp(y)] - \sqrt{y} \le \sqrt{y+1} - \sqrt{y} = \frac{(y+1)-y}{\sqrt{y+1}+\sqrt{y}} < \frac{1}{2}.$$

Таким образом, мы показали, что для арифметики вещественных чисел существуют функция округления $SelRoundUp$ и справочная таблица $UpRoot$, которые делают предусловие спецификации (6) истинным; следовательно, при использовании этих функций и таблиц алгоритм $LANIR$ завершает работу, после чего выполнено постусловие спецификации (6).

2. Разработка и спецификация прототипа алгоритма аппроксимации квадратного корня в машинной арифметике с фиксированной запятой

2.1. Машинная арифметика с фиксированной запятой

Не все вещественные числа представимы в компьютере. Более точно, представимы только некоторые рациональные числа. Для представления существуют два формата: с фиксированной запятой и с плавающей запятой. Все рациональные числа, представимые в формате с фиксированной запятой, будем называть *числами с фиксированной запятой*, а все рациональные числа, представимые в формате с плавающей запятой, — *числами с плавающей запятой*. К формализации чисел с плавающей запятой мы вернёмся позже, а сейчас рассмотрим формализацию чисел с фиксированной запятой.

Мы определим числа с фиксированной запятой как тип данных $\mathbb{D}$, удовлетворяющий следующим аксиомам:

- множество значений этого типа $Val_{\mathbb{D}}$ — это некоторое конечное подмножество рациональных (и, следовательно, вещественных) чисел $\mathbb{Q}$ такое, что
  - оно содержит наименьшее $\inf_{\mathbb{D}} < 0$ и наибольшее $\sup_{\mathbb{D}} > 0$ числа,

[1] Здесь и далее $\lambda\ z \in \mathbb{R}.\lceil z \rceil$ — операции округления до ближайшего сверху целого числа.

  - а также все числа из диапазона $[\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$ с шагом $\delta_{\mathbb{D}} > 0$,
  - и включает все целые числа $Int_{\mathbb{D}}$ из этого диапазона $[\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$;

- допустимые операции со значениями этого типа — это
  - машинное сложение $\oplus$ и вычитание $\ominus$. Если результат математического сложения (вычитания) принадлежит диапазону $[\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$, то результат машинного сложения (соответственно — машинного вычитания) совпадает с результатом математических операций (и в этих случаях сами операции могут обозначаться обычным образом как $+$ and $-$);
  - машинное умножение $\otimes$ и машинное деление $\oslash$. Эти операции возвращают округлённое значение соответствующих математических операций с округлением к ближайшему числу с фиксированной запятой, причём для любых $x, y \in Val_{\mathbb{D}}$
    - если $x \times y \in Val_{\mathbb{D}}$, то $x \otimes y = x \times y$;
    - если $x/y \in Val_{\mathbb{D}}$, то $x \oslash y = x/y$;
    - если $x \times y \in [\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$, то $|x \otimes y - x \times y| \le \delta_{\mathbb{D}}/2$;
    - если $x/y \in [\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$, то $|x \oslash y - x/y| \le \delta_{\mathbb{D}}/2$;
  - округление вверх $\lceil\ \rceil$ и вниз $\lfloor\ \rfloor$ до ближайшего целого. Обе операции являются всюду определёнными на $Val_{\mathbb{D}}$;

- допустимые бинарные отношения со значениями этого типа — все стандартные равенства и неравенства в рамках диапазона $[\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$ (и поэтому обозначаются обычным образом как $=$, $\neq$, $\le$, $\ge$, $<$, $>$).

Легко видеть, что

- если $\inf_{\mathbb{D}} \le -1$ или $1 \le \sup_{\mathbb{D}}$, то $\frac{1}{\delta_{\mathbb{D}}} \in \mathbb{N}$;
- $Val_{\mathbb{D}} = \{k\delta_{\mathbb{D}} \ :\ k \in \mathbb{Z},\ \inf_{\mathbb{D}} \le k\delta_{\mathbb{D}} \le \sup_{\mathbb{D}}\}$;
- для любых $n \in \mathbb{Z}$ и $v \in Val_{\mathbb{D}}$, если $(n \times v) \in [\inf_{\mathbb{D}}, \sup_{\mathbb{D}}]$, то $n \otimes v = n \times v$;
- так как операции $\lceil\ \rceil$ и $\lfloor\ \rfloor$ всюду определены, то $inf_{\mathbb{D}}, sup_{\mathbb{D}} \in Int_{\mathbb{D}}$.

Вариант реализации описанного выше типа данных (так сказать, виртуальная машина) доступен по ссылке (https://bitbucket.org/ainoneko/lib_verify/src/).

2.2. Прототип алгоритма и спецификации в арифметике с фиксированной запятой

При переходе от “идеальной” арифметики вещественных чисел к арифметике с фиксированной запятой алгоритм $LANIR$ с Рис. 3 должен измениться, так как теперь вместо обычных операций сложения, вычитания, умножения и деления над вещественными числами $\mathbb{R}$ используются операции $\oplus$, $\ominus$, $\otimes$ и $\oslash$. На Рис. 4 представлен вариант такого изменения — алгоритм $LANIFAv1$ (*Look-up table Adaptive Newton In Fix-point Arithmetic, version 1*). В этом изменённом алгоритме

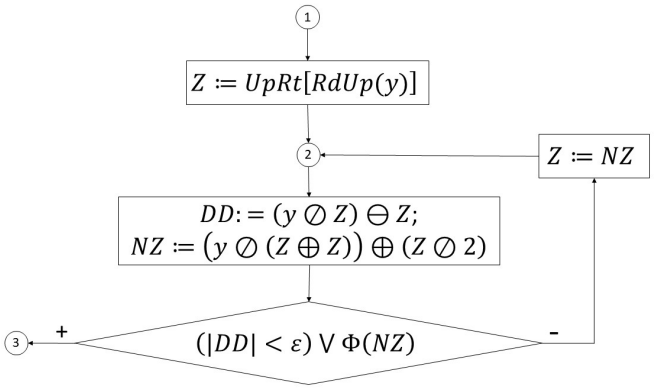


Рис. 4. Блок-схема алгоритма $LANIFAv1$
Fig. 4. The flowchart of the algorithm $LANIFAv1$

- вместо переменной X алгоритма $LANIR$ используется другая переменная Z (чтобы не возникало путаницы, да и типы этих переменных разные: вещественный $\mathbb{R}$ у X и с фиксированной запятой $\mathbb{D}$ у Z);
- присваивание $D := \frac{y-X^2}{2X}$ и следующая за ним проверка $|D| < \frac{\varepsilon}{2}$ должны были бы "превратиться" (поскольку $\frac{y-X^2}{2X} = \frac{y}{2X} - \frac{X}{2}$) в присваивание $D := (y \oslash (Z \oplus Z)) \ominus (Z \oslash 2)$ и проверку $|D| < (\varepsilon \oslash 2)$, но превращаются в присваивание $DD := (y \oslash Z) \ominus Z$ для вычисления удвоенного значения и проверку $(|DD| < \varepsilon)\&\Phi(NZ)$, где $\Phi(NZ)$ — дополнительная проверка (которую тоже ещё предстоит определить) на следующее значение переменной Z, вычисленное и сохранённое в новой переменной NZ;
- присваивание $X := X + D$ превращается (так как $D = \frac{y-X^2}{2X} = \frac{y}{2X} + \frac{X}{2}$) в два присваивания $NZ := \left(y \oslash (Z \oplus Z)\right) \oplus (Z \oslash 2)$ для вычисления следующего значения переменной Z и сохранения его в переменной NZ ($Next\ Z$) и $Z := NZ$.

Кроме того, в этом алгоритме вместо бесконечной таблицы $UpRoot : Sel \to \mathbb{R}$, которая (как было показано выше в разделе 1.3.) может "хранить" график функции $\lambda\ z \in \mathbb{R}^+.\sqrt{z}$, используется "настоящий" (конечный) предвычисленный массив $UpRt : Int^+_{\mathbb{D}} \to Val_{\mathbb{D}}$ для начальных приближений квадратного корня.

Нам бы хотелось специфицировать алгоритм $LANIFAv1$ по аналогии со спецификацией (6) алгоритма $LANIR$, например так:

$$\left[1 < y\ \&\ 0 < \varepsilon\ \&\ \sqrt{y} \le UpRt\left[\lceil y \rceil\right] \le \sqrt{y} + \tfrac{1}{2}\right] \quad LANIFAv1 \quad \left[|\sqrt{y} - Z| \le g(\varepsilon)\right], \tag{7}$$

где $g : Val_{\mathbb{D}} \to \mathbb{R}$ — некоторая функция (которую ещё предстоит определить), но, однако, эта спецификация (7) нуждается в модификации и предусловия, и постусловия.

Начнём с модификации предусловия спецификации (7). Во-первых, совершенно естественно, что два первых условия $1 < y$ и $0 < \varepsilon$ в предусловии при переходе к

арифметике с фиксированной запятой должны быть дополнены условиями $y \in Val_{\mathbb{D}}$ и $\varepsilon \in Val_{\mathbb{D}}$, означающими, что входные данные представимы в машине. Во-вторых, совершенно не очевидно, существует ли такой массив $UpRt : Int^{+}_{\mathbb{D}} \to Val_{\mathbb{D}}$, который удовлетворяет третьему условию $\sqrt{y} \le UpRt\big[\lceil y \rceil\big] \le \sqrt{y} + \frac{1}{2}$ из предусловия: если такого массива вообще не существует, то доказывать спецификацию (7) не нужно — она верна, так как из предусловия $FALSE$ следует и завершаемость любого алгоритма, и истинность любого постусловия. Однако в следующем абзаце будет предложено простое достаточное условие на $\delta_{\mathbb{D}}$, гарантирующее существование такого массива.

Достаточным условием для существования массива $UpRt : Int^{+}_{\mathbb{D}} \to Val_{\mathbb{D}}$, удовлетворяющего третьему условию из предусловия спецификации (7), является условие

$$\delta_{\mathbb{D}} \le \frac{1}{12}. \tag{8}$$

Действительно, если $\delta_{\mathbb{D}} \le \frac{1}{12}$, то для любого $m \ge 1$, $m, (m+1) \in Int_{\mathbb{D}}$, имеем

$$\begin{aligned} \tfrac{1}{2} \ge\ \delta_{\mathbb{D}} + \tfrac{1}{2{,}4}\ \ge (\text{так как } \sqrt{m} \ge 1 \text{ и } \sqrt{m+1} \ge \sqrt{2} > 1{,}4)\ \delta_{\mathbb{D}} + \tfrac{1}{\sqrt{m+1}+\sqrt{m}}\ = \\ =\ \delta_{\mathbb{D}} + \tfrac{(m+1)-m}{\sqrt{m+1}+\sqrt{m}}\ =\ \delta_{\mathbb{D}} + (\sqrt{m+1} - \sqrt{m}), \end{aligned}$$

то есть $(\sqrt{m} + \frac{1}{2}) - \sqrt{m+1} \ge \delta_{\mathbb{D}}$. Следовательно, интервал $[\sqrt{m+1},\ (\sqrt{m} + \frac{1}{2})]$ "шире" $\delta_{\mathbb{D}}$ и поэтому содержит некоторое число a_{m+1} вида $k\delta_{\mathbb{D}}$, которое мы примем в качестве $UpRt[m+1]$: $UpRt : m \mapsto a_m$. Для такого массива $UpRt$ и любого $y > 1$ имеем

$$\sqrt{y} \le \sqrt{\lceil y \rceil} = \sqrt{\lfloor y \rfloor + 1} \le a_{\lfloor y \rfloor + 1} = a_{\lceil y \rceil} = UpLd[\lceil y \rceil] \le \sqrt{\lfloor y \rfloor} + \frac{1}{2} \le \sqrt{y} + \frac{1}{2},$$

что и требовалось доказать для массива $UpRt$.

Доказательство существования типа данных $\mathbb{D}$ и массива $UpRt$ выполнено с использованием системы ACL2 и доступно на Github по ссылке https://github.com/apple2-66/c-light/tree/master/experiments/square-root.

2.3. Ошибки вычислений прототипа алгоритма в арифметике с фиксированной запятой

Проанализируем, как меняется ошибка $|Z - \sqrt{y}|$ за одну итерацию цикла $(2-2)$ в алгоритме $LANIFAv1$. Выберем произвольное $n > 0$, и пусть z_n и z_{n+1} — значения переменной Z непосредственно перед и сразу после n-й итерации этого цикла, а $\Delta_n = z_n - \sqrt{y}$ и $\Delta_{n+1} = z_{n+1} - \sqrt{y}$. Мы выбрали обозначения z_n и z_{n+1}, Δ_n и Δ_{n+1} чтобы не путать эти значения с соответствующими значениями x_n и x_{n+1}, d_n и d_{n+1} переменной X непосредственно перед и сразу после n-й итерации цикла $(2-2)$ в алгоритме $ANIR$ в предыдущей части 1. Имеем

$$\begin{aligned} \Delta_{n+1} = z_{n+1} - \sqrt{y} = \Big(\big(y \oslash (z_n \oplus z_n)\big) \oplus (z_n \oslash 2)\Big) - \sqrt{y}\ = \\ (\text{если } \big(y \oslash (z_n \oplus z_n)\big) \oplus (z_n \oslash 2) = \big(y \oslash (z_n \oplus z_n)\big) + (z_n \oslash 2)) \\ =\ \Big(\big(y \oslash (z_n \oplus z_n)\big) + (z_n \oslash 2)\Big) - \sqrt{y}\ = \end{aligned}$$

$$(\text{если } z_n \oplus z_n = 2z_n) = \Big(\big(y \oslash (2z_n)\big) + (z_n \oslash 2)\Big) - \sqrt{y} =$$
$$= \big((\tfrac{y}{2z_n} + \delta') + (\tfrac{z_n}{2} + \delta'')\big) - \sqrt{y} \text{ (где } |\delta'| \le \tfrac{\delta_{\mathbb{D}}}{2} \text{ и } |\delta''| \le \tfrac{\delta_{\mathbb{D}}}{2}) =$$
$$= \big((\tfrac{y}{2(\sqrt{y}+\Delta_n)} + \tfrac{\sqrt{y}+\Delta_n}{2}) + \delta\big) - \sqrt{y} \text{ (где } \delta = \delta' + \delta'' \text{ и } |\delta| \le \delta_{\mathbb{D}}) =$$
$$= \big(\tfrac{y}{2(\sqrt{y}+\Delta_n)} - \tfrac{\sqrt{y}}{2}\big) + \tfrac{\Delta_n}{2} + \delta = \big(-\tfrac{\Delta_n\sqrt{y}}{2(\sqrt{y}+\Delta_n)} + \tfrac{\Delta_n}{2}\big) + \delta = \tfrac{\Delta_n^2}{2(\sqrt{y}+\Delta_n)} + \delta = \tfrac{\Delta_n^2}{2z_n} + \delta.$$

Заметим, что в приведённых выше выкладках мы использовали два предположения $z_n \oplus z_n = 2z_n$ и $\big(y \oslash (2z_n)\big) \oplus (z_n \oslash 2) = \big(y \oslash (2z_n)\big) + (z_n \oslash 2)$ о значении z_n, которые можно сформулировать в виде двух следующих условий на переменную Z (так как z_n — это значение переменной Z):

$$\begin{cases} Z \oplus Z = 2Z \\ \big(y \oslash (2Z)\big) \oplus (Z \oslash 2) = \big(y \oslash (2Z)\big) + (Z \oslash 2) \end{cases}. \tag{9}$$

Следовательно,

$$|\Delta_{n+1}| = \big|\tfrac{\Delta_n^2}{2z_n} + \delta\big| \le \tfrac{\Delta_n^2}{2z_n} + \delta_{\mathbb{D}} \le$$
$$(\text{если } 1 \le z_n) \le \tfrac{\Delta_n^2}{2} + \delta_{\mathbb{D}} < (\text{если } |\Delta_n| < 1) < \tfrac{|\Delta_n|}{2} + \delta_{\mathbb{D}},$$

то есть

$$\text{если } |\Delta_n| < 1 \text{ и } 1 \le z_n, \text{ то } |\Delta_{n+1}| < \frac{|\Delta_n|}{2} + \delta_{\mathbb{D}}. \tag{10}$$

Поэтому, если верно условие (8) и $|\Delta_n| < 1$, то $|\Delta_{n+1}| < 1$. В этих выкладках мы использовали предположение $z_n \ge 1$, которое можно сформулировать в виде следующего условия на переменную Z (так как z_n — это значение переменной Z):

$$Z \ge 1. \tag{11}$$

Далее, если $|\Delta_1| = |\sqrt{y} - z_1| < \frac{1}{2}$ и $z_1 \ge 1, \ \dots \ z_m \ge 1$, то в силу (10) имеем

$$|\Delta_{m+1}| < \tfrac{|\Delta_m|}{2} + \delta_{\mathbb{D}} < \tfrac{\frac{|\Delta_{m-1}|}{2}+\delta_{\mathbb{D}}}{2} + \delta_{\mathbb{D}} = \tfrac{|\Delta_{m-1}|}{4} + (\tfrac{\delta_{\mathbb{D}}}{2} + \delta_{\mathbb{D}}) <$$
$$< \tfrac{|\Delta_{m-2}|}{8} + (\tfrac{\delta_{\mathbb{D}}}{4} + \tfrac{\delta_{\mathbb{D}}}{2} + \delta_{\mathbb{D}}) < \dots < \tfrac{|\Delta_1|}{2^m} + \Sigma_{k=0}^{k=(m-1)} \tfrac{\delta_{\mathbb{D}}}{2^k} <$$
$$< \tfrac{|\Delta_1|}{2^m} + \Sigma_{k>0} \tfrac{\delta_{\mathbb{D}}}{2^k} = \tfrac{|\Delta_1|}{2^m} + 2\delta_{\mathbb{D}} < \tfrac{1}{2^{m+1}} + 2\delta_{\mathbb{D}},$$

то есть

$$\text{если } |\Delta_1| < \frac{1}{2} \text{ и } z_1 \ge 1, \ \dots \ z_m \ge 1, \text{ то } |\Delta_{m+1}| < \frac{1}{2^{m+1}} + 2\delta_{\mathbb{D}}. \tag{12}$$

2.4. Завершаемость работы прототипа алгоритма в арифметике с фиксированной запятой

Таким образом, из (12) следует, что алгоритму $LANIFAv1$ следует остановиться, как только нельзя продолжить "победную" серию $z_1 \ge 1, \ \dots \ z_m \ge 1$, то есть как только $z_{m+1} < 1$. Если $z_{m+1} < 1$, то мы не можем применить (10) для оценки ошибки на следующей $(m+1)$-й итерации цикла. Так как по смыслу z_m — это значение переменной Z, то z_{m+1} — это значение переменной NZ, поэтому в качестве "кандидата" для $\Phi(NZ)$ в условии выхода из цикла в алгоритме $LANIFAv1$ можно предложить условие

$$\Phi(NZ) \equiv (NZ < 1). \tag{13}$$

Важное следствие (12) состоит в том, что если $\varepsilon > 5\frac{1}{6}\delta_{\mathbb{D}}$, то цикл в алгоритме $LANIFAv1$ обязательно завершается. Действительно, этот цикл может завершиться

- или потому, что для некоторого $m \geq 0$ верно $z_1 \geq 1, \ldots\ z_m \geq 1$, но $z_{m+1} < 1$, и произойдёт выход по условию $\Phi(NZ)$,
- или потому, что на всех итерациях $z_1 \geq 1, \ldots\ z_m \geq 1, \ldots$, что гарантирует в конечном счёте выход по условию $|DD| < \varepsilon$.

Так как в первом из этих случаев завершение алгоритма очевидно, то сразу перейдём к рассмотрению второго случая. Выберем произвольное $m \geq 0$ такое, что $\frac{1}{2^m} \leq \delta_{\mathbb{D}}$. В силу (12) на m-й итерации цикла $(2-2)$ верно $|z_m - \sqrt{y}| \leq 3\delta_{\mathbb{D}}$ и, следовательно, мы имеем следующее:

$$
\begin{aligned}
|DD| = |(y \oslash z_m) \ominus z_m| = \\
\text{(если } (y \oslash z_m) \ominus z_m = (y \oslash z_m) - z_m) = |(y \oslash z_m) - z_m| = \\
= |(\tfrac{y}{z_m} + \delta) - z_m| \text{ (где } |\delta| \leq \tfrac{\delta_{\mathbb{D}}}{2}) = |\tfrac{y - z_m^2}{z_m} + \delta| \leq \\
\leq |\tfrac{y-z_m^2}{z_m}| + |\delta| = |\tfrac{(\sqrt{y}-z_m)(\sqrt{y}+z_m)}{z_m}| + |\delta| = \\
= |\Delta_m|\big(1 + \tfrac{\sqrt{y}}{z_m}\big) + |\delta| \leq |\Delta_m|\big(1 + \tfrac{\sqrt{y}}{z_m}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \leq \\
\text{(так как } |z_m - \sqrt{y}| \geq 3\delta_{\mathbb{D}})) \leq |\Delta_m|\big(1 + \tfrac{\sqrt{y}}{\sqrt{y} - 3\delta_{\mathbb{D}}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \leq \\
\leq |\Delta_m|\big(2 + \tfrac{3\delta_{\mathbb{D}}}{\sqrt{y} - 3\delta_{\mathbb{D}}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \leq \\
\text{(так как } \delta_{\mathbb{D}} \leq \tfrac{1}{12} \text{ и } 1 \leq \sqrt{y}) \leq |\Delta_m|\big(2 + \tfrac{\frac{3}{12}}{1 - \frac{3}{12}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} = \\
= \tfrac{7}{3}|\Delta_m| + \tfrac{\delta_{\mathbb{D}}}{2}.
\end{aligned}
$$

Заметим, что в этих выкладках мы использовали предположение о значении z_m, которое можно сформулировать в виде следующего условия на переменную Z (так как z_m — это опять значение именно переменной Z):

$$(y \oslash Z) \ominus Z = (y \oslash Z) - Z. \tag{14}$$

В силу свойства (12) мы имеем

$$|DD| \leq \frac{7}{3}\big(\frac{1}{2^m} + 2\delta_{\mathbb{D}}\big) + \frac{\delta_{\mathbb{D}}}{2} = \frac{7}{3} \times \frac{1}{2^m} + \frac{31}{6}\delta_{\mathbb{D}} == \frac{7}{3} \times \frac{1}{2^m} + 5\frac{1}{6}\delta_{\mathbb{D}};$$

поэтому для любого

$$\varepsilon > 5\frac{1}{6}\delta_{\mathbb{D}} \tag{15}$$

обязательно найдется такое число m, что $\big(5\frac{1}{6}\delta_{\mathbb{D}} + \frac{7}{3} \times \frac{1}{2^m}\big) < \varepsilon$; следовательно, после m итераций цикла $(2-2)$ в алгоритме $LANIFAv1$ обязательно будет выполнено условие $|DD| < \varepsilon$ завершения этого цикла, причём (забегая несколько вперёд — см. свойство (17)) после выхода из цикла будет верно условие $|\sqrt{y} - Z| < \varepsilon + \frac{\delta_{\mathbb{D}}}{2}$. Поэтому (на основании (10) и (15)) мы можем оценить ошибку вычислений следующим образом:

$$
\begin{aligned}
|\sqrt{y} - NZ| < \\
< \big(\tfrac{1}{2}|\sqrt{y} - NZ| + \delta_{\mathbb{D}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \ < \ \big(\tfrac{1}{2}\big(\varepsilon + \tfrac{1}{2}\delta_{\mathbb{D}}\big) + \delta_{\mathbb{D}}\big) + \tfrac{\delta_{\mathbb{D}}}{2} \ < \ \tfrac{1}{2}\varepsilon + \tfrac{7}{4}\delta_{\mathbb{D}} \ < \\
< \varepsilon + 2\delta_{\mathbb{D}};
\end{aligned}
$$

следовательно, окончательно мы получаем оценку для ошибки вычислений

$$|\sqrt{y} - NZ| < \frac{1}{2}\varepsilon + \frac{7}{4}\delta_{\mathbb{D}} < \varepsilon + 2\delta_{\mathbb{D}}. \tag{16}$$

2.5. Анализ постусловия для прототипа алгоритма в арифметике с фиксированной запятой

Теперь проанализируем, как зависит оценка ошибки $|\sqrt{y} - Z| < g(\varepsilon)$ в постусловии спецификации (7) от того, каким образом была завершена работа алгоритма $LANIFAv1$ —

- или по условию $|DD| < \varepsilon$,
- или по условию $\Phi(NZ)$, то есть $NZ < 1$ (см. (13)).

Начнём со случая выхода по условию $|DD| < \varepsilon$. При доказательстве пути $(2+3)$ для алгоритма $ANIR$ в разделе 1.1. корректность постусловия $|\sqrt{y} - X| \leq \varepsilon$ была следствием истинности условия выхода из цикла и завершения алгоритма $|D| < \frac{\varepsilon}{2}$ (разумеется, при истинности инварианта). Поэтому попробуем применить такой же подход и к алгоритму $LANIFAv1$.

Пусть для любого положительного $\varepsilon \in Val_{\mathbb{D}}$

$$g(\varepsilon) = \varepsilon + \frac{\delta_{\mathbb{D}}}{2}, \tag{17}$$

а $z \in Val_{\mathbb{D}}$ — произвольное положительное значение переменной Z. Тогда из $\varepsilon > |DD|$ следует, что

$$\begin{aligned} g(\varepsilon) = \varepsilon + \tfrac{\delta_{\mathbb{D}}}{2} > |DD| + \tfrac{\delta_{\mathbb{D}}}{2} \geq \\ \geq |DD| + |\delta| \text{ (где } \delta = (y \oslash z) - \tfrac{y}{z} \text{ и } |\delta| \leq \tfrac{\delta_{\mathbb{D}}}{2}) \geq |DD - \delta| = \\ = \Big|\big((y \oslash z) \ominus z\big) - \delta\Big| = (\text{если } (y \oslash z) \ominus z = (y \oslash z) - z) = \Big|\big((y \oslash z) - z\big) - \delta\Big| = \\ \Big|\big((y \oslash z) - \delta\big) - z\Big| = \Big|\tfrac{y}{z} - z\Big| = \Big|\tfrac{y - z^2}{z}\Big| = \Big|\tfrac{(\sqrt{y} - z)(\sqrt{y} + z)}{z}\Big| = \\ = |\sqrt{y} - z| \times \tfrac{\sqrt{y} + z}{z}. \end{aligned}$$

С другой стороны, имеем $g(\varepsilon) \leq g(\varepsilon) \times \frac{\sqrt{y}+z}{z}$; поэтому из приведённой выше цепочки неравенств следует, что $g(\varepsilon) \times \frac{\sqrt{y}+z}{z} > |\sqrt{y} - z| \times \frac{\sqrt{y}+z}{z}$ и что (в случае, когда $z > 0$) $g(\varepsilon) > |\sqrt{y} - z|$. Но это ровно то свойство, которое мы хотим от функции g: при выходе из цикла и завершении алгоритма по условию $\varepsilon > |DD|$ обязательно выполнено постусловие $g(\varepsilon) > |\sqrt{y} - Z|$ и, следовательно, условие (16)

$$|\sqrt{y} - NZ| < \frac{1}{2}\varepsilon + \frac{7}{4}\delta_{\mathbb{D}} < \varepsilon + 2\delta_{\mathbb{D}}.$$

(Заметим, что в приведённом рассуждении мы использовали условие (14), а так же условие $z > 0$, являющееся следствием из более сильного условия (11).)

Теперь рассмотрим случай выхода по условию $\Phi(NZ)$ (то есть $NZ < 1$). Пусть $n > 0$ — номер итерации цикла $(2-2)$, во время выполнения которой произошёл выход из этого цикла по этому условию, и пусть $z_n \geq 1$ и $z_{n+1} < 1$ — значения

переменной Z непосредственно перед и сразу после n-ой итерации этого цикла. Очевидно, что z_{n+1} — это значение переменной NZ перед проверкой условия выхода из цикла. Имеем

$$\begin{aligned} z_{n+1} = \ & \big(y \oslash (z_n \oplus z_n)\big) \oplus (z_n \oslash 2) \ = \\ & (\text{если } \big(y \oslash (z_n \oplus z_n)\big) \oplus (z_n \oslash 2) = \big(y \oslash (z_n \oplus z_n)\big) + (z_n \oslash 2)) \\ = \ & \big(y \oslash (z_n \oplus z_n)\big) + (z_n \oslash 2) \ = \\ & (\text{если } z_n \oplus z_n = 2z_n) = \ \big(y \oslash (2z_n)\big) + (z_n \oslash 2) \ = \\ = \ & (\tfrac{y}{2z_n} + \delta') + (\tfrac{z_n}{2} + \delta'') \ (\text{где } |\delta'| \le \tfrac{\delta_{\mathbb{D}}}{2} \text{ и } |\delta''| \le \tfrac{\delta_{\mathbb{D}}}{2}) = \\ & = \ \tfrac{y}{2z_n} + \tfrac{z_n}{2} + \delta \ (\text{где } \delta = \delta' + \delta'' \text{ и } |\delta| \le \delta_{\mathbb{D}}) = \ \tfrac{y+z_n^2}{2z_n} + \delta. \end{aligned}$$

Заметим, что в этих преобразованиях мы опять использовали условия (9). Так как $z_{n+1} < 1$, то

$$\frac{y + z_n^2}{2z_n} + \delta < 1. \tag{18}$$

Рассмотрим два случая в зависимости от того, какое из следующих двух неравенств имеет место: или $z_n < \sqrt{y}$, или $z_n \ge \sqrt{y}$.

В первом из этих случаев (если $z_n < \sqrt{y}$) вычтем $z_n \ge 1$ из (18):

$$0 = 1 - 1 > \big(\frac{y + z_n^2}{2z_n} + \delta\big) - z_n = \frac{(\sqrt{y} - z_n)(\sqrt{y} + z_n)}{2z_n} + \delta.$$

Следовательно,

$$0 < |\sqrt{y} - z_n| = \sqrt{y} - z_n < (\sqrt{y} - z_n) \times \frac{\sqrt{y} + z_n}{2z_n} < -\delta \le \delta_{\mathbb{D}},$$

то есть в этом случае имеем $|\sqrt{y} - z_n| \le \delta_{\mathbb{D}}$ и, так как z_{n+1} — это значение NZ в момент выхода из цикла, то в силу (10) имеем

$$|\sqrt{y} - NZ| \le 1\frac{1}{2}\delta_{\mathbb{D}} < \frac{1}{2}\varepsilon + \frac{7}{4}\delta_{\mathbb{D}} < \varepsilon + 2\delta_{\mathbb{D}}. \tag{19}$$

Во втором из перечисленных выше случаев (если $z_n \ge \sqrt{y}$) давайте примем z_n как начальное приближение $x_1 = ini(y)$ для квадратного корня $\sqrt{y}$ в алгоритме $ANIR$ (Рис. 2). Тогда $x_2 = \frac{y+x_n^2}{2x_1} = \frac{y+z_n^2}{2z_n} = z_{n+1} - \delta$. Согласно доказательству корректности пути $(2-2)$ алгоритма $ANAIR$ (см. раздел 1.1.) $\sqrt{y} < x_2 < x_1$. Следовательно, $\sqrt{y} < (z_{n+1} - \delta) < z_n$. Но, так как $z_{n+1} < 1$ и $1 < \sqrt{y}$, то $z_{n+1} < \sqrt{y} < z_{n+1} - \delta$, то есть (так как z_{n+1} — это значение NZ в момент выхода из цикла) имеем

$$|\sqrt{y} - NZ| \le \delta_{\mathbb{D}} < \frac{1}{2}\varepsilon + \frac{7}{4}\delta_{\mathbb{D}} < \varepsilon + 2\delta_{\mathbb{D}}. \tag{20}$$

Подведём некоторый итог результатов, полученных в данной части 2.:

- в результате анализа прототипа алгоритма $LANIFAv1$ (Рис. 4) в разделе 2.4. мы пришли к выводу (см. (13)), что условие $\Phi(NZ)$ — это $NZ < 1$;
- обновлённая спецификация алгоритма должна учесть в предусловии все свойства, выявленные в части 2. при анализе прототипа спецификации 7.

3. Алгоритмы аппроксимации квадратного корня в машинной арифметике, их спецификация и верификация

3.1. От прототипа к аннотированному алгоритму в арифметике с фиксированной запятой

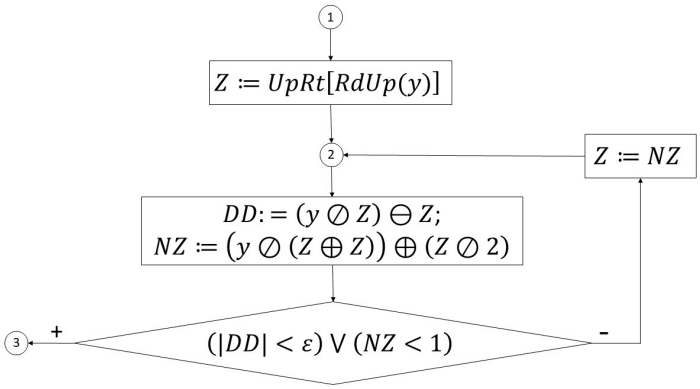


Рис. 5. Блок-схема алгоритма $LANIFAv2$
Fig. 5. The flowchart of the algorithm $LANIFAv2$

Окончательный вариант алгоритма $LANIFAv2$ (*Look-up table Adaptive Newton In Fix-point Arithmetic, version 2*, см. Рис. 5) аппроксимации квадратного корня в машинной арифметике с фиксированной запятой получается из алгоритма $LANIFAv1$ в результате подстановки $NZ < 1$ вместо $\Phi(NZ)$. А спецификация этого алгоритма $LANIFAv2$ получается из спецификации (7) прототипа алгоритма $LANIFAv1$ и выглядит следующим образом:

$$\left[\begin{array}{ll} (a) & 1 < y \in Val_{\mathbb{D}} \ \& \\ (b) & \delta_{\mathbb{D}} \le \frac{1}{12} \ \& \\ (c) & 5\frac{1}{6}\delta_{\mathbb{D}} < \varepsilon \in Val_{\mathbb{D}} \ \& \\ (d) & \sqrt{y} \le UpRt\left[\lceil y \rceil\right] \le \sqrt{y} + \frac{1}{2} \ \& \\ (e) & 2\left(\sqrt{y} + \frac{1}{2}\right) < \sup_{\mathbb{D}} - 2\delta_{\mathbb{D}} \end{array}\right] \quad LANIFAv2 \quad \left[|\sqrt{y} - NZ| \le \left(\varepsilon + 2\delta_{\mathbb{D}}\right)\right]. \tag{21}$$

Однако для верификации этой спецификации нам удобно “обогатить” алгоритм $LANIFAv2$ (Рис. 5) вспомогательной переменной M для счётчика итераций. Эта вспомогательная переменная никак не влияет на работу алгоритма, но нужна для аннотации контрольной точки 2 инвариантом цикла $(2-2)$. Соответствующий алгоритм $LANIFAv3$ представлен на Рис. 6.

Аннотируем контрольные точки алгоритма $LANIFAv3$ следующим образом: точки 1 и 3 — предусловием и постусловием в соответствии со спецификацией (21), а точку 2 — инвариантом, являющимся конъюнкцией предусловия спецификации (21) и трёх новых условий (f) $Z \ge 1$, (g) $|\sqrt{y} - Z| < \left(\frac{1}{2^M} + 2\delta_{\mathbb{D}}\right)$ и (h) $M > 0$.

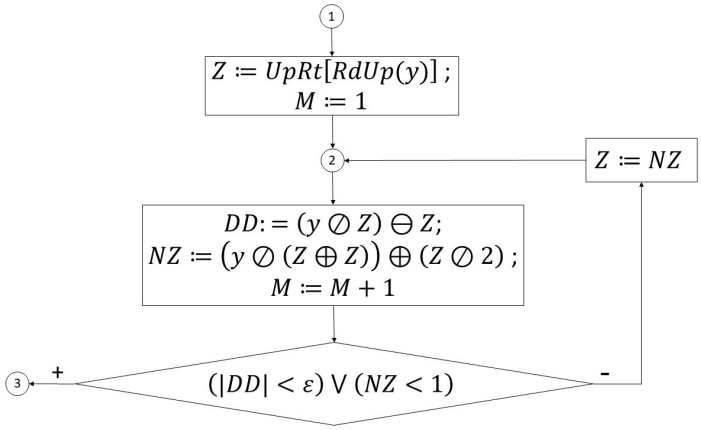


Рис. 6. Блок-схема алгоритма $LANIFAv3$, отличающаяся от $LANIFAv2$ вспомогательной переменной-счётчиком M
Fig. 6. The flowchart of the algorithm $LANIFAv3$ different from $LANIFAv2$ by a fresh counter M

3.2. Верификация аннотированного алгоритма в арифметике с фиксированной запятой

Доказательство утверждения тотальной корректности для алгоритма $LANIFAv3$ с предусловием и постусловием такими же, как в спецификации (21), разобьем на доказательство утверждения частичной корректности и доказательство завершаемости.

