СОДЕРЖАНИЕ

Номер 4, 2022

CONTENTS

No. 4, 2022

EDN: QYPZYK
ПРОГРАММИРОВАНИЕ, 2022, № 4, с. 3–16

ПРОГРАММНАЯ ИНЖЕНЕРИЯ, ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ ПРОГРАММ

УДК 004.421.6

ПРОГНОЗИРОВАНИЕ СТЕПЕНИ ПОРАЖЕНИЯ ЛЕГКИХ ПРИ COVID-19 НА ОСНОВЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ

© 2022 г. Ю. А. Васильев[a,*] (ORCID: 0000-0001-9210-5544),
М. И. Петровский[a,] (ORCID: 0000-0002-1236-398X),**
И. В. Машечкин[a,*] (ORCID: 0000-0002-9837-585X),**
Л. Л. Панкратьева[b,**] (ORCID: 0000-0002-1339-4155)**

[a] МГУ им М.В. Ломоносова, факультет вычислительной математики и кибернетики, кафедра интеллектуальных информационных технологий, 119991, Москва, ГСП-1, ул. Колмогорова, д. 1, стр. 52, Россия

[b] Федеральный научно-клинический центр детской гематологии, онкологии и иммунологии им. Дмитрия Рогачева Минздрава России, 117997, Москва, ул. Саморы Машела, д. 1, Россия

**E-mail: iuliivasilev@gmail.com*

***E-mail: michael@cs.msu.su*

****E-mail: mash@cs.msu.su*

*****E-mail: liudmila.pankratyeva@gmail.com*

Поступила в редакцию 27.12.2021 г.
После доработки 16.01.2022 г.
Принята к публикации 29.01.2022 г.

Работа посвящена решению задачи прогнозирования течения заболевания у пациентов с COVID-19. На основе данных об анамнезе, осмотре, результатах клинико-лабораторного анализа и других факторов, потенциально связанных с тяжестью течения заболевания и вероятностью смерти пациентов с COVID-19, был разработан комплекс моделей, построенных с использованием методов машинного обучения и прикладного статистического анализа, для прогнозирования тяжести течения и исхода заболевания у пациентов, получающих лечение в амбулаторных и стационарных условиях.

Одним из ключевых результатов проведенной работы является создание сервиса “КТ-калькулятор”, встроенного в городскую медицинскую информационную систему – способа оценки степени изменения легочной ткани при COVID-19 в экспресс-режиме без использования компьютерной томографии, и позволяющего на основе физикальных и лабораторных признаков спрогнозировать степень поражения легких пациентов.

Построенные в рамках данного проекта модели машинного обучения дают возможность судить о степени риска легкой и тяжелой формы течения заболевания в зависимости от различных категорий факторов.

DOI: 10.31857/S0132347422040069

1. ВВЕДЕНИЕ

Всемирная организация здравоохранения 11 марта 2020 г. объявила пандемию по заболеванию COVID-19, вызываемому вирусом SARS-CoV-2. С начала пандемии Systems Science and Engineering (CSSE) at Johns Hopkins University[1] в Российской Федерации зарегистрировано более 10 млн случаев заражения COVID-19 и более 292 тыс. случаев летального исхода по состоянию на 20 декабря 2021 года.

Ежедневный рост количества зараженных приводит к увеличению нагрузки на врачей и медицинское оборудование, ухудшению качественной оценки состояния пациентов и увеличению затрат на здравоохранение в целом.

Пандемия резко ускорила внедрение цифровых сервисов в работу московского здравоохранения. Начиная с марта 2019 года московскими поликлиниками и стационарами был накоплен огромный объем данных по истории болезни более чем 2 млн человек.

Пациенту с подтвержденным COVID-19, проводится комплекс клинического обследования.

Во-первых, происходит сбор анамнеза, включающего индивидуальные характеристики пациента (хронические заболевания, пол, возраст и т.д.). Далее, врач проводит физикальное обследование (ЧДД, сатурация, температура, степень тя-

[1] https://origin-coronavirus.jhu.edu/map.html

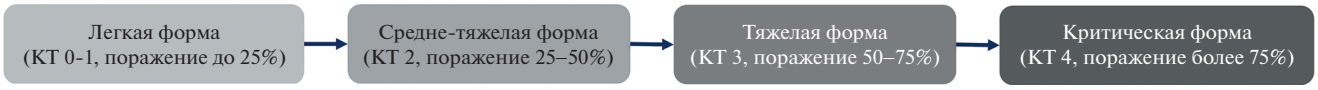


Рис. 1. Порядковое сопоставление степени поражения легких по КТ и форму COVID-19.

жести). Наконец, производится сбор анализов по направлениям на лабораторные исследования: клинический анализ крови и мочи, биохимический анализ крови.

Массовое применение оценки изменений легких получила компьютерная томография (далее – КТ). Результаты КТ могут служить предикторами необходимости госпитализации в стационар и вероятности неблагоприятного исхода в отделении интенсивной терапии. Порядковое сопоставление степени поражения легких по КТ и формы течения COVID-19 пациента представлено на рис. 1.

Однако, подход оценки изменений легких путем КТ имеет и недостатки, например, риск создания искусственных эпидемических очагов, нерациональная работа бригад скорой помощи, экономические затраты на проведение КТ. Также, возникают проблемы, связанные с радиационной безопасностью пациентов и врачей.

Актуальным является построение прогнозных моделей оценки поражения легких пациентов, основанных на собранных данных, а также выявление наиболее важных факторов, влияющих на развитие легочной пневмонии. В качестве альтернативного диагностического инструмента для оценки изменений легких в рамках данной работы предлагается исследовать и разработать методы интеллектуального анализа данных для решения задачи оценки степени изменения легочной ткани при COVID-19 в экспресс-режиме на основе физикальных и клинических признаков пациента.

Важной особенностью данной работы является использование реальных данных из нескольких источников: КТ-центров, лабораторий, поликлиник и стационаров. При сборе данных из нескольких источников возникает множество проблем: ошибки ввода и противоречия признаков, неоднородное заполнение признаков (в частности, клинические и биохимические анализы являются достаточно редкими для рядового пациента), разный объем и полнота данных. Без детального исследования и построения алгоритмов решения данных проблем не может быть достигнута большая точность прогнозных моделей.

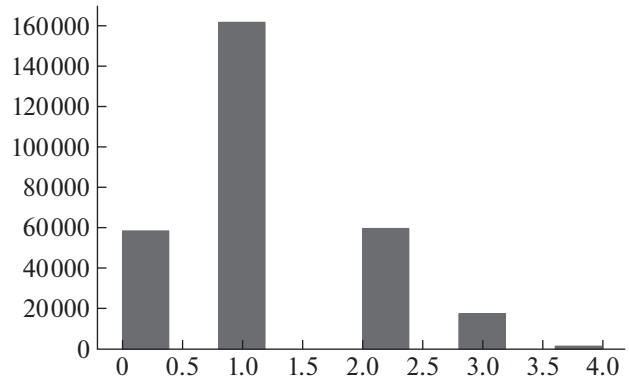


Рис. 2. Распределение результатов КТ для набора муниципальных данных.

Разработанные прогнозные модели могут быть использованы в качестве системы поддержки принятия врачебных решений, которая позволит сократить количество исследований для пациента и уменьшить облучение. Также, система снизит нагрузку на реальное оборудование – на КТ-центры и может быть применена в тех регионах, в которых доступ к компьютерному томографу ограничен, либо этого оборудования нет.

Данная работа имеет следующую структуру: в разделе 2 описываются существующие работы по применению методов машинного обучения для анализа данных COVID-19. В разделе 3 описываются особенности решаемой задачи, а также рассматривается возможность применения моделей машинного обучения. В разделе 4 описываются используемые наборы данных, метрики качества и полный алгоритм сбора и предобработки данных, включая удаление противоречий и артефактов и унификацию значений признаков. Также, в разделе описывается экспериментальное исследование оценки качества предложенных методов. В разделе 5 описывается практическая реализация сервиса “Калькулятор-КТ”, методы взаимодействия с сервисом и способы интеграции во внешнюю среду. В разделе 6 представлены основные результаты работы.

2. ОБЗОР ЛИТЕРАТУРЫ

В данном разделе рассматриваются несколько типов исследовательских работ, посвященных применению методов машинного обучения для анализа данных COVID-19, в частности, по анализу распространяемости COVID-19 [1], анализу снимков КТ [2], анализу клинических данных [3–5].

2.1. Анализ распространяемости COVID-19

На данный момент, существует несколько типов исследовательских работ, посвященных прогнозированию COVID-19.

Во-первых, существуют работы по анализу распространяемости COVID-19, прогнозированию ежедневной заболеваемости и смертности. В статье Garca-Cremades, Santi and Morales-Garca [1] решается задача анализа и прогнозирования временных рядов кумулятивной заболеваемости по COVID-19 на 14-дневный период. В работе рассматриваются модели нейронных сетей LSTM и GRU, а также статистические модели AR и ARIMA.

В качестве набора данных используются данные о мобильности в Испании, собранные с помощью инструмента Google (GMD)[2]. Он показывает набор агрегированных и анонимизированных данных, полученных от продуктов Google (в частности, Google Maps). Основными признаками набора данных являются тенденции мобильности граждан по различным категориям мест: парки, супермаркеты, общественный транспорт, рабочие места, жилье.

Лучшие результаты прогнозирования в краткосрочном периоде показал ансамблевый подход рассматриваемых методов ($0.93R^2$, $4.16RMSE$, $1.08MAE$).

2.2. Анализ снимков КТ

Во-вторых, существуют работы по анализу снимков КТ и прогнозированию степени тяжести течения COVID-19. В статье Elaziz, Mohamed Abd and Hosny [2] решается задача бинарной классификации рентгеновских снимков грудной клетки по наличию COVID-19 у пациента. Предлагаемый способ извлекает особенности из рентгеновских снимков грудной клетки с использованием ортогональных многочленов с дробными порядками FrMEMs.