Для доказательства частичной корректности методом Флойда надо доказать, что

- если в контрольной точке 1 выполнена её аннотация (то есть предусловие) и путь (1..2) завершается, то после выполнения этого пути в контрольной точке 2 выполнена её аннотация (то есть инвариант);
- если в контрольной точке 2 выполнена её аннотация (то есть инвариант) и путь $(2-2)$ завершается, то после выполнения этого пути в контрольной точке 2 выполнена её аннотация (то есть тот же инвариант, но при новых значениях переменных);
- если в контрольной точке 2 выполнена её аннотация (то есть инвариант) и путь $(2+3)$ завершается, то после выполнения этого пути в контрольной точке 3 выполнена её аннотация (то есть постусловие).

Доказательство пути (1..2) тривиально. Так как значение переменной Z в конце этого пути равно $UpRt\big[\lceil y \rceil\big]$, дополнительные условия (f) и (g) следуют из условий (a) и (d) предусловия. Так как значение переменной M в конце этого пути равно 1, то выполнение условия (h) очевидно.

Доказательство пути $(2+3)$ фактически выполнено в разделе 2.5.

Докажем путь $(2-2)$. Пусть a и m — значения переменных Z и M в начале этого пути, путь завершается, а b и $(m+1)$ — значения этих же переменных после завершения этого пути. Так как этот путь завершается, то b — это и значение переменной

NZ в конце этого пути, а так как на этом пути выполнено условие $NZ \geq 1$, то $b \geq 1$, то есть выполнено условие (f) в конце этого пути. Если условие (h) выполнено в начале пути $(2-2)$, то, очевидным образом, это условие выполнено и в конце этого пути. В силу свойства (g) для a имеет место $1 \leq a < \left(\sqrt{y} + \frac{1}{2}\right)$, и, с учётом свойства (e), $2a < (\sup_{\mathbb{D}} - 2\delta_{\mathbb{D}})$. Следовательно, $2a = (a \oplus a)$. Далее имеем

$$\frac{y}{2a} + \frac{a}{2} - \sqrt{y} = \frac{y}{2(\sqrt{y}+\Delta)} + \frac{\sqrt{y}+\Delta}{2} - \sqrt{y} \text{ (где } \Delta = a - \sqrt{y}) =$$
$$= \left(\frac{y}{2(\sqrt{y}+\Delta)} - \frac{\sqrt{y}}{2}\right) + \frac{\Delta}{2} = -\frac{\Delta\sqrt{y}}{2(\sqrt{y}+\Delta)} + \frac{\Delta}{2} = \frac{\Delta^2}{2(\sqrt{y}+\Delta)} = \frac{\Delta^2}{2a} <$$
$$< \frac{a-\sqrt{y}}{2} \text{ (в силу } (f) \text{ и } (g) \text{ для } a).$$

Пусть $\left(\frac{y}{2a} + \delta'\right) = (y \oslash 2a)$ и $\left(\frac{a}{2} + \delta''\right) = (a \oslash 2)$, где $|\delta'| \leq \frac{\delta_{\mathbb{D}}}{2}$ и $|\delta''| \leq \frac{\delta_{\mathbb{D}}}{2}$. Тогда

$$((y \oslash (a \oplus a)) + (a \oslash 2)) = \left(\frac{y}{2a} + \frac{a}{2}\right) + (\delta' + \delta'') < \sqrt{y} + \frac{a-\sqrt{y}}{2} = \frac{a+\sqrt{y}}{2} < \sup_{\mathbb{D}},$$

и поэтому

$$b = ((y \oslash (a \oplus a)) \oplus (a \oslash 2)) = ((y \oslash 2a) + (a \oslash 2))$$

и

$$|b - \sqrt{y}| = \left|\left(\frac{y}{2a} + \frac{a}{2} - \sqrt{y}\right) + (\delta' + \delta'')\right| \leq \left|\frac{y}{2a} + \frac{a}{2} - \sqrt{y}\right| + |\delta' + \delta''| <$$
$$< \frac{|a-\sqrt{y}|}{2} + \delta_{\mathbb{D}} < \frac{\left(\frac{1}{2^m} + 2\delta_{\mathbb{D}}\right)}{2} + \delta_{\mathbb{D}} = \frac{1}{2^{(m+1)}} + 2\delta_{\mathbb{D}}.$$

Следовательно, условие (g) тоже выполнено в конце пути $(2-2)$.

Таким образом, доказана частичная корректность обоих алгоритмов $LANIFAv2$ (Рис. 5) и $LANIFAv3$ (Рис. 6) для вычислений аппроксимаций квадратного корня в арифметике с фиксированной запятой. Это означает, в частности, что доказан инвариант контрольной точки 2 алгоритма $LANIFAv3$ и его условия (f) и (g). Кроме того, значение переменной M в этом алгоритме $LANIFAv3$ возрастает на каждой легальной итерации цикла $(2-2)$. Поэтому для доказательства завершаемости цикла $(2-2)$ в алгоритме $LANIFAv3$ мы можем применить рассуждения, уже описанные в разделе 2.4. Остаётся повторить замечание, что переменная M никак не влияет на работу алгоритма $LANIFAv3$ и, следовательно, оба алгоритма $LANIFAv2$ и $LANIFAv3$ на одних и тех же значениях y и ε или завершаются одновременно (выполнив одно и то же число итераций цикла $(2-2)$), или оба не завершаются. Так как мы доказали завершаемость алгоритма $LANIFAv3$, если выполнено предусловие (21), тем самым доказана и завершаемость алгоритма $LANIFAv2$, если также выполнено предусловие. Тем самым, мы доказали завершаемость и тотальную корректность обоих алгоритмов $LANIFAv2$ (Рис. 5) и $LANIFAv3$ (Рис. 6).

3.3. Алгоритм аппроксимации квадратного корня в арифметике с плавающей запятой, его спецификация и верификация

Идея алгоритма вычисления квадратного корня в арифметике с плавающей запятой математически может быть изложена очень просто: если вещественное число $a \in \mathbb{R}$, $a \geq 0$ представлено в форме $a = m \times \beta^p$, где $m \in \mathbb{R}$ — “мантисса” этого числа,

$\beta \in \mathbb{R}^+$ — (положительное) “основание” (экспоненты), а $p \in \mathbb{Z}$ — “‘экспонента” (вернее, показатель экспоненты) этого числа, то

$$\sqrt{a} = \begin{cases} \sqrt{m} \times \beta^{\frac{p}{2}}, & \text{если } p \text{ — чётное число;} \\ \sqrt{m \times \beta} \times \beta^{\frac{p-1}{2}}, & \text{если } p \text{ — нечётное число.} \end{cases} \tag{22}$$

Алгоритм $FSQRT$ на Рис. 7 реализует эту простую математическую идею для чисел с плавающей запятой с использованием чисел с фиксированной запятой для мантиссы, основания и экспоненты.

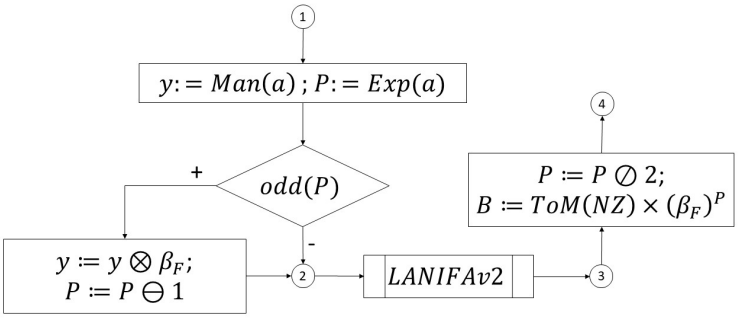


Рис. 7. Блок-схема алгоритма $FSQRT$
Fig. 7. The flowchart of the algorithm $FSQRT$

В отличие от аксиоматизации арифметики с фиксированной запятой, описанной в разделе 2.1., мы не будем аксиоматизировать *всю* арифметику с плавающей запятой, а только аксиоматизируем несколько свойств и операций, которые преобразуют числа с плавающей запятой в числа с фиксированной запятой и наоборот. Пусть $\mathbb{D}$ — тип для чисел с фиксированной запятой, удовлетворяющий аксиоматизации, описанной в 2.1. Определим тип для чисел с плавающей запятой $\mathbb{F}$ следующим образом:

- множество значений этого типа $Val_{\mathbb{F}}$ — это конечное подмножество множества вещественных чисел $\mathbb{R}$, состоящее из некоторых чисел из диапазона $[\inf_{\mathbb{F}}, \sup_{\mathbb{F}}]$, где $\inf_{\mathbb{F}} < 2$, $\sup_{\mathbb{F}} > 2$ и $\{0, \inf_{\mathbb{F}}, \sup_{\mathbb{F}}\} \subseteq Val_{\mathbb{F}}$;
- есть три унарные операции:
  - *мантисса* $Man : Val_{\mathbb{F}} \to Val_{\mathbb{D}}$,
  - *экспонента* $Exp : Val_F \to Int_{\mathbb{D}}$,
  - *преобразование в мантиссу* $ToM : Val_{\mathbb{D}} \to Val_{\mathbb{D}}$,

  а также константы
  - *основание* (целое число) $\beta_T \in Int^+_{\mathbb{D}}$ (натуральное число с фиксированной запятой),

– *дефект* (мантиссы) $\mu_{\mathbb{F}} \in Val^{+}_{\mathbb{D}}$ (положительное число с фиксированной запятой)

такие, что:

– для любого положительного числа $x \in Val_{\mathbb{F}}$
  * $1 < Man(x) < \frac{sup_{\mathbb{D}}}{\beta_{\mathbb{F}}}$, и $x = Man(x) \times \beta_{\mathbb{F}}^{Exp(x)}$ (где "$\times$" — математическая операция умножения);
  * если $inf_{\mathbb{D}}$ нечётное число, то $inf_{\mathbb{D}} < Exp(x)$, и $inf_{\mathbb{D}} \leq Exp(x)$, если $inf_{\mathbb{D}}$ — чётное;
– $|x - ToM(x)| \leq \mu_{\mathbb{F}}$ для любого положительного числа $x \in Val_{\mathbb{D}}$.

Во-первых, заметим, что по нашему определению диапазон изменения мантиссы — это некоторый подынтервал $(1, \frac{sup_{\mathbb{D}}}{\beta_{\mathbb{F}}})$, в то время как общепринятое определение фиксирует диапазон $[0.1, 1)$. Причина нашего выбора диапазона изменения мантиссы простая: в предыдущем разделе 3.1. был описан, а в разделе 3.2. верифицирован алгоритм $LANIFAv2$ (Рис. 5) аппроксимации квадратного корня в арифметике с фиксированной запятой для чисел, больших 1 (см. спецификацию (21)).

Во-вторых, заметим, что свойство $x = Man(x) \times \beta_{\mathbb{F}}^{Exp(x)}$ использует именно *математическую* операцию умножения, а не её "машинный" аналог. Но это использование операции умножения на самом деле соответствует представлению числа с плавающей запятой в компьютере в машинном слове в виде мантиссы и экспоненты, и поэтому операция умножения в данном случае — это просто операция "сборки" числа с плавающей запятой из его мантиссы и экспоненты (без потери точности).

В-третьих, "обратная" операция преобразования в мантиссу по нашему определению может приводить к потере точности мантиссы, что означает, что не все числа с фиксированной запятой являются мантиссами чисел с плавающей запятой (мантисса числа с плавающей запятой имеет меньше разрядов, чем число с фиксированной запятой).

Как уже было сказано, алгоритм аппроксимации квадратного корня для чисел с плавающей запятой $FSQRT$ представлен на Рис. 7. В этом алгоритме

- $LANIFAv2$ — это уже знакомый нам алгоритм с Рис. 5 с входными переменными y и ε и выходной переменной NZ типа $\mathbb{D}$ (с фиксированной запятой),
- переменные a и B типа $\mathbb{F}$ (с плавающей запятой) являются ещё одной входной и, соответственно, единственной выходной переменной этого алгоритма $FSQRT$,
- переменная P имеет тип с фиксированной запятой $\mathbb{D}$ (но принимает только целые значения из $Int_{\mathbb{D}}$),
- операции $\otimes$, $\otimes$ и $\oslash$ — это машинные операции умножения, вычитания и деления для арифметики с фиксированной запятой, описанные в разделе 2.1.),
- а константа $\beta_{\mathbb{F}}$ — это основание экспоненты (то есть число с фиксированной запятой типа $\mathbb{D}$).

Заметим также, что алгоритм $FSQRT$ выглядит ациклическим, но на самом деле он содержит цикл, "спрятанный" в алгоритме $LANIFAv2$.

Так как алгоритм $FSQRT$ основан на алгоритме $LANIFAv2$, то его спецификация получается из спецификации (21), но по отдельности рассматривает случаи чётной и нечётной экспоненты входной переменной a (как это уже сделано в (22)). Для удобства выделим общую часть предусловия

$$(a,b,c) \equiv \left\{ \begin{array}{ll} (a) & 0 < a \in Val_{\mathbb{F}} \ \& \\ (b) & \delta_{\mathbb{D}} \le \frac{1}{12} \ \& \\ (c) & 5\frac{1}{6}\delta_{\mathbb{D}} < \varepsilon \in Val_{\mathbb{D}}. \end{array} \right. \tag{23}$$

Тогда спецификация алгоритма $FSQRT$ выглядит следующим образом:

Чётная экспонента:

$$\left[\begin{array}{ll} (a,b,c) & \& \\ (d) & \sqrt{Man(a)} \le UpRt\big[\lceil Man(a) \rceil\big] \le \\ & \qquad \le \sqrt{Man(a)} + \frac{1}{2} \ \& \\ (e) & 2\left(\sqrt{Man(a)} + \frac{1}{2}\right) < \sup_{\mathbb{D}} - 2\delta_{\mathbb{D}} \end{array} \right] \tag{24}$$

$$FSQRT$$

$$\left[|\sqrt{a} - B| \le (\varepsilon + 2\delta_{\mathbb{D}} + \mu_{\mathbb{F}}) \times \beta_{\mathbb{F}}^{\frac{Exp(a)}{2}} \right]$$

Нечётная экспонента:

$$\left[\begin{array}{ll} (a,b,c) & \& \\ (d) & \sqrt{Man(a) \times \beta_{\mathbb{F}}} \le UpRt\big[\lceil man(a) \rceil\big] \le \\ & \qquad \le \sqrt{Man(a) \times \beta_{\mathbb{F}}} + \frac{1}{2} \ \& \\ (e) & 2\left(\sqrt{Man(a) \times \beta_{\mathbb{F}}} + \frac{1}{2}\right) < \sup_{\mathbb{D}} - 2\delta_{\mathbb{D}} \end{array} \right] \tag{25}$$

$$FSQRT$$

$$\left[|\sqrt{a} - B| \le (\varepsilon + 2\delta_{\mathbb{D}} + \mu_{\mathbb{F}}) \times \beta_{\mathbb{F}}^{\frac{Exp(a)-1}{2}} \right]$$

Фактически в этой спецификации просто явно использовано то, что в алгоритме $FSQRT$ (Рис. 7) на вход алгоритма $LANIFAv2$ (Рис. 5) в качестве значения переменной y передаётся или значение $Man(a)$, или значение $Man(a) \times \beta_{\mathbb{F}}$ (в зависимости от чётности показателя экспоненты), и, следовательно, предусловия в (24) и (25) получаются из предусловия в (21) в результате подстановки $Man(a)$ или $Man(a) \times \beta_{\mathbb{F}}$ вместо y. Аналогично, постусловия в (24) и (25) получаются из постусловия в (21) в результате прямого просачивания постусловия из (21) через присваивания

$$P := P \oslash 2 \ ; \ B := ToM(NZ) \times \beta_{\mathbb{F}}^{P},$$

в результате чего в оценке ошибки в арифметике с фиксированной запятой $\varepsilon + 2\delta_{\mathbb{D}}$ появляется дефект мантиссы $\mu_{\mathbb{F}}$ в качестве "довеска".

4. Заключение

4.1. Сумма результатов статьи

В качестве эпиграфа к нашей статье (и проекту "*Платформенно-независимый подход к формальной спецификации и верификации стандартных математических функций*") можно было бы взять следующую цитату из аннотации работы [20]:

> *Current critical systems commonly use a lot of floating-point computations, and thus the testing or static analysis of programs containing floating-point operators has become a priority. However, correctly defining the semantics of common implementations of floating-point is tricky, because semantics may change with many factors beyond source-code level, such as choices made by compilers.*

Однако есть существенная разница между [20] и нашей статьей. Цитируемая статья посвящена вопросам представления чисел с плавающей запятой и реализации арифметических операций, а наша статья — представлению платформенно-независимого инкрементального комбинированного подхода к спецификации и верификации стандартных функций и его применению для спецификации и верификации стандартной математической функции квадратного корня.

Платформенная независимость отличает нашу работу от работ по формальной спецификации и верификации стандартных функций для конкретных процессорных архитектур Intel [14,15,17] и Oracle [11], или чётко описанных форматов представления чисел с фиксированной и плавающей запятой [5,13,20]. Мы только формулируем достаточно общие свойства арифметики с фиксированной запятой и только минимально необходимые свойства арифметики с плавающей запятой, которые, как мы надеемся, будет легко проверить для конкретной архитектуры и платформы. Отметим сразу, что на данный момент мы только прототипировали вариант реализации арифметики с фиксированной запятой, а проверка, что существующие реализации арифметики с фиксированной запятой удовлетворяют нашей модели, пока остаётся задачей на перспективу.

Кроме того, подход, представленный в данной статье, отличает инкрементальность, состоящая в том, что сначала мы специфицируем и вручную верифицируем алгоритм для идеальной арифметики вещественных чисел, потом используем эту спецификацию и верификацию как эскиз для спецификации и ручной верификации алгоритма в машинной арифметике с фиксированной запятой, а затем комбинируем ручную верификацию с автоматической, то есть формализуем и проверяем наше доказательство целиком средствами автоматизированной системы построения доказательств. На данный момент мы располагаем только формализованным доказательством существования справочной таблицы начальных приближений для квадратного корня, выполненное с использованием системы ACL2.

Следует обратить внимание на то, что согласно стандарту IEEE-754 [29], аппроксимация квадратного корня должна быть "точной" (exact) в следующем смысле: возвращаемый результат должен отличаться от истинного математического результата не более чем на $\frac{1}{2}ULP$, в то время как лучшая из доказанных нами в настоящей

статье оценок для ошибки (в арифметике с фиксированной запятой)

$$\varepsilon + 2\delta_{\mathbb{D}} \text{ (причём } 5\frac{1}{6}\delta_{\mathbb{D}} < \varepsilon),$$

которая фигурирует в утверждении тотальной корректности (21), в пределе даёт оценку $7\frac{1}{6}\delta_{\mathbb{D}}$. Может сложиться впечатление, что наш алгоритм $LANIFAv2$ для арифметики с фиксированной запятой и, следовательно, алгоритм для арифметики с плавающей запятой $FSQRT$, не являются точными. Однако, если принять, что наш тип данных $\mathbb{D}$, использованный для вычислений с фиксированной запятой, — это "внутренний" тип данных, используемый только для обеспечения нужной точности вычислений во "внешнем" типе $\mathbb{T}$, то ситуация изменится: если всякое число с плавающей запятой типа $\mathbb{T}$ автоматически является числом типа $\mathbb{D}$, а при обратном преобразовании из типа $\mathbb{D}$ в тип $\mathbb{T}$ происходит округление с "потерей" 4 младших двоичных разрядов (то есть $16\delta_{\mathbb{D}}$), то вычисления, выполненные в типе $\mathbb{D}$ с ошибкой $7\frac{1}{6}\delta_{\mathbb{D}}$, становятся точными для типа $\mathbb{T}$ (так как $7\frac{1}{6}\delta_{\mathbb{D}} < 8\delta_{\mathbb{D}} = \frac{16}{2}\delta_{\mathbb{D}}$). Такая интерпретация соответствует практике реализации вычислений со значениями типа `float`, во время которых все вычисления реально проходят с типом `double`, а по завершении вычислений результат типа `double` округляется до значения типа `float`. Именно по этой причине мы выбрали обозначение $\mathbb{D}$ (Double) для "внутреннего" типа с фиксированной запятой. Аналогичная ситуация у нас возникала с мантиссами в разделе 3.3.: всякая мантисса определялась числом с фиксированной запятой, но при обратном преобразовании происходила потеря точности.

4.2. Обзор литературы

Актуальность проблемы спецификации и валидации стандартных функций хорошо осознана научным и инженерным сообществом. Можно, например, сослаться на работы [1,19], в которых всесторонне обсуждается проблема спецификации и тестирования стандартных математических функций, причём во внимание принимается не только точность вычислений (ошибка или невязка), но и такие алгебраические свойства стандартных функций, как, например, чётность и нечётность (cos и sin), нормализация ($\sin^2 x + \cos^2 x = 1$). Образовательное значение проблемы спецификации и верификации стандартных функций также нашло отражение в литературе [24,25].

Достаточно обширная литература посвящена формальной верификации реализации стандартных математических функций в арифметике с плавающей запятой для конкретных платформ и архитектур: в работах [8,15] верифицирована функция (операция) деления, в работах [3,8,15,23] — функция квадратного корня, в [14] — некоторые тригонометрические функции, в [16] — степенная функция, а в [27] — гамма-функция. (Кстати, даже автоматизированная верификация целочисленных стандартных функций является нетривиальной задачей, см., например, работу [3], в которой доказана с использованием системы PVS функция целой части квадратного корня.)

Есть и публикации по аксиоматизации машинной арифметики с плавающей запятой и, прежде всего, бинарной машинной арифметики в соответствии со стандартом IEEE-754, с доказательством основных свойств и корректности некоторых стандартных математических функций [4–6,13,22].

Так, в работе [5] представлена формализация машинной арифметики в Z-нотации и её реализация на языке Occam. В частности, в цитируемой работе описаны следующие операции и функции в арифметике с плавающей запятой: округления, сложения, умножения, квадратного корня, преобразования в целый тип, сравнения и другие. Кроме того, в работе [5] формализованы пять классов чисел с плавающей запятой (NaN, Inf, $zero$, нормализованные и денормализованные числа) и четыре варианта округления, в то время как в нашей статье мы используем только один вариант округления к ближайшему числу. Реализация на языке Occam включает представление чисел с плавающей запятой, операции округления и другие. Шаблон работы с числами с плавающей запятой в [5]) состоит в следующем:

1. "распаковка" числа, то есть вычисление знака, мантиссы и экспоненты числа;
2. денормализация числа (выравнивание экспонент);
3. выполнение операции с денормализованными числами;
4. "упаковка" результатов операции (то есть "сборка" числа из знака, мантиссы и экспоненты).

В работе [13] представлен подход с использованием системы HOL Light к спецификации и верификации нескольких операций с плавающей запятой в архитектуре Intel IA-64:

> *Correctness of the mathematical software starts from the assumption that the underlying hardware floating point operations behave according to the IEEE standard 754 for binary floating point arithmetic. Actually, IEEE-754 doesn't explicitly address floating-point machine arithmetic operations, and it leaves underspecified certain significant questions, e.g. NaN propagation and underflow detection. Thus, we not only need to specify the key IEEE concepts but also some details specific to IA-64.*

В цитируемой статье сначала представлена платформенно-независимая теория чисел с плавающей запятой, которая затем специализируется для архитектуры IA-64: в теории числа с плавающей запятой представляются в весьма общем виде $\pm k \times 2^{E-N}$, но затем используются стандартные форматы представления, распаковки и упаковки, денормализации и нормализации. В работе [13] также обсуждается проблема определения понятия ULP, которое имеет определённые разночтения (но, в конце концов, в цитируемой работе принимается определение ULP из [21]). В [13] (так же, как и в [5]) рассмотрены 4 варианта округления результата выполнения операций над числами с плавающей запятой:

- к ближайшему числу с плавающей запятой (как у нас в статье),
- а также вниз, вверх и по направлению к нулю.

В принципе числа с плавающей запятой в [13] могут иметь неограниченную экспоненту (что отличается от стандарта IEEE-754), но во время выполнения операций переполнение памяти обрабатывается должным образом.

В работе [6] предпринята попытка описать логическую теорию (синтаксис и семантика) арифметики чисел с плавающей запятой и использовать SMT-решатели (Satisfiability Modulo Theories) в качестве разрешающей процедуры. Заметим, что в нашей статье мы не стремились дать сколько-либо полную теорию ни для арифметики с фиксированной запятой, ни для арифметики с плавающей запятой, а ограничились только свойствами, которые нам понадобились для спецификации и верификации конкретной проблемы — аппроксимации квадратного корня.

Верификация аппроксимаций квадратного корня вызывает особый интерес. Так, работа [8] посвящена верификации алгоритма вычисления аппроксимации функции квадратного корня, основанного на так называемом методе "цифра за цифрой" (digit-by-digit или CORDIC — COordinate Rotation DIgital Computer), который вычисляет цифры вещественного числа. В работах [9, 23] доказывается корректность алгоритма аппроксимации квадратного корня, использующего многочлены Чебышёва и реализованного в процессоре Power4 (хотя алгоритм, основанный на многочленах Чебышёва, выполняет больше шагов, чем метод Ньютона для достижения нужной точности). Верификация алгоритма аппроксимации квадратного корня в работах [9, 23] разделена на два этапа: доказательство корректности разложения функции квадратного корня в ряд Тейлора и доказательство свойств функции квадратного корня с использованием этого разложения. Все доказательства в работе [23] выполнены с использованием библиотеки ACL2, формализующей нестандартный вещественный анализ.

В конце литературного обзора мы остановимся на сравнении нашего препринта [26] с настоящей статьей. В препринте [26] описаны, специфицированы и (вручную) верифицированы *неадаптивные* алгоритмы аппроксимации квадратного корня для чисел с фиксированной и плавающей запятой с *накоплением ошибки* вплоть (для чисел с фиксированной запятой) до $2\delta_{\mathbb{D}}(2 + \log_2 \frac{S}{\varepsilon})$, где S — "шаг" справочной таблицы начальных приближений для квадратного корня (в настоящей статье $S = 1$). В настоящей работе описаны, специфицированы и (вручную) верифицированы *адаптивные* алгоритмы аппроксимации квадратного корня, достигающие (для чисел с фиксированной запятой) точности $7\frac{1}{6}\delta_{\mathbb{D}}$, рассмотрено прототипирование модели чисел с фиксированной запятой и автоматизированное доказательство (в системе ACL2) существования массива начальных приближений квадратного корня.

4.3. План дальнейших исследований

Первая ближайшая практическая задача — полная автоматизированная (с использованием, вероятно, системы ACL2) проверка ручного доказательства утверждения тотальной корректности (21) для алгоритма аппроксимации квадратного корня $LANIFAv2$ (Рис. 5) для чисел с фиксированной запятой и утверждений тотальной корректности (25) и (24) для алгоритма аппроксимации квадратного корня $FSQRT$ (Рис. 7) для чисел с плавающей запятой. Напомним, что на данном этапе пока выполнена автоматизированная верификация только существования массива начальных приближений квадратного корня в арифметике с фиксированной запятой.

Вторая ближайшая практическая задача — прототипирование алгоритма $LANIFAv2$ с использованием уже реализованной модели арифметики с фиксированной

запятой, прототипирование модели арифметики с плавающей запятой, а затем — прототипирование алгоритма $FSQRT$ в этой модели. Связанная задача — поиск примеров реальных платформ, соответствующих нашим моделям машинной арифметики.

Ближайшие теоретические задачи — исследование применимости интервальной математики [12] в контексте спецификации и верификации стандартных математических функций (на примере квадратного корня) и выбор метода и алгоритма спецификации и (сначала ручной) верификации стандартных тригонометрических функций sin и cos.

Благодарность: Дмитрию Юрьевичу Надёжину за техническую помощь с доказательством в системе ACL2 и за обсуждение материалов препринта [26].

Список литературы / References

[1] Кулямин В.В., “Стандартизация и тестирование реализаций математических функций, работающих с числами с плавающей точкой”, *Программирование*, **33**:3 (2007), 1–29; English transl.: Kuliamin V.V., “Standardization and testing of implementations of mathematical functions in floating point numbers”, *Programming and Computer Software*, **33**:3 (2007), 154–173.

[2] Никитин В.Ф., *Вариант вычисления квадратного корня (алгоритм Ньютона)*, http://algolist.manual.ru/maths/count_fast/sqrt.php; [Nikitin V.F., *Variant vychisleniya kvadratnogo kornya (algoritm Nyutona)*, http://algolist.manual.ru/maths/count_fast/sqrt.php, (in Russian).]

[3] Шелехов В.И., “Верификация и синтез эффективных программ стандартных функций floor, isqrt и ilog2 в технологии предикатного программирования”, *Проблемы управления и моделирования в сложных системах*, Труды 12-й межд. конф. (Самара, Самарский научный центр РАН), 2010, 622–630; [Shelekhov V.I., “Verifikatsiya i sintez effektivnykh programm standartnykh funktsiy floor, isqrt i ilog2 v tekhnologii predikatnogo programmirovaniya”, *Problemy upravleniya i modelirovaniya v slozhnykh sistemakh*, Trudy 12-y mezhd. konf. (Samara, Samarskiy nauchnyy tsentr RAN), 2010, 622–630, (in Russian).]

[4] Ayad A., Marché C., “Multi-prover verification of floating-point programs”, *Perspectives of System Informatics. PSI 2014*, Lecture Notes in Artificial Intelligence, **6173**, Springer, Berlin, Heidelberg, 2010, 127–141.

[5] Barret G., “Formal Methods Applied to a Floating-Point Number System”, *IEEE Trans. Softw. Eng.*, **15**:5 (1989), 611–621.

[6] Brain M. et al., “An Automatable Formal Semantics for IEEE-754 Floating-Point Arithmetic”, *Computer Arithmetic (ARITH ’15)*, Proc. of the 2015 IEEE 22nd Symposium (IEEE Computer Society), 2015, 160–167.

[7] El-Magdoub M.H., *Best Square Root Method – Algorithm – Function (Precision VS Speed)*, https://www.codeproject.com/Articles/69941/Best-Square-Root-Method-Algorithm-Function-Precisi.

[8] Ferguson W.E. et al., “Digit serial methods with applications to division and square root (with mechanically checked correctness proofs)”, 2017, 102–114, arXiv: arXiv:1708.00140.

[9] Gamboa R.A., *Square Roots in Acl2: a Study in Sonata Form*, Technical Report. University of Texas at Austin, USA, 1997.

[10] Грис Д., *Наука программирования*, Мир, М., 1986; in English: Gries D., *The Science of Programming*, Springer-Verlag, New York, 1981.

[11] Grohoski G., “Verifying Oracle’s SPARC Processors with ACL2 (Slides of the Invited talk)”, *The ACL2 Theorem Prover and Its Applications*, Proc. Int. Workshop, 2017,

http://www.cs.utexas.edu/users/moore/acl2/workshop-2017/slides-accepted/grohoski-ACL2_talk.pdf.

[12] Gutowski M.W., "Power and beauty of interval methods", arXiv: arXiv:physics/0302034.

[13] Harrison J., "A Machine-Checked Theory of Floating Point Arithmetic", Lecture Notes in Computer Science, **1690**, Springer, Berlin, Heidelberg, 1999, 113–130.

[14] Harrison J., "Formal Verification of Floating Point Trigonometric Functions", Lecture Notes in Computer Science, **1954**, Springer, Berlin, Heidelberg, 2000, 217–233.

[15] Harrison J., "Formal Verification of IA-64 Division Algorithms", Lecture Notes in Computer Science, **1869**, Springer, Berlin, Heidelberg, 2000, 233–251.

[16] Harrison J., "Floating Point Verification in HOL Light: The Exponential Function", *Formal Methods in System Design*, **16**:3 (2000), 271–305.

[17] Harrison J., "Formal Verification of Square Root Algorithms", *Formal Methods in System Design*, **22**:2 (2003), 143–153.

[18] Hoare C.A.R., "The Verifying Compiler: A Grand Challenge for Computing Research", Lecture Notes in Computer Science, **2890**, Springer, Berlin, Heidelberg, 2003, 1–12.

[19] Kuliamin V.V., "Standardization and Testing of Mathematical Functions in floating point numbers", Lecture Notes in Computer Science, **5947**, Springer, Berlin, Heidelberg, 2010, 257–268.

[20] Monniaux D., "The pitfalls of verifying floating-point computations", *ACM Transactions on Programming Languages and Systems*, **30**:3 (2008), 1–41.

[21] Muller J.-M., *Elementary Functions: Algorithms and Implementation*, Birkhaüser, 2005.

[22] Muller J.-M. et al., *Handbook of Floating-Point Arithmetic, 2nd eddition*, Birkhüser, 2018.

[23] Sawada J., Gamboa R., "Mechanical Verification of a Square Root Algorithm Using Taylor's Theorem", Lecture Notes in Computer Science, **2517**, Springer, Berlin, Heidelberg, 2002, 274–291.

[24] Shilov N.V., "On the need to specify and verify standard functions", *The Bulletin of the Novosibirsk Computing Center (Series: Computer Science)*, **38** (2015), 105–119.

[25] Shilov N.V., Promsky A.V., "On specification andd verification of standard mathematical functions", *Humanities and Science University Journal*, **19** (2016), 57–68.

[26] Shilov N.V. et al., "Towards platform-independent verification of the standard mathematical functions: the square root function", arXiv: arXiv:abs/1801.00969.

[27] Siddique U., Hasan O., "On the Formalization of Gamma Function in HOL", *J. Autom. Reason.*, **53**:4 (2014), 407–429.

[28] *C reference. Sqrt, sqrtf, sqrtl*, http://en.cppreference.com/w/c/numeric/math/sqrt.

[29] *IEEE 754-2008*, http://ieeexplore.ieee.org/document/4610935.

[30] *ISO/IEC/IEEE 60559:2011. Information technology -- Microprocessor Systems -- Floating-Point arithmetic*, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57469.

[31] *Алгоритм нахождения корня n-ной степени*, https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%BD%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BA%D0%BE%D1%80%D0%BD%D1%8F_n-%D0%BD%D0%BE%D0%B9_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D0%B8; in English: *nth root algorithm*, https://en.wikipedia.org/wiki/Nth_root_algorithm.

[32] *"Роскосмос" назвал причину неудачного запуска с космодрома Восточный*, https://www.rbc.ru/politics/12/12/2017/5a2ebcd59a79479d29667115; [*"Roskosmos" nazval prichinu neudachnogo zapuska s kosmodroma Vostochnyy*, https://www.rbc.ru/politics/12/12/2017/5a2ebcd59a79479d29667115, (in Russian).]

Shilov N. V., Kondratyev D. A., Anureev I. S., Bodin E. V, Promsky A. V., "Platform-independent Specification and Verification of the Standard Mathematical Square Root Function", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 637–666.

DOI: 10.18255/1818-1015-2018-6-637-666

Abstract. The project "Platform-independent approach to formal specification and verification of standard mathematical functions" is aimed onto the development of incremental combined approach to specification and verification of standard Mathematical functions like `sqrt`, `cos`, `sin`, etc. Platform-independence means that we attempt to design a relatively simple axiomatization of the computer arithmetics in terms of real arithmetics (i.e. the field $\mathbb{R}$ of real numbers) but do not specify neither base of the computer arithmetics, nor a format of numbers representation. Incrementality means that we start with the most straightforward specification of the simplest case to verify the algorithm in real numbers and finish with a realistic specification and a verification of the algorithm in computer arithmetics. We call our approach combined because we start with manual (pen-and-paper) verification of the algorithm in real numbers, then use this verification as proof-outlines for a manual verification of the algorithm in computer arithmetics, and finish with a computer-aided validation of the manual proofs with a proof-assistant system (to avoid appeals to "obviousness" that are common in human-carried proofs). In the paper, we apply our platform-independent incremental combined approach to specification and verification of the standard Mathematical square root function. Currently a computer-aided validation was carried for correctness (consistency) of our fix-point arithmetics and for the existence of a look-up table with the initial approximations of the square roots for fix-point numbers.