Далее, используется алгоритм отбора признаков MRFODE, заключающийся в генерации набора решений и вычисления значения пригодности для каждого из признаков с использованием классификатора KNN (K ближайших соседей) на основе обучающего набора с определением наилучшего из признаков. Процесс выбора лучшего признака производится до достижения предельных условий. На основе дифференциальной эволюции (DE) генерируются бинарные вектора по выбранным признакам и обучается классификатор KNN.

В работе рассматриваются два разных набора данных. Первый набор данных, собранный по педиатрическим пациентам в возрасте от одного до пяти лет из медицинского центра Гуанчжоу, содержит изображения обычной и вирусной пневмонии. Набор содержит 216 положительных и 1675 отрицательных изображений COVID-19. Второй набор данных представляет выгрузку из базы данных Итальянского общества медицинской и интервенционной радиологии (SIRM) по пациентам с COVID-19. Набор данных состоит из 219 положительных и 1341 отрицательного изображения COVID-19.

Экспериментальное исследование качества методов проводилось по метрике Accuracy и включало сравнение предложенного метода с обученной глубокой нейронной сетью MobileNet. Предложенный алгоритм показал лучшие результаты и достиг показателей точности 0.9609 и 0.9809 для первого и второго наборов данных соответственно.

2.3. Анализ клинических данных

Наконец, существуют работы по анализу клинических данных госпитализированных пациентов и прогнозированию смертности от COVID-19. В статье Levy, Todd J. and Richardson [3] решается задача прогнозирования 7-дневной выживаемости у пациентов, госпитализированных с COVID-19. Предлагаемый метод основан на предварительном отборе наиболее значимых признаков с использованием регрессии LASSO и прогнозированием 7-дневной выживаемости на основе теоремы Байеса.

В качестве набора данных использовалась выгрузка пациентов, госпитализированных в период с 1 марта по 6 мая 2020 года, из 13 неотложных больниц Нью-Йорка. Набор содержит более 11000 пациентов со средним возрастом 65 лет и общей 7-дневной выживаемостью 89%. На основе электронной медицинской карты были извлечены 42 признака, включая демографические, лабораторные и клинические данные пациентов.

Разработанный калькулятор NOCOS (Northwell COVID-19) основан на 6 наиболее значимых признаках: азот мочевины сыворотки крови, возраст, абсолютное количество нейтрофилов, ширина распределения эритроцитов, насыщение кислородом и натрий, и на тестовой выборке достигает значения AUC 0.86.

В статье Jin, Jin and Agarwala [4] решается задача прогнозирования риска смертности от COVID-19. Предлагаемый подход основан на построении множества моделей пропорциональных рисков Кокса на подвыборках, построенных на основе местоположения и возрастной группы (рассматриваются группы 18–44, 45–74, 75+). На основе расчитанных рисков, для каждого местоположения также строится модель логистической регрессии.

[2] https://www.google.com/covid19/mobility/

В качестве набора данных использовалась выгрузка по пациентам, застрахованным в системе National Health Insurance Scheme с 7 июня 2020 года по 1 октября 2020 года. Набор данных содержит более 4.1 млн пациентов по 259 округам США, а также более 15 признаков, включая анамнез, осмотр и клинические данные. Предложенный метод достигает на тестовой выборке значения AUC 0.895.

В статье Yadaw, Arjun S. and Li [5] решается задача прогнозирования смертности госпитализированных пациентов с диагностированным COVID-19. В работе рассматриваются модели машинного обучения: Logistic Regression, Support Vector Machine, Random Forest и XGBoost.

В качестве набора данных используется выгрузка по пациентам, проходивших лечение в Системе здравоохранения Маунт-Синай в Нью-Йорке, штат Нью-Йорк, США. Набор содержит более 3800 наблюдений и более 20 признаков, включая анамнез (возраст, пол, хронические заболевания) и осмотр врача (сатурация, артериальное давление, температура).

Лучшие результаты прогнозирования смертности пациентов по метрике AUC показал метод XGBoost (0.91 AUC), основанный на идее градиентного бустинга и фокусирующийся на более сложных для прогнозирования подмножествах обучающих данных. Наибольшую значимость в прогнозных моделях получили признаки: сатурация, возраст, артериальное давление.

3. ИССЛЕДОВАНИЕ И ПОСТРОЕНИЕ РЕШЕНИЯ ЗАДАЧИ

Задача прогнозирования степени поражения легких по КТ пациентов может быть представлена в виде решения двух задач бинарной классификации:

1. Определение вероятности легкой степени поражения (КТ 0–1) – пациент не требует госпитализации и может проходить лечение на дому.

2. Определение вероятности тяжелой степени поражения (КТ 3–4) – пациент должен быть незамедлительно госпитализирован в стационар для проведения интенсивной терапии без промежуточного посещения КТ-центра или поликлиники.

3.1. Особенности решения задачи

Важной особенностью данной работы является использование реальных медицинских данных, имеющих несколько значимых проблем.

Во-первых, возникает проблема сложности сбора данных: для составления полной картины лечения пациента требуется обработать данные из нескольких источников (КТ-центр, поликлиника, стационар, лаборатории клинических анализов и тестов). Каждый источник обладает разным объемом данных и может содержать различные ошибки ввода.

При агрегации данных из различных источников, могут возникать противоречия в данных и пропуски в признаках пациентов. Для корректной работы моделей прогнозирования требуется предварительно провести удаление артефактов, противоречий и исследовать подходы для обработки пропусков.

Из-за различного объема данных в источниках также возникает проблема несбалансированности заполнения признаков пациентов. Например, признаки из КТ-центров будут заполнены у всех пациентов, так как целевым признаком является результат степени поражения по КТ. А признаки клинических или биохимических анализов могут быть не заполнены (по причине не проведения тестов или потери данных) или быть неактуальными из-за давнего срока сдачи.

Также, возникает проблема полноты данных, так как доступ к некоторым данным ограничен в силу сложности сбора. В частности, в медицинских исследованиях [6] отмечается значимость признака SpO2 (сатурация) на течение COVID-19. Однако, в доступных источниках данных сатурация встречается у малого числа пациентов. Для повышения заполненности признаков могут быть задействованы дополнительные источники данных, а также произведена аппорксимация на основе имеющихся признаков.

Наконец, существует проблема насбалансированности набора данных относительно степени поражения легких по КТ. Доминирующее большинство пациентов имеют легкую степень поражения КТ-1, а критические степени КТ-3 и КТ-4 имеются у малой доли выборки. Многоклассовая классификация с балансировкой классов приведет к использованию только части имеющихся данных. В данной работе предлагается разработка одноклассовых классификаторов высокой и легкой степени, каждый из которых построен на основе сбалансированных выборок по степени поражения по КТ.

Данный раздел содержит описание базовых моделей машинного обучения с учителем [7, 8], в частности, бинарные классификаторы: метод случайного леса [9, 10], нейронные сети [11, 14] и градиентный бустинг [15, 16].

Выбор данных моделей обуславливается наличием сложных нелинейных зависимостей в данных. Однако, так как заполненность признаков набора данных неравномерна, при использовании базовых моделей может наблюдаться нестабильность работы моделей и переобучение.

Для повышения устойчивости прогнозов базовых моделей и учета неоднородной заполненно-

сти признаков, в данном разделе также рассматриваются ансамбли базовых моделей прогнозирования.

3.2. Метод случайного леса (Random Forest)

Предложенный в статье [9], алгоритм Random Forest основан на идее построения ансамбля деревьев решений [10] и агрегации их прогнозов:

1. Строится N bootstrap выборок (с возвращением) из исходной выборки. Каждая bootstrap подвыборка в среднем исключает 37% данных, которые называются out-of-bag (OOB).

2. На каждой bootstrap выборке строится дерево решений, в каждом узле дерева выбирается P произвольных признаков для поиска лучшего разбиения. Выбирается разбиение, которое максимизирует разницу между дочерними узлами (в частности, максимизует logrank статистику).

3. Деревья выживаемости строятся до исчерпания bootstrap выборки.

Классификация объектов проводится путем голосования: каждое дерево ансамбля относит классифицируемый объект к одному из классов, и побеждает класс, за который проголосовало наибольшее число деревьев.

3.3. Нейронные сети (Neural Networks)

Предложенный в статье [11], алгоритм нейронной сети имитирует функционирование человеческой нейронной системы мозга. С помощью нейронных сетей можно сколь угодно точно аппроксимировать любую непрерывную функцию и имитировать любой непрерывный автомат.

Нейрон представляет собой единицу обработки информации в нейронной сети. В этой модели можно выделить три основных элемента. Во-первых, это набор синапсов x_j, связанных с нейронами k, характеризуемых весом $w_k j$. Во-вторых, сумматор, складывающий входные сигналы, взвешенные относительно соответствующих синапсов нейрона, и добавляющий смещение b_k. Наконец, к полученной сумме применяется функция активации $\varphi(.)$, формирующая выходной сигнал нейрона y_k.

Таким образом, в математическом представлении функционирование нейрона k описывается уравнением:

$$y_k = \varphi\left(\sum_{j=1}^{m} w_{kj}x_j + b_k\right) \qquad (3.1)$$

Поскольку модель нейрона реализует функцию от его входов, нейроны можно объединять в соответствии с правилами суперпозиции функций, получая более сложные модели, называемые персептронами [12] или искусственными нейронными сетями прямого распространения.

Многослойный персептрон [13] имеет несколько отличительных признаков: каждый нейрон имеет нелинейную функцию активации, сеть содержит один или несколько слоев скрытых нейронов.

Для обучения многослойного персептрона используется метод обратного распространения ошибки (от англ. Back propagation) – алгоритм обучения, основанный на вычислении градиента функции ошибок. В процессе обучения веса нейронов каждого слоя нейросети корректируются с учетом сигналов, поступивших с предыдущего слоя, и невязки (отклонения) каждого слоя, которая вычисляется рекурсивно в обратном направлении от последнего слоя к первому.

При одноклассовой классификации в качестве функции ошибок наиболее распространена бинарная кросс-энтропия, а в качестве функции активации – логистическая функция [17].