Keywords: fix-point numbers, floating-point numbers, computer arithmetic, formal verification, partial and total correctness, Hoare triples, Floyd verification method of inductive assertions, exact function, square root function, Newton–Raphson method, look-up table

On the authors:
Nikolay V. Shilov, orcid.org/0000-0001-7515-9647, PhD
Autonomous noncommercial organization of higher education "Innopolis University",
1 Universitetskaya str., Innopolis, Tatarstan Republic, 420500, Russia, e-mail: shiloviis@mail.ru

Dmitry A. Kondratyev, orcid.org/0000-0002-9387-6735, postgraduate student
A.P. Ershov Institute of Informatics Systems,
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: apple-66@mail.ru

Igor S. Anureev, orcid.org/0000-0001-9574-128X, PhD, senior researcher
A.P. Ershov Institute of Informatics Systems,
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: anureev@iis.nsk.su

Eugene V. Bodin, orcid.org/0000-0002-5882-0365, researcher
A.P. Ershov Institute of Informatics Systems,
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: bodin@iis.nsk.su

Alexei V. Promsky, orcid.org/0000-0002-5963-2390, PhD, scientific secretary
A.P. Ershov Institute of Informatics Systems,
6, Acad. Lavrentjev pr., Novosibirsk 630090, Russia, e-mail: promsky@iis.nsk.su

Acknowledgments:
The research has been supported by Russian Foundation for Basic Research (grant 17-01-00789).

Моделирование и анализ информационных систем. Т. 25, №6 (2018), с. 667–679
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 667–679

Анализ сигналов
Signal Analysis

DOI: 10.18255/1818-1015-2018-6-667-679

УДК 004.032.26

Применение нейронных сетей для распознавания конструктивных элементов рельсов на магнитных и вихретоковых дефектограммах

Кузьмин Е. В., Горбунов О. Е., Плотников П. О., Тюкин В. А., Башкин В. А.

Поступила в редакцию 1 октября 2018
После доработки 23 ноября 2018
Принята к публикации 30 ноября 2018

Аннотация. Для обеспечения безопасности движения на железнодорожном транспорте регулярно проводится неразрушающий контроль рельсов с применением различных подходов и методов, включая методы магнитной и вихретоковой дефектоскопии. Актуальной задачей является автоматический анализ больших массивов данных (дефектограмм), которые поступают от соответствующего оборудования. Под анализом понимается процесс определения по дефектограммам наличия дефектных участков наряду с выявлением конструктивных элементов рельсового пути. Данная статья посвящена задаче распознавания образов конструктивных элементов железнодорожных рельсов по дефектограммам многоканальных магнитных и вихретоковых дефектоскопов. Рассматриваются три класса конструктивных элементов рельсового пути: 1) болтовой стык с прямым или скошенным соединением рельсов, 2) электроконтактная сварка рельсов и 3) алюмотермитная сварка рельсов. Образы, которые не могут быть отнесены к этим трем классам, условно считаются дефектами и выносятся в отдельный четвертый класс. Для распознавания образов конструктивных элементов на дефектограммах применяется нейронная сеть, реализованная в рамках открытой библиотеки TensorFlow. С этой целью каждая выделенная для анализа область дефектограммы преобразуется в графический образ в градации серого цвета размером 20 на 39 пикселей.

Ключевые слова: неразрушающий контроль рельсов, магнитная и вихретоковая дефектоскопия, обнаружение дефектов, автоматический анализ дефектограмм, нейронные сети

Для цитирования: Кузьмин Е. В., Горбунов О. Е., Плотников П. О., Тюкин В. А., Башкин В. А., "Применение нейронных сетей для распознавания конструктивных элементов рельсов на магнитных и вихретоковых дефектограммах", *Моделирование и анализ информационных систем*, **25**:6 (2018), 667–679.

Об авторах:
Кузьмин Егор Владимирович, orcid.org/0000-0003-0500-306X, д-р физ.-мат. наук, профессор кафедры теоретической информатики, Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: kuzmin@uniyar.ac.ru, kuzminev@nddlab.com

Горбунов Олег Евгеньевич, orcid.org/0000-0001-6274-9971, канд. физ.-мат. наук, генеральный директор,
ООО «Центр инновационного программирования», NDDLab, ул. Союзная, 144, г. Ярославль, 150008 Россия,
e-mail: gorbunovoe@nddlab.com

Плотников Петр Олегович, orcid.org/0000-0001-5687-7969, инженер-технолог, ООО «Центр инновационного программирования», NDDLab, ул. Союзная, 144, г. Ярославль, 150008 Россия, e-mail: plotnikovpo@nddlab.com

Тюкин Вадим Александрович, orcid.org/0000-0001-9149-7435, руков. сектора разработки, ООО «Центр инновационного программирования», NDDLab, ул. Союзная, 144, г. Ярославль, 150008 Россия, e-mail: tyukinva@nddlab.com

Башкин Владимир Анатольевич, orcid.org/0000-0002-2534-1026, д-р физ.-мат. наук, профессор кафедры теоретической информатики, Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: bashkinva@nddlab.com

Введение

Для обеспечения безопасности движения на железнодорожном транспорте регулярно проводится неразрушающий контроль рельсов с применением различных подходов и методов, включая методы магнитной и вихретоковой дефектоскопии. Актуальной задачей является автоматический анализ [3–5] больших массивов данных (дефектограмм), которые поступают от соответствующего оборудования. Под анализом понимается процесс определения по дефектограммам наличия дефектных участков наряду с выявлением конструктивных элементов рельсового пути.

Данная статья посвящена задаче распознавания образов конструктивных элементов железнодорожных рельсов по дефектограммам магнитных и вихретоковых дефектоскопов. Статья является логическим продолжением работ [1, 2], посвященных определению порогового уровня шума рельсов. В этой работе мы ограничимся рассмотрением обобщения реального устройства в виде 10-разрядного вихретокового дефектоскопа с 10-ю каналами данных (анализ данных магнитных дефектоскопов проводится аналогичным образом). Каналы данных соответствуют физическим датчиками, которые последовательно располагаются на поверхности рельса перпендикулярно направлению движения дефектоскопа. Значения амплитуд сигналов каждого канала регистрируются дефектоскопом в виде натуральных чисел от 1 до 1024 (в 10-разрядном случае).

При автоматическом анализе дефектограммы обычно разбиваются на фрагменты, которые, например, могут соответствовать 50-метровым участкам рельсового пути, т. е. при снятии показаний 10-канального дефектоскопа с каждого миллиметра пути блок анализа представляет собой матрицу размером 10 строк на 50000 столбцов, где элемент матрицы — это значение амплитуды сигнала соответствующего канала данных.

Интерес представляют амплитуды только полезных сигналов. Сигнал считается полезным (и подлежит дальнейшему анализу), если отклонение его значения от среднего значения всех сигналов (в рамках одного канала) как минимум в два раза превосходит пороговый уровень шума рельсов. Как и ранее [1, 2], под пороговым уровнем шума (одного канала) будем понимать отклонение Level от среднего значения μ сигналов рассматриваемого фрагмента дефектограммы (данные с 50-метрового участка), при котором сигналы со значениями амплитуд из диапазона $[\mu - \text{Level}; \mu + \text{Level}]$ являются шумом рельсов. Например, на рис. 1 в виде линий отсечки показаны пороговые уровень шума рельсов и уровень начала полезных сигналов для данных одного канала вихретокового дефектоскопа на 50-метровом участке рельсового пути.

Найденные полезные сигналы всех каналов группируются в отметки, которые представляют собой секции (дефектограммы) длиной 157 мм. Отметки формируются таким образом, чтобы иметь максимальное суммарное амплитудное отклонение по всем каналам на 157-миллиметровом участке дефектограммы среди близлежащих полезных сигналов. Также важным условием при формировании отметки является необходимость того, чтобы центр отметки совпадал с возможным центром реального конструктивного элемента, который может быть представлен на дефектограмме в виде записи данных. Алгоритм нахождения центра отметки в данной статье не рассматривается.

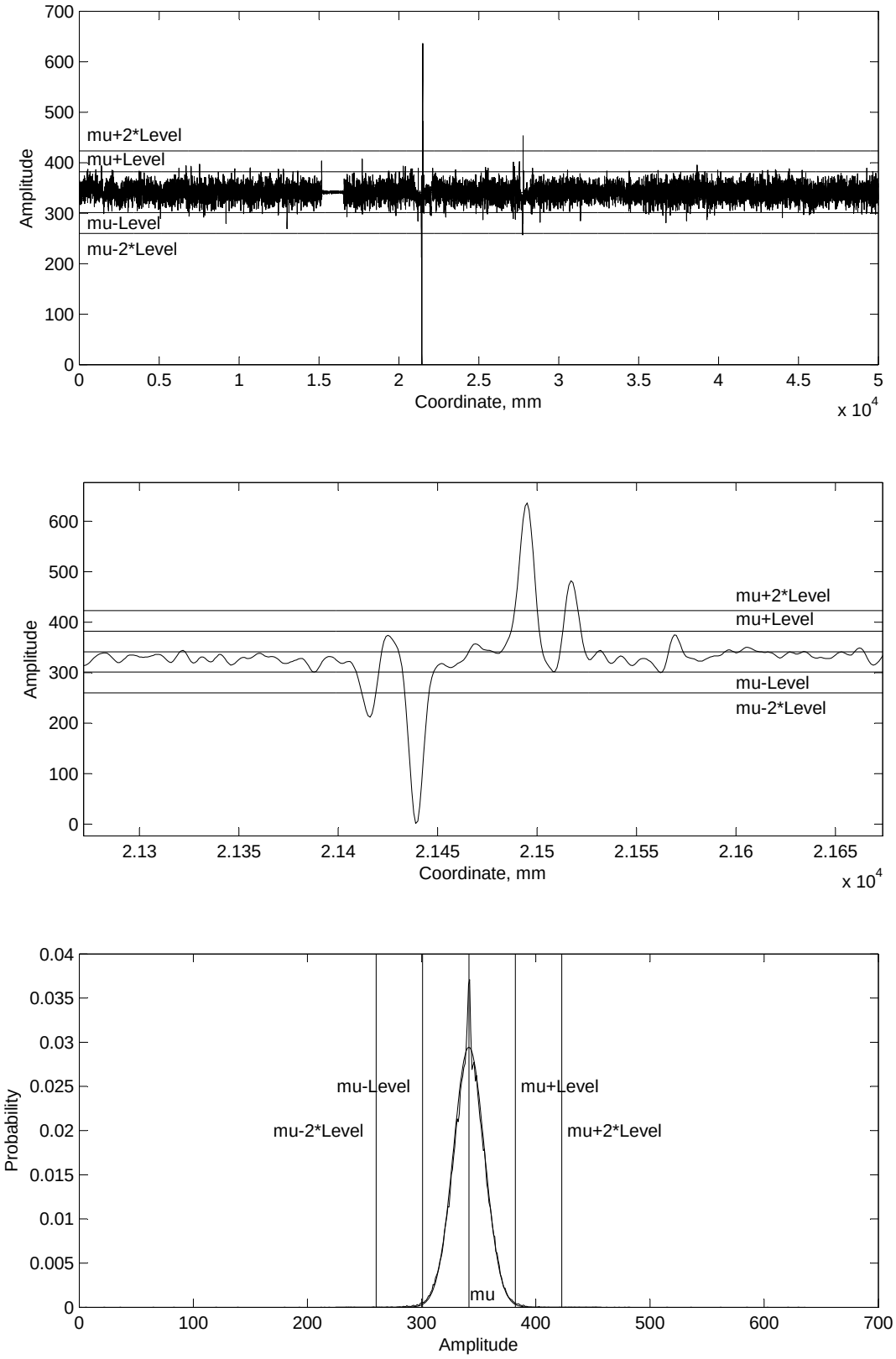


Рис. 1. Данные канала дефекстоскопа с линиями отсечки полезных сигналов
Fig. 1. Data of one testing system channel with useful signal threshold lines

В данной статье рассматриваются следующие три класса конструктивных элементов рельсового пути: 1) болтовой стык с прямым или скошенным соединением рельсов, 2) электроконтактная сварка рельсов и 3) алюмотермитная сварка рельсов. Отметки, которые не могут быть отнесены к этим трем классам, условно считаются дефектами и выносятся в отдельный (четвертый) класс отметок.

Для распознавания образов конструктивных элементов на вихретоковых дефектограммах применяется нейронная сеть, реализованная в рамках открытой библиотеки TensorFlow (в состав которой входит библиотека Keros) [9]. С этой целью каждая отметка преобразуются в графический образ в градации серого цвета размером 20 на 157 пикселей с последующим построчным сжатием до размера 20 на 39 пикселей. Полносвязная нейронная сеть, состоящая из входного (780 нейронов), внутреннего (800 нейронов) и выходного (4 нейрона) слоев, обучается на базе эталонных образов указанных четырех классов (всего около 50 тыс. образцов).

Обученная нейронная сеть в составе аппаратно-программного комплекса рельсовой дефектоскопии успешно используется на практике для распознавания образов конструктивных элементов рельсового пути.

1. Формирование образа отметки

После того как по полезным сигналам на дефектограмме выделена отметка в виде матрицы амплитуд размером 10 строк (по числу каналов) на 157 столбцов (длина отметки в мм), предполагается преобразование отметки в графический образ размером 20 на 157 пикселей в градации серого цвета с последующим построчным сжатием образа до размера 20 на 39 пикселей.

Отметим, что все эти преобразования делаются для того, чтобы свести хорошо известную задачу распознавания рукописных цифр в поле размером 28 на 28 точек к текущей задаче распознавания конструктивных элементов рельсов на дефектограмме. Во многом размер и конфигурация выбранной нейронной сети продиктована результатами работы [6].

Удвоение количества строк в образе отметки. Первым шагом в изменении представления отметки является добавление в матрицу амплитуд по одной строке для каждого канала. При этом в нечетной строке остаются только те амплитуды, которые больше среднего значения μ амплитуд сигналов (одного канала) в текущем фрагменте дефектограммы, а также превосходят пороговый уровень полезных сигналов $\mu + 2 * \mathrm{Level}$. Остальные значения в нечетной строке устанавливаются в ноль. Аналогичным образом, в четной строке остаются только те амплитуды, которые меньше порогового уровня полезных сигналов $\mu - 2 * \mathrm{Level}$. Остальные значения в четной строке также устанавливаются в ноль. В результате имеем матрицу амплитуд полезных сигналов с их дифференциацией и последующим построчным размещением по направлению отклонения от среднего значения (соответствующего канала) размером 20 строк на 157 столбцов.

Переход к 8-битовым значениям. Следующим шагом является представление текущего образа отметки в виде картинки в градации серого цвета. Для этого необходимо перейти к 8-битовым значениям матрицы. При этом осуществляется перевод амплитуд сигналов в децибелы с числом $2 * \mathrm{Level}$, т. е. уровнем начала полезных

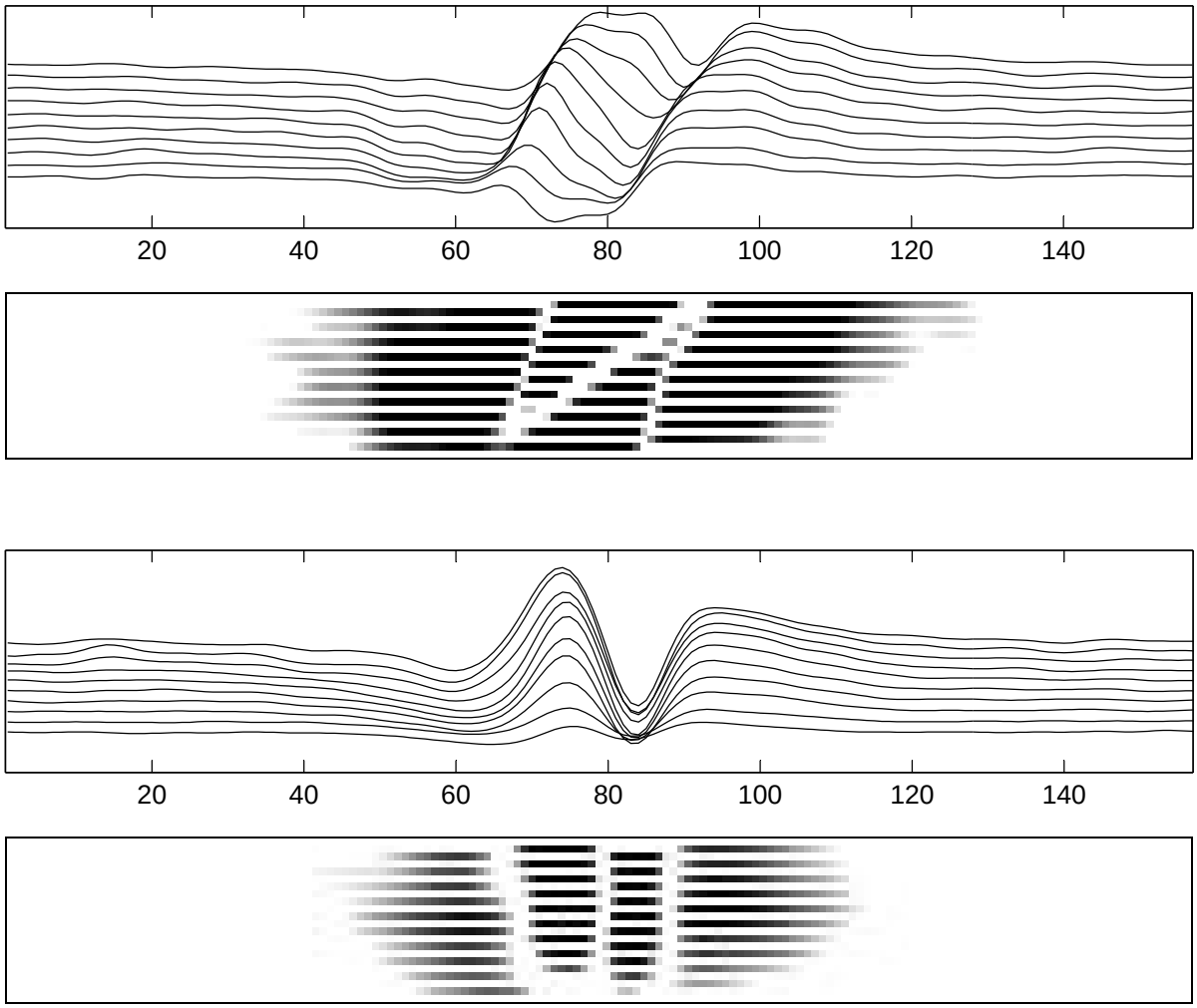


Рис. 2. Скошенный рельсовый стык и прямой рельсовый стык
Fig. 2. Two types of rail bolted joints

сигналов (в каждом канале свой уровень), в качестве опорной/исходной величины. Полученное значение проецируется на шкалу 0–255 простым умножением на 10 и присваиванием 255 всем элементам, которые превосходят это число.

Построчное сжатие. Заключительный шаг – построчное сжатие графического образа отметки до размера 20 строк на 39 столбцов. Для этого в каждой строке из каждых идущих подряд четырех элементов (по направлению от концов строки к середине) выбирается максимальный, который и будет являться элементом в новой битовой матрице. Единственное исключение – центральные элементы строки. В этом случае максимальное значение выбирается из пяти центральных элементов.

Ниже приведена функция на языке Python 3, формирующая графический образ отметки в виде bmp-матрицы размером 20 строк на 157 столбцов в градации серого цвета. В этой функции EC[0:49999,0:9] – текущий фрагмент анализируемых исходных вихретоковых данных, т. е. фрагмент дефектограммы длиной 50 метров

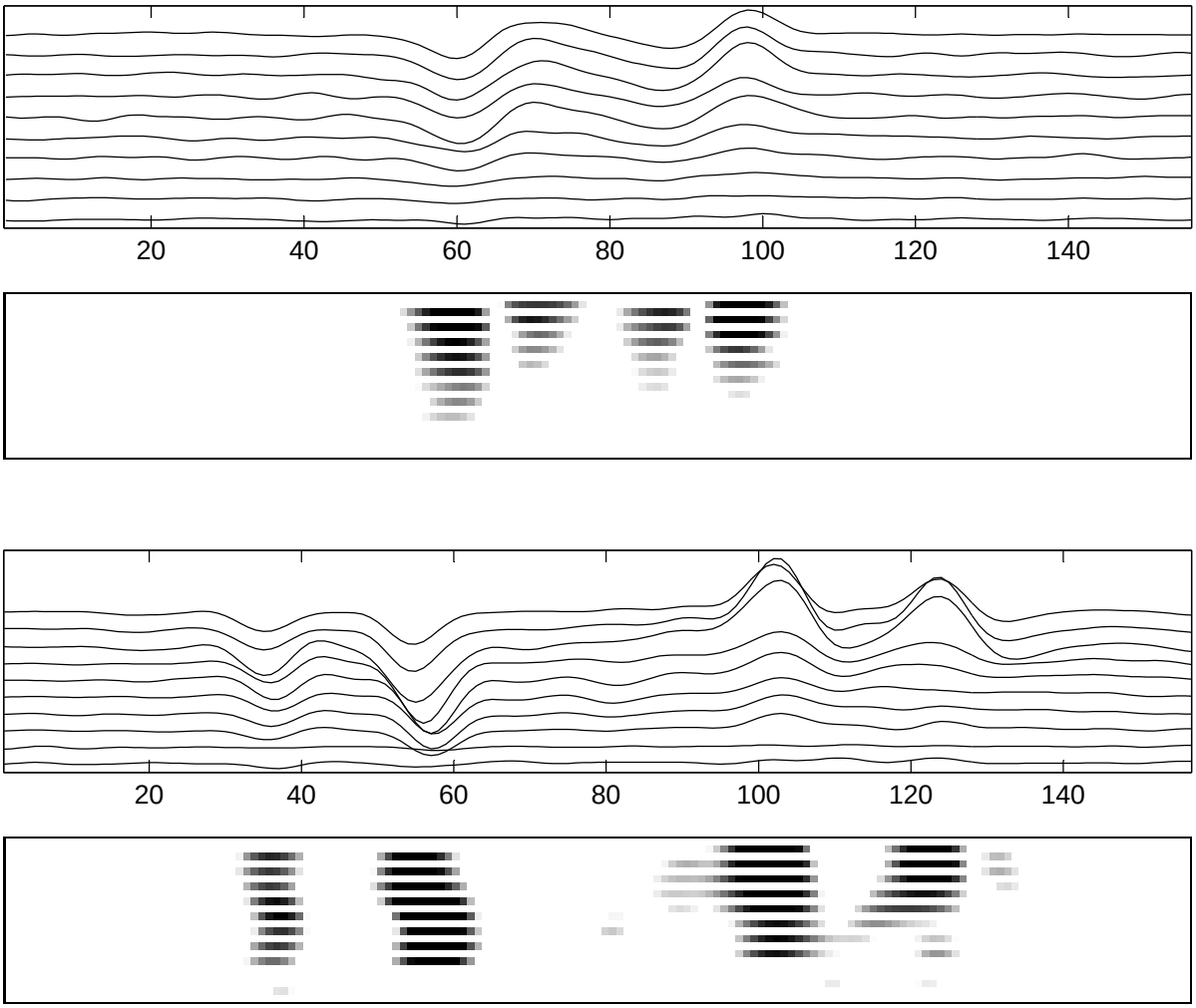


Рис. 3. Электроконтактная и алюмотермитная сварки рельсов
Fig. 3. Flash butt weld and aluminothermic weld

с данными от 10 каналов, Threshold[0:9,0:1] – массив пороговых значений амплитуд полезных сигналов для 10 каналов в соответствии с направлением отклонения от среднего значения, centr – координата (номер строки в EC) середины текущей отметки, A[0:19,0:156] – искомая (изначально состоящая полностью из нулей) bmp-матрица размером 20 строк на 157 столбцов в градации серого цвета, которая возвращается в качестве результата работы функции.

```
def Image_Preparation(EC, Threshold, centr, A):
    (ec_row, ec_col) = EC.shape
    start = int(centr - 78)
    finis = int(centr + 78)
    for i in range(1, 10 + 1):
        n = int(0)
        for j in range(start, finis + 1):
            n = int(n + 1)
            if j >= 1 and j <= ec_row:
```

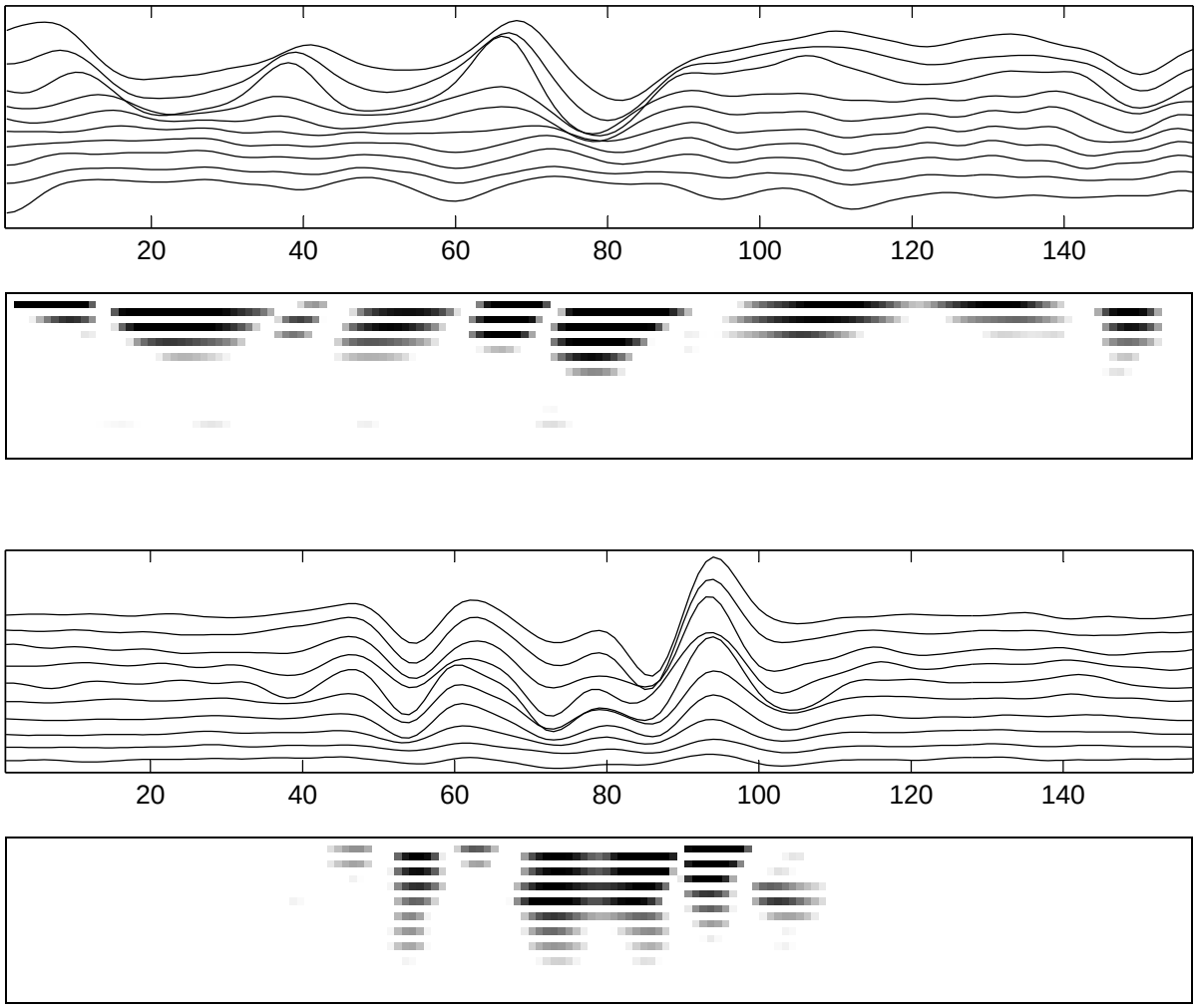


Рис. 4. Дефекты на поверхности головки рельса
Fig. 4. Defects of a rail head surface

```
if EC[j - 1, i - 1] > Threshold[i - 1, 1 - 1]:
    mu = (Threshold[i - 1, 1 - 1] + Threshold[i - 1, 2 - 1]) / 2
    rslt = (EC[j - 1, i - 1] - mu) / (Threshold[i - 1, 1 - 1] - mu)
    rslt = 20 * math.log10(rslt)
    rslt = 10 * rslt
    rslt = min(rslt, 255)
    rslt = max(rslt, 1)
    A[i * 2 - 1 - 1, n - 1] = int(rslt + 0.5)
if EC[j - 1, i - 1] < Threshold[i - 1, 2 - 1]:
    mu = (Threshold[i - 1, 1 - 1] + Threshold[i - 1, 2 - 1]) / 2
    rslt = (EC[j - 1, i - 1] - mu) / (Threshold[i - 1, 2 - 1] - mu)
    rslt = 20 * math.log10(rslt)
    rslt = 10 * rslt
    rslt = min(rslt, 255)
    rslt = max(rslt, 1)
    A[i * 2 - 1, n - 1] = int(rslt + 0.5)
```

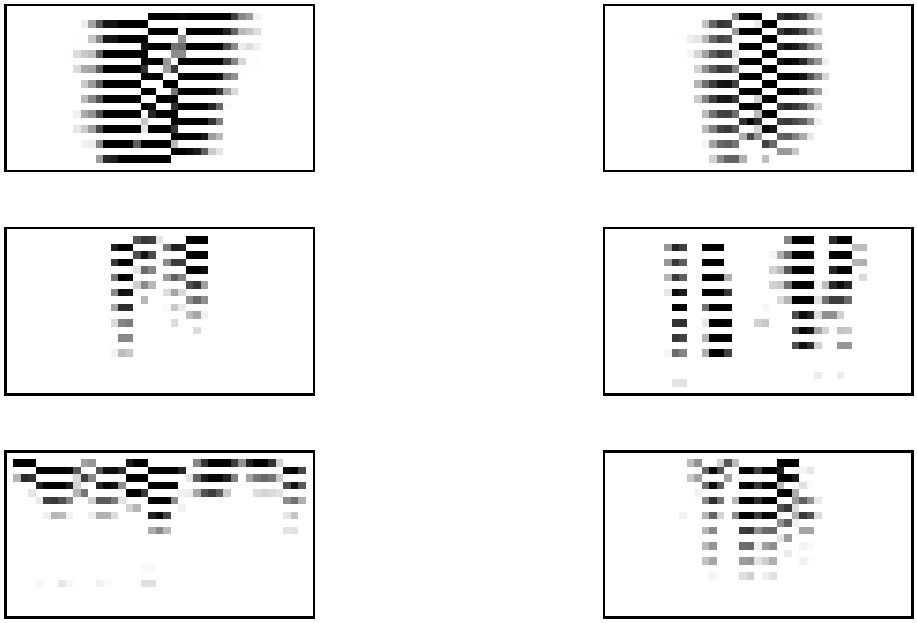


Рис. 5. Сжатые графические образы конструктивных элементов и дефектов
Fig. 5. Shrunk graphic images of structural elements and defects

Следующая функция на языке Python 3 сжимает графический образ отметки (bmp-матрицу A) до размера 20 строк на 39 столбцов. Происходит подготовка итоговой матрицы B (сжатого в четыре раза по строкам графического образа отметки, полученного на предыдущей стадии).

```
def Image_Compression(A, B):
    for i in range(1, 20 + 1):
        mxA = 0
        mxA = max(mxA, A[i - 1, 77 - 1])
        mxA = max(mxA, A[i - 1, 78 - 1])
        mxA = max(mxA, A[i - 1, 79 - 1])
        mxA = max(mxA, A[i - 1, 80 - 1])
        mxA = max(mxA, A[i - 1, 81 - 1])
        B[i - 1, 20 - 1, 1 - 1] = int(mxA)
        for j in range(1, 19 + 1):
            mxA = 0
            mxA = max(mxA, A[i - 1, j * 4 - 3 - 1])
            mxA = max(mxA, A[i - 1, j * 4 - 2 - 1])
            mxA = max(mxA, A[i - 1, j * 4 - 1 - 1])
            mxA = max(mxA, A[i - 1, j * 4 - 0 - 1])
            B[i - 1, j - 1, 1 - 1] = int(mxA)
            mxA = 0
            mxA = max(mxA, A[i - 1, (40 - j) * 4 - 2 - 1])
            mxA = max(mxA, A[i - 1, (40 - j) * 4 - 1 - 1])
            mxA = max(mxA, A[i - 1, (40 - j) * 4 - 0 - 1])
            mxA = max(mxA, A[i - 1, (40 - j) * 4 + 1 - 1])
            B[i - 1, (40 - j) - 1, 1 - 1] = int(mxA)
```

На рис. 2, 3, 4 представлены примеры скошенного и прямого болтовых рельсовых стыков, электроконтактной и алюмотермитной сварок рельсов и дефектов поверхности головки рельса в виде вихретоковых данных (фрагментов выхретоковых дефектограмм) и построенных графических образов (bmp-матриц в градации серого цвета) соответственно. На этих рисунках в графических образах отметок интенсивность серого цвета возрастает с увеличением значения элемента bmp-матрицы, т. е. от 0 (чистый белый цвет) до 255 (полный черный цвет). Рис. 5 содержит сжатые построчно (в четыре раза) графические образы приведенных конструктивных элементов и дефектов.

2. Построение и применение нейронной сети

Подготовка данных для обучения нейронной сети. Для обучения нейронной сети был подготовлен набор данных, состоящий из 48028 графических образов (файлов в формате bmp) размером 20 строк на 39 столбцов в градации серого цвета. Этот набор данных содержит в себе следующие три выборки, каждая из которых разбита на четыре класса – три класса файлов для конструктивных элементов и один для условных дефектов. Обучающая выборка (training set) используется непосредственно для обучения нейронной сети. С помощью проверочной выборки (validation set) в процессе обучения сети осуществляется оценка его качества. Итоговая оценка качества обучения сети после завершения процесса обучения проводится на тестовой выборке (test set).

В подготовленном наборе данных наряду с bmp-файлами, которые были созданы по реальным образам конструктивных элементов и условных дефектов, представленным на вихретоковых дефектограммах, содержатся также и искусственные образцы, сгенерированные на основе реальных по принципу симметрии. При построении искусственного образца в графическом образе реальной отметки (bmp-матрице) меняются местами четные и нечетные строки соответствующих каналов, а их данные (элементы строк) переписываются в обратном порядке.

В обучающей выборке находятся 16287 реальных и 8323 сгенерированных образца конструктивных элементов трех классов и 11417 реальных образцов условных дефектов. Проверочная выборка содержит 4098 сгенерированных образцов конструктивных элементов трех классов и 1903 реальных образца условных дефектов. Тестовая выборка составлена из 4098 сгенерированных образцов конструктивных элементов трех классов и 1902 реальных образцов условных дефектов. Таким образом, проверочная и тестовые выборки занимают лишь по 12,5% от общего числа подготовленных для обучения графических файлов. А все входящие в них образцы конструктивных элементов являются сгенерированными. Наличие в обучающей выборке почти всех реальных образцов связано с необходимостью «запоминания» нейронной сетью всех подтвержденных экспертами образов конструктивных элементов, которые были найдены на реальных вихретоковых дефектограммах.

Создание нейронной сети. Следующий код на языке Python 3 с опорой на библиотеки TensorFlow и Keras позволяет сформировать нейронную сеть, состоящую из трех слоев, расположенных последовательно друг за другом. Входной слой содержит 780 входных нейронов, внутренний слой имеет 800 нейронов и выходной слой

состоит всего лишь из 4-х нейронов (по числу классов распознавания). Все нейроны внутреннего слоя соединяются со всеми нейронами входного слоя, а все нейроны выходного слоя – со всеми нейронами внутреннего слоя. Начальные значения весовых коэффициентов нейронов назначаются случайным образом в соответствии с нормальным распределением вероятностей. Функция активации нейронов внутреннего слоя – ReLu (Rectified Linear Unit $f(x) = \max(0, x)$). В качестве функции активации нейронов выходного слоя используется нормализованная экспоненциальная функция SoftMax, при которой сумма значений всех выходных нейронов равняется 1. Эта функция позволяет выдавать результат в виде набора вероятностей принадлежности распознаваемого объекта тому или иному классу (искомый класс определяется по большему значению из набора).

Слой регуляризации служит для борьбы с проблемой переобучения сети (адаптации весов только к обучающей выборке), который выключает нейроны внутреннего слоя из процесса обучения с вероятностью 0,25 (на один нейрон) при каждом новом объекте, подаваемом на сеть для обучения.

Метод обучения сети – метод стохастического градиентного спуска в модификации Adam. Мера ошибки при обучении – перекрестная энтропия (для четырех категорий). Метрика оценки результатов обучения – точность (accuracy, отношение количества правильно классифицированных объектов к общему числу классифицируемых объектов).

```
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense
img_width, img_height = 39, 20 # Размеры изображения
input_shape = (img_height, img_width, 1) # Размерность тензора
model = Sequential() # создание последовательной модели нейронной сети
model.add(Flatten(input_shape=input_shape)) # добавление входного слоя нейронов
# добавление полносвязного внутреннего слоя нейронов
model.add(Dense(800, input_dim=780, activation='relu', kernel_initializer='normal'))
model.add(Dropout(0.25))  # добавление слоя регуляризации
# добавление выходного слоя из четырех нейронов
model.add(Dense(4, activation='softmax', kernel_initializer='normal'))
# компиляция модели
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Обучение нейронной сети. Ниже приводится код на языке Python 3, с помощью которого происходит обучение нейронной сети для распознавания конструктивных элементов и условных дефектов. Данные для обучения, проверки и тестирования подаются порциями по 32 изображения из соответствующих каталогов (для этого задействованы генераторы изображений библиотеки Keros с размером мини-выборки, равным 32). При этом осуществляется нормализация данных – значения элементов bmp-матриц делятся на 255 (приводятся к вещественным числам от 0 до 1). Количество эпох обучения – 400.

Время, затраченное на обучение описанной модели нейронной сети, составило около 3-х часов. Обучение сети проводилось на компьютере с центральным процессором Intel Core i7-4770K CPU 3.50 GHz и оперативной памятью 8 GB. В результате обучения была достигнута следующая точность (accuracy):

на данных для обучения – 99,83%,

на данных для проверки – 99,45%,

на тестовых данных – 99,23%.

```
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
epochs = 400     # количество эпох обучения
batch_size = 32  # размер мини-выборки
nb_train_samples = (8112 + 11417 + 6640 + 9858) # количество изображений для обучения
nb_validation_samples = (1352 + 1903 + 1104 + 1642) # кол-во изображений для проверки
nb_test_samples = (1352 + 1902 + 1104 + 1642)   # кол-во изображений для тестирования
# создание генератора изображений с нормализацией данных
datagen = ImageDataGenerator(rescale=1. / 255)
# генерация данных для обучения (на основе изображений из каталога)
train_generator = datagen.flow_from_directory(
    train_dir, # каталог с данными для обучения
    target_size=(img_height, img_width), # размер изображения
    color_mode='grayscale', # изображение в градации серого цвета
    batch_size=batch_size, # размер мини-выборки
    class_mode='categorical')
# генерация данных для проверки (на основе изображений из каталога)
val_generator = datagen.flow_from_directory(
    val_dir, # каталог с данными для проверки
    target_size=(img_height, img_width), # размер изображения
    color_mode='grayscale', # изображение в градации серого цвета
    batch_size=batch_size, # размер мини-выборки
    class_mode='categorical')
# генерация данных для тестирования (на основе изображений из каталога)
test_generator = datagen.flow_from_directory(
    test_dir, # каталог с данными для тестирования
    target_size=(img_height, img_width), # размер изображения
    color_mode='grayscale', # изображение в градации серого цвета
    batch_size=batch_size, # размер мини-выборки
    class_mode='categorical')
# обучение модели нейронной сети с помощью генераторов данных
model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs, # количество эпох обучения
    validation_data=val_generator,
    validation_steps=nb_validation_samples // batch_size)
# оценка качества обучения на тестовых данных
scores = model.evaluate_generator(test_generator, nb_test_samples // batch_size)
```

Распознавание графических образов конструктивных элементов. Отметим, что в каждом из каталогов train_dir, val_dir и test_dir данные (картинки) четырех классов находились в соответствующих папках: для графических образов алюмотермитной сварки была отведена папка Alt, для образов электроконтактной сварки – папка Wfb, для болтовых стыков – Gap, а для условных дефектов – Def. Поэтому при обучении нейронной сети классы образцов получили имена в соответствии с названиями этих папок.

С применением следующего кода на языке Python 3 происходит распознавание графических образов четырех классов с помощью обученной нейронной сети. На вход сети подается подготовленный и нормализованный графический образ B найденной на дефектограмме отметки, выходом является название класса, к которому принадлежит исследуемый образ. Нейронная сеть выдает набор значений вероятностей принадлежности объекта к соответствующему классу. Класс объекта выбирается по максимальному значению из этого набора.