Также, для избежания переобучения, в архитектуру нейронной сети может быть добавлен слой регуляризации, ограничивающий размер весов. На практике, наибольшее распространение получили слои dropout, зануляющие часть весов перед входом в следующий слой. При использовании слоя dropout, нейросетевая архитектура обучается на частично заполненных данных, предотвращая формирование глобальных зависимостей от малого числа признаков. Используя полную информацию доступных признаков, повышается усточивость архитектуры нейронной сети на реальных данных.

3.4. Градиентный бустинг (Gradient Boosting Machines)

Альтернативным подходом построения ансамбля деревьев решений является Gradient Boosting, представленный в статье Friedman, Jerome H. (2001) [15].

В отличии от Random Forest, алгоритм Gradient Boosting основывается не на независимом построении деревьев решений и усреднении их прогнозов, а на итеративном алгоритме обучения очередного дерева решений на ошибках предыдущего. Агрегация прогнозов деревьев основана на весовых коэффициентах, рассчитываемых при добавлении нового дерева решений в ансамбль.

Целью алгоритма является минимизация loss-функции L, на основе которой производится расчет ошибки ансамбля. По умолчанию, в качестве loss-функции используется логарифмическая функция потерь (log loss) [17]. Пусть $\{(x_i, y_i)\}_{i=1}^{n}$ – обучающий набор, L – функция потерь, M – раз-

мер ансамбля. Алгоритм представляет собой следующую последовательность шагов:

1. Модель инициализируется константным значением α:

$$F_0(x) = \text{argmin}_\alpha \sum_{i=1}^{n} L(y_i, \alpha)$$

2. Вычисляются псевдоостатки для всех наблюдений в обучающей выборке $i = 1, \ldots, n$:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

3. На обучающей выборке $\{(x_i, r_{im})\}_{i=1}^{n}$ строится дерево решений $h_m(x)$.

4. Вычисляется вес v_m $(0 < v_m < 1)$ дерева решений с помощью решения следующей оптимизационной задачи:

$$v_m = \arg\min_v \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + v \cdot h_m(x_i)).$$

5. В ансамбль добавляется обученное дерево с весом:

$$F_m(x) = F_{m-1}(x) + v_m \times h_m(x).$$

6. Если размер ансамбля m не равен M, то переходим на шаг 2.

7. Итоговой моделью является F_M.

Классификация объектов проводится путем композиции откликов моделей ансамбля: каждое дерево решений $h_m(x)$ возвращает вещественную "степень" принадлежности объекта к некоторому классу, а результирующий ответ F_M получается применением порогового правила к композиции.

3.5. Ансамбль случайного леса RF и нейронных сетей NN

Для раздельной обработки признаков с разной заполненностью, предлагается использовать агрегацию базовых моделей прогнозирования. Предлагается использовать ансамбль двух моделей прогнозирования: первая модель обучается на признаках с большим количеством пропусков, а вторая модель обучается на хорошо заполненных признаках и отклике первой модели.

Наибольшее количество пропусков выявлено у группы признаков лабораторных анализов. В данную группу входят все показатели общего анализа крови и биохимических анализов. На основе лабораторных признаков предлагается обучить нейронную сеть. Отклик модели добавляется к группе оставшихся признаков, на основе которой производится обучение случайного леса.

Вероятностный прогноз случайного леса используется при оценке качества модели. Выбор нейронной сети для обучения на малозаполненных признаках обуславливается лучшей аппроксимацией на большом признаковом пространстве числовых признаков (дополненного бинарными признаками наличия пропуска), которые могут коррелировать друг с другом. В случае использования древовидных ансамблей на признаках с пропусками, часть признаков могут быть вовсе не использованы или иметь малую значимость.

3.6. Бустинг ансамбль деревьев решений LGBM

LGBM (Light Gradient Boosting Machine) [16] – это одна из наиболее эффективных реализаций процедуры градиентного бустинга. Поскольку данный метод является ансамблем деревьев решений, LGBM позволяет оценивать важность признаков из обученной модели. Как правило, важность обеспечивает оценку, которая указывает, насколько полезным был каждый признак при построении деревьев решений в модели. Чем больше атрибут используется для принятия ключевых решений, тем выше его относительная важность. Важность рассчитывается для отдельного дерева решений, затем значения характеристик усредняются по всем деревьям решений в модели.

Особенность LGBM заключается в обучении только на тех данных, которые приводят к большему градиенту, что способствует ускорению работы алгоритма и уменьшению его вычислительной сложности.

Также, LGBM позволяет обрабатывать пропуски во входных признаках. Для числовых признаков пропущенные значения соотносятся той ветви разбиения, которая в наибольшей степени уменьшает функцию потерь. Для категориальных признаков пропущенные значения используются в качестве отдельной категории. В таком случае, нет необходимости использовать ансамбль нескольких видов моделей машинного обучения, обрабатывающий признаки с разной заполненностью.

3.7. Ансамбль регуляризованных нейросетей

В чистом виде метод обратного распространения ошибки работает плохо [11, 18]. Возникают проблемы медленной сходимости или расходимости, застревания в локальных минимумах функционала. Для стабилизации процесса обучения была добавлена инициализация и регуляризация слоев нейросети. Так как в качестве функции активации нейросети используется сигмоида, отклик нейросети может быть интерпретирован как вероятность.