```
import numpy
classes = ['alt', 'def', 'gap', 'wfb']
B /= 255 # нормализация матрицы B
B = numpy.expand_dims(B, axis=0) # переход к 'линейному' массиву данных
preds = model.predict(B)  # классификация входного объекта
if classes[preds.argmax()] == 'def':  # расшифровка результата классификации
    mark_result = 'условный дефект'
if classes[preds.argmax()] == 'wfb':  # расшифровка результата классификации
    mark_result = 'электроконтактная сварка'
if classes[preds.argmax()] == 'alt':  # расшифровка результата классификации
    mark_result = 'алюмотермитная сварка'
if classes[preds.argmax()] == 'gap':  # расшифровка результата классификации
    mark_result = 'болтовой стык'
```

Заключение. По результатам обучения можно сделать вывод, что выбранная конфигурация модели нейронной сети является удачной и вполне достаточной для решения поставленной задачи распознавания конструктивных элементов. Построение глубоких (сверточных) нейронных сетей не требуется, так как, с одной стороны, удалось провести явное сжатие исходного графического образа отметки до относительно небольшого размера без существенного влияния на результат распознавания отметок, с другой стороны, образы конструктивных элементов имеют жесткую простую структуру и характерный рисунок, четко выраженный для соответствующего класса. При этом ни масштабирования, ни поворота, ни сдвигов ключевых признаков на изображении не происходит, что делает простую последовательную полносвязную 3-слойную модель нейронной сети достаточной для решения данной задачи распознавания.

Список литературы / References

[1] Кузьмин Е. В., Горбунов О. Е., Плотников П. О., Тюкин В. А., "Об определении уровня полезных сигналов при расшифровке магнитных и вихретоковых дефектограмм", *Модел. и анализ информ. систем*, **24**:6 (2017), 760–771; [Kuzmin E. V., Gorbunov O. E., Plotnikov P. O., Tyukin V. A., "On Finding a Threshold of Useful Signals in the Analysis of Magnetic and Eddy Current Defectograms", *Modeling and Analysis of Information Systems*, **24**:6 (2017), 760–771, (in Russian).]

[2] Кузьмин Е. В., Горбунов О. Е., Плотников П. О., Тюкин В. А., "Эффективный алгоритм определения уровня полезных сигналов при расшифровке магнитных и вихретоковых дефектограмм", *Модел. и анализ информ. систем*, **25**:4 (2018), 382–387; [Kuzmin E. V., Gorbunov O. E., Plotnikov P. O., Tyukin V. A., "An Efficient Algorithm for Finding a Threshold of Useful Signals in the Analysis of Magnetic and Eddy Current Defectograms", *Modeling and Analysis of Information Systems*, **25**:4 (2018), 382–387, (in Russian).]

[3] Марков А. А., Кузнецова Е. А., *Дефектоскопия рельсов. Формирование и анализ сигналов. Кн. 1. Основы*, КультИнформПресс, СПб., 2010; [Markov A. A., Kuznetsova E. A., *Rails flaw detection. Formation and analysis of signals. Book 1. Principles*, KultInformPress, St. Petersburg, 2010, (in Russian).]

[4] Марков А. А., Кузнецова Е. А., *Дефектоскопия рельсов. Формирование и анализ сигналов. Кн. 2. Расшифровка дефектограмм*, Ультра Принт, СПб., 2014; [Markov A. A., Kuznetsova E. A., *Rails flaw detection. Formation and analysis of signals. Book 2. Data interpretation*, Ultra Print, St. Petersburg, 2014, (in Russian).]

[5] Тарабрин В. Ф., Зверев А. В., Горбунов О. Е., Кузьмин Е. В., "О фильтрации данных при автоматической расшифровке дефектограмм АПК «АСТРА»", *В мире неразрушающего контроля*, **64**:2 (2014), 5–9; [Tarabrin V. F., Zverev A. V., Gorbunov O. E.,

Kuzmin E. V., "About Data Filtration of the Defectogram Automatic Interpretation by Hardware and Software Complex "ASTRA"", *NDT World*, **64**:2 (2014), 5–9, (in Russian).]

[6] Simard P. Y., Steinkraus D., Platt J. C., "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis", *7th International Conference on Document Analysis and Recognition (ICDAR-2003), 2-Volume Set*, (3–6 August 2003, Edinburgh, Scotland, UK), IEEE Computer Society, 2003, 958–962.

[7] Goodfellow I., Bengio Y., Courville A., *Deep Learning*, MIT Press, 2016.

[8] Chollet F., *Deep Learning with Python*, Manning Publications Co., 2018.

[9] *TensorFlow* https://www.tensorflow.org/.

Kuzmin E.V., Gorbunov O.E., Plotnikov P.O., Tyukin V.A., Bashkin V.A., "Application of Neural Networks for Recognizing Rail Structural Elements in Magnetic and Eddy Current Defectograms", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 667–679.

DOI: 10.18255/1818-1015-2018-6-667-679

Abstract. To ensure traffic safety of railway transport, non-destructive test of rails is regularly carried out by using various approaches and methods, including magnetic and eddy current flaw detection methods. An automatic analysis of large data sets (defectgrams) that come from the corresponding equipment is an actual problem. The analysis means a process of determining the presence of defective sections along with identifying structural elements of railway tracks on defectograms. This article is devoted to the problem of recognition of rail structural element images in magnetic and eddy current defectograms. Three classes of rail track structural elements are considered: 1) a bolted joint with straight or beveled connection of rails, 2) a butt weld of rails, and 3) an aluminothermic weld of rails. Images that cannot be assigned to these three classes are conditionally considered as defects and are placed in a separate fourth class. For image recognition of structural elements in defectograms a neural network is applied. The neural network is implemented by using the open library TensorFlow. To this purpose each selected (picked out) area of a defectogram is converted into a graphic image in a grayscale with size of 20 x 39 pixels.

Keywords: nondestructive testing, magnetic and eddy current testing, rail flaw detection, automated analysis of defectograms, neural networks

On the authors:
Egor V. Kuzmin, orcid.org/0000-0003-0500-306X, doctor of science, associate professor,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003 Russia, e-mail: kuzmin@uniyar.ac.ru, kuzminev@nddlab.com

Oleg E. Gorbunov, orcid.org/0000-0001-6274-9971, PhD, general director,
Center of Innovative Programming, NDDLab,
144 Soyuznaya str., Yaroslavl, 150008, Russia, e-mail: gorbunovoe@nddlab.com

Petr O. Plotnikov, orcid.org/0000-0001-5687-7969, production engineer,
Center of Innovative Programming, NDDLab,
144 Soyuznaya str., Yaroslavl, 150008, Russia, e-mail: plotnikovpo@nddlab.com

Vadim A. Tyukin, orcid.org/0000-0001-9149-7435, head of software development,
Center of Innovative Programming, NDDLab,
144 Soyuznaya str., Yaroslavl, 150008, Russia, e-mail: tyukinva@nddlab.com

Vladimir A. Bashkin, orcid.org/0000-0002-2534-1026, doctor of science, associate professor,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl, 150003 Russia, e-mail: bashkinva@nddlab.com

Моделирование и анализ информационных систем. Т. 25, №6 (2018), с. 680–691
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 680–691

Вычислительная геометрия
Computational Geometry

DOI: 10.18255/1818-1015-2018-6-680-691

УДК 514.17+517.51+519.6

О некоторых задачах для симплекса и шара в $\mathbb{R}^n$

Невский М. В.

Поступила в редакцию 20 сентября 2018
После доработки 30 октября 2018
Принята к публикации 10 ноября 2018

Аннотация. Пусть C — выпуклое тело, S невырожденный симплекс в $\mathbb{R}^n$. Через τS обозначим образ S при гомотетии относительно центра тяжести S с коэффициентом τ. Под $\xi(C;S)$ понимается минимальное $\tau > 0$, для которого C является подмножеством симплекса τS. По определению, $\alpha(C;S)$ есть минимальное $\tau > 0$, такое что C принадлежит трансляту симплекса τS. Ранее автор доказал, что справедливы равенства $\xi(C;S) = (n+1)\max\limits_{1\le j\le n+1}\max\limits_{x\in C}(-\lambda_j(x)) + 1$ (если $C \not\subset S$), $\alpha(C;S) = \sum\limits_{j=1}^{n+1}\max\limits_{x\in C}(-\lambda_j(x)) + 1$. Здесь λ_j — линейные функции, называемые базисными многочленами Лагранжа симплекса S. Они таковы, что числа $\lambda_j(x),\dots,\lambda_{n+1}(x)$ являются барицентрическими координатами точки $x \in \mathbb{R}^n$. В предыдущих работах автора указанные формулы исследовались в ситуации, когда C представляет собой n-мерный единичный куб $Q_n = [0,1]^n$. В статье рассматривается случай, когда C есть единичный евклидов шар $B_n = \{x : \|x\| \le 1\}$, где $\|x\| = \left(\sum\limits_{i=1}^{n} x_i^2\right)^{1/2}$. Устанавливаются различные соотношения для $\xi(B_n;S)$ и $\alpha(B_n;S)$, а также приводится их геометрическая интерпретация. Например, если $\lambda_j(x) = l_{1j}x_1 + \ldots + l_{nj}x_n + l_{n+1,j}$, то $\alpha(B_n;S) = \sum\limits_{j=1}^{n+1}\left(\sum\limits_{i=1}^{n} l_{ij}^2\right)^{1/2}$. Минимальное возможное значение каждой из величин $\xi(B_n;S)$, $\alpha(B_n;S)$ для $S \subset B_n$ равно n и соответствует правильному симплексу, вписанному в B_n. Даётся сравнение с результатами, полученными ранее для $C = Q_n$.

Ключевые слова: n-мерный симплекс, n-мерный шар, гомотетия, коэффициент поглощения

Для цитирования: Невский М. В., "О некоторых задачах для симплекса и шара в $\mathbb{R}^n$", *Моделирование и анализ информационных систем*, **25**:6 (2018), 680–691.

Об авторах:
Невский Михаил Викторович, orcid.org/0000-0002-6392-7618, доктор физ.-мат. наук, доцент,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: mnevsk55@yandex.ru

1. Определения и предварительные сведения

Начнём с основных определений и обозначений. Всюду далее $n \in \mathbb{N}$. Элемент $x \in \mathbb{R}^n$ будем записывать в виде $x = (x_1, \ldots, x_n)$. Положим

$$\|x\| = \sqrt{(x,x)} = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2},$$

$$B\left(x^{(0)}; \varrho\right) := \{x \in \mathbb{R}^n : \|x - x^{(0)}\| \le \varrho\} \quad \left(x^{(0)} \in \mathbb{R}^n, \varrho > 0\right),$$

$$B_n := B(0;1), \quad Q_n := [0,1]^n, \quad Q_n' := [-1,1]^n.$$

Пусть C — выпуклое тело в $\mathbb{R}^n$. Через τC обозначим результат гомотетии C относительно центра тяжести с коэффициентом τ. Для невырожденного n-мерного симплекса S введём в рассмотрение величину $\xi(C;S) := \min\{\sigma \ge 1 : C \subset \sigma S\}$. Эту величину мы называем *коэффициентом поглощения (absorption index) выпуклого тела C симплексом S*. Обозначим через $\alpha(C;S)$ минимальное $\tau > 0$, для которого выпуклое тело C принадлежит трансляту симплекса τS. Под $\mathrm{ver}(G)$ будем понимать совокупность вершин выпуклого многогранника G.

Обозначим вершины симплекса S через $x^{(j)} = \left(x_1^{(j)}, \ldots, x_n^{(j)}\right)$, $1 \le j \le n+1$. Матрица

$$\mathbf{A} := \begin{pmatrix} x_1^{(1)} & \ldots & x_n^{(1)} & 1 \\ x_1^{(2)} & \ldots & x_n^{(2)} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(n+1)} & \ldots & x_n^{(n+1)} & 1 \end{pmatrix}$$

является невырожденной. Пусть $\mathbf{A}^{-1} = (l_{ij})$. Линейные многочлены $\lambda_j(x) = l_{1j}x_1 + \ldots + l_{nj}x_n + l_{n+1,j}$, коэффициенты которых составляют столбцы $\mathbf{A}^{-1}$, обладают свойством $\lambda_j\left(x^{(k)}\right) = \delta_j^k$, где δ_j^k — символ Кронекера. Мы называем λ_j *базисными многочленами Лагранжа, соответствующими S*. Числа $\lambda_j(x)$ являются барицентрическими координатами точки $x \in \mathbb{R}^n$ относительно S. Симплекс S задаётся системой линейных неравенств $\lambda_j(x) \ge 0$. Подробнее о многочленах λ_j см. [3, гл. 1].

Равенство $\xi(C;S) = 1$ эквивалентно включению $C \subset S$. В [2] (см. также [3, § 1.3]) установлено, что если $C \not\subset S$, то

$$\xi(C;S) = (n+1) \max_{1 \le j \le n+1} \max_{x \in C}(-\lambda_j(x)) + 1. \tag{1}$$

Соотношение

$$\max_{x \in C}(-\lambda_1(x)) = \ldots = \max_{x \in C}(-\lambda_{n+1}(x)) \tag{2}$$

эквивалентно тому, что симплекс $\xi(C;S)S$ описан вокруг C. В случае $C = Q_n$ равенство (1) приводится к виду

$$\xi(Q_n;S) = (n+1) \max_{1 \le j \le n+1} \max_{x \in \mathrm{ver}(Q_n)}(-\lambda_j(x)) + 1,$$

а (2) сводится к соотношению

$$\max_{x \in \mathrm{ver}(Q_n)}(-\lambda_1(x)) = \ldots = \max_{x \in \mathrm{ver}(Q_n)}(-\lambda_{n+1}(x)). \tag{3}$$

Для любых C и S выполняется $\xi(C;S) \geq \alpha(C;S)$. При этом $\xi(C;S) = \alpha(C;S)$ тогда и только тогда, когда симплекс $\xi(C;S)S$ описан вокруг выпуклого тела C. Последнее равносильно соотношению (2), а для $C = Q_n$ — соотношению (3).

В [4] (см. также [3, § 1.4]) доказано равенство

$$\alpha(C;S) = \sum_{j=1}^{n+1} \max_{x \in C}(-\lambda_j(x)) + 1, \tag{4}$$

Если $C = Q_n$, то эта формула может быть записана в более геометрическом виде:

$$\alpha(Q_n;S) = \sum_{i=1}^{n} \frac{1}{d_i(S)}. \tag{5}$$

Здесь $d_i(S)$ — *i-й осевой диаметр симплекса S,* представляющий собой максимальную длину отрезка, содержащегося в S и параллельного i-й координатной оси. Равенство (5) установлено в [11]. Если $S \subset Q_n$, то $d_i(S) \leq 1$, поэтому для таких симплексов (5) даёт

$$\xi(Q_n;S) \geq \alpha(Q_n;S) = \sum_{i=1}^{n} \frac{1}{d_i(S)} \geq n. \tag{6}$$

Как ранее доказал автор (см. [2]),

$$\frac{1}{d_i(S)} = \frac{1}{2} \sum_{j=1}^{n+1} |l_{ij}|. \tag{7}$$

Из (5) и (7) следует соотношение

$$\alpha(Q_n;S) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n+1} |l_{ij}|. \tag{8}$$

Заметим, что $\alpha(C;S)$ не меняется при параллельном переносе множеств и при $\tau > 0$ верно $\alpha(\tau C;S) = \tau\alpha(C;S)$. Поскольку куб $Q'_n = [-1,1]^n$ есть транслят куба $2Q_n$, то при замене Q_n на Q'_n из (8) получается ещё более простая формула:

$$\alpha(Q'_n;S) = \sum_{i=1}^{n} \sum_{j=1}^{n+1} |l_{ij}|. \tag{9}$$

Положим

$$\xi_n := \min\{\xi(Q_n;S) : \ S \text{ — } n\text{-мерный симплекс}, S \subset Q_n, \operatorname{vol}(S) \neq 0\}.$$

Оценкам чисел ξ_n посвящены многие работы автора, а также автора и А. Ю. Ухалова (см., например, статьи [1], [2], [5], [6], [7], [8], [12] и монографию [3]). Всегда $n \leq \xi_n < n+1$. Точные значения ξ_n к настоящему моменту известны для $n = 2, 5, 9$ и бесконечной совокупности нечётных n, для каждого из которых существует матрица Адамара порядка $n+1$. При $n \neq 2$ каждое известное ξ_n равно n, а

$\xi_2 = 1 + \frac{3\sqrt{5}}{5} = 2.34\ldots$ До сих пор неизвестно, существует ли хотя бы одно чётное n со свойством $\xi_n = n$. Относительно чисел ξ_n есть и другие открытые вопросы.

В настоящей статье обсуждаются аналоги введённых выше характеристик и соотношений для симплекса и евклидова шара. Замена куба на шар делает многие рассматриваемые вопросы существенно более простыми. Однако геометрическая интерпретация отмеченных общих результатов и в этом частном случае имеет определённый интерес. Кроме того, мы отметим новые приложения базисных многочленов Лагранжа.

Числовые характеристики, связывающие симплексы с подмножествами $\mathbb{R}^n$, имеют важные приложения к оцениванию норм проекторов, возникающих при полиномиальной интерполяции функций многих переменных. Этот подход и соответствующая методология подробно описаны в [3]. В последнее время в круге этих вопросов удалось эффективно применить, кроме аналитических, и компьютерные методы (см., например, [5], [6], [8], [12]).

2. Величина $\alpha(B_n; S)$

Инрадиусом n-мерного симплекса S (inradius) называется максимальный радиус шара, содержащегося в S. Центр этого единственного максимального шара называется *инцентром S (incenter)*. Граница максимального в симплексе шара — сфера, которая имеет с каждой $(n-1)$-мерной гранью единственную общую точку. Под *внешним радиусом, или циркумрадиусом S (circumradius)*, будем понимать минимальный радиус шара, содержащего S. Граница этого единственного минимального шара не обязательно содержит все вершины S. Это имеет место тогда и только тогда, когда центр минимального шара принадлежит S.

Инрадиус r и внешний радиус R симплекса S связаны *неравенством Эйлера*

$$R \geq nr. \tag{10}$$

Равенство в (10) имеет место тогда и только тогда, когда S — правильный симплекс. По поводу доказательств n-мерного неравенства Эйлера, а также его истории и обобщений см., например, [10], [13], [14].

По поводу неравенства (10) отметим аналог следующего свойства, справедливого для параллелепипедов (см. [11], [3, § 1.8]). *Пусть S — невырожденный симплекс, D, D^* — параллелепипеды в $\mathbb{R}^n$. Предположим, что D^* есть результат гомотетии D с коэффициентом $\tau > 1$. Если $D \subset S \subset D^*$, то $\tau \geq n$.* Это утверждение верно и для шаров. Нетрудно видеть, что неравенство Эйлера эквивалентно следующему свойству. *Если B — шар радиуса r_1, B^* — шар радиуса r_2 и $B \subset S \subset B^*$, то $r_1 \leq nr_2$. Равенство выполняется тогда и только тогда, когда S — правильный симплекс, вписанный в B^*, а B — шар, вписанный в S.* Другая эквивалентная форма этих утверждений — приводимая ниже теорема 2 (см. замечание после её доказательства).

Пусть $x^{(1)}, \ldots, x^{(n+1)}$ — вершины, $\lambda_1, \ldots, \lambda_{n+1}$ — базисные многочлены Лагранжа невырожденного симплекса $S \subset \mathbb{R}^n$ (см. п. 1). Через Γ_j обозначим $(n-1)$-мерную гиперплоскость, задаваемую уравнением $\lambda_j(x) = 0$, через Σ_j — $(n-1)$-мерную грань симплекса, содержащуюся в Γ_j, через h_j — высоту, опущенную из вершины $x^{(j)}$

на Γ_j, через r — инрадиус S. Пусть σ_j — $(n-1)$-мера Σ_j и $\sigma := \sum\limits_{j=1}^{n+1} \sigma_j$. Обозначим $a_j := \{l_{1j}, \ldots, l_{nj}\}$. Этот вектор ортогонален Γ_j и направлен в полупространство, содержащее вершину $x^{(j)}$. Ясно, что

$$\lambda_j(x) = l_{1j}x_1 + \ldots + l_{nj}x_n + l_{n+1,j} = (a_j, x) + l_{n+1,j} = (a_j, x) + \lambda_j(0).$$

Теорема 1. *Справедливы равенства*

$$\alpha(B_n; S) = \sum_{j=1}^{n+1} \left(\sum_{i=1}^{n} l_{ij}^2 \right)^{1/2}, \tag{11}$$

$$\alpha(B_n; S) = \sum_{j=1}^{n+1} \frac{1}{h_j}, \tag{12}$$

$$\alpha(B_n; S) = \frac{1}{r}, \tag{13}$$

$$\alpha(B_n; S) = \frac{\sigma}{n\mathrm{vol}(S)}. \tag{14}$$

Доказательство. Получим эти попарно эквивалентные равенства, двигаясь сверху вниз. Сначала запишем

$$\alpha(B_n; S) = \sum_{j=1}^{n+1} \max_{x \in B} (-\lambda_j(x)) + 1. \tag{15}$$

Формула (15) есть частный случай (4) в ситуации $C = B_n$. По неравенству Коши имеем $|(a_j, x)| \le \|a_j\|\|x\|$, откуда

$$-\|a_j\|\|x\| \le (a_j, x) \le \|a_j\|\|x\|, \tag{16}$$

$$-\|a_j\|\|x\| - \lambda_j(0) \le -\lambda_j(x) \le \|a_j\|\|x\| - \lambda_j(0).$$

Так как верхняя и нижняя границы в неравенстве (16) достигаются, то

$$\max_{x \in B_n} (-\lambda_j(x)) = \max_{\|x\| \le 1} (-\lambda_j(x)) = \|a_j\| - \lambda_j(0).$$

Поэтому

$$\alpha(B_n; S) = \sum_{j=1}^{n+1} \max_{x \in B_n} (-\lambda_j(x)) + 1 = \sum_{j=1}^{n+1} \|a_j\| - \sum_{j=1}^{n+1} \lambda_j(0) + 1 = \sum_{j=1}^{n+1} \left(\sum_{i=1}^{n} l_{ij}^2 \right)^{1/2}.$$

Мы воспользовались тем, что $\sum\limits_{j=1}^{n+1} \lambda_j(0) = 1$. Поскольку $\lambda_j\left(x^{(j)}\right) = 1$, имеем

$$h_j = \mathrm{dist}\left(x^{(j)}; \Gamma_j\right) = \frac{\left|\lambda_j\left(x^{(j)}\right)\right|}{\|a_j\|} = \frac{1}{\|a_j\|} = \frac{1}{\left(\sum\limits_{i=1}^{n} l_{ij}^2 \right)^{1/2}}.$$

Следовательно,

$$\alpha(B_n; S) = \sum_{j=1}^{n+1} \left(\sum_{i=1}^{n} l_{ij}^2 \right)^{1/2} = \sum_{j=1}^{n+1} \frac{1}{h_j}.$$

Итак, мы получили (11) и (12).

Докажем (13). Шар B есть подмножество транслята симплекса $\alpha(B_n; S)S$. Значит, транслят шара $\frac{1}{\alpha(B_n;S)} B_n$ содержится в S. Так как максимальный радиус шара, принадлежащего S, равен r, то $\frac{1}{\alpha(B_n;S)} \leq r$, т. е. $\alpha(B_n; S) \geq \frac{1}{r}$. Для получения противоположного неравенства обозначим через B' шар радиуса r, вписанный в S. Тогда шар $B_n = \frac{1}{r} B'$ содержится в некотором трансляте симплекса $\frac{1}{r} S$. По определению $\alpha(B_n; S)$ это даёт $\alpha(B_n; S) \leq \frac{1}{r}$. Стало быть, верно равенство $\alpha(B_n; S) = \frac{1}{r}$.

Наконец, для установления (14) достаточно привлечь (13) и формулу $\mathrm{vol}(S) = \frac{1}{n} \sigma r$. Последнее равенство получается из обычной формулы для объёма симплекса после разбиения S на $n+1$ симплекс, каждый из которых имеет вершину в центре вписанного шара и опирается на одну из граней Σ_j.

□

Следствие 1. *Имеет место равенство*

$$\frac{1}{r} = \sum_{j=1}^{n+1} \frac{1}{h_j}.$$

Для доказательства достаточно применить (12) и (13). Представляется интересным, что это красивое геометрическое соотношение, которое может быть получено и непосредственно, оказывается эквивалентным общей формуле для $\alpha(C; S)$ в случае, когда выпуклое тело C является евклидовым единичным шаром.

Следствие 2. *Инрадиус r и инцентр z симплекса S могут быть вычислены по формулам*

$$r = \frac{1}{\sum\limits_{j=1}^{n+1} \left(\sum\limits_{i=1}^{n} l_{ij}^2 \right)^{1/2}}, \tag{17}$$

$$z = \frac{1}{\sum\limits_{j=1}^{n+1} \left(\sum\limits_{i=1}^{n} l_{ij}^2 \right)^{1/2}} \sum_{j=1}^{n+1} \left(\sum_{i=1}^{n} l_{ij}^2 \right)^{1/2} x^{(j)}. \tag{18}$$

Точка касания шара $B(z; r)$ с $(n-1)$-мерной гранью Σ_k симплекса S имеет вид

$$y^{(k)} = \frac{1}{\sum\limits_{j=1}^{n+1} \left(\sum\limits_{i=1}^{n} l_{ij}^2 \right)^{1/2}} \left[\sum_{j=1}^{n+1} \left(\sum_{i=1}^{n} l_{ij}^2 \right)^{1/2} x^{(j)} - \frac{1}{\left(\sum\limits_{i=1}^{n} l_{ik}^2 \right)^{1/2}} (l_{1k}, \ldots, l_{nk}) \right]. \tag{19}$$

Доказательство. Равенство (17) немедленно следует из (11) и (13). Для получения (18) заметим, что

$$r = \mathrm{dist}(z; \Gamma_j) = \frac{|\lambda_j(z)|}{\|a_j\|}.$$

Так как точка z лежит строго внутри S, каждая её барицентрическая координата $\lambda_j(z)$ положительна. Поэтому $\lambda_j(z) = r\|a_j\|$. Следовательно,

$$z = \sum_{j=1}^{n+1} \lambda_j(z) x^{(j)} = r \sum_{j=1}^{n+1} \|a_j\| x^{(j)},$$

что совпадает с (18). Наконец, поскольку вектор $a_k = \{l_{1k}, \ldots, l_{nk}\}$ ортогонален грани Σ_k и направлен от неё внутрь симплекса, единственная общая точка шара $B(z; r)$ и этой грани имеет вид

$$y^{(k)} = z - \frac{r}{\|a_k\|} a_k = r \left(\sum_{j=1}^{n+1} \|a_j\| x^{(j)} - \frac{1}{\|a_k\|} a_k \right).$$

Последнее равенство равносильно (19). □

Формулу (11) интересно сравнить с формулой (9) для $\alpha(Q'_n; S)$. Так как шар B_n есть подмножество куба $Q'_n = [-1, 1]^n$, то $\alpha(B_n; S) \leq \alpha(Q'_n; S)$. Аналитически же это неравенство следует из оценки

$$\left(\sum_{i=1}^{n} l_{ij}^2 \right)^{1/2} \leq \sum_{i=1}^{n} |l_{ij}|.$$

Для произвольных $x^{(0)}$ и $\varrho > 0$ величина $\alpha\left(B(x^{(0)}; \varrho); S\right)$ может быть вычислена с помощью формул теоремы 1 и равенства $\alpha(B(x^{(0)}; \varrho); S) = \varrho\alpha(B_n; S)$.

Если $S \subset Q_n$, то все осевые диаметры $d_i(S) \leq 1$. Формула (5) немедленно даёт $\alpha(Q_n; S) \geq n$. Если же для некоторого S выполняется $\alpha(Q_n; S) = n$, то каждый осевой диаметр S равен 1. Следующая теорема даёт аналоги этих утверждений для симплексов, содержащихся в шаре.

Теорема 2. *Если $S \subset B_n$, то $\alpha(B_n; S) \geq n$. Равенство $\alpha(B_n; S) = n$ выполняется тогда и только тогда, когда S — правильный симплекс, вписанный в B_n.*

Доказательство. По определению $\alpha(B_n; S)$, шар B_n содержится в трансляте симплекса $\alpha(B_n; S)S$. Поэтому некоторый транслят B' шара $\frac{1}{\alpha(B_n;S)} B_n$ есть подмножество S. Итак, $B' \subset S \subset B_n$. Так как радиус B' равен $\frac{1}{\alpha(B_n;S)}$, то для инрадиуса r и внешнего радиуса R симплекса S справедливы неравенства $\frac{1}{\alpha(B_n;S)} \leq r$, $R \leq 1$. Применяя неравенство Эйлера $R \geq nr$, запишем

$$\frac{1}{\alpha(B_n; S)} \leq r \leq \frac{R}{n} \leq \frac{1}{n}, \tag{20}$$

откуда $\alpha(B_n; S) \geq n$.

Если же $\alpha(B_n; S) = n$, то левая величина в (20) совпадает с правой, значит, все неравенства в этой цепочке обращаются в равенства. Это даёт $R = 1$, $r = \frac{1}{n}$. Так как в этом случае неравенство Эйлера (10) обращается в равенство, S является правильным симплексом, вписанным в B_n. Наоборот, если S — правильный симплекс, вписанный в B_n, то для него, очевидно, $r = \frac{1}{n}$, поэтому $\alpha(B_n; S) = \frac{1}{r} = n$. □

Итак, теорема 2 следует из неравенства Эйлера (10). Фактически же эти утверждения эквивалентны. Действительно, пусть S — произвольный n-мерный симплекс, r — инрадиус, R — внешний радиус S. Обозначим через B шар радиуса R, содержащий S. Тогда некоторый транслят S' симплекса $\frac{1}{R}S$ содержится в шаре $\frac{1}{R}B = B_n$. По теореме 1, $\alpha(B_n; S')$ есть величина, обратная инрадиусу S', т. е. равна $\frac{R}{r}$. Предположим теперь, что теорема 2 верна, и применим её к симплексу $S' \subset B_n$. Это даст $\alpha(B_n; S') = \frac{R}{r} \geq n$, т. е. неравенство (10). Если же $R = nr$, то $\alpha(B_n; S') = n$. Из теоремы 2 получаем, что в этом случае S', а с ним и S являются правильными симплексами.

Как следует из (6), минимальное значение $\alpha(Q_n; S)$ для $S \subset Q_n$ также равно n. Это минимальное значение достигается на тех и только тех $S \subset Q_n$, для которых каждый осевой диаметр $d_i(S)$ равен 1. Указанным свойством обладают, например, симплексы максимального объёма в Q_n (подробнее см. [3]), но при $n > 2$ не только они.

3. Величина $\xi(B_n; S)$

В этом пункте приводится формула для коэффициента поглощения симплексом евклидова шара. Мы применяем предыдущие обозначения.

Теорема 3. *Пусть S — невырожденный симплекс в $\mathbb{R}^n$, $x^{(0)} \in \mathbb{R}^n$, $\varrho > 0$. Если $B\left(x^{(0)}; \varrho\right) \not\subset S$, то*

$$\xi\left(B\left(x^{(0)}; \varrho\right); S\right) = (n+1) \max_{1 \leq j \leq n+1} \left[\varrho \left(\sum_{i=1}^{n} l_{ij}^2 \right)^{1/2} - \sum_{i=1}^{n} l_{ij} x_i^{(0)} - l_{n+1,j} \right] + 1. \tag{21}$$

В частности, если $B_n \not\subset S$, то

$$\xi(B_n; S) = (n+1) \max_{1 \leq j \leq n+1} \left[\left(\sum_{i=1}^{n} l_{ij}^2 \right)^{1/2} - l_{n+1,j} \right] + 1. \tag{22}$$

Доказательство. Применим общую формулу (1) для случая $C = B\left(x^{(0)}; \varrho\right)$. Неравенство Коши даёт

$$-\|a_j\|\|x - x^{(0)}\| \leq (a_j, x - x^{(0)}) \leq \|a_j\|\|x - x^{(0)}\|. \tag{23}$$

Для $\|x - x^{(0)}\| \leq \varrho$ имеем

$$-\varrho\|a_j\| \leq (a_j, x) - (a_j, x^{(0)}) \leq \varrho\|a_j\|,$$

$$-\lambda_j(x) = -(a_j, x) - l_{n+1,j} \leq \varrho\|a_j\| - (a_j, x^{(0)}) - l_{n+1,j}.$$

Так как верхняя и нижняя границы в неравенстве (23) достигаются, то

$$\max_{\|x - x^{(0)}\| \leq \varrho} (-\lambda_(x)) = \varrho \left(\sum_{i=1}^{n} l_{ij}^2 \right)^{1/2} - \sum_{i=1}^{n} l_{ij} x_i^{(0)} - l_{n+1,j}.$$

С учётом этого

$$\xi\left(B\left(x^{(0)};\varrho\right);S\right)=(n+1)\max_{1\le j\le n+1,\|x-x^{(0)}\|\le\varrho}(-\lambda_j(x))+1=$$

$$=(n+1)\max_{1\le j\le n+1}\left[\varrho\left(\sum_{i=1}^{n}l_{ij}^2\right)^{1/2}-\sum_{i=1}^{n}l_{ij}x_i^{(0)}-l_{n+1,j}\right]+1,$$

и мы пришли к (21). Равенство (22) получается из (21) при $x^{(0)}=0,\varrho=1$. □

4. Равенство $\beta_n=n$. Комментарии

Теорема 4. *Если $S\subset B_n$, то $\xi(B_n;S)\ge n$. Равенство $\xi(B_n;S)=n$ выполняется тогда и только тогда, когда S — правильный симплекс, вписанный в B_n.*

Доказательство. Утверждение теоремы немедленно следует из теоремы 2 и неравенства $\xi(B_n;S)\ge\alpha(B_n;S)$. Приведём непосредственное доказательство без применения неравенства Эйлера, с помощью которого получена оценка $\alpha(B_n;S)\ge n$.

Пусть сначала S — правильный симплекс, вписанный в B_n. Тогда $\alpha(B_n;S)=n$, что равносильно тому, что инрадиус S равен $\frac{1}{n}$. Так как симплекс $\xi(B_n;S)S$ описан вокруг B_n, то $\xi(S;B_n)=\alpha(S;B_n)=n$ и выполняется соотношение (2), в котором надо взять $C=B_n$. Из (1) следует, что при любом $j=1,\dots,n+1$

$$\max_{x\in B_n}(-\lambda_j(x))=\frac{n-1}{n+1},$$

где λ_j — базисные многочлены Лагранжа симплекса S.

Пусть теперь симплекс S содержится в B_n, но не является правильным симплексом или не вписан в этот шар. Его многочлены Лагранжа обозначим через μ_j. Найдётся такой правильный симплекс S^*, вписанный в B_n, и такое k, что S принадлежит полосе $0\le\lambda_k(x)\le 1$, k-е $(n-1)$-мерные грани S и S^* параллельны и при этом S не имеет общих точек хотя бы с одной граничной гиперплоскостью этой полосы. Здесь λ_j — базисные многочлены Лагранжа симплекса S^*. Вершина $x^{(k)}$ симплекса S^* не принадлежит его k-й грани. Пусть u есть точка границы шара, наиболее удалённая от $x^{(k)}$, тогда u есть точка максимума многочлена $-\lambda_k(x)$, т. е. $-\lambda_j(u)=\frac{n-1}{n+1}$. Рассмотрим прямую, соединяющую $x^{(k)}$ и u. Обозначим через y,z и t точки пересечения этой прямой с попарно параллельными гиперплоскостями $\mu_k(x)=1$, $\mu_k=0$ и $\lambda_k(x)=0$ соответственно. Выполняются неравенства

$$\|x^{(k)}-t\|\ge\|y-z\|,\quad \|t-u\|\le\|z-u\|, \tag{24}$$

хотя бы одно из которых наверняка строгое. В силу линейности базисных многочленов Лагранжа справедливы соотношения

$$\frac{\mu_k(z)-\mu_k(u)}{\mu_k(y)-\mu_k(z)}=\frac{\|z-u\|}{\|y-z\|},\quad \frac{\lambda_k(t)-\lambda_k(u)}{\lambda_k\left(x^{(k)}\right)-\lambda_k(t)}=\frac{\|t-u\|}{\|x^{(k)}-t\|}.$$

Поскольку $\mu_k(y)=1,\mu_k(z)=0,\lambda_k\left(x^{(k)}\right)=1,\lambda_k(t)=0$, имеем

$$-\mu_k(u)=\frac{\|z-u\|}{\|y-z\|}>\frac{\|t-u\|}{\|x^{(k)}-t\|}=-\lambda_k(u)=\frac{n-1}{n+1}.$$

Мы применили неравенства (24) и учли, что хотя бы одно из них является строгим. По формуле (1)

$$\xi(B_n; S) = (n+1) \max_{1\le j\le n+1} \max_{x\in B_n}(-\mu_j(x)) + 1 \ge (n+1)(-\mu_k(u)) + 1 > n.$$

Значит, если S не является правильным симплексом, вписанным в B_n, то имеет место строгое неравенство $\xi(B_n; S) > n$.

Таким образом, для любого симплекса $S \subset B_n$ выполняется $\xi(B_n; S) \ge n$. Равенство здесь эквивалентно тому, что S — правильный симплекс, вписанный в B_n. □

По аналогии с величиной $\xi_n = \min\{\xi(Q_n; S) : S \subset Q_n\}$, определяемой через единичный куб $Q_n = [0,1]^n$, введём в рассмотрение числовую характеристику, задаваемую единичным шаром:

$$\beta_n := \min\{\xi(B_n; S) : S \text{ — } n\text{-мерный симплекс}, S \subset B_n, \operatorname{vol}(S) \ne 0\}.$$

Многие задачи о числах ξ_n ещё не решены. Например, единственным точным значением ξ_n для чётного n до сих пор остаётся $\xi_2 = 1 + \frac{3\sqrt{5}}{5}$, да и оно было найдено весьма непростым способом (см. [3, гл. 2]). По сравненению с ξ_n задача о числах β_n оказывается тривиальной.

Следствие 3. *Для любого n справедливо $\beta_n = n$. Экстремальным относительно β_n симплексом $S \subset B_n$ является любой правильный симплекс, вписанный в B_n. Других экстремальных симплексов нет.*

Достаточно применить теорему 4. Техника, связанная с шаром, позволяет проиллюстрировать некоторые результаты, полученные ранее для куба. Отметим здесь доказательство следующего известного утверждения, отличное от приведённых в [3, § 3.2] и [12].

Следствие 4. *Пусть существует матрица Адамара порядка $n+1$. Тогда $\xi_n = n$.*

Доказательство. Известно (см., например, [9]), что для такого и только такого n в Q_n можно вписать правильный симплекс S так, что вершины симплекса будут находиться в вершинах куба. Обозначим через B шар с центром в центре куба и радиусом $\frac{\sqrt{n}}{2}$. Очевидно, куб Q_n вписан в B, поэтому и симплекс S вписан в этот шар. Так как S является правильным, то по теореме 4 и из соображений подобия имеем $\xi(B; S) = n$. Включение $Q_n \subset B$ означает, что $\xi(Q_n; S) \le \xi(B; S)$, т. е. $\xi(Q_n; S) \le n$. В силу (6) выполнено и противоположное неравенство $\xi(Q_n; S) \ge n$, поэтому $\xi(Q_n; S) = n$. Одновременно из (6) следует, что $\xi_n = \xi(Q_n; S) = n$. □

Это рассуждение базируется на том, что если S — правильный симплекс, вершины которого находятся в вершинах Q_n, то симплекс nS поглощает не только Q_n, но и шар B, описанный вокруг куба. При этом коэффициент поглощения n оказывается минимально возможным и для куба, и для шара. Дополнительно отметим такое свойство.

Следствие 5. *Пусть $S \subset Q_n \subset nS$, причём симплекс S не является правильным. Тогда $B \not\subset nS$.*

Доказательство. Включение $B \subset nS$ означало бы, что $\xi(B;S) = n$. Тогда S был бы правильным симплексом, вписанным в шар B, однако это не так. Поэтому B не является подмножеством nS. □

Симплексы, удовлетворяющие условию следствия 5, существуют по крайней мере при $n = 3, 5$ и 9 (см. [12]).

Соотношения (6) означают, что всегда $\xi_n \geq n$. Поскольку $\xi_2 = 1 + \frac{3\sqrt{5}}{5} > 2$, существуют n, для которых $\xi_n > n$. Кроме ситуаций, когда $n+1$ есть число Адамара, равенство $\xi_n = n$ установлено при $n = 5$ и $n = 9$ (экстремальные симплексы в $\mathbb{R}^5$ и $\mathbb{R}^9$ найдены в [12]). Для всех таких размерностей $\xi_n = \beta_n$, т. е. с точки зрения минимального коэффициента поглощения находящимся внутри симплексом оба множества — n-мерный куб и n-мерный шар — ведут себя одинаково.

Равенство $\xi_n = n$ равносильно существованию симплексов, удовлетворяющих включениям $S \subset Q_n \subset nS$. Некоторые свойства таких симплексов (например, то, что центр тяжести S совпадает с центром куба) аналогичны свойствам правильных симплексов, вписанных в B_n (см. [7]). Однако вопрос о полном описании тех размерностей n, для которых существуют симплексы с таким условием, является весьма трудным и пока далёк от решения.

Список литературы / References

[1] Невский М. В., “Об одном соотношении для минимальной нормы интерполяционного проектора”, *Модел. и анализ информ. систем*, **16**:1 (2009), 24–43; [Nevskij M. V., “On a certain relation for the minimal norm of an interpolational projection”, *Modeling and Analysis of Information Systems*, **16**:1 (2009), 24–43, (in Russian).]

[2] Невский М. В., “Об одном свойстве n-мерного симплекса”, *Матем. заметки*, **87**:4 (2010), 580–593; English transl.: Nevskii M. V., “On a property of n-dimensional simplices”, *Math. Notes*, **87**:4 (2010), 543–555.

[3] Невский М. В., *Геометрические оценки в полиномиальной интерполяции*, Ярославль: Ярославский государственный университет им. П. Г. Демидова, 2012; [Nevskii M. V., *Geometricheskie ocenki v polinomialnoy interpolyacii*, Yaroslavl: P. G. Demidov Yaroslavl State University, 2012, (in Russian).]

[4] Невский М. В., “О минимальном положительном гомотетическом образе симплекса, содержащем выпуклое тело”, *Матем. заметки*, **93**:3 (2013), 448–456; English transl.: Nevskii M. V., “On the minimal positive homothetic image of a simplex containing a convex body”, *Math. Notes*, **93**:3–4 (2013), 470–478.

[5] Невский М. В., Ухалов А. Ю., “О числовых характеристиках симплекса и их оценках”, *Модел. и анализ информ. систем*, **23**:5 (2016), 603–619; English transl.: Nevskii M. V., Ukhalov A. Yu., “On numerical charasteristics of a simplex and their estimates”, *Aut. Control Comp. Sci.*, **51**:7 (2017), 757–769.

[6] Невский М. В., Ухалов А. Ю., “Новые оценки числовых величин, связанных с симплексом”, *Модел. и анализ информ. систем*, **24**:1 (2017), 94–101; English transl.: Nevskii M. V., Ukhalov A. Yu., “New estimates of numerical values related to a simplex”, *Aut. Control Comp. Sci.*, **51**:7 (2017), 770–782.

[7] Невский М. В., Ухалов А. Ю., “Об n-мерных симплексах, удовлетворяющих включениям $S \subset [0,1]^n \subset nS$”, *Модел. и анализ информ. систем*, **24**:5 (2017), 578–595; [Nevskii M. V., Ukhalov A. Yu., “On n-Dimensional Simplices Satisfying Inclusions $S \subset [0,1]^n \subset nS$”, *Modeling and Analysis of Information Systems*, **24**:5 (2017), 578–595, (in Russian).]

[8] Невский М. В., Ухалов А. Ю., "О минимальном коэффициенте поглощения для n-мерного симплекса", *Модел. и анализ информ. систем*, **25**:1 (2018), 140–150; [Nevskii M. V., Ukhalov A. Yu., "On Minimal Absorption Index for an n-Dimensional Simplex", *Modeling and Analysis of Information Systems*, **25**:1 (2018), 140–150, (in Russian).]

[9] Hudelson M., Klee V., Larman D., "Largest j-simplices in d-cubes: some relatives of the Hadamard maximum determinant problem", *Linear Algebra Appl.*, **241–243** (1996), 519–598.

[10] Klamkin M. S., Tsifinis G. A., "Circumradius–inradius inequality for a simplex", *Mathematics Magazine*, **52**:1 (1979), 20–22.

[11] Nevskii M., "Properties of axial diameters of a simplex", *Discrete Comput. Geom.*, **46**:2 (2011), 301–312.

[12] Nevskii M., Ukhalov A., "Perfect simplices in $\mathbb{R}^5$", *Beiträge zur Algebra und Geometrie / Contributions to Algebra and Geometry*, **59**:3 (2018), 501–521.

[13] Yang S., Wang J., "Improvements of n-dimensional Euler inequality", *Journal of Geometry*, **51** (1994), 190–195.

[14] Vince A., "A simplex contained in a sphere", *Journal of Geometry*, **89**:1–2 (2008), 169–178.

Nevskii M. V., "On Some Problems for a Simplex and a Ball in $\mathbb{R}^n$", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 680–691.

DOI: 10.18255/1818-1015-2018-6-680-691

Abstract. Let C be a convex body and let S be a nondegenerate simplex in $\mathbb{R}^n$. Denote by τS the image of S under homothety with a center of homothety in the center of gravity of S and the ratio τ. We mean by $\xi(C;S)$ the minimal $\tau>0$ such that C is a subset of the simplex τS. Define $\alpha(C;S)$ as the minimal $\tau>0$ such that C is contained in a translate of τS. Earlier the author has proved the equalities $\xi(C;S)=(n+1)\max\limits_{1\le j\le n+1}\max\limits_{x\in C}(-\lambda_j(x))+1$ (if $C\not\subset S$), $\alpha(C;S)=\sum\limits_{j=1}^{n+1}\max\limits_{x\in C}(-\lambda_j(x))+1$. Here λ_j are the linear functions that are called the basic Lagrange polynomials corresponding to S. The numbers $\lambda_j(x),\dots,\lambda_{n+1}(x)$ are the barycentric coordinates of a point $x\in\mathbb{R}^n$. In his previous papers, the author investigated these formulae in the case when C is the n-dimensional unit cube $Q_n=[0,1]^n$. The present paper is related to the case when C coincides with the unit Euclidean ball $B_n=\{x:\|x\|\le 1\}$, where $\|x\|=\left(\sum\limits_{i=1}^{n}x_i^2\right)^{1/2}$. We establish various relations for $\xi(B_n;S)$ and $\alpha(B_n;S)$, as well as we give their geometric interpretation. For example, if $\lambda_j(x)=l_{1j}x_1+\ldots+l_{nj}x_n+l_{n+1,j}$, then $\alpha(B_n;S)=\sum\limits_{j=1}^{n+1}\left(\sum\limits_{i=1}^{n}l_{ij}^2\right)^{1/2}$. The minimal possible value of each characteristics $\xi(B_n;S)$ and $\alpha(B_n;S)$ for $S\subset B_n$ is equal to n. This value corresponds to a regular simplex inscribed into B_n. Also we compare our results with those obtained in the case $C=Q_n$.

Keywords: n-dimensional simplex, n-dimensional ball, homothety, absorption index

On the authors:
Mikhail V. Nevskii, orcid.org/0000-0002-6392-7618, Doctor of Science,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: mnevsk55@yandex.ru

Моделирование и анализ информационных систем. Т. 25, №6 (2018), с. 692–710
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 692–710

DOI: 10.18255/1818-1015-2018-6-692-710

УДК 517.9

Особые точки кривых

Уваров А. Д.

Поступила в редакцию 3 сентября 2018
После доработки 30 октября 2018
Принята к публикации 10 ноября 2018

Аннотация. В данной работе затрагивается ключевая проблема геометрического моделирования, связанная с построением кривых пересечения поверхностей. Найдены способы построения кривых пересечения в сложных случаях: при касании и при прохождении через особые точки поверхностей. В первой части работы рассматривается проблема определения линии касания двух поверхностей, заданных в параметрическом виде. Анализируется несколько подходов к решению задачи. Выявляются достоинства и недостатки приведенных подходов. Описываются итерационные алгоритмы поиска точки на линии касания. Вторая часть работы посвящена методам преодоления возникающих трудностей решения задачи для сингулярных точек кривых пересечения, в которых нарушается регулярный итерационный процесс. В зависимости от типа поставленной задачи автор останавливается на двух методах. Первый из них предполагает находить особые точки кривых без использования итерационных методов, что уменьшает время работы алгоритма по построению кривой пересечения. Второй метод, рассматриваемый в заключительной части статьи, относится к численным методам. В этой части автор вводит функцию, достигающую глобального минимума только в особых точках кривых пересечения, и решает задачу минимизации этой функции. Применение этого метода является весьма эффективным в некоторых частных случаях, накладывающих ограничения на поверхности и их расположение. В заключение рассматривается использование этого метода в случае, когда функция имеет такой рельеф, что в окрестности точки минимума поверхности уровня являются сильно вытянутыми эллипсоидами. Все изображения, приведенные в данной статье, являются результатом работы алгоритмов по методам, предложенным автором. Изображения получены с помощью авторской программной среды.

Ключевые слова: особая точка, пересечение, касание, матрица, градиент

Для цитирования: Уваров А. Д., "Особые точки кривых", *Моделирование и анализ информационных систем*, **25**:6 (2018), 692–710.

Об авторах:
Уваров Артем Дмитриевич, orcid.org/0000-0002-0624-3877, канд. физ.-мат. наук,
Ярославский государственный педагогический университет им. К.Д. Ушинского,
ул. Республиканская, 108, г. Ярославль, 150000 Россия, e-mail: artiom_uvarov@inbox.ru

Введение

Сложные геометрические объекты, например, поверхности, ограниченные контурами, являются результатом некоторых операций над более простыми объектами. Базовыми логическими операциями над геометрическими объектами можно считать их пересечение, объединение и вычитание. Для применения этих операций

требуется определение еще одного геометрического объекта – линии пересечения исходных поверхностей. Алгоритм построения такого объекта хорошо известен и описан в литературе (см., например, [1], [2], [3]). На каждом шаге итерационного процесса требуется информация о точке, лежащей на кривой пересечения, о направлении касательного вектора к этой кривой в этой точке и о величине шага вдоль касательного вектора. Этот вектор определяется как вектор, ортогональный двум нормалям к обеим поверхностям, проведенным из точки, лежащей на кривой пересечения. Частным случаем пересечения поверхностей является их полное или частичное касание (частичное касание предполагает, что поверхности касаются в конечном числе точек). В двух последних случаях невозможно применение традиционных алгоритмов, поскольку в случае касания поверхностей их нормали, построенные в точках касания – коллинеарны. Однако в некоторых специальных случаях касания поверхностей, накладывающих ограничения на их взаимное расположение, можно построить касательную кривую, используя узко направленные алгоритмы. Стоит заметить, что сами поверхности и касательная кривая должны быть гладкими. Первая часть статьи посвящена анализу таких алгоритмов.

В случае частичного касания поверхностей кривая пересечения содержит особенности – точки самопересечения или самокасания. При "подходе" к особой точке кривой пересечения традиционный алгоритм становится мало пригодным, поскольку на каждом шаге итерационного процесса он предполагает численное решение системы линейных уравнений, которая становится плохо обусловленной в особой точке. Общим подходом для решения этой проблемы является применение сингулярного разложения (см. [4], [5]). Однако такой метод требует больших вычислительных затрат. В случае, когда не требуется высокая точность координат особой точки и важна скорость работы алгоритма, возможно определить особую точку как барицентр концов ветвей кривой пересечения, сходящихся в этой точке. Плюсы и минусы этого метода рассматриваются во второй части статьи.

Задачу о пересечении поверхностей можно рассматривать как задачу минимизации. В случае параметрического задания поверхностей она была решена Баттерфильдом (см. [6]), но только для функции трех переменных. Если поверхности обозначить $\mathbf{r}(u,v)$ и $\mathbf{s}(a,b)$, то необходимо найти минимум функции $|\mathbf{r}(u,v)-\mathbf{s}(a,b)|$. Однако в этом случае один из параметров должен быть фиксирован. Если глобальный минимум последняя функция достигает в особой точке кривой пересечения, то фиксировать какой-либо параметр не представляется возможным. В третей части статьи вводится функция $f(u,v,a,b)$ которая достигает глобального минимума только в особых точках кривых пересечения. При этом для некоторых частных случаев минимум функции ищется через решение системы линейных уравнений. В общем случае, не накладывая на поверхности и их расположение дополнительных ограничений, глобальный минимум находится с помощью метода "оврагов" и покоординатного спуска [7].

1. Гладкие кривые

В статье [8] для поиска точки пересечения поверхностей авторы предлагают использовать метод градиентного спуска вместо метода, в котором используется закрепле-

ние одной из параметрических координат. Предполагается, что кривая пересечения поверхностей – гладкая.

Пересекающиеся поверхности обозначаются $\mathbf{r}(u, v)$ и $\mathbf{s}(a, b)$. Искомая точка пересечения обозначается $\mathbf{p}^*$, при этом

$$\mathbf{p}^* = \begin{bmatrix} u \\ v \\ a \\ b \end{bmatrix}. \tag{1}$$

Точка пересечения ищется из следующего векторного уравнения:

$$\mathbf{r}(u, v) - \mathbf{s}(a, b) = 0. \tag{2}$$

Обозначим через

$$\mathbf{p}_k = \begin{bmatrix} u \\ v \\ a \\ b \end{bmatrix}$$

некоторую точку исходного параметрического пространства. А через $\mathbf{F}(\mathbf{p}_k)$ вектор невязки векторного уравнения (2). Тогда

$$\mathbf{F}(\mathbf{p}_k) = \begin{bmatrix} F_x(\mathbf{p}_k) \\ F_y(\mathbf{p}_k) \\ F_x(\mathbf{p}_k) \end{bmatrix}. \tag{3}$$

Фиксировав одну из параметрических координат, мы приходим к необходимости решения следующей системы уравнений методом Ньютона:

$$J(\mathbf{p}_k)\Delta_k = -\mathbf{F}(\mathbf{p}_k), \tag{4}$$

где

$$J(\mathbf{p}_k) = \begin{bmatrix} \nabla F_x \\ \nabla F_y \\ \nabla F_z \end{bmatrix} \tag{5}$$

– матрица Якоби, вычисленная в точке $\mathbf{p}_k$. При этом $\nabla F_x, \nabla F_y, \nabla F_z$ – градиенты функций F_x, F_y, F_z. При фиксированной координате последние функции есть функции трех переменных, а Δ_k – трехкомпонентный вектор. В этом случае вектор приращения Δ_k ищется из следующего уравнения:

$$\Delta_k = -J^{-1}(\mathbf{p}_k)\mathbf{F}(\mathbf{p}_k), \tag{6}$$

то есть $\mathbf{p}_{k+1} = \mathbf{p}_k + \Delta_k$.

Система (4) является вырожденной (плохо согласованной) в том случае, когда

$$\det J(\mathbf{p}_k) = 0. \tag{7}$$

В общем виде матрица Якоби – это прямоугольная матрица размера 3×4, а именно:

$$J(\mathbf{p}_k) = \begin{bmatrix} \frac{\partial r}{\partial u} & \frac{\partial r}{\partial v} & -\frac{\partial s}{\partial a} & -\frac{\partial s}{\partial b} \end{bmatrix}, \tag{8}$$

отсюда следует,что равенство (7) возможно в том случае, когда векторы $\frac{\partial r}{\partial u}, \frac{\partial r}{\partial v}, \frac{\partial s}{\partial a}, \frac{\partial s}{\partial b}$ лежат в одной плоскости. Последняя ситуация возможна в предельном случае пересечения – касании поверхностей, например, когда искомая кривая вся состоит из точек касания двух поверхностей. Заметим, что в случае, когда кривая – плоская и нормаль в каждой ее точке параллельна какой-либо координатной оси, равенство (7) возникает из-за того, что один из градиентов $\nabla F_x, \nabla F_y, \nabla F_z$ становится равным нулю, то есть строка матрицы (5) зануляется. При этом ни один из столбцов этой матрицы не будет равным нулю. Шаг по методу Ньютона в этом случае выполнить нельзя. Также, например, равенство (7) может возникать в точках, принадлежащих параллельным участкам параметрических линий двух семейств, в которых касательные к параметрическим линиям коллинеарные. При этом первый столбец матрицы (5) подобен третьему, а второй подобен четвертому.

В обоих случаях применение метода Ньютона не дает необходимой точности поэтому целесообразней использовать метод градиентного спуска, который является более стабильным по отношению к последним двум случаям.

В случае, когда кривая не содержит особых точек, представим решение уравнения (2) в пространстве новых четырех координат. Три оси направим параллельно векторам градиентов компонент невязки в точке $\mathbf{p}_k$. Вектор $\mathbf{t}$, задающий направление четвертой оси, выберем ортогональным градиентам. Возможность такого перехода обосновывается в работе [9]. Решение уравнения (2) в новых координатах примет вид

$$\Delta_k = \alpha_k \mathbf{t} + \beta_k \nabla F_x + \gamma_k \nabla F_y + \delta_k \nabla F_z. \tag{9}$$

В окрестности неособой точки $\mathbf{p}_k$ шаг в направлении $\mathbf{t}$ приводит к незначительному изменению величины $\mathbf{F}$. Поэтому координата $\alpha_k = 0$. С учетом последнего равенства имеем

$$\Delta_k = J^T(\mathbf{p}_k)\Delta'_k, \tag{10}$$

где

$$\Delta'_k = \begin{bmatrix} \beta_k \\ \gamma_k \\ \delta_k \end{bmatrix}$$

– шаг в новом пространстве. В формуле (10) Δ_k – трехкомпонентный вектор, который может быть найден из уравнения (4). Действительно, подставим равенство (10) в равенство (4) и выразим Δ'_k :

$$\Delta'_k = -(J'(\mathbf{p}_k))^{-1}\mathbf{F}(\mathbf{p}_k),$$

где

$$J' = JJ^T = \begin{bmatrix} \nabla F_x \nabla F_x & \nabla F_x \nabla F_y & \nabla F_x \nabla F_z \\ \nabla F_y \nabla F_x & \nabla F_y \nabla F_y & \nabla F_y \nabla F_z \\ \nabla F_z \nabla F_x & \nabla F_z \nabla F_y & \nabla F_z \nabla F_z \end{bmatrix}. \tag{11}$$

Рассмотрим ситуации, в которых матрица J' вырождается.

1. Один из градиентов $\nabla F_x, \nabla F_y, \nabla F_z$ становится равным нулю. При этом один из столбцов матрицы (11) также будет равен нулю, поскольку эта матрица симметричная. Вычеркиванием строки и столбца приходим к квадратной матрице, из которой определяется шаг.

2. Подобие строк матрицы J' влечет за собой подобие столбцов. При этом вычеркивание подобных строк и столбцов снова приводит к квадратной матрице, из которой также определяется шаг.

Далее рассмотрим систему

$$\begin{array}{l}(\mathbf{r}(u,v)-\mathbf{s}(a,b))\cdot\frac{\partial\mathbf{r}}{\partial u}=0\\(\mathbf{r}(u,v)-\mathbf{s}(a,b))\cdot\frac{\partial\mathbf{r}}{\partial v}=0\\(\mathbf{s}(a,b)-\mathbf{r}(u,v))\cdot\frac{\partial\mathbf{s}}{\partial a}=0\\(\mathbf{s}(a,b)-\mathbf{r}(u,v))\cdot\frac{\partial\mathbf{s}}{\partial b}=0,\end{array} \tag{12}$$

приведенную в книге [1] на странице 251. Если ввести обозначения:

$$a=|\frac{\partial\mathbf{r}}{\partial u}|^2+\rho\cdot\frac{\partial^2\mathbf{r}}{\partial u^2};b=|\frac{\partial\mathbf{r}}{\partial v}|^2+\rho\cdot\frac{\partial^2\mathbf{r}}{\partial v^2};c=|\frac{\partial\mathbf{s}}{\partial a}|^2+\rho\cdot\frac{\partial^2\mathbf{s}}{\partial a^2};d=-|\frac{\partial\mathbf{s}}{\partial b}|^2+\rho\cdot\frac{\partial^2\mathbf{s}}{\partial b^2};$$

$$e=\frac{\partial\mathbf{r}}{\partial v}\cdot\frac{\partial\mathbf{r}}{\partial u}+\rho\cdot\frac{\partial^2\mathbf{r}}{\partial u\partial v};f=\frac{\partial\mathbf{s}}{\partial a}\cdot\frac{\partial\mathbf{r}}{\partial u};g=\frac{\partial\mathbf{s}}{\partial b}\cdot\frac{\partial\mathbf{r}}{\partial u};h=\frac{\partial\mathbf{s}}{\partial a}\cdot\frac{\partial\mathbf{r}}{\partial v};$$

$$i=\frac{\partial\mathbf{s}}{\partial b}\cdot\frac{\partial\mathbf{r}}{\partial v};k=-\frac{\partial\mathbf{s}}{\partial b}\cdot\frac{\partial\mathbf{s}}{\partial a}+\rho\cdot\frac{\partial^2\mathbf{s}}{\partial a\partial b},$$

где $\rho=\mathbf{r}(u,v)-\mathbf{s}(a,b)$, то шаг Δ_k находится из решения системы (4), при этом матрица $J(\mathbf{p}_k)$ имеет вид

$$J=\begin{bmatrix}a&e&f&g\\e&b&h&i\\f&h&c&k\\g&i&k&d\end{bmatrix}. \tag{13}$$

Последняя матрица хорошо подходит для выполнения шага в предельном случае пересечения – касании поверхностей. При этом нужно учитывать, что кривая пересечения – гладкая.

Можно показать, что если кривая содержит особую точку, то в ней матрица (13) вырождается (строки становятся подобными). Поскольку последняя матрица является симметрической, подобие строк приводит к подобию столбцов, а значит, можно вычеркиванием строки и столбца снова прийти к квадратной матрице меньшего ранга и вычислить шаг.

Вывод: если по условию задачи предполагается, что касание поверхностей приводит только к гладкой кривой и при этом нормаль коллинеарна одной из осей координат, то целесообразно выполнять шаг с помощью матрицы (11), поскольку мы приходим к матрице ранга ≤ 2. Если же кривая касания остается гладкой, но нормаль не предполагается коллинеарной какой-либо оси, то можно использовать матрицу (13).

2. Кривые с особенностями

В статье [8] в качестве общего подхода для численного решения плохо обусловленных систем предлагается использовать метод сингулярного разложения. В этом случае с помощью итерационного процесса матрицу (8) можно привести к виду

$$J = UOD,$$

где U – ортогональная матрица размера 4×4, O – ортогональная матрица размера 4×3, а D – диагональная матрица размера 3×3. Однако в случае полного вырождения матрицы J, то есть когда $\mathrm{Rank}(J) = 0$, невозможно получить шаг с помощью сингулярной матрицы. В статье предлагается преодолеть данную ситуацию с помощью малого возмущения начального приближения. Очевидно, что такой метод обладает рядом недостатков. Во-первых, он неточен, во-вторых, он не стабилен.

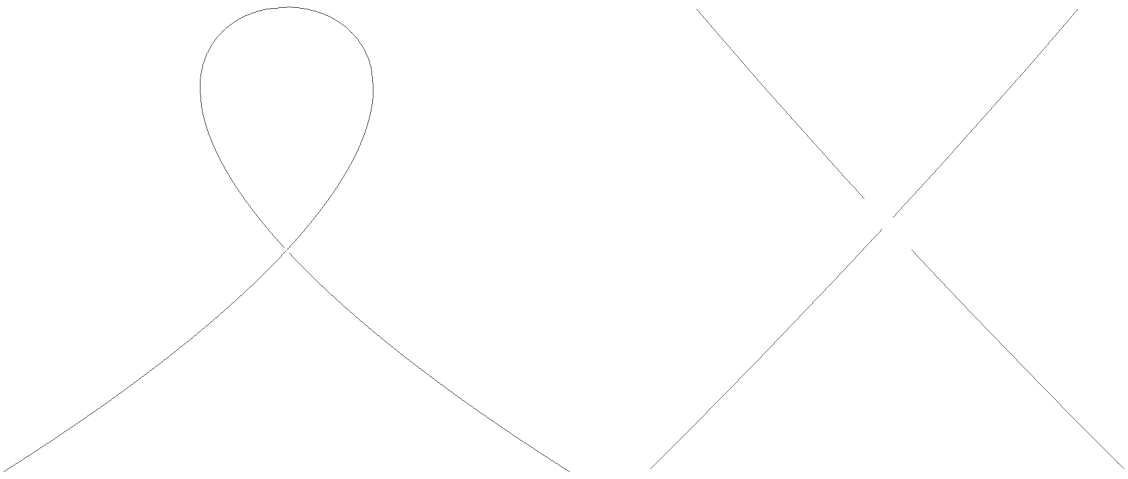
В качестве альтернативы предлагается использовать следующий подход.

1. Подойти к особой точке достаточно близко. Например, пока $|\mathbf{t}| \leq \varepsilon$, где $\mathbf{t}$ – касательный вектор к кривой в точке $A(\mathbf{p}^*) = \frac{\mathbf{r}(u^*,v^*)+\mathbf{s}(a^*,b^*)}{2}$, при этом $\mathbf{p}^*$ – точка из (1).

1.1. В случае перехода через особую точку отбросить последний сегмент кривой, затем к точке, полученной на предыдущем шаге, прибавить касательный вектор, длина которого уменьшилась в два раза. Переход через особую точку определяется тем, что угол между касательными векторами в текущей и предыдущей точках близок к 180^0.

1.2. В том случае, когда при переходе через особую точку угол между касательными векторами в текущей и предыдущей точках больше 90^0, необходимо просто отбросить последний сегмент кривой.

На следующих рисунках под буквой (a) изображены кривые пересечения, а под буквой (b) – увеличенная область вокруг особой точки.



(a) (b)

Рис. 1: $z = x^3 - x^2 + y^2 \cap z = 0$
Fig. 1: $z = x^3 - x^2 + y^2 \cap z = 0$

(a) (b)

Рис. 2: $z = 2x^4 - 3x^2y + y^2 - 2y^3 + y^4 \cap z = 0$
Fig. 2: $z = 2x^4 - 3x^2y + y^2 - 2y^3 + y^4 \cap z = 0$

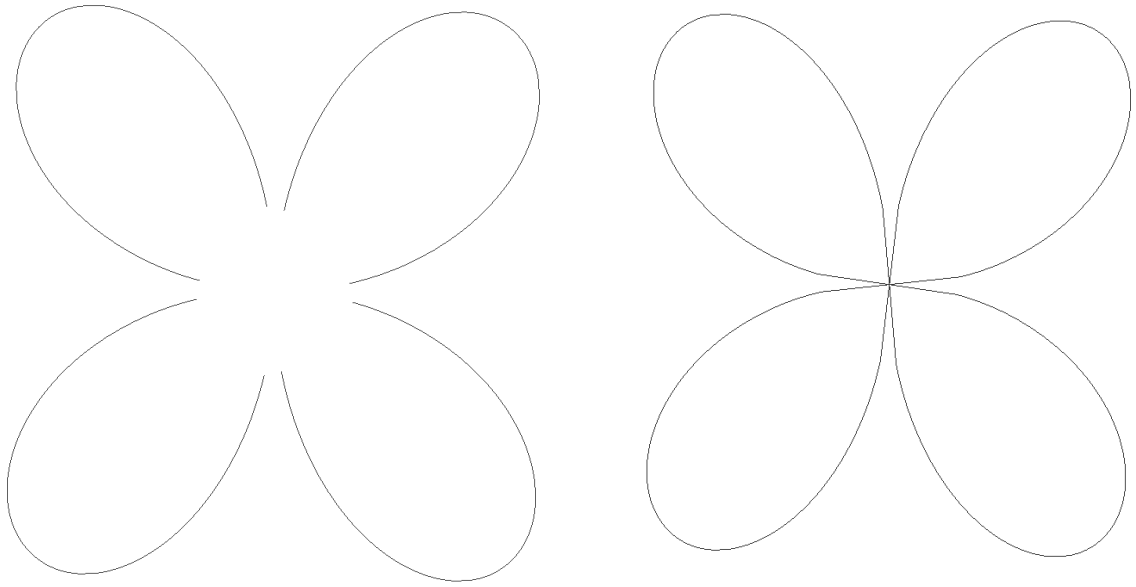
(a) (b)

Рис. 3: $z = x^4 + x^2y^2 - 2x^2y - xy^2 + y^2 \cap z = 0$
Fig. 3: $z = x^4 + x^2y^2 - 2x^2y - xy^2 + y^2 \cap z = 0$

2. Приближенно вычислить особую точку.

Несложно видеть, что если особая точка алгебраической кривой есть точка самопересечения кратности p (см. рис. 1), то число ветвей, сходящихся к данной точке, равно $2p$. Поэтому характер сходимости решения к особой точке при движении к ней с противоположных сторон должен быть один и тот же, то есть особая точка должна быть равноудалена от последних сегментов противоположных ветвей. В связи с

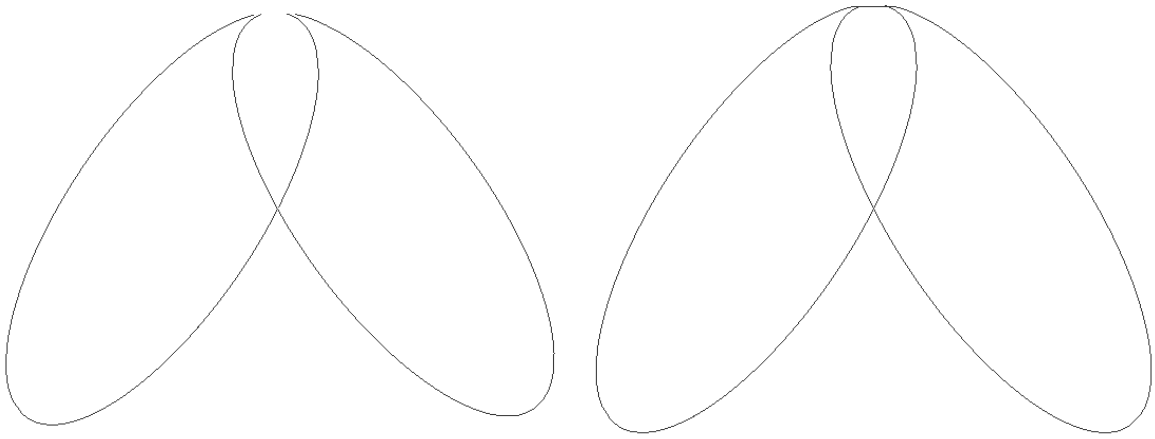
этим фактом в качестве особой точки можно взять центр симметрии концов каждой ветви, сходящейся к этой точке. В роли центра симметрии выступает барицентр.



(a) (b)

Рис. 4: $z = (x^2 + y^2)^3 - 4x^2y^2 \cap z = 0$
Fig. 4: $z = (x^2 + y^2)^3 - 4x^2y^2 \cap z = 0$

В случае точки самокасания (см. рис. 2) можно поступить так же, как с точкой самопересечения, однако точность будет несколько хуже.



(a) (b)

Рис. 5: $z = 2x^4 - 3x^2y + y^2 - 2y^3 + y^4 \cap z = 0$
Fig. 5: $z = 2x^4 - 3x^2y + y^2 - 2y^3 + y^4 \cap z = 0$

Если предполагается, что кривые используются в качестве наглядного материала или в методических целях, то данной точности будет достаточно.

Вычисление координат особой точки типа "острие" (см. рис. 3) плохо укладывается в рамки предыдущего алгоритма, однако если одна из ветвей подошла к особой точке намного ближе другой ветви, то последний алгоритм тоже применим.

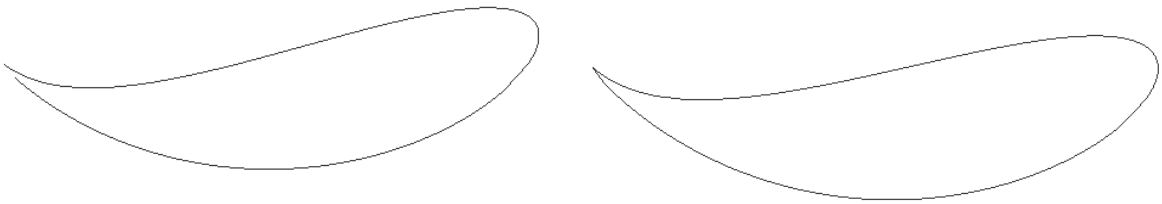


Рис. 6: $z = x^4 + x^2y^2 - 2x^2y - xy^2 + y^2 \cap z = 0$
Fig. 6: $z = x^4 + x^2y^2 - 2x^2y - xy^2 + y^2 \cap z = 0$

Все типы особых точек плоских алгебраических кривых возникают и на пространственных кривых, поскольку локально их можно рассматривать как плоские.

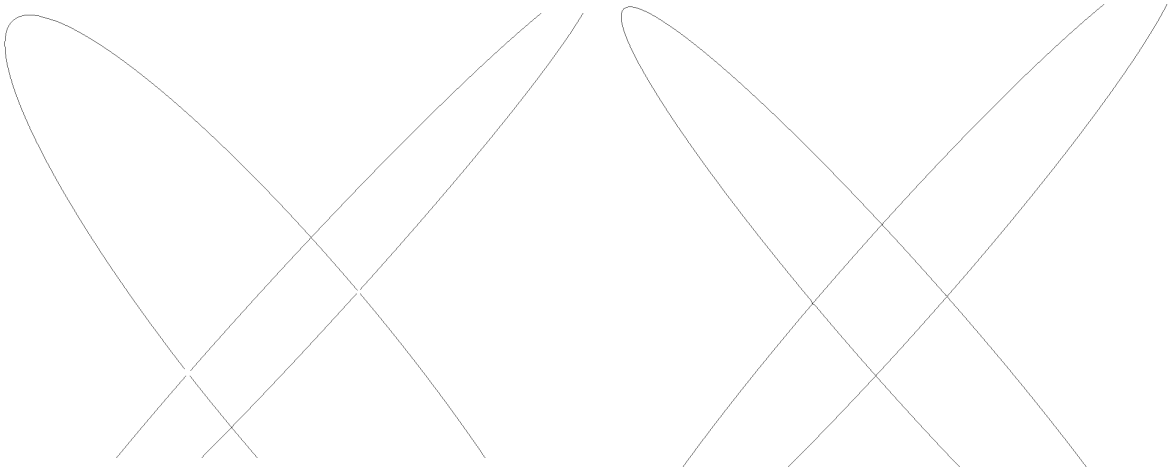


Рис. 7: Пересечение двух цилиндров
Fig. 7: Intersection of two cylinders

Вывод: можно считать, что при решении технических задач метод поиска и вычисления особых точек кривых пересечения, предложенный в статье, не совсем приемлем из-за своей низкой точности. Последний метод хорошо подходит для решения методических задач, связанных с наглядным моделированием.

3. Численные методы нахождения особых точек

Система (12) возникает при попытке минимизировать следующую функцию :

$$g(u, v, a, b) = |\mathbf{r}(u, v) - \mathbf{s}(a, b)|^2.$$

Поиск локального минимума функции g сводится к нахождению нуля градиента функции g. А именно, к решению уравнения $\nabla g = 0$. Если ввести обозначения

$$\Delta_k = \begin{bmatrix} \Delta_u \\ \Delta_v \\ \Delta_a \\ \Delta_b \end{bmatrix}, \tag{14}$$

где $\Delta_u, \Delta_v, \Delta_a, \Delta_b$ – приращения аргументов функции g на k-м шаге итерационного процесса, то

$$\Delta_k = H^{-1}\nabla g,$$

где H – матрица Гессе функции g.

Решение последней системы методом Ньютона–Рафсона невозможно, поскольку $\det H = 0$. Докажем последнее равенство.

Доказательство. Градиент ∇g имеет вид:

$$\nabla g = \begin{bmatrix} 2\mathbf{t} \cdot \frac{\partial \mathbf{r}}{\partial u} \\ 2\mathbf{t} \cdot \frac{\partial \mathbf{r}}{\partial v} \\ -2\mathbf{t} \cdot \frac{\partial \mathbf{s}}{\partial a} \\ -2\mathbf{t} \cdot \frac{\partial \mathbf{s}}{\partial b} \end{bmatrix}, \tag{15}$$

где $\mathbf{t} = \mathbf{r}(u, v) - \mathbf{s}(a, b)$. Обозначим координаты вектора ∇g через g_u, g_v, g_a, g_b. Тогда матрица Гессе $H(g)$ функции g примет вид:

$$H(g) = \begin{bmatrix} \nabla g_u \\ \nabla g_v \\ \nabla g_a \\ \nabla g_b \end{bmatrix}. \tag{16}$$

Покажем, что векторы ∇g_u и ∇g_v линейно зависят от векторов ∇g_a и ∇g_b. Действительно, полагая, что $|\mathbf{t}| \approx 0$, слагаемым, содержащим $\mathbf{t}$, можно пренебречь. Учитывая этот факт и вектор (15), матрицу (16) можно записать так:

$$H(g) = \begin{bmatrix} 2\mathbf{t}'_u \cdot \frac{\partial \mathbf{r}}{\partial u} & 2\mathbf{t}'_v \cdot \frac{\partial \mathbf{r}}{\partial u} & 2\mathbf{t}'_a \cdot \frac{\partial \mathbf{r}}{\partial u} & 2\mathbf{t}'_b \cdot \frac{\partial \mathbf{r}}{\partial u} \\ 2\mathbf{t}'_u \cdot \frac{\partial \mathbf{r}}{\partial v} & 2\mathbf{t}'_v \cdot \frac{\partial \mathbf{r}}{\partial v} & 2\mathbf{t}'_a \cdot \frac{\partial \mathbf{r}}{\partial v} & 2\mathbf{t}'_b \cdot \frac{\partial \mathbf{r}}{\partial v} \\ -2\mathbf{t}'_u \cdot \frac{\partial \mathbf{s}}{\partial a} & -2\mathbf{t}'_v \cdot \frac{\partial \mathbf{s}}{\partial a} & -2\mathbf{t}'_a \cdot \frac{\partial \mathbf{s}}{\partial a} & -2\mathbf{t}'_b \cdot \frac{\partial \mathbf{s}}{\partial a} \\ -2\mathbf{t}'_u \cdot \frac{\partial \mathbf{s}}{\partial b} & -2\mathbf{t}'_v \cdot \frac{\partial \mathbf{s}}{\partial b} & -2\mathbf{t}'_a \cdot \frac{\partial \mathbf{s}}{\partial b} & -2\mathbf{t}'_b \cdot \frac{\partial \mathbf{s}}{\partial b} \end{bmatrix}. \tag{17}$$

В особой точке нормали обеих поверхностей коллинеарны и, соответственно, касательные плоскости этих поверхностей совпадают, то есть векторы $\frac{\partial \mathbf{r}}{\partial u}, \frac{\partial \mathbf{r}}{\partial v}, \frac{\partial \mathbf{s}}{\partial a}, \frac{\partial \mathbf{s}}{\partial b}$ линейно зависимы. Отсюда следует, что $\frac{\partial \mathbf{r}}{\partial u} = a_1 * \frac{\partial \mathbf{s}}{\partial a} + b_1 * \frac{\partial \mathbf{s}}{\partial b}$ и $\frac{\partial \mathbf{r}}{\partial v} = a_2 * \frac{\partial \mathbf{s}}{\partial a} + b_2 * \frac{\partial \mathbf{s}}{\partial b}$, где $a_1^2 + b_1^2 + a_2^2 + b_2^2 \neq 0$. Учитывая последние равенства, билинейность скалярного произведения и матрицу (17), легко видеть, что $\nabla g_u = a_1 * \nabla g_a + b_1 * \nabla g_b$ и $\nabla g_v = a_2 * \nabla g_a + b_2 * \nabla g_b$, то есть $rank H = 2$ и, соответственно, $\det H = 0$. □

Поставим следующую задачу: какова должна быть функция от переменных u, v, a, b, чтобы ее глобальный минимум достигался только в особых точках кривых пересечения. Для ее решения рассмотрим следующую функцию:

$$f(u, v, a, b) = |\mathbf{r} - \mathbf{s}|^2 + 1 - |\mathbf{n}_r \cdot \mathbf{n}_s|, \tag{18}$$

где $\mathbf{r} = \mathbf{r}(u,v), \mathbf{s} = \mathbf{s}(a,b)$, а $\mathbf{n}_r$ и $\mathbf{n}_s$ – нормали к поверхностям $\mathbf{r}$ и $\mathbf{s}$. Функция (18) достигает глобального минимума только в особых точках кривых пересечения, при этом $\min f = 0$. Отсюда получаем, что из равенства $\nabla f = 0$ немедленно следует равенство $f = 0$, учитывая, что нулевое приближение u_0, v_0, a_0, b_0 является достаточно "хорошим". Последнее словосочетание означает следующее: пусть $\mathbf{p}_m(u_m, v_m, a_m, b_m)$ – точка глобального минимума, при этом существует ε-окрестность точки $\mathbf{p}_m$ такая, что для любой точки $\mathbf{p}$, лежащей внутри этой окрестности, выполняется неравенство $f(\mathbf{p}_m) - f(\mathbf{p}) \leq 0$. Полагаем, что $|\mathbf{p}_m - \mathbf{p}(u_0, v_0, a_0, b_0)| \leq \varepsilon$. Несложно показать, что для любой точки $\mathbf{p}$ такой, что $|\mathbf{p}_m - \mathbf{p}| \leq \varepsilon$, скалярное произведение $\mathbf{n}_r \cdot \mathbf{n}_s$ имеет определенный знак ($f = 0 \Leftrightarrow |\mathbf{r} - \mathbf{s}|^2 = 0$ и $|\mathbf{n}_r \cdot \mathbf{n}_s| - 1 = 0 \Rightarrow \mathbf{n}_r \cdot \mathbf{n}_s = \pm 1$), поэтому модуль, встречающийся в функции f, при численном дифференцировании не учитывается. В дальнейшем для определенности будем полагать, что $\mathbf{n}_r \cdot \mathbf{n}_s \geq 0$ и поэтому функция f приобретает вид

$$f(u,v,a,b) = |\mathbf{r} - \mathbf{s}|^2 + 1 - \mathbf{n}_r \cdot \mathbf{n}_s. \tag{19}$$

Обозначим через $f_1(u,v,a,b) = |\mathbf{r} - \mathbf{s}|^2$ и через $f_2(u,v,a,b) = 1 - \mathbf{n}_r \cdot \mathbf{n}_s$. С учетом введенных обозначений имеем: $\nabla f = \nabla f_1 + \nabla f_2$ и, соответственно, $H(f) = H(f_1) + H(f_2)$. Матрица Гессе $H(f_1)$ имеет вид (17), а матрица Гессе $H(f_2)$ может быть получена аналогично матрице $H(f_1)$.

Покажем это. Градиент ∇f_2 имеет вид:

$$\nabla f_2 = - \begin{bmatrix} \mathbf{n}_s \cdot \frac{\partial \mathbf{n}_r}{\partial u} \\ \mathbf{n}_s \cdot \frac{\partial \mathbf{n}_r}{\partial v} \\ \mathbf{n}_r \cdot \frac{\partial \mathbf{n}_s}{\partial a} \\ \mathbf{n}_r \cdot \frac{\partial \mathbf{n}_s}{\partial b} \end{bmatrix}. \tag{20}$$

Пусть в особой точке $\mathbf{n}_r \cdot \mathbf{n}_s = 1$, то есть $\mathbf{n}_r = \mathbf{n}_s$. С учетом этого факта и того, что вектор градиента ∇f_2 имеет вид (20), матрица Гессе $H(f_2)$ примет следующий вид:

$$H(f_2) = - \begin{bmatrix} \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_r}{\partial u^2} & \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_r}{\partial u \partial v} & \frac{\partial \mathbf{n}_r}{\partial u} \cdot \frac{\partial \mathbf{n}_s}{\partial a} & \frac{\partial \mathbf{n}_r}{\partial u} \cdot \frac{\partial \mathbf{n}_s}{\partial b} \\ \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_r}{\partial u \partial v} & \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_r}{\partial v^2} & \frac{\partial \mathbf{n}_r}{\partial v} \cdot \frac{\partial \mathbf{n}_s}{\partial a} & \frac{\partial \mathbf{n}_r}{\partial v} \cdot \frac{\partial \mathbf{n}_s}{\partial b} \\ \frac{\partial \mathbf{n}_r}{\partial u} \cdot \frac{\partial \mathbf{n}_s}{\partial a} & \frac{\partial \mathbf{n}_r}{\partial v} \cdot \frac{\partial \mathbf{n}_s}{\partial a} & \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_s}{\partial a^2} & \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_s}{\partial a \partial b} \\ \frac{\partial \mathbf{n}_r}{\partial u} \cdot \frac{\partial \mathbf{n}_s}{\partial b} & \frac{\partial \mathbf{n}_r}{\partial v} \cdot \frac{\partial \mathbf{n}_s}{\partial b} & \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_s}{\partial a \partial b} & \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_s}{\partial b^2} \end{bmatrix}. \tag{21}$$

Нельзя утверждать, что Гессиан матрицы $H(f_2)$ всегда равен нулю. Однако в некоторых случаях $\det H(f_2) = 0$. Например, когда одна из поверхностей является плоскостью.

Пусть это будет поверхность $\mathbf{s}(a,b)$. В этом случае все частные производные от $\mathbf{n}_s$ равны нулю, и матрица (21) примет вид

$$H(f_2) = - \begin{bmatrix} \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_r}{\partial u^2} & \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_r}{\partial u \partial v} & 0 & 0 \\ \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_r}{\partial u \partial v} & \mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_r}{\partial v^2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{22}$$

Замечание. Предполагается, что хотя бы один элемент матрицы (22) как в первой, так и во второй строке не равен нулю. Для этого, например, потребуем, чтобы

$$\mathbf{n}_s \cdot \frac{\partial^2 \mathbf{n}_r}{\partial u \partial v} \neq 0. \tag{23}$$

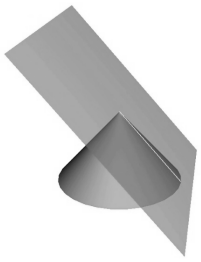


Рис. 8: Касание конуса и плоскости
Fig. 8: A cone and a plane touching

Это означает, что вектор $\frac{\partial^2 \mathbf{n}_r}{\partial u \partial v}$ не лежит в касательной плоскости, образованной векторами $\frac{\partial \mathbf{r}}{\partial u}$ и $\frac{\partial \mathbf{r}}{\partial v}$. Матрцу $H(f_2)$ с учетом (22) и последнего замечания можно записать следующим образом:

$$H(f_2) = -\begin{bmatrix} \nabla f_{2u} \\ \nabla f_{2v} \\ 0 \\ 0 \end{bmatrix}, \tag{24}$$

где $f_{2u}, f_{2v}, 0, 0$ – координаты вектора ∇f_2 градиента функции f_2.

Покажем, что в этом случае $\det H(f) \neq 0$, где $H(f) = H(f_1) + H(f_2)$. В наших обозначениях $H(f_1) = H(g)$ и, как мы ранее доказали, $\det H(f_1) = 0$, поскольку первая и вторая строки в матрице (17) являются линейными комбинациями третьей и четвертой строки этой матрицы. Таким образом, с учетом равенства (24) остается показать, что векторы ∇f_{2u} и ∇f_{2v} не являются линейными комбинациями третьей и четвертой строки матрицы (17). Действительно, предположим противное. Пусть, например, $\nabla f_{2u} = \gamma_1 * \nabla g_a + \varepsilon_1 * \nabla g_b$, где ∇g_a и ∇g_b – обозначения, введенные в равенстве (16). Тогда с учетом (17) и (22) получаем следующую систему относительно γ_1 и ε_1 :

$$\begin{cases} \gamma_1 * -2\mathbf{t}'_a \cdot \frac{\partial \mathbf{s}}{\partial a} + \varepsilon_1 * -2\mathbf{t}'_b \cdot \frac{\partial \mathbf{s}}{\partial a} = 0 \\ \gamma_1 * -2\mathbf{t}'_a \cdot \frac{\partial \mathbf{s}}{\partial b} + \varepsilon_1 * -2\mathbf{t}'_b \cdot \frac{\partial \mathbf{s}}{\partial b} = 0. \end{cases} \tag{25}$$

Ненулевые решения этой системы возможны только в том случае, когда ее определитель равен нулю. То есть

$$\begin{vmatrix} -2\mathbf{t}'_a \cdot \frac{\partial \mathbf{s}}{\partial a} & -2\mathbf{t}'_b \cdot \frac{\partial \mathbf{s}}{\partial a} \\ -2\mathbf{t}'_a \cdot \frac{\partial \mathbf{s}}{\partial b} & -2\mathbf{t}'_b \cdot \frac{\partial \mathbf{s}}{\partial b} \end{vmatrix} = 0.$$

Последний определитель, очевидно, может быть равен нулю только в случае, когда векторы $\frac{\partial \mathbf{s}}{\partial a}$ и $\frac{\partial \mathbf{s}}{\partial b}$ линейно зависимы, что невозможно, поскольку поверхность $\mathbf{s}(a, b)$ является гладкой в особой точке кривой пересечения. Отсюда получаем, что наше предположение неверно и вектор ∇f_{2u} не выражается через векторы ∇g_a и ∇g_b. Аналогично можно показать, что вектор ∇f_{2v} также не выражается через векторы ∇g_a и ∇g_b. На основе двух последних утверждений можно сделать вывод, что

$\det H(f) \neq 0$ и, соответственно, для поиска глобального минимума функции (18) можно методом Ньютона–Рафсона решить систему

$$\Delta_k = H^{-1}(f)\nabla f, \tag{26}$$

где Δ_k – вектор приращений аргументов функции f на k-м шаге итерационного процесса. Этот вектор определен в формуле (14).

Недостатком метода, описанного выше, является использование дополнительного условия (23), наложенного на поверхность $\mathbf{r}(u, v)$, что сужает разнообразие поверхностей, используемых в пересечении. Однако если предполагается, что $\mathbf{r}(u, v)$ является графиком явной функции и его пересечение с плоскостью $z = 0$ есть алгебраическая кривая с особой точкой кратности не менее трех, то условие (23) является излишним.