Также, при различных исходных инициализациях *random_state*, различаются как результаты

Таблица 1. Структура набора муниципальных данных

Источник	Объем данных	Признаки пациента	Результат анализа	Доп. признаки
Данные КТ-центров	303’628	степень тяжести, ЧДД, температура тела, наличие и тип кашля	время проведения КТ, степень поражения КТ, результат КТ, наличие пневмонии	основной диагноз, сопутствующие диагнозы
Данные амбулаторных тестов	43’348’273	–	название теста, время проведения, время сбора, результат теста, референсные значения	название исследования, единицы измерения
Данные тестов ПЦР и ИФА	4’466’407	дата рождения, пол	тип теста, название и код диагноза, время проведения теста, даты получения и выдачи результатов теста, результат теста	наименование пункта сбора тестов, наименование филиала, наименование лаборатории
Данные по сатурации	661’353	–	время проведения, значение сатурации	–
Данные по госпитализированным пациентам	880’352	дата рождения, пол, группа риска	тяжесть заболевания, ОРИТ, ИВЛ, ЭКМО	дата госпитализации, причина поступления
Данные по умершим пациентам	48’415	дата рождения, пол	дата смерти, причина смерти, код смерти	связь смерти с COVID-19

сходимости нейросети, так и порог бинаризации отклика. Для повышения устойчивости прогноза частных нейросетей, был предложен подход формирования ансамбля нейросетей. Для входного наблюдения рассчитывается множество откликов по всем обученным нейросетям, отклики объединяются суммированием и нормируются. Полученный отклик принимается за отклик ансамбля.

Также, для контроля обучения нейронных сетей использовался обработчик прекращения обучения при ухудшении метрики качества. Максимальное количество эпох без возможного улучшения параметра – 10 эпох. Также, использовался обработчик уменьшения скорости обучения (learning rate) при ухудшении метрики качества.

3.8. Калибровка результатов и выбор порогов

При выполнении классификации часто требуется не только предсказать метку класса, но и получить вероятность соответствующей метки. Эта вероятность определяет уверенность в предсказании.

Распределение вероятностей может быть скорректировано, чтобы лучше соответствовать ожидаемому распределению, наблюдаемому в данных. Такая корректировка называется калибровкой моделей [19, 20] и используется для сведения откликов прогнозных моделей на вероятностную шкалу. Это важно как при интерпретации прогнозов, так и для принятия решений о внедрении моделей и анализа их работы.

Пусть y_i – эталонная вероятность наблюдения x_i, $p(x_i)$ – отклик модели. Задача методов калибровки – построить скорректированный отклик $\hat{p}(x)$.

В данной работе в качестве метода калибровки рассматривается калибровка Платта [21], которая работает путем подгонки модели логистической регрессии к оценкам классификатора:

$$\hat{p}(x) = \frac{1}{1 + \exp(a \cdot p(x) + b)} \qquad (3.2)$$

Параметры a, b определяются методом максимального правдоподобия на отложенной выборке. Калибровка Платта наиболее эффективна, когда искажение в предсказанных вероятностях имеет сигмовидную форму.

Также, возникает проблема соотнесения наблюдений к классам на основе спрогнозированной вероятности. В действительности, не все построенные модели имеют порог 0.5 для выходных вероятностей, а также, веса ошибок при “занижении” или “завышении” результата могут отличаться. В данной задаче ошибки “занижения” прогноза критичнее ошибок “завышения” и не-

обходимо максимизировать метрики Recall (полнота) или Sensitivity (чувствительность).

Экспертами может быть установлено минимальное допустимое значение Recall, на основе которого выбираются пороги, максимизирующие Precision (точность). Данный подход позволяет использовать множество частных порогов в различных организациях использования модели, однако не имеет унифицируемого подхода расчета порогов.

4. ЭКСПЕРИМЕНТЫ

4.1. Описание метрик качества

Для оценки качества моделей прогнозирования в данной работе рассматривается метрика ROC-AUC [22] – площадь под ROC-кривой.

ROC-кривая – график, отображающий соотношение между долей объектов от общего количества носителей признака, верно классифицированных как несущие признак, и долей объектов от общего количества объектов, не несущих признака, ошибочно классифицированных как несущие признак, при варьировании порога решающего правила (ошибок I рода).

Площадь под ROC-кривой AUC (англ. Area Under Curve) принимает значение от 0 до 1 и интерпретируема как вероятность того, что классификатор присвоит больший вес случайно выбранному положительному наблюдению, чем случайно выбранному отрицательному наблюдению.

4.2. Описание наборов данных

В работе рассматриваются два набора данных.

Первый набор данных представляет собой выгрузку муниципальных медицинских данных по заболеваемости клинических и амбулаторных COVID-19 пациентов города Москва с 03.2020 по 02.2021. Выгрузка по пациентам состоит из шести источников данных, структура выгрузки и содержание источников представлено в табл. 1. Все источники данных имеют общий идентификатор пациента. Выгрузка является достаточно полной, на ее основе формируется набор данных для обучения и тестирования прогнозных моделей.

Второй набор данных госпитализированных пациентов городской больницы города Москва представляет собой выгрузку из 95 наблюдений, содержащую: идентификатор пациента, признаки анамнеза (пол, возраст, хронические заболевания), признаки осмотра врача (степень тяжести, сатурация, ЧДД, наличие одышки, слабость, заложенность, температура тела, наличие и тип кашля), результаты лабораторных тестов (PCR, WBC, PDV, MON, GRA, LYM, PLT, HGB, RBC, MPV, HCT, RDW).

В силу малого размера и явного дисбаланса в сторону степени поражения КТ 1, набор данных может использоваться только в виде тестовой выборки для оценки качества моделей.

4.3. Подготовка данных

Перед построением прогнозных моделей необходима обработка исходных медицинских данных по пациентам с диагностированным COVID-19, с целью очистки некорректных данных и приведения значений признаков к общим шкалам, а также нахождения и обработки артефактов, выбросов и противоречивых данных.

В данной работе предлагается следующий алгоритм формирования набора данных на основе множества источников.

Для каждого наблюдения из КТ-центра производится поиск ближайших (окно – неделя) тестов пациентов. Каждому пациенту сопоставляется список тестов на основе уникального идетификатора. Тесты фильтруются по рассматриваемому множеству тестов и сортируются по близости к дате проведения КТ. Выбираются ближайшие тесты с тремя признаками: значение теста, референсные значения, дата взятия теста.

Рассматриваются множества тестов общего клинического анализа крови и наиболее заполненные тесты биохимического анализа: АЛТ, альбумин, АСТ, билирубин общий, билирубин прямой, калий общий, креатинин, лактатдегидрогеназ, мочевина, натрий общий, белок общий, хлор, щелочная фосфатаза, относительное количество нормобластов.

Производится фильтрация исходного набора данных по рассматриваемым тестам. Таким образом, производится поиск теста (а не анализа) и возможно получение результатов теста от нескольких источников исследований. Также, тесты от разных исследований объединяются в один признак теста и унифицируется размерность.

Выделяются наиболее заполненные признаки среди близких тестов. Например, для тромбоцитов существует множество признаков после объединения тестов от разных исследований: общий объем тромбоцитов в крови (тромбокрит, PCT), количество тромбоцитов, средний объем тромбоцитов в крови, ширина распределения тромбоцитов по объему. Данные признаки коррелируют между собой, поэтому в ходе формирования признакового пространства выбирается наиболее заполненный признак.

Сформированный набор данных дополняется тестами: ПЦР, ИФА, сатурация (для каждого пациента выделяются ближайшие тесты с окном – неделя). Также набор дополняется признаками хронических болезней. Каждому пациенту сопоставляется список диагнозов в формате кодов

Таблица 2. Соответствие результата КТ и степени поражения КТ

	Нулевая	Легкая	Средн.	Тяжел.	Крит.
0	**536**	1423	409	78	0
1	510	**157298**	2578	126	7
2	48	3208	**56516**	463	8
3	12	402	1285	**15814**	9
4	3	30	37	169	**1196**

МКБ-10, из которого выделяются хронические диагнозы: ишемическая болезнь сердца (I11, I20, I24, I25, I51), артериальная гипертензия (I10, O10-13, G97, I27, K76, P29, I15), сахарный диабет (G63, E10-14, H36, M14, G59, E23, N08, O24), ожирение (E66).

4.3.1. Унификация значений признаков

В данном разделе рассматривается задача очистки и приведения значений показателей к общим шкалам и словарям, включая учет референсных значений.

В ходе обработки признаков набора муниципальных данных, были построены несколько правил унификации значений признаков. Значения, выходящие за допустимые границы признака, удаляются. Например, в данных из КТ-центров были найдены значения температуры: 3.0, 3.6, 3.8, а также значения ЧДД: 0, 1 и более 150.

Для унификации единиц измерения значений признака предложен алгоритм приведения единиц измерений к наиболее частой (целевой). Изначально, выделяются префиксы и постфиксы целевой единицы измерения. Далее, для остальных единиц измерения вычисляется расстояние до целевой. Обрабатываются префиксы и постфиксы “м, мк, н, мл, к, М”, а также степенные показатели. Неунифицируемые единицы измерения удаляются.

Для унификации категориальных признаков были составлены словари всех возможных категорий.

Для унификации непрерывных признаков прозводится удаление незначащих символов с преобразованием к вещественному виду (в случае множественного значения производится разбиение по разделителям и выбирается первое значение).

Для унификации референсных значений признака были выделены 2 основных типа референсных значений: интервал $x - -y$ и полуинтервалы $< x$, $>y$. Все референсные значения приводятся к данным типам или считаются некорректными.

Для унификации результатов теста ИФА созданы 4 признака: 2 вещественных значения IGG, IGM и 2 бинарных показателя обнаружения (IGG > 10, IGM > 2.0).

4.3.2. Обработка аномальных значений

В данном разделе рассматривается задача поиска, удаления или исправления артефактов, выбросов и противоречивых данных.