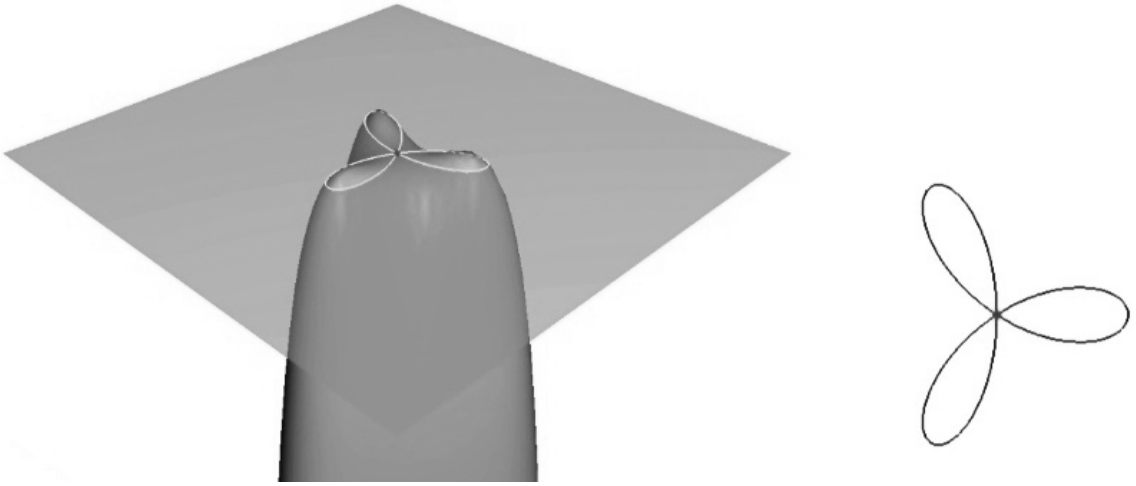


Рис. 9: $z = (x^2 + y^2)^2 + 3x^2y - y^3 \cap z = 0$
Fig. 9: $z = (x^2 + y^2)^2 + 3x^2y - y^3 \cap z = 0$

Действительно, пусть $\mathbf{r}(u, v) = (u, v, f(u, v))$, а $\mathbf{s}(a, b) = (a, b, 0)$. При этом $\frac{\partial \mathbf{r}}{\partial u} = (1, 0, \frac{\partial f}{\partial u})$ и $\frac{\partial \mathbf{r}}{\partial v} = (0, 1, \frac{\partial f}{\partial v})$. С другой стороны, $\mathbf{n}_r = \frac{1}{\sqrt{\frac{\partial f}{\partial u}^2 + \frac{\partial f}{\partial v}^2 + 1}}(\frac{\partial f}{\partial u}, \frac{\partial f}{\partial v}, -1) = (0, 0, -1)$ и, соответственно, $\frac{\partial f}{\partial u} = \frac{\partial f}{\partial v} = 0$. С учетом последних замечаний матрица $H(f_1)$ имеет вид

$$H(f_1) = 2\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{27}$$

Матрица $H(f_2)$ полностью вырождается, а именно:

$$H(f_2) = \begin{bmatrix} (\frac{\partial^2 \mathbf{n}_r}{\partial u^2})_z & (\frac{\partial^2 \mathbf{n}_r}{\partial y \partial v})_z & 0 & 0 \\ (\frac{\partial^2 \mathbf{n}_r}{\partial u \partial v})_z & (\frac{\partial^2 \mathbf{n}_r}{\partial v^2})_z & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{28}$$

поскольку все смешанные частные производные второго порядка от функции

$$(\mathbf{n}_r)_z = -\frac{1}{\sqrt{\frac{\partial f}{\partial u}^2 + \frac{\partial f}{\partial v}^2 + 1}},$$

как несложно видеть, равны нулю в силу следующего условия: $\frac{\partial^2 f}{\partial u^2} = \frac{\partial^2 f}{\partial v^2} = \frac{\partial^2 f}{\partial u \partial v} = 0$.

В данном случае можно положить, что $a = u$ и $b = v$. Таким образом, матрица $H(f)$ примет вид

$$H(f) = 2\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{29}$$

и система (26) решается методом Ньютона–Рафсона (как система двух уравнений с двумя неизвестными).

Если на поверхности $\mathbf{r}(u, v)$ и $\mathbf{s}(a, b)$ не накладываются дополнительные условия, то задача о поиске особых точек кривой пересечения поверхностей может быть решена одним из следующих способов.

Первый способ предполагает использование псевдообратной матрицы $H(f)^+$ для матрицы $H(f)$, где f – функция, определенная в (18). Соответственно, наилучшее приближенное решение $\widetilde{\Delta_k}$ имеет вид

$$\widetilde{\Delta_k} = H(f)^+ \nabla f.$$

Однако и этот метод не лишен недостатков. Во-первых, если $rank H(f) = 0$, то псевдообратная матрица $H(f)^+$ не определена. Во-вторых, если $rank H(f) > 0$, то матрицу $H(f)$ можно представить в виде $H(f) = U\Lambda V^T$, где U и V – ортогональные матрицы, а Λ – диагональная матрица, состоящая из собственных чисел матрицы $H(f)$. При этом $H(f)^+ = V\Lambda^{-1}U^T$. Для поиска диагональной матрицы Λ используется спектральное разложение, которое строится итерационными методами. В случае, когда $\mathbf{r}(u, v) = (u, v, f(u, v))$ – график явной функции, а $\mathbf{s}(a, b) = (a, b, 0)$ – координатная плоскость $z = 0$, вместо функции f можно рассматривать функцию $\widetilde{f}$:

$$\widetilde{f}(u, v, a, b) = 1 - |\mathbf{n}_r \cdot \mathbf{n}_s|. \tag{30}$$

Последняя функция является упрощенным вариантом функции (18). Как уже говорилось, функция (18) достигает глобального минимума только в особых точках кривых пересечения, при этом $\min f = 0$. Функция (30) также достигает глобального минимума в особых точках кривых пересечения и при этом $\min \widetilde{f} = 0$. Однако $\widetilde{f}$ может достигать глобального минимума и в других точках из области определения (в случае, когда $\mathbf{r}(u, v)$ – график функции, а $\mathbf{s}(a, b)$ – плоскость, множество таких точек – конечно). Пусть (u_s, v_s, a_s, b_s) – особая точка кривой пересечения и (u_0, v_0, a_0, b_0) – точка нулевого приближения. Обозначим через

$$\varepsilon = \sqrt{(u_0 - u_s)^2 + (v_0 - v_s)^2 + (a_0 - a_s)^2 + (b_0 - b_s)^2}$$

расстояние между этими точками. Тогда, если полагать, что точка нулевого приближения (u_0, v_0, a_0, b_0) выбрана достаточно близко к особой точке кривой пересечения, то можно быть уверенным, что ε-окрестность точки (u_s, v_s, a_s, b_s) не содержит других точек глобального минимума. Соответственно, при решении задачи минимизации вместо функции (18) можно использовать функцию (30). В качестве нулевого

приближения можно использовать точку барицентра, о которой подробно написано во второй части статьи.

Второй способ также основан на использовании псевдообратной матрицы. Пусть f – функция, определенная равенством (19). Тогда

$$f = 0 \Leftrightarrow \begin{cases} |\mathbf{r} - \mathbf{s}|^2 = 0 \\ 1 - \mathbf{n}_r \cdot \mathbf{n}_s = 0 \end{cases} \Leftrightarrow \begin{cases} \mathbf{r}_x - \mathbf{s}_x = 0 \\ \mathbf{r}_y - \mathbf{s}_y = 0 \\ \mathbf{r}_z - \mathbf{s}_z = 0 \\ 1 - \mathbf{n}_r \cdot \mathbf{n}_s = 0 \end{cases} . \tag{31}$$

Решение последний системы находится с помощью матрицы A_f, где

$$A_f = \begin{bmatrix} \frac{\partial \mathbf{r}_x}{\partial u} & \frac{\partial \mathbf{r}_\mathbf{x}}{\partial v} & -\frac{\partial \mathbf{s}_\mathbf{x}}{\partial a} & -\frac{\partial \mathbf{s}_\mathbf{x}}{\partial b} \\ \frac{\partial \mathbf{r}_y}{\partial u} & \frac{\partial \mathbf{r}_\mathbf{y}}{\partial v} & -\frac{\partial \mathbf{s}_\mathbf{y}}{\partial a} & -\frac{\partial \mathbf{s}_\mathbf{y}}{\partial b} \\ \frac{\partial \mathbf{r}_z}{\partial u} & \frac{\partial \mathbf{r}_\mathbf{z}}{\partial v} & -\frac{\partial \mathbf{s}_\mathbf{z}}{\partial a} & -\frac{\partial \mathbf{s}_\mathbf{z}}{\partial b} \\ \frac{\partial p}{\partial u} & \frac{\partial p}{\partial v} & \frac{\partial p}{\partial a} & \frac{\partial p}{\partial b} \end{bmatrix}, \tag{32}$$

где $p = p(u, v, a, b) = 1 - \mathbf{n}_r \cdot \mathbf{n}_s$. Нельзя утверждать, что $rank A_f = 4$ (например, когда совпавшие касательные плоскости в особой точке кривой пересечения параллельны координатным плоскостям).

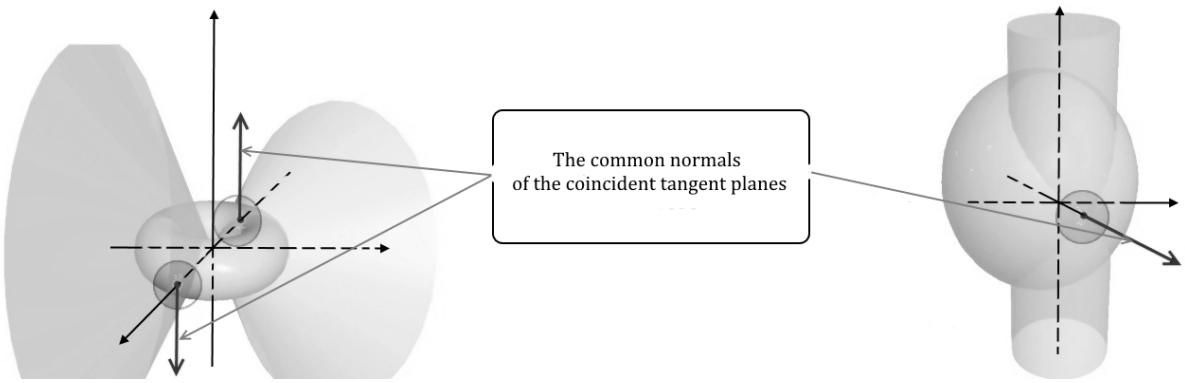


Рис. 10: Параллельность нормалей осям координат
Fig. 10: The normals parallel to the coordinate axes

В этом случае какая-то из первых трех строк матрицы (32) становится нулевой. Также $rank A_f < 4$ в случае, когда $\mathbf{r}(u, v) = (u, v, f(u, v))$ – график явной функции, а $\mathbf{s}(a, b) = (a, b, 0)$ – координатная плоскость $z = 0$. В этом случае матрица A_f примет вид

$$A_f = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ \frac{\partial f}{\partial u} & \frac{\partial f}{\partial v} & 0 & 0 \\ \frac{\partial p}{\partial u} & \frac{\partial p}{\partial v} & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ \frac{\partial p}{\partial u} & \frac{\partial p}{\partial v} & 0 & 0 \end{bmatrix}. \tag{33}$$

Очевидно, однако, что $rank A_f > 0$, поскольку в противном случае $\frac{\partial \mathbf{r}}{\partial u} = 0$, $\frac{\partial \mathbf{r}}{\partial v} = 0$, $\frac{\partial \mathbf{s}}{\partial a} = 0$, $\frac{\partial \mathbf{s}}{\partial b} = 0$, что невозможно в силу гладкости поверхностей. Отсюда следует, что псевдообратная матрица A_f^+ существует. По своей структуре матрица (32) проще

матрицы $H(f)$, поэтому при компьютерной реализации алгоритма поиска особой точки кривой пересечения ее использование более предпочтительно.

К недостатком первого и второго способа можно отнести тот факт, что они не являются универсальными. Третий способ поиска глобального минимума функции (18) основан на комбинировании методов покоординатного спуска и оврагов. Этот способ вполне можно считать универсальным, поскольку он применим к любым гладким поверхностям.

Будем предполагать, что $rankH(f) > 0$, в противном случае для решения задачи минимизации рационально применить второй способ, описанный выше. В силу последнего допущения, функция f, определенная в формуле (18), в окрестности глобального минимума раскладывается в ряд Тейлора следующим образом:

$$f(u,v,a,b) = f(\mathbf{p}_m) + \frac{\partial^2 f}{\partial u^2}\Delta u^2 + \frac{\partial^2 f}{\partial v^2}\Delta v^2 + \frac{\partial^2 f}{\partial a^2}\Delta a^2 + \frac{\partial^2 f}{\partial b^2}\Delta b^2 + \frac{\partial^2 f}{\partial u \partial v}\Delta u \Delta v +$$

$$\frac{\partial^2 f}{\partial a \partial b}\Delta a \Delta b + \frac{\partial^2 f}{\partial a \partial v}\Delta a \Delta v + \frac{\partial^2 f}{\partial b \partial v}\Delta b \Delta v + \frac{\partial^2 f}{\partial a \partial u}\Delta a \Delta u + \frac{\partial^2 f}{\partial b \partial u}\Delta b \Delta u + ..., \qquad (34)$$

где $\mathbf{p}_m(u_m, v_m, a_m, b_m)$ – точка глобального минимума. Коэффициенты ряда Тейлора (34) при вторых степенях переменных есть в точности элементы матрицы $H(f)$, которая в окрестности точки $\mathbf{p}_m$ является положительно определенной ($\mathbf{p}_m$ – точка минимума). Пользуясь положительной определенностью матрицы $H(f)$, пренебрегая членами порядка≥ 3 и учитывая, что $rankH(f) > 0$, можно сделать вывод, что поверхности уровня функции f в окрестности точки $\mathbf{p}_m$ есть трехмерные эллипсоиды. Поверхности уровня определяют рельеф функции в окрестности точки $\mathbf{p}_m$. Рельеф является овражным, если эллипсоиды сильно вытянуты вдоль какой-то полуоси. Поскольку $rankH(f) > 0$ матрицу $H(f)$ можно перевести в диагональную матрицу Λ, составленную из собственных чисел матрицы $H(f)$. Тогда в базисе, составленном из собственных векторов матрицы $H(f)$, разложение (34) примет вид

$$f(u',v',a',b') = f(\mathbf{p'}_m) + \frac{\partial^2 f}{\partial u'^2}\Delta u'^2 + \frac{\partial^2 f}{\partial v'^2}\Delta v'^2 + \frac{\partial^2 f}{\partial a'^2}\Delta a'^2 + \frac{\partial^2 f}{\partial b'^2}\Delta b'^2 + ..., \qquad (35)$$

где u', v', a', b' – координаты точек в новом базисе, а $\mathbf{p}'_m$ – точка глобального минимума. Разложение (35) задает канонические представления эллипсоидов. При этом, если $rankH(f) < 4$, как минимум один из коэффициентов разложения (35) равен нулю и, тем самым, эллипсоиды вырождаются, определяя овражный рельеф вблизи точки глобального минимума. В связи с этим в случае, когда $0 < rankH(f) \leq 4$, для минимизации функции f можно использовать метод "оврагов".

Суть метода заключается в следующем. Выбирается точка $\mathbf{p}_0(u_0, v_0, a_0, b_0)$, достаточно близкая к глобальному минимуму (например точка барицентра (см. вторую часть статьи)). Из нее производится покоординатный спуск. То есть решается задача одномерной минимизации функции $f(\mathbf{p}(t)) = f(u(t), v_0, a_0, b_0)$ с помощью метода Ньютона. Пусть t_0 – найденное решение. Далее решается задача, аналогичная предыдущей, но для второй координаты, а именно, $f(\mathbf{p}(t)) = f(u(t_0), v(t), a_0, b_0)$. После поочередного прохождения всех координат точка $\mathbf{p}_0$ имеет координаты $(u(t_0), v(t_1), a(t_2), b(t_3))$ и расположена на дне оврага. Следует заметить, что если поверхности уровня ориентированы по осям координат, то покоординатный спуск

за четыре шага дает достаточно "хорошее" приближение к точке глобального минимума.

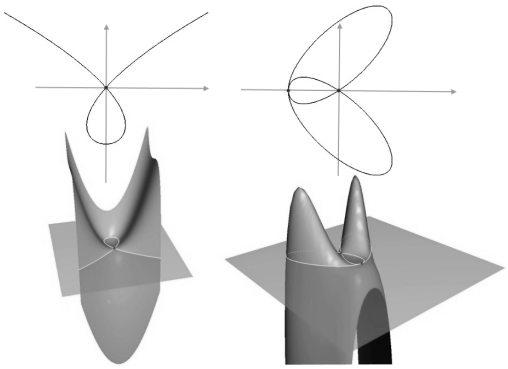


Рис. 11: Кривые уровня ориентированы по осям координат
Fig. 11: The level curves are oriented along the coordinate axes

Однако заранее неизвестна ориентация эллипсоидов, поэтому выполняется еще один покоординатный спуск из точки $\mathbf{p}_1(u_1, v_1, a_1, b_1)$. В качестве точки $\mathbf{p}_1$ можно взять конец одной из ветвей кривой пересечения вблизи особой точки (см. вторую часть статьи). Наконец, для того чтобы продвинуться по дну оврага к точке глобального минимума, решается задача одномерной минимизации для функции

$$f(\mathbf{p}_1 \pm (\mathbf{p}_1 - \mathbf{p}_0)h) \tag{36}$$

как функции от параметра h с помощью метода Ньютона. В формуле (36) ставится знак плюс, если $f(\mathbf{p}_1) < f(\mathbf{p}_0)$, и минус в противном случае.

Два покоординатных спуска из точек $\mathbf{p}_0$ и $\mathbf{p}_1$ вместе с минимизацией функции (36) составляют один цикл работы алгоритма. Приближение определяется как достаточно "хорошее", если число циклов $\geq c$, где c – некоторая константа (например $c = 20$) или $|f(\mathbf{p}_1) - f(\mathbf{p}_0)| \leq \varepsilon$, где ε – также некоторая константа (например $\varepsilon = 10^{-6}$).

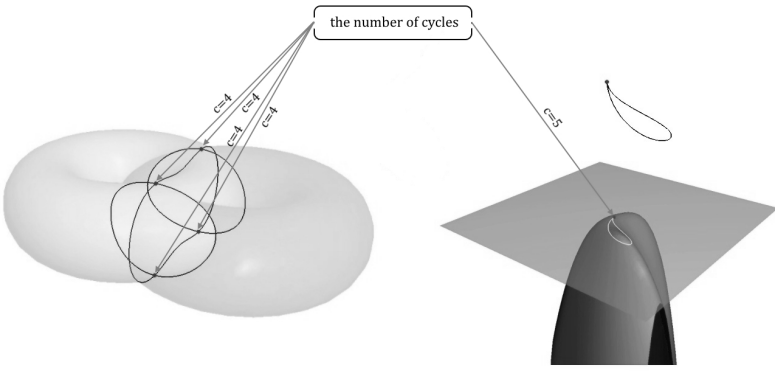


Рис. 12: Особые точки кривых как результаты работы циклов длины 4 и 5
Fig. 12: Singular points of curves as the results of operation of cycles of lengths 4 and 5

Список литературы / References

[1] Голованов Н. Н., *Геометрическое моделирование*, Издательство физико-математической литературы, М., 2002, 472 с.; English transl.: Golovanov N. N., *Geometric modeling*, Create Space Independent Publishing Platform, 2014, 348 pp.

[2] Шенен П. и др., *Математика и САПР*, в 2-х томах, Мир, М., 1988; [Shenen P. et al, *Matematika i SAPR*, in 2 vol., Mir, M., 1988, (in Russian).]

[3] Фокс А., Пратт М., *Вычислительная геометрия. Применение в проектировании и на производстве*, Мир, М., 1982, 304 с.; in English: Faux I. D., Pratt M. J., *Computational geometry for design and manufacture*, Ellis Horwood Ltd., 1979.

[4] Hopcroft J., Kannan R., *Computer Science Theory for the Information Age*, Springer, 2012.

[5] Golub J., Reinsch C., "Singular Value Decomposition and Least Squares Solutions", *Numerische Mathematik*, **14**:5 (1970), 403–420.

[6] Butterfield K., *Ph. D. Thesis*, Brunel University Uxbridge, 1978.

[7] Калиткин Н. Н., *Численные методы*, учебное пособие, БХВ-Петербург, 2011, 592 с.; [Kalitkin N. N., *Chislennyye metody*, uchebnoye posobiye, BKHV-Peterburg, 2011, 592 pp., (in Russian).]

[8] Вермель А. В., Вермель В. Д., Калитин Е. И., "Пересечение поверхностей агрегатов в аэродинамической компоновке самолета", *Ученые записки ЦАГИ*, **XLII**:5 (2011), 92–106; [Vermel A. V., Vermel V. D., Kalitin E. I., "Intersection of airplane component surfaces for design of an aerodynamic layout", *Uchenyye zapiski TsAGI*, **XLII**:5 (2011), 92–106, (in Russian).]

[9] Bajaj C. L., Hoffmann C. M., Lynch R. E., Hoptcroft J. T., "Tracing surface intersections", *Computer Aided Geometric Design*, **5** (1988), 285–307.

Uvarov A. D., "Singular Points of Curves", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 692–710.

DOI: 10.18255/1818-1015-2018-6-692-710

Abstract. In this paper, we consider the key problem of geometric modeling, connected with the construction of the intersection curves of surfaces. Methods for constructing the intersection curves in complex cases are found: by touching and passing through singular points of surfaces. In the first part of the paper, the problem of determining the tangent line of two surfaces given in parametric form is considered. Several approaches to the solution of the problem are analyzed. The advantages and disadvantages of these approaches are revealed. The iterative algorithms for finding a point on the line of tangency are described. The second part of the paper is devoted to methods for overcoming the difficulties encountered in solving a problem for singular points of intersection curves, in which a regular iterative process is violated. Depending on the type of problem, the author dwells on two methods. The first of them suggests finding singular points of curves without using iterative methods, which reduces the running time of the algorithm of plotting the intersection curve. The second method, considered in the final part of the article, is a numerical method. In this part, the author introduces a function that achieves a global minimum only at singular points of the intersection curves and solves the problem of minimizing this function. The application of this method is very effective in some particular cases, which impose restrictions on the surfaces and their arrangement. In conclusion, this method is considered in the case when the function has such a relief, that in the neighborhood of the minimum point the level surfaces are strongly elongated ellipsoids. All the images given in this article are the result of the work of algorithms on methods proposed by the author. Images are built in the author's software environment.

Keywords: singular point, intersection, touching, matrix, gradient

On the authors:
Artem D. Uvarov, orcid.org/0000-0002-0624-3877, PhD,
K.D. Ushinsky Yaroslavl State Pedagogical University,
108 Respublikanskaya str., Yaroslavl 150000, Russia, e-mail: artiom_uvarov@inbox.ru

Моделирование и анализ информационных систем. Т. 25, № 6 (2018), с. 711–725
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 711–725

Модели процессов
Process Modeling

DOI: 10.18255/1818-1015-2018-6-711-725

УДК 004.023

Применение генетического алгоритма для нахождения редакционного расстояния между моделями процессов

Каленкова А. А., Колесников Д. А.

Поступила в редакцию 1 сентября 2018
После доработки 10 ноября 2018
Принята к публикации 20 ноября 2018

Аннотация. Поиск редакционного расстояния между графовыми моделями (определение схожести графовых моделей) является важной задачей в различных областях компьютерных наук, таких как анализ изображений, машинное обучение, химическая информатика. В последнее время, в связи с развитием методов извлечения и анализа процессов, появилась необходимость в адаптации существующих методов сравнения графовых моделей для анализа моделей процессов (аннотированных графов), извлекаемых из логов событий информационных систем. Методы нахождения минимального редакционного расстояния между графами могут быть использованы для обнаружения шаблонов (подпроцессов), а также для сравнения извлекаемых моделей процессов. Как было показано экспериментально и теоретически обосновано, точные методы нахождения минимального редакционного расстояния между извлекаемыми моделями процессов (и графами в общем случае) имеют большую временную сложность и могут быть применены лишь к небольшим моделям процессов. В этой статье мы оцениваем точность и временные характеристики генетического алгоритма, применяемого для нахождения расстояний между моделями процессов, извлекаемых из логов событий. В частности мы находим расстояния между BPMN (Business Process Model and Notation) моделями, извлекаемыми из логов событий с помощью различных алгоритмов синтеза. В этой работе показано, что представленный генетический алгоритм позволяет в значительной степени уменьшить время вычислений, при этом показывая результаты, близкие к оптимальным (минимальным редакционным расстояниям).

Ключевые слова: минимальное редакционное расстояние между графами, извлечение и анализ процессов, BPMN (Business Process Model and Notation), генетический алгоритм

Для цитирования: Каленкова А. А., Колесников Д. А., "Применение генетического алгоритма для нахождения редакционного расстояния между моделями процессов", *Моделирование и анализ информационных систем*, **25**:6 (2018), 711–725.

Об авторах: Каленкова Анна Алексеевна, orcid.org/0000-0002-5088-7602, ст. науч. сотр.,
Национальный исследовательский университет «Высшая школа экономики», лаборатория ПОИС
ул. Мясницкая, 20, г. Москва, 101000 Россия, e-mail: akalenkova@hse.ru

Колесников Данил Александрович, orcid.org/0000-0002-9010-8415, студент
Национальный исследовательский университет «Высшая школа экономики», факультет компьютерных наук
ул. Мясницкая, 20, г. Москва, 101000 Россия, e-mail: dakolesnikov@edu.hse.ru

Благодарности:
Исследование выполнено при поддержке Гранта Президента РФ для молодых российских ученых — кандидатов наук МК-4188.2018.9.

Введение

Информационные системы, автоматизирующие рабочие процессы в различных областях, таких как медицина, образование, предоставление государственных услуг, финансы, сохраняют историю своей работы в виде логов событий. Технологии извлечения и анализа процессов (известные также как process mining) позволяют строить модели процессов, обобщающие поведение систем, представленное в логах событий [1]. Несмотря на то что извлекаемые модели, как правило, достаточно наглядно визуализируют процесс, их структура также может быть более детально проанализирована. Так, например, шаблоны моделирования процессов, такие как «последовательность», «выбор», «параллельное исполнение», «цикл» и другие более сложные структуры, могут быть обнаружены и выделены автоматически. Кроме того, извлекаемые модели процессов могут быть сопоставлены с моделями, определяющими эталонное (ожидаемое) поведение, и, таким образом, отклонения в поведении (использовании) системы могут быть явно определены. В работах [4,6] было предложено использовать редакционное расстояние между графами для сопоставления моделей процессов, извлекаемых из логов событий. Редакционное расстояние показывает степень схожести/различия двух графов. Более точно оно определяется как количество элементарных шагов (добавление/удаление вершины, добавление/удаление дуги), которые необходимо выполнить, чтобы преобразовать один граф в другой. Рассматривая модели процессов как ориентированные графы, методы нахождения минимального редакционного расстояния можно адаптировать для их сопоставления. В этом случае названия вершин и дуг, а также их типы должны быть дополнительно учтены. Точный алгоритм поиска A* [7] был реализован в инструменте [6] для сопоставления BPMN (Business Process Model and Notation) [8] моделей процессов. Этот инструмент был использован для сопоставления BPMN моделей, извлеченных из логов событий сервисов предоставления государственных услуг [4]. Несмотря на то что для сокращения времени нахождения минимального редакционного расстояния были использованы некоторые процессно-ориентированные эвристики [4], точный алгоритм нахождения минимального расстояния был применим лишь к небольшим моделям процессов. Это может быть объяснено в том числе тем обстоятельством, что задача нахождения минимального редакционного расстояния между графами является NP-полной [5].

В этой работе мы исследуем возможность нахождения минимального редакционного расстояния между моделями процессов с помощью генетического алгоритма. Методы нахождения минимального редакционного расстояния между моделями процессов, основанные на использовании генетических алгоритмов, были описаны в [9]. Генетические алгоритмы были также использованы в комбинации с другими алгоритмами нахождения минимального редакционного расстояния в [10]. В отличие от работы [9] мы предлагаем алгоритм, который также учитывает специфику графовых моделей процессов (названия и типы вершин). Для оценки времени и точности предложенного алгоритма мы сравниваем модели процессов, извлеченные из логов событий с помощью альфа алгоритма [2] и индукционного алгоритма [3], используя методы приведения извлекаемых моделей к BPMN формату [11].

Статья имеет следующую структуру. В разделе 1 представлены основные понятия и определения, использующиеся в тексте статьи. Раздел 2 содержит описание

точного алгоритма сопоставления BPMN моделей процессов. В разделе 3 представлен новый метод сопоставления BPMN моделей, основанный на принципах генетического алгоритма. Раздел 4 содержит результаты экспериментов (точность и время работы предложенного алгоритма).

1. Основные определения

В этом разделе вводится понятие плоских BPMN (Business Process Model and Notation) моделей, также даются определения графов бизнес-процессов и расстояний между ними. Эти понятия используются далее в статье.

Несмотря на большое количество классов BPMN моделей, предложенных Object Management Group (OMG) в формальной спецификации [8], мы рассматриваем только плоские модели процессов, формализующие поток управления. Эти модели могут быть получены из логов событий в два шага: (1) низкоуровневая модель процесса (сеть Петри, дерево процесса, каузальная сеть) синтезируется из лога событий; (2) эта модель конвертируется в высокоуровневую модель с помощью алгоритмов, предложенных в работе [11].

Плоские BPMN модели, формализующие поток управления и полученные с помощью алгоритмов преобразования из сетей Петри, деревьев процессов и каузальных сетей, представлены следующим базовым набором элементов: *действия*, *исключающие* и *параллельные маршрутизаторы*, *начальные* и *конечные события*, *потоки управления* (Рис. 1).

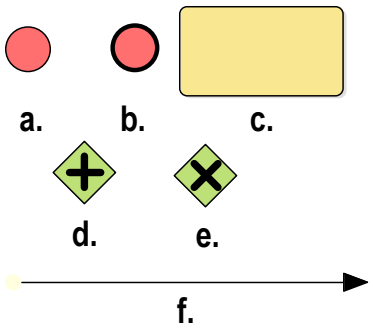


Рис. 1. Элементы плоских BPMN моделей
Fig. 1. Elements of flat BPMN models

Начальные (Рис. 1 a) и *конечные события* (Рис. 1 b) обозначают начало и завершение процесса соответственно. *Действия* (Рис. 1 c) моделируют атомарные шаги процесса. *Параллельные* (Рис. 1 d) и *исключающие маршрутизаторы* (Рис. 1 e) используются для моделирования ветвей процесса, исполняющихся параллельно или взаимоисключающих друг друга.

Пример BPMN модели, описывающей простую процедуру бронирования, приведен на Рис. 2. Сначала пользователь регистрируется, бронирует самолет и гостиницу (эти действия выполняются параллельно и каждое из них может быть пропущено) и оплачивает заказ.

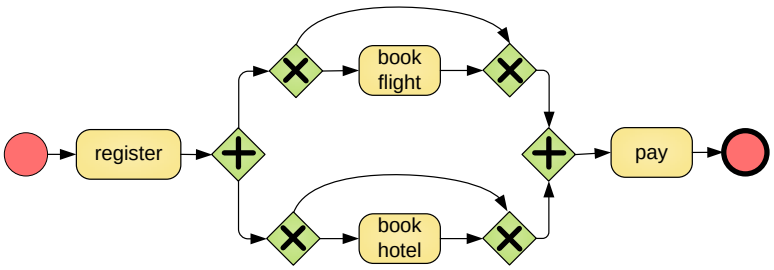


Рис. 2. BPMN модель, описывающая процедуру бронирования
Fig. 2. A BPMN model of a booking procedure

Плоские BPMN модели могут быть рассмотрены как ориентированные графы с типами и именами вершин. Мы будем называть их *графы бизнес-процессов*. Дадим их формальное определение.

Графом бизнес-процесса называется кортеж $G = (N, E, t, l)$, в котором N – это множество вершин, $E \subseteq (N \times N)$ – множество направленных дуг, $t : N \to T$, где $T = \{start\ event, end\ event, task, exclusive\ gateway, parallel\ gateway\}$ – функция, определяющая типы вершин, $l : N \to L$ – функция, задающая имена вершин, L – множество имен.

Рассмотрим два графа бизнес-процессов: $G_k = (N_k, E_k, t_k, l_k)$, $k = 1, 2$. *Расстояние* между ними вычисляется путем построения *редакционного отношения* $R \subseteq (N_1 \cup E_1 \cup \{\epsilon, \delta\}) \times (N_2 \cup E_2 \cup \{\epsilon, \delta\})$, где $\epsilon, \delta \notin N_1 \cup N_2 \cup E_1 \cup E_2$, такого, что выполняются следующие условия:

1. для каждого элемента $i_1 \in N_1 \cup E_1$ ($i_2 \in N_2 \cup E_2$) существует один и только один элемент $i_2 \in N_2 \cup E_2 \cup \{\epsilon, \delta\}$ ($i_1 \in N_1 \cup E_1 \cup \{\epsilon, \delta\}$), такой что $(i_1, i_2) \in R$;

2. если $i_1 \in \{\epsilon, \delta\}$ ($i_2 \in \{\epsilon, \delta\}$), тогда $i_2 \notin \{\epsilon, \delta\}$ ($i_1 \notin \{\epsilon, \delta\}$);

3. если $i_1 \in N_1$ и $(i_1, i_2) \in R$, тогда $i_2 \in N_2 \cup \{\epsilon, \delta\}$, и если при этом $i_2 \in N_2$, тогда типы вершин совпадают, т.е. $t_1(i_1) = t_2(i_2)$;

4. если $i_1 \in E_1$ и $(i_1, i_2) \in R$, тогда $i_2 \in E_2 \cup \{\epsilon, \delta\}$;

5. для двух любых дуг $(i_1, i_1') \in E_1$, $(i_2, i_2') \in E_2$ условие $((i_1, i_1'), (i_2, i_2')) \in R$ выполняется тогда и только тогда, когда $(i_1, i_2) \in R$ и $(i_1', i_2') \in R$.

Если для некоторого элемента i выполняется $(i, \epsilon) \in R$ ($(\epsilon, i) \in R$), то мы говорим, что элемент i *удаляется* (*добавляется*). Если $(i, \delta) \in R$ ($(\delta, i) \in R$), то мы говорим, что для i не существует соответствующего элемента (это будет использовано для представления промежуточных результатов сравнения).

Сравним два графа бизнес-процессов $G_k = (N_k, E_k, t_k, l_k)$, $k = 1, 2$, построив редакционное отношение R. Для каждого $r = (i_1, i_2) \in R$ стоимость будет определена следующим образом:

$$cost(r) = \begin{cases} lev(l_1(i_1), l_2(i_2)) \cdot c_{lev}, & i_1 \in N_1, i_2 \in N_2, \\ 0, & i_1 \in E_1, i_2 \in E_2, \\ c_{delete}, & i_2 = \epsilon, \\ c_{insert}, & i_1 = \epsilon, \\ 0, & i_1 = \delta \vee i_2 = \delta. \end{cases}$$

Функция *lev* будет определять расстояние Левенштейна [12] между двумя строками, c_{lev} – специальный коэффициент; c_{delete} и c_{insert} обозначают стоимости операций удаления и добавления соответственно.

Общая стоимость редакционного отношения R определяется как сумма стоимостей всех пар, принадлежащих этому отношению: $cost(R) = \sum_{r \in R} cost(r)$.

Минимальное редакционное отношение между двумя графами бизнес-процессов – это редакционное отношение между этими графами, имеющее минимальную стоимость, такое, что оно не содержит пар с элементом δ (соответствия для всех элементов определены). Стоимость этого отношения называется *минимальным расстоянием редактирования*.

2. Точный алгоритм нахождения минимального редакционного расстояния

В этом разделе приведено описание точного алгоритма нахождения минимального редакционного расстояния между графами бизнес-процессов (Алгоритм 1).

На каждом шаге алгоритм берет из очереди редакционное отношение с минимальной стоимостью, затем из этого отношения создаются все возможные новые отношения, в которых некоторой вершине, пока не имеющей соответствия (соответствующей δ), подбираются в соответствие элементы другой модели того же типа или ϵ. Инцидентные дуги этой вершины также обрабатываются. После этого все новые редакционные отношения добавляются в очередь. Алгоритм останавливает свою работу, когда минимальное по стоимости редакционное отношение не имеет пар, содержащих δ, то есть для всех элементов моделей определены соответствия. Стоимость этого редакционного отношения является минимальным редакционным расстоянием между графами бизнес-процессов, и она возвращается алгоритмом в качестве результата.

На Рис. 3 приведен пример сопоставления графов бизнес-процессов, моделирующих процедуру бронирования. Граф бизнес-процесса, представленный на Рис. 3 a, моделирует процедуру бронирования, представленную ранее (Рис. 2). Модель, представленная на Рис. 3 b, предполагает возможность сброса. Для того чтобы преобразовать первый граф бизнес-процесса (Рис. 3 a) во второй (Рис. 3 b), необходимо удалить две дуги и добавить структуру, соответствующую сбросу. Сама процедура бронирования является единой для обеих графовых моделей.

Для того чтобы сделать поиск минимального редакционного расстояния более быстрым, может быть использована так называемая *эвристическая функция*.

Data: $G_1 = (N_1, E_1, t_1, l_1)$ и $G_2 = (N_2, E_2, t_2, l_2)$ – графы бизнес-процессов;
Result: минимальное редакционное расстояние между G_1 и G_2;
\\R_{init} – *начальное редакционное отношение*;
$R_{init} \leftarrow \{\}$;
for $i_1 \in N_1 \cup E_1$ **do**
  $R_{init} \leftarrow R_{init} \cup \{(i_1, \delta)\}$;
end
for $i_2 \in N_2 \cup E_2$ **do**
  $R_{init} \leftarrow R_{init} \cup \{(\delta, i_2)\}$;
end
\\Q - *очередь редакционных отношений, отсортированных по стоимости*;
$Q \leftarrow \langle R_{init} \rangle$;
while *true* **do**
  взять редакционное отношение с наименьшей стоимостью;
  $R_{min} \leftarrow takeMinCostRelation(Q)$;
  if (R_{min} *содержит пары с* δ) **then**
    $i \leftarrow takeNodeRelatedtoDelta(R_{min})$;
    $Q.remove(R_{min})$;
    добавить все возможные пары для элементы i *в отношении* R_{min};
    $Q.add(expand(R_{min}, i))$;
  else
    return $cost(R_{min})$;
  end
end

Алгоритм 1: Нахождение минимального редакционного расстояния между графами бизнес-процессов.

Она определяется на множестве редакционных отношений и задается следующим образом:

$$H(R) = \sum\nolimits_{t \in T} \begin{cases} |I_1^t| - |I_2^t| \cdot c_{delete}, & |I_1^t| \geq |I_2^t|, \\ |I_2^t| - |I_1^t| \cdot c_{insert}, & |I_2^t| > |I_1^t|. \end{cases}$$

$I_1^t \subseteq N_1$ и $I_2^t \subseteq N_2$ обозначают множества вершин моделей типа t, для которых еще не были определены соответствия.

Тогда общая стоимость определяется как: $cost(R) = \sum_{r \in R} cost(r) + H(R)$. Действительно, используя эвристическую функцию, можно "предугадать" количество вершин, для которых не будет найдено соответствующих вершин в другом графе, поэтому они гарантированно будут удалены или добавлены. Этот подход, основанный на использовании эвристической функции, также известен как алгоритм поиска A* [7].

Другая эвристика, которая может быть использована при реализации алгоритма, заключается в том, что мы сначала можем рассматривать те вершины, которые соединены дугами с вершинами, для которых уже определено соответствие. Эта эвристика позволяет быстрее находить решение при сопоставлении схожих моделей процессов.

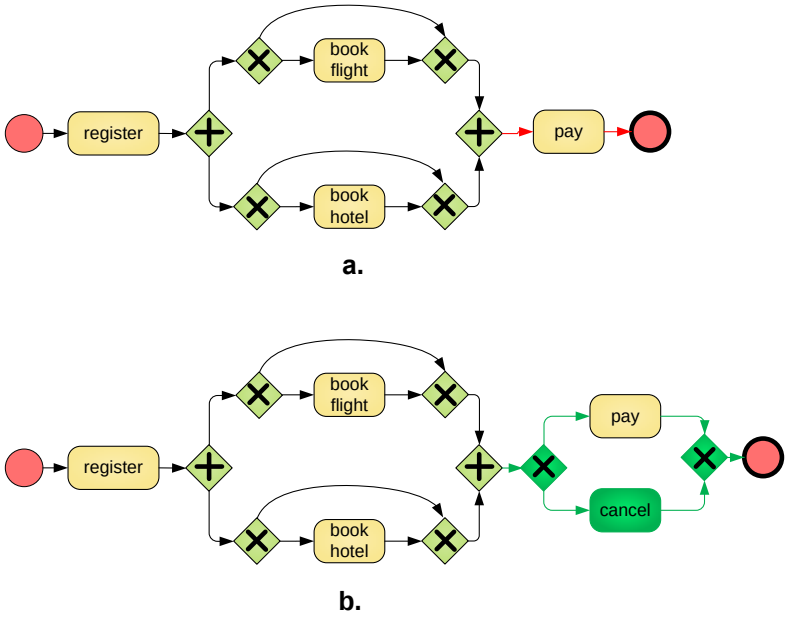


Рис. 3. Результат сопоставления графов бизнес-процессов, моделирующих процедуру бронирования
Fig. 3. A result of the comparison of business process graphs, modeling the booking procedure

Предложенные эвристики были реализованы [6] и протестированы при сопоставлении графов бизнес-процессов, синтезированных по реальным логам событий информационных систем [4]. Несмотря на то что они позволяют сделать процесс сравнения графовых моделей быстрее, сложность задачи, связанная с ее NP-полнотой, остается. Это обусловливает необходимость разработки неточных, но быстрых методов, которые позволят находить минимальное редакционное расстояние между моделями процессов произвольного размера.

3. Применение генетического алгоритма для нахождения минимального редакционного расстояния

В этом разделе будет представлено описание генетического алгоритма [13], позволяющего находить минимальное редакционное расстояние между графами бизнес-процессов.

3.1. Базовый алгоритм и структура гена

Генетический алгоритм основан на идее естественного отбора в природе. Имеется популяция – множество особей, каждая особь однозначно определяется своим геном. Пары особей дают потомство, тем самым порождают новую популяцию. Гены потомков являются продуктом перемешивания генов родителей. Из полученного множества особей выживают только сильнейшие, после чего процесс размножения

продолжается. Таким образом в популяции закрепляются те генетические признаки, которые способствуют выживанию особи.

Предположим, что нам необходимо найти минимальное редакционное расстояние между графами бизнес-процессов $G_1 = (N_1, E_1, t_1, l_1)$ и $G_2 = (N_2, E_2, t_2, l_2)$. Каждое решение (ген) будет определяться некоторым редакционным отношением R между этими графами бизнес-процессов. При этом мы будем рассматривать только те редакционные отношения, в которых для каждого элемента есть соответствие, то есть эти редакционные отношения не содержат пар с δ. Структура и стоимость редакционного отношения однозначно определяются соответствиями элементов первой модели, поэтому в качестве гена возьмем проекцию некоторого редакционного отношения R на первый граф бизнес-процесса. Она будет определена как: $\bar{R} = \{(i_1, i_2) | (i_1, i_2) \in R, i_1 \in N_1\}$. Далее мы будем называть эту проекцию также редакционным отношением.