В ходе обработки признаков набора муниципальных данных, было найдено несколько типов противоречивых данных. Выявление противоречий проводилось как на основе анализа признаков, так и с помощью экспертной оценки.

Во-первых, наблюдается несогласованность полей “время определения температуры при проведении КТ” и “время проведения КТ” – между датами может пройти несколько дней. В 299 тысячах наблюдений данной проблемы не наблюдается, однако более 3800 наблюдений имеют ненулевое расстояние между датами, в частности, 1 день имеют 2142 наблюдения, более двух дней имеют 1694 наблюдения, более семи дней имеют 763 наблюдения. Максимальная разница во времени составила 176 дней. В данной работе считается некорректным расстояние более 7 дней, такие наблюдения удаляются из набора данных.

Во-вторых, наблюдается несогласованность полей “степень поражения КТ” и “результат КТ”. Согласно описанию исходного набора данных, корректно следующее сопоставление между категорией КТ и степенью поражения КТ: КТ-0 – нулевая, КТ-1 – легкая, КТ-2 – средне-тяжелая, КТ-3 – тяжелая, КТ-4 – критическая. Однако, в наборе данных такому сопоставлению отвечают только 167 тысяч наблюдений. Полное соответствие категории КТ и степени поражения КТ представлено в табл. 2. Также, в 61 тысяче наблюдений значение степени поражения не указано. В данной работе будем считать, что пациент имеет степень поражения КТ-N, если его “результат КТ” КТ-N, а “степень поражения КТ” не ниже КТ-N или не указана.

В-третьих, с помощью экспертной оценки найдено противоречие, основанное на неприводимых единицах измерения. В признаках абсолютного количества эозинофилов, базофилов, моноцитов, гранулоцитов, лимфоцитов, нейтрофилов существует 2 категории значений, в процентном и количественном содержании. Данные категории невозможно преобразовать друг в друга общим подходом преобразования единиц измерения. Для корректного преобразования требуется умножить значения процентного содержания на количество лейкоцитов (WBC). При их отсутствии, значение принимается пустым.

Наконец, с помощью экспертной оценки найдено противоречие, основанное на отсутствии единиц измерения у признаков RDW, PDW, D-димер. Для устранения данной проблемы было принято решение анализировать референсные значения признаков.

Для признака RDW (и PDW) определено следующее правило: если правая референсная граница меньше 20 или написание содержит запятую, то значение представлено в виде **RDW-CV** (и **PDW-CV**) и измеряется в %, иначе представлено в виде **RDW-SD** (и **PDW-SD**) и измеряется в фемтолитрах. Сопоставление единиц измерения производится по формуле $RDW - CV = \frac{RDW - SD}{MCV} \times 100$, где MCV – средний объем эритроцитов (или по формуле $PDW - CV = \frac{PDW - SD}{MPV} \times 100$, где MPV – средний объем тромбоцитов).

Для признака D-димера, определено следующее правило: если правая референсная граница меньше 1, то значение измеряется в мкг/мл, иначе измеряется в нг/мл. Сопоставление единиц измерения производится по правилу: 1 мкг/мл = = 1000 нг/мл.

4.3.3. Обогащение набора данных

В данном разделе рассматривается задача расширения признакового пространства за счет включения информации о датах проведения анализов и осмотра.

Как было описано ранее, при формировании набора данных наблюдения с проведенной КТ обогащаются клиническими анализами, тестами ПЦР и ИФА, а также сатурацией, взятыми за недельный период от даты проведения КТ.

На практике данные анализов не всегда являются актуальными. Для контроля актуальности данных были добавлены 2 признака: дата общего анализа крови и дата биохимического анализа. На их основе вычисляется количество дней между анализами и датой КТ.

На основе данных признаков можно проводить фильтрацию актуальности входных данных, а при количестве дней, меньшем 7, использовать признак в прогнозной модели.

Также, для каждого признака N набора данных формируется признак $none\_N$, отображающий наличие в оригинальном признаке пропусков (1, если признак имел пропуск, 0 иначе).

4.3.4. Доопределение сатурации в данных

В данном разделе рассматривается задача взаимообратного учета показателя сатурации в оценке степени тяжести осмотра с пересчетом по баллам шкалы NEWS2.

При прогнозировании ожидаемой степени поражения КТ одним из важнейших факторов является показатель сатурации (насыщение крови кислородом в процентном соотношении). Однако, в сформированном наборе данных признак сатурации заполнен лишь на 103 тысячах наблюдений (из 299 тысяч).

Для увеличения заполненности по сатурации предлагается использовать международную шкалу NEWS2 [23] (National Early Warning Score), предложенную в 2020 году для оценки тяжести течения COVID-19 Королевским колледжем врачей (Royal College of Physicians).

На основе шкалы NEWS2 могут быть сформированы признаки b_spo2, bw_spo2. Признак $b\_spo2 = \frac{97 - SpO2}{2}$ определяет баллы по заполненной сатурации, а признак bw_spo2 определяет ожидаемые баллы за сатурацию на основе весовой схемы по заполненным признакам ЧДД, температуры