Основная структура генетического алгоритма (Алгоритм 2) приведена ниже. Сначала генерируется начальная популяция особей, представленных редакционными отношениями между графами бизнес-процессов, затем выбираются особи из текущей популяции, они скрещиваются, потомство добавляется в новую популяцию. Далее происходит отбор, и в популяции остаются только наиболее сильные особи (отношения с наименьшей стоимостью). В ходе отбора используется специальный коэффициент $selectionFactor$, который определяет, какую часть от всей популяции особей необходимо оставить. Алгоритм продолжает работу до тех пор, пока в популяции не останется одна особь.

Data: $G_1 = (N_1, E_1, t_1, l_1)$ и $G_2 = (N_2, E_2, t_2, l_2)$ – графы бизнес-процессов;
Result: редакционное отношение между G_1 и G_2;
population $\leftarrow$ *generatrePopulation*(G_1, G_2);
while *population.size*()! = 1 **do**
  while !*population.empty*() **do**
    parent1 $\leftarrow$ population.get(); parent2 $\leftarrow$ population.get();
    children $\leftarrow$ crossingover(parent1, parent2);
    newpopulation.add(children);
  end
  newpopulation.sort(); *сортировка по стоимости;*
  newpopulation.resize(newpopulation.size()$*$ selectionFactor); *отбор;*
  population $\leftarrow$ newpopulation;
end
return population.get();

Алгоритм 2: Нахождение редакционного отношения между графами бизнес-процессов с помощью генетического алгоритма

3.2. Генерация начальной популяции

Приведем алгоритм генерации начальной популяции редакционных отношений между графами бизнес-процессов (Алгоритм 3).

Data: $G_1 = (N_1, E_1, t_1, l_1)$ и $G_2 = (N_2, E_2, t_2, l_2)$ – графы бизнес-процессов;
Result: популяция редакционных отношений между G_1 и G_2;
$\bar{R}_0 \leftarrow \{\}$;
for $i_i \in N_1$ **do**
  $\bar{R}_0 \leftarrow \bar{R}_0 \cup \{(i_1, \epsilon)\}$;
end
population.add($\bar{R}_0$);
for $i_1 \in N_1$ **do**
  for $i_2 \in N_2$ **do**
    if $t_1(i_1) = t_2(i_2)$ **then**
      $\bar{R}_{i_1,i_2} \leftarrow \bar{R}_0 \backslash \{(i_1, \epsilon)\} \cup \{(i_1, i_2)\}$;
      population.add($\bar{R}_{i_1,i_2}$);
    else
    end
  end
end
return population;

Алгоритм 3: Генерация начальной популяции

Сначала создается редакционное отношение $\bar{R}_0$, которое подразумевает удаление всех элементов первой модели и как следствие добавление всех элементов второй модели. Это редакционное отношение добавляется в начальную популяцию. Далее, для каждой вершины первой модели i_1 и для каждой вершины второй модели i_2, если их типы совпадают, создается новая особь $\bar{R}_{i_1,i_2}$ такая, что она отличается от $\bar{R}_0$ лишь тем, что в ней между элементами i_1 и i_2 устанавливается соответствие. Это редакционное отношение также добавляется в начальную популяцию. Для всех особей начальной популяции вычисляется их стоимость и выполняется сортировка начальной популяции.

3.3. Генерация потомков

Кроссинговер – процесс обмена генетической информацией предков, в результате которого получаются гены потомков (Рис. 4).

Из пары генов предков создаются два оппозитных гена потомков. Происходит одновременный перебор редакционных отношений по первому и второму предку. Фактически перебираются вершины первого графа бизнес-процессов $i_1 \in N_1$. Каждая пара $(i_1, i_2) \in \bar{R}_1$ первого предка и каждая соответствующая пара $(i_1, i'_2) \in \bar{R}_2$ второго предка случайным образом распределяются между двумя потомками. При этом, если в результате кроссинговера некоторому потомку принадлежат две различные пары (i_1, i_2) и (i'_1, i_2), где $i_2 \neq \epsilon$ и $i'_2 \neq \epsilon$ (одной вершине второго графа бизнес-процесса соответствуют две различные вершины первого графа), то также случайным образом выбирается одна из них и преобразуется в (i_1, ϵ) (или (i'_1, ϵ) соответственно). После этого вычисляются стоимости новых полученных редакционных отношений.

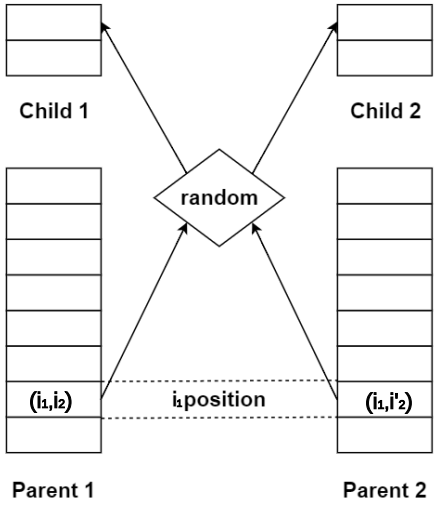


Рис. 4. Кроссинговер
Fig. 4. Crossover

3.4. Параметры генетического алгоритма

Генетический алгоритм имеет набор внутренних параметров, которые отвечают за качество его работы. Далее в ходе экспериментов нами будут рассмотрены следующие параметры: (1) коэффициент отбора популяции ($selectionFactor$), определяющий степень сокращения популяции после каждой итерации размножения; (2) метод выбора из отсортированного массива популяции пар предков для скрещивания. Рассмотрим три основных метода формирования пар из множества предков:

1. Алгоритм первый-второй (FS). Пары образуются последовательно в порядке возрастания стоимости (Рис. 5).

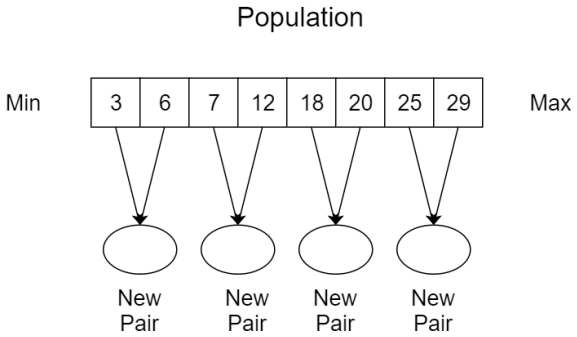


Рис. 5. Алгоритм первый-второй образования пар предков
Fig. 5. An algorithm of first-second parent matching strategy

2. Алгоритм первый-середина (FM). В пару предку, находящемуся в i-й позиции ($i \in [1, \lfloor size/2 \rfloor]$) отсортированного по стоимости массива, ставится предок с номером позиции $i + \lfloor size/2 \rfloor]$, где size – размер массива (Рис. 6).

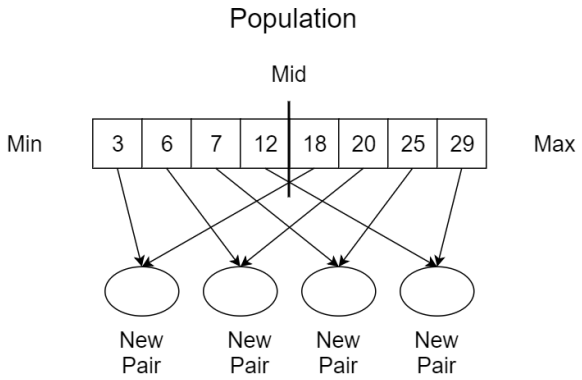


Рис. 6. Алгоритм первый-середина образования пар предков
Fig. 6. An algorithm of first-middle parent matching strategy

3. Алгоритм первый-последний (FL). Предку, находящемуся в i-й позиции ($i \in [1, size - 1]$) отсортированного массива, ставится в пару предок с номером позиции size - i, size – размер массива (Рис. 7).

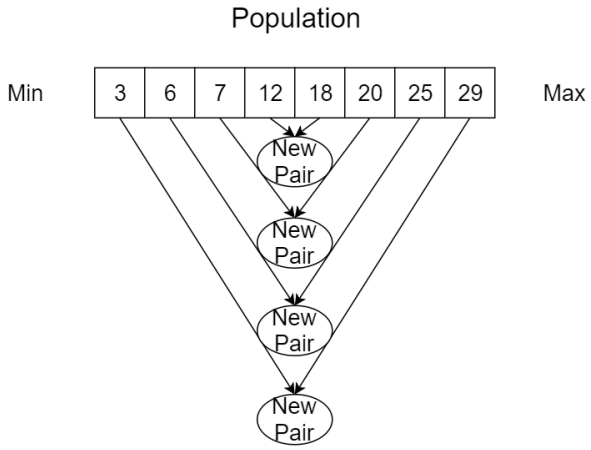


Рис. 7. Алгоритм первый-последний образования пар предков
Fig. 7. An algorithm of first-last parent matching strategy

4. Результаты экспериментов

В этом разделе представлены результаты экспериментов по нахождению минимального редакционного расстояния между BPMN моделями, синтезированными разными алгоритмами [2,3] из одних и тех же логов событий[1]. Были установлены единичные стоимости удаления и добавления элементов, а также символов их названий, т.е. $c_{delete} = 1$, $c_{insert} = 1$ и $c_{lev} = 1$. Только стоимости добавления и удаления действий были определены как 10 для того, чтобы переименования могли конкурировать с удалением и добавлением действий.

[1] http://www.processmining.org.

Все эксперименты были проведены на компьютере Acer Aspire V3, обладающем следующими характеристиками: процессор Intel i5-5200U 2.20GZ, объем оперативной памяти – 8ГБ.

На Рис. 8 показаны зависимости точности решения от размера модели для каждого коэффициента отбора и метода образования пар. Алгоритм A* дает точное минимальное редакционное расстояние. Следует отметить, что нет необходимости сравнивать между собой методы образования пар для одного коэффициента по времени, так как все они имеют одинаковое время работы.

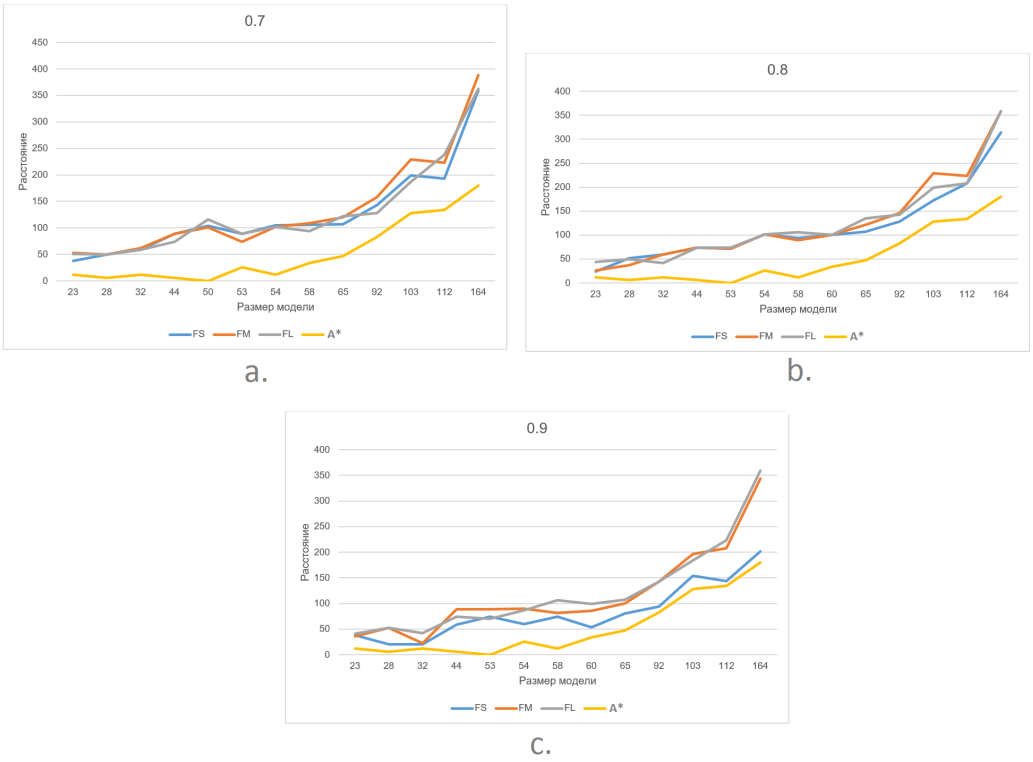


Рис. 8. Зависимость точности результата от характеристик алгоритма
Fig. 8. A dependence of a result accuracy from the characteristics of the algorithm

Как видно из Рис. 8, на небольших размерах моделей разные методы образования пар дают примерно одинаковый результат, а для больших размеров наиболее близкий к точному ответу результат получен для метода FS независимо от коэффициента отбора. Далее будем анализировать только алгоритм FS.

На Рис. 9 видна зависимость точности решения от значения коэффициента отбора: чем больше коэффициент, тем точнее ответ. Действительно, при «строгом» отборе вероятность того, что «полезные качества» успеют зафиксироваться в решении, намного меньше, чем при большем коэффициенте отбора.

Наконец, проанализируем время работы алгоритма для различных коэффициентов (Рис. 10).

Как видно из Рис. 10, коэффициент отбора влияет на время работы алгоритма. При большем значении коэффициента популяция сокращается медленнее, что

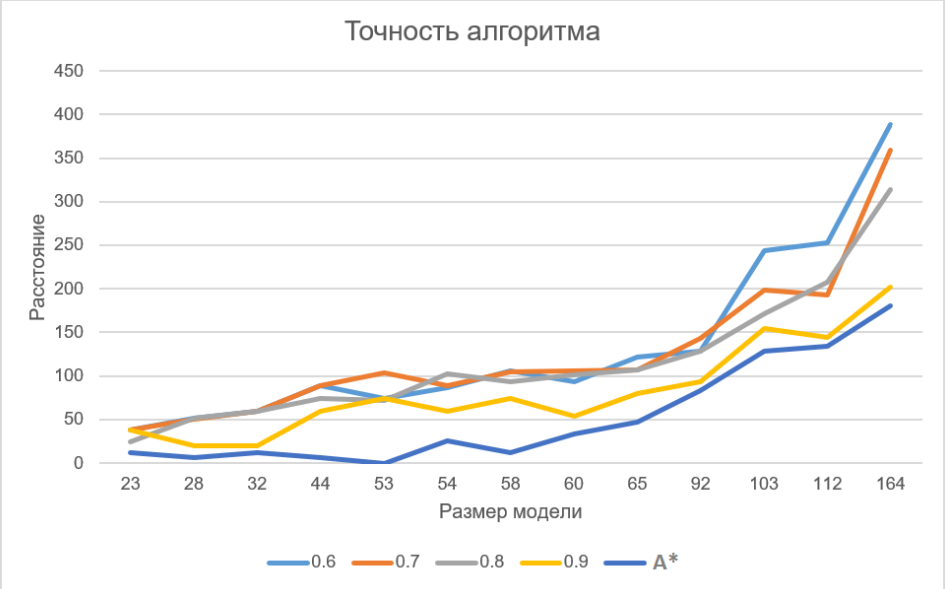


Рис. 9. Зависимость точности результата от коэффициента отбора
Fig. 9. A dependence of a result accuracy from the selection factor

Рис. 10. Зависимость времени работы алгоритма от коэффициента отбора
Fig. 10. A dependence of a computation time from the selection factor

сказывается на времени работы. Следует заметить, что алгоритм A* работает на порядок медленнее.

Итак, наиболее эффективным является метод образования пар первый-второй. Обнаружилась прямая зависимость времени работы и точности алгоритма от коэффициента отбора, его значение устанавливается в зависимости от необходимых пользователю критериев.

Заключение

В настоящее время большую популярность приобретают алгоритмы анализа моделей процессов по логам событий информационных систем (process mining). Существует множество коммерческих программ построения и анализа моделей процессов по логам событий информационных систем, которые широко используются в индустрии, в том числе в крупных компаниях. В связи с развитием теории process mining задача сопоставления модель-модель становится крайне актуальной и важной, так как не только помогает найти различия в ожидаемом и реальном поведении, но и наглядно визуализировать результат (это качество алгоритмов анализа процессов особенно ценится пользователями). В силу того что в общем случае задача сопоставления графовых моделей процессов является NP-полной и точное сравнение моделей занимает значительное время, эвристические методы сопоставления моделей процессов становятся особенно ценными. В этой работе мы адаптировали генетический алгоритм сравнения графов для задачи сопоставления моделей процессов, извлекаемых из логов событий. Этот алгоритм был реализован и протестирован на BPMN моделях, синтезированных по логам событий. Была проведена как оценка точности алгоритма в зависимости от параметров, так и его временных характеристик.

Список литературы / References

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

[2] Van der Aalst W. M. P., Weijters T., Maruster L., “Workflow Mining: Discovering Process Models from Event Logs”, *IEEE Transactions on Knowledge and Data Engineering*, **16**:9 (2004), 1128–1142.

[3] Leemans S. J. J., Fahland D., van der Aalst W. M. P., “Discovering Block-Structured Process Models from Incomplete Event Logs”, *Application and Theory of Petri Nets and Concurrency*, Lecture Notes in Computer Science, **8489**, Springer, 2014, 91–110.

[4] Kalenkova A. A., Ageev A. A., Lomazova I. A., van der Aalst W. M. P., “E-Government Services: Comparing Real and Expected User Behavior”, *Business Process Management Workshops*, Springer International Publishing, 2018, 484–496.

[5] Garey M. R., Johnson D. S., *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., 1990, ISBN: 0716710455.

[6] Ivanov S. Y., Kalenkova A. A., van der Aalst W. M. P., “BPMNDiffViz: A Tool for BPMN Models Comparison”, Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015) (Innsbruck, Austria, September 2, 2015), 2015, 35–39.

[7] Hart P. E, Nilsson N. J, Raphael B., “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”, *IEEE Transactions on Systems Science and Cybernetics*, **4**:2 (1968), 100–107.

[8] *Business Process Model and Notation (BPMN)*, Object Management Group, formal/2013-12-09, 2013.

[9] Cross A. D. J., Wilson R. C., Hancock E. R., “Inexact Graph Matching Using Genetic Search”, *Pattern Recognition*, **30**:6 (1997), 953 – 970.

[10] Riesen K., Fischer A., Bunke H., “Improving Approximate Graph Edit Distance Using Genetic Algorithms”, *Structural, Syntactic, and Statistical Pattern Recognition*, Springer Berlin Heidelberg, 2014, 63–72.

[11] Kalenkova A. A., van der Aalst W. M. P., Lomazova I. A., Rubin V. A., "Process Mining Using BPMN: Relating Event Logs and Process Models", *Software & Systems Modeling*, **16**:4 (2017), 1019–1048.

[12] Levenshtein V. I., "Binary Codes Capable of Correcting Deletions, Insertions and Reversals", *Soviet Physics Doklady*, **10** (1966), 707.

[13] Гладков Л.А., Курейчик В.В, Курейчик В.М., *Генетические алгоритмы*, Физматлит, М., 2010, 368 с.; [Gladkov L. A., Kureichik V. V., Kureichik V. M., *Genetic Algorithms*, Fizmatlit, Moscow, 2010, 368 pp., (in Russian).]

Kalenkova A. A., Kolesnikov D. A., "Application of Genetic Algorithms for Finding Edit Distance between Process Models", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 711–725.

DOI: 10.18255/1818-1015-2018-6-711-725

Abstract. Finding graph-edit distance (graph similarity) is an important task in many computer science areas, such as image analysis, machine learning, chemicalinformatics. Recently, with the development of process mining techniques, it became important to adapt and apply existing graph analysis methods to examine process models (annotated graphs) discovered from event data. In particular, finding graph-edit distance techniques can be used to reveal patterns (subprocesses), compare discovered process models. As it was shown experimentally and theoretically justified, exact methods for finding graph-edit distances between discovered process models (and graphs in general) are computationally expensive and can be applied to small models only. In this paper, we present and assess accuracy and performance characteristics of an inexact genetic algorithm applied to find distances between process models discovered from event logs. In particular, we find distances between BPMN (Business Process Model and Notation) models discovered from event logs by using different process discovery algorithms. We show that the genetic algorithm allows us to dramatically reduce the time of comparison and produces results which are close to the optimal solutions (minimal graph edit distances calculated by the exact search algorithm).

Keywords: minimal graph edit distance, process mining, BPMN (Business Process Model and Notation), genetic algorithm

On the authors:
Anna A. Kalenkova, orcid.org/0000-0002-5088-7602, senior research fellow,
National Research University Higher School of Economics, Laboratory of Process-Aware Information Systems,
20 Myasnitskaya St., Moscow 101000, Russia, e-mail: akalenkova@hse.ru

Danil A. Kolesnikov, orcid.org/0000-0002-9010-8415, student
National Research University Higher School of Economics, Faculty of Computer Science
20 Myasnitskaya St., Moscow 101000, Russia, e-mail: dakolesnikov@edu.hse.ru

Acknowledgments:
This work was funded by the President Grant MK-4188.2018.9.

Моделирование и анализ информационных систем. Т. 25, №6 (2018), с. 726–733
Modeling and Analysis of Information Systems. Vol. 25, No 6 (2018), pp. 726–733

Тезаурусы
Thesauri

DOI: 10.18255/1818-1015-2018-6-726-733

УДК 004.912

Векторное представление слов с семантическими отношениями: экспериментальные наблюдения

Каряева М. С.[1], **Браславский П. И.**, **Соколов В. А.**[1]

Поступила в редакцию 1 сентября 2018
После доработки 20 ноября 2018
Принята к публикации 25 ноября 2018

Аннотация. Возможность идентификации семантической близости между словами сделала модель word2vec широко используемой в NLP-задачах. Идея word2vec основана на контекстной близости слов. Каждое слово может быть представлено в виде вектора, близкие координаты векторов могут быть интерпретированы как близкие по смыслу слова. Таким образом, извлечение семантических отношений (отношение синонимии, родо-видовые отношения и другие) может быть автоматизировано. Установление семантических отношений вручную считается трудоемкой и необъективной задачей, требующей большого количества времени и привлечения экспертов. Но среди ассоциативных слов, сформированных с использованием модели word2vec, встречаются слова, не представляющие никаких отношений с главным словом, для которого был представлен ассоциативный ряд. В работе рассматриваются дополнительные критерии, которые могут быть применимы для решения данной проблемы. Наблюдения и проведенные эксперименты с общеизвестными характеристиками, такими как частота слов, позиция в ассоциативном ряду, могут быть использованы для улучшения результатов при работе с векторным представлением слов в части определения семантических отношений для русского языка. В экспериментах используется обученная на корпусах Флибусты модель word2vec и размеченные данные Викисловаря в качестве образцовых примеров, в которых отражены семантические отношения. Семантически связанные слова (или термины) нашли свое применение в тезаурусах, онтологиях, интеллектуальных системах для обработки естественного языка.

Ключевые слова: векторное представление слов, word2vec, семантические отношения, тезаурус, гипонимы, гиперонимы, синонимы

Для цитирования: Каряева М. С., Браславский П. И. Соколов В. А., "Векторное представление слов с семантическими отношениями: экспериментальные наблюдения", *Моделирование и анализ информационных систем*, **25**:6 (2018), 726–733.

Об авторах: Каряева Мария Сергеевна, orcid.org/0000-0003-4466-1735, аспирант
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: mari.karyaeva@gmail.com

Браславский Павел Исаакович, orcid.org/0000-0002-6964-458X, канд. техн. наук, доцент,
Уральский федеральный университет,
г. Екатеринбург, ул. Мира, 19, 620002 Россия, e-mail: pbras@yandex.ru

Соколов Валерий Анатольевич, orcid.org/0000-0003-1427-4937, доктор физ.-мат. наук, профессор,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150003 Россия, e-mail: sokolov@uniyar.ac.ru

Благодарности:
[1] Исследование выполнено при финансовой поддержке РФФИ в рамках научных проектов №16-07-01180 и №16-06-00497

Введение

Изменение метода распознания семантических ролей на основе нейронной сети [1], разработанного в 2013 году Томасом Миколовым, привело к разработке подхода векторного представления слов word2vec[1], который является на сегодняшний день одним из самых распространенных методов семантического моделирования при работе с текстовой информацией. Векторное представление слов применяется в широком спектре задач [2], [3], [4], [5] обработки естественного языка. На вход алгоритму word2vec подается корпус текстов, после обучения модели с определенными параметрами на выходе формируются векторные представления слов, которые отражают их семантику. Для оценки полученных пар часто используют метрику – косинусная мера близости, которая является оптимальной для отображения семантического сходства слов.

Представление слов в виде векторов позволяет применять математические операции. В большинстве примеров можно встретить вычитание векторов, когда результат вычисления vec('Madrid') - vec('Spain') + vec('France') будет ближе к vec('Paris'), чем к другим векторам из распределения. Таким образом, разница векторов может быть использована для поиска семантических отношений между словами.

Word2vec не возвращает напрямую семантические отношения между словами. В ассоциативном ряду, который может быть возвращен в качестве близких слов к запрашиваемому (главному) слову, отражаются слова, которые часто употребляются рядом в контексте. Бесспорно, в ассоциативном ряду встречаются синонимы, антонимы, гипонимы, гиперонимы, холонимы, меронимы, ассоциации и другие типы, которые могут быть определены как семантические отношения.

Наиболее популярным и эффективным методом для извлечения семантических отношений были [7] и остаются [8], [9], [10] лексико-синтаксические шаблоны. Наша работа не перекрывает полученные результаты использованием векторных представлений слов, в данном исследовании мы стремимся поделиться наблюдениями и закономерностями, которые могут быть полезны при работе с word2vec.

1. Связанные работы

Обучение модели с использованием word2vec строится по принципу наличия семантических отношений между словами в схожих контекстах. Получение связанных слов с оценкой может послужить базой для поиска связей между терминами при автоматическом создании тезауруса.

Исследование [6], посвященное задачам семантической близости, показало, что морфология русского языка не является препятствием для обучения модели с использованием word2vec на русских корпусах, в частности был рассмотрен Национальный корпус русского языка (НКРЯ).

В основу исследования [12] положена идея поиска пар гипоним-гипероним на основе векторного представления: сначала разности векторов кластеризуются, после этого для каждого кластера обучается отдельная проекция вектора на основе обуча-

[1] https://code.google.com/p/word2vec/

ющей выборки, полученной из тезауруса. В результате пара слов, соответствующих векторам, может быть классифицирована как “род-вид”. Данная работа послужила фундаментом для поиска семантических отношений с применением word2vec. В работе [13] дополнительно к применению концепции [12] было изучено негативное влияние примеров на прогнозирование наличия родо-видовых отношений.

Кроме родо-видовых отношений, с помощью word2vec возможно извлечение синонимии. В работе [14] были проведены эксперименты по автоматическому поиску синонимов в сфере медицины. Данное исследование не только демонстрирует жизнеспособность извлечения синонимов с использованием word2vec, но и подтверждает применимость векторного представления слов в предметных областях.

2. Векторное представление слов

Примеры ассоциативных рядов (выборка) с указанием метрики близости представлены в Таблице 1. Полужирным шрифтом выделены слова-ассоциаты, которые связаны с главным словом отношением род-вид. Среди слов-ассоциатов встречаются слова, характеризующие главное слово: голос хрипловатый, спокойный и т.д. Таким образом, существует вероятность благоприятного результата при использовании векторного представления слов для извлечения пар-кандидатов, имеющих семантические отношения с главным словом, в данном случае родо-видовые.

Таблица 1. Примеры векторных представлений
с оценкой близости (косинусная мера)
Table 1. Some examples of pairs with semantic relations
and their scores of cosine similarity

голос		оружие		лук	
хрипловатый	0.80	огнестрельное	0.71	луком	0.72
голосок	0.76	стрелковое	0.66	**порей**	0.65
звучал	0.72	**ружье**	0.62	колчан	0.65
баритон	0.72	**пистолет**	0.61	**репчатый**	0.62
бас	0.64	метательное	0.61	чеснок	0.59
женский	0.61	**копье**	0.58	шинкованный	0.58
спокойный	0.56	**лучеметы**	0.52	зелень	0.57

Кандидат из ассоциативного ряда с главным словом, для которого был автоматически сгенерирован ассоциативный ряд, не является симметричным относительно позиции кандидата в ассоциативном ряду. Другими словами, если заранее предположить, что между X и Y есть семантические отношения, а затем для слова X сгенерировать ассоциативный ряд и найти в нем позицию слова Y, то данная позиция не будет равна той позиции, которая установлена при проведении поиска кандидата X в ассоциативном ряду, построенном для Y как главного слова.

Власть – диктатура (31-е место в ассоциативном ряду),
Диктатура – власть (6-е место в ассоциативном ряду).

Следовательно, при поиске семантических несимметричных отношений необходимо определить вертикаль главного слова и кандидата. Например, для родовидовых отношений определить – гипонимом или гиперонимом будет являться главное слово.

В данной работе для экспериментов с родо-видовыми отношениями главное слово является видом, а искомый кандидат ассоциативного ряда – родом.

3. Данные

3.1. Флибуста

Обучение модели word2vec на корпусе Флибусты показало следующие результаты: сгенерировано 931 896 векторных представлений в виде главного слова и слов-ассоциатов с оценкой близости, которая представлена косинусной мерой между векторами главного слова и рассматриваемого ассоциата. Среди слов-ассоциатов встречаются как слова окружения, характеризующие главное слово, так и слова, имеющие семантические отношения с главным словом.

3.2. Викисловарь

Викисловарь[2] (общий объем – более 173 тыс. словарных входов) как лексикографический проект с прямым указанием семантических свойств слов (в том числе ссылками на гипонимы и гиперонимы) подходит для тестирования данных и получения автоматической оценки извлеченных пар-кандидатов. Для проведения экспериментов использовался дамп Викисловаря, содержащий 59 582 родо-видовых пар.

3.3. НКРЯ

Частотные списки Национального корпуса русского языка[3] были использованы для определения частоты встречаемости слов.

4. Экспериментальные результаты

4.1. Определение границы

В работе [15] приводится сравнительный анализ метрик близости с целью векторного представления слов для поиска семантических отношений. Наилучший результат показала косинусная мера близости. Изучение результатов распределения косинусной меры близости может быть полезно с точки зрения сужения границ ассоциативного ряда для увеличения вероятности нахождения кандидата родо-видовых отношений. В данном эксперименте в качестве данных были использованы пары из Википедии с родо-видовыми отношениями, для них были сформированы векторные

[2] https://ru.wiktionary.org/
[3] http://www.ruscorpora.ru/

представления и определена косинусная мера близости. Главное слово представлено видом, а слово из ассоциативного списка – родом.

На рис. 1 изображено распределение, позволяющее детектировать границы, в которых могут быть найдены семантические отношения. Анализируя распределение можно с уверенностью сказать, что вероятность встретить кандидата в ассоциативном ряду с высокой или низкой косинусной мерой близости очень низка. Основная концентрация потенциальных кандидатов будет расположена в пределах 0.53–0.63.

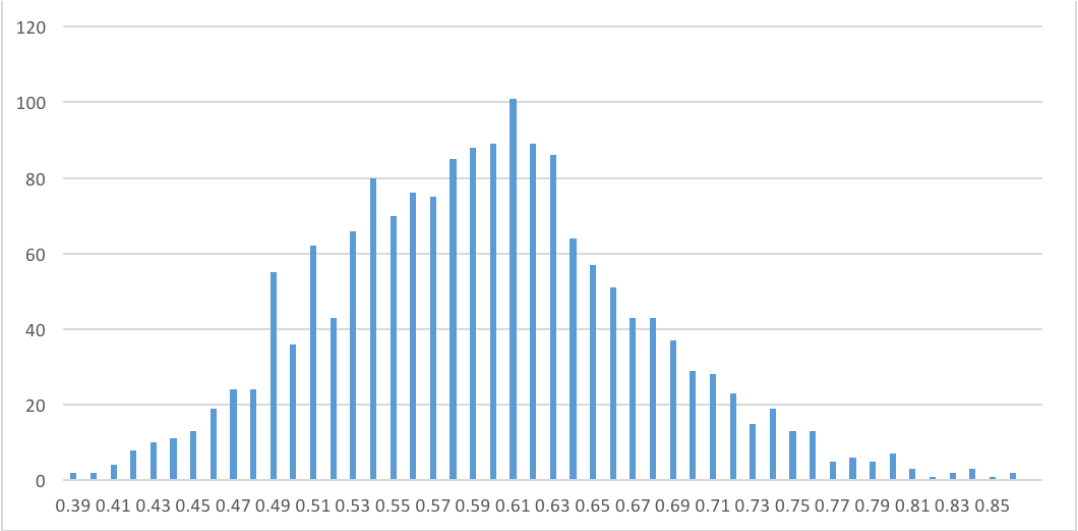


Рис. 1. Распределение значения косинусной меры
по парам вид-род из Викисловаря (1 688 пар)
Fig. 1. Cosine similarity score distribution for hyponym/hypernym pairs
(1 688 pairs) from Wiktionary

4.2. Ассоциативный ряд

Был проведен эксперимент, который показал, что *расстояние от главного слова до слова-кандидата является наименьшим, если главное слово представлено видом, а слово-кандидат – родом.* Данное расстояние измеряется не косинусной мерой, а количеством слов-ассоциатов, упорядоченных по косинусной мере и стоящих выше потенциального кандидата.

Для примера рассмотрим пару *мясо-конина.* Для слова *мясо* составим пронумерованный ряд слов-ассоциатов в порядке убывания косинусной меры. Таким образом, в ряду слов-ассоциатов *конина* окажется на 64-м месте. Аналогично для главного слова *конина* в ряду слов-ассоциатов найдем *мясо*, которое оказалось на 5-й позиции от главного слова.

Мясо – жареное, . . . , говядина, баранина, вареное, копченое, . . . , нарезанное, зажаренное, вяленое, курятина, полусырое, . . . , ягнятина, . . . , конина, . . .

Конина – говядина, свинина, баранина, конину, мясо, козлятина, . . .

Для подтверждения гипотезы был проведен следующий эксперимент. Имена собственные были удалены из пар дампа Викисловаря, таким образом, работа производилась с 39,5 тыс. пар. В результате поиска данных пар в векторном представлении были получены 1 688 пары. Из них:

- 52% пар, где позиция рода в списке слов-ассоциатов меньше позиции вида в списке слов-ассоциатов;
- 36% пар, где позиция рода в списке слов-ассоциатов больше позиции вида в списке слов-ассоциатов;
- 12% пар, где позиция рода в списке слов-ассоциатов равна позиции вида в списке слов-ассоциатов.

Результат второго места (36%) можно объяснить наличием вида с высокой частотностью, как, например, в паре *'мясо-говядина'*. Тем не менее, таких часто встречающихся пар оказывается не так много.

4.3. Частотность слов

Информация о частоте встречаемости слов является одной из ключевых характеристик в задачах компьютерной лингвистики. Опираясь на успешный опыт применения статистики использования слов [16], [17], мы рассмотрели зависимость – частотность рода и вида среди пар Викисловаря, которые имеют векторное представление. Примеры пар род-вид с частотностью:

Игра – 5 815*, прятки – 306, жмурки – 122.*
Мясо – 4 900*, говядина – 231, свинина – 157, конина – 29, курятина – 25.*

88% составляют пары, где частотность рода больше, чем частотность вида, например: *песня (29 220) – баллада (900)*. Остальные 12% составляют пары, где частотность вида больше, чем частотность рода. Такие пары можно отнести к исключениям, поскольку они отображают: слабую родо-видовую связь: *директор (27 442) – начальник (40 352)*; смещение рода к виду или вида к роду: *помещение (14 659) – коридор (17 507)*

На рис. 2 приведены графики распределения частоты от встречаемости отдельно для рода и вида.

5. Заключение

Работа с векторным представлением слов предоставляет огромный спектр исследований как с точки зрения лингвистики, так и статистики. Поиск закономерностей и эвристик, полученных таким образом, позволит улучшать качество извлекаемых сущностей. Представленные в данной работе заключения могут служить дополнительными критериями для устранения избыточных кандидатов при извлечении семантических отношений.

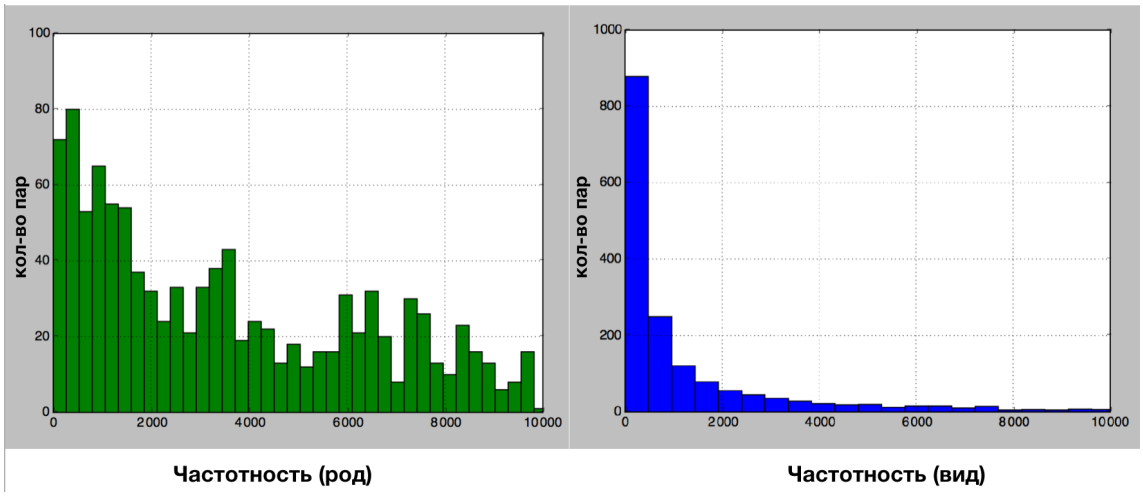


Рис. 2. Распределение частоты от встречаемости (количества пар)
Fig. 2. Word frequency distribution and their occurrence

Список литературы / References

[1] Mikolov T., Yih W., Zweig G., "Linguistic Regularities in Continuous Space Word Representations", *HLT-NAACL*, 2013, 746–751.

[2] Sienčnik S.K., "Adapting word2vec to named entity recognition", *Proceedings of the 20th nordic conference of computational linguistics*, 2015, 239–243.

[3] Lilleberg J., Zhu Y., Zhang Y., "Support vector machines and word2vec for text classification with semantic features", *Cognitive Informatics & Cognitive Computing*, IEEE 14th International Conference, 2015, 136–140.

[4] Ling W. et al., "Two/too simple adaptations of word2vec for syntax problems", *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, 1299–1304.

[5] Najafabadi M.M. et al., "Deep learning applications and challenges in big data analytics", *Journal of Big Data*, **2** (2015), 1.

[6] Kutuzov A., Andreev I., "Texts in, meaning out: neural language models in semantic similarity task for Russian", 2015, https://arxiv.org/abs/1504.08183.

[7] Hearst M. A., "Automatic acquisition of hyponyms from large text corpora", *Proceedings of the 14th conference on Computational linguistics – Association for Computational Linguistics*, **2** (1992), 539–545.

[8] Klaussner C., Zhekova D., "Lexico-syntactic patterns for automatic ontology building", *Proceedings of the Second Student Research Workshop associated with RANLP*, 2011, 109–114.

[9] Maedche A., Pekar V., Staab S., "Ontology learning part one—on discovering taxonomic relations from the web", *Web Intelligence*, 2003, 301–319.

[10] Snow R., Jurafsky D., Ng A. Y., "Learning syntactic patterns for automatic hypernym discovery", *Advances in Neural Information Processing Systems*, 2005, 1297–1304.

[11] Panchenko A., et al., "Human and Machine Judgements for Russian Semantic Relatedness", *Analysis of Images, Social Networks and Texts: 5th International Conference, AIST 2016*, (Yekaterinburg, Russia, April 7–9, 2016, Revised Selected Papers), 2017, 221–235.

[12] Fu R., et al., "Learning semantic hierarchies via word embeddings", *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, **1** (2014), 1199–1209.

[13] Ustalov D., Arefyev N., Biemann C., Panchenko A., "Negative sampling improves hypernymy extraction based on projection learning", 2017, https://arxiv.org/pdf/1707.03903.pdf.

[14] Wang C., Cao L., Zhou B., “Medical Synonym Extraction with Concept Space Models”, 2015, https://arxiv.org/pdf/1506.00528.pdf.

[15] Rei M., Briscoe T., “Looking for hyponyms in vector space”, *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, 2014, 68–77.

[16] Turney P., Pantel P., “From frequency to meaning: Vector space models of semantics”, *Journal of artificial intelligence research*, **37** (2010), 141–188.

[17] Matsuo Y., Ishizuka M., “Keyword extraction from a single document using word co-occurrence statistical information”, *International Journal on Artificial Intelligence Tools*, **13**:1 (2004), 157–169.

Karyaeva M. S., Braslavski P. I., Sokolov V. A., "Word Embedding for Semantically Relative Words: an Experimental Study", *Modeling and Analysis of Information Systems*, **25**:6 (2018), 726–733.

DOI: 10.18255/1818-1015-2018-6-726-733

Abstract. The ability to identify semantic relations between words has made a word2vec model widely used in NLP tasks. The idea of word2vec is based on a simple rule that a higher similarity can be reached if two words have a similar context. Each word can be represented as a vector, so the closest coordinates of vectors can be interpreted as similar words. It allows to establish semantic relations (synonymy, relations of hypernymy and hyponymy and other semantic relations) by applying an automatic extraction. The extraction of semantic relations by hand is considered as a time-consuming and biased task, requiring a large amount of time and some help of experts. Unfortunately, the word2vec model provides an associative list of words which does not consist of relative words only. In this paper, we show some additional criteria that may be applicable to solve this problem. Observations and experiments with well-known characteristics, such as word frequency, a position in an associative list, might be useful for improving results for the task of extraction of semantic relations for the Russian language by using word embedding. In the experiments, the word2vec model trained on the Flibusta and pairs from Wiktionary are used as examples with semantic relationships. Semantically related words are applicable to thesauri, ontologies and intelligent systems for natural language processing.

Keywords: word embedding, word2vec, semantic relations, thesaurus, hyponymy, hypernymy, synonymy

On the authors:
Maria Karyaeva, orcid.org/0000-0003-4466-1735, graduate student,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail: mari.karyaeva@gmail.com

Pavel Braslavski, orcid.org/0000-0002-6964-458X, PhD, Docent,
Ural Federal University,
19 Mira str., Ekaterinburg 620002, Russia, e-mail: pbras@yandex.ru

Valery A. Sokolov, orcid.org/0000-0003-1427-4937, Doctor, Professor,
P.G. Demidov Yaroslavl State University,
14 Sovetskaya str., Yaroslavl 150003, Russia, e-mail:sokolov@uniyar.ac.ru

Acknowledgments:
[1] The reported study was funded by RFBR according to the research projects №16-07-01180 и №16-06-00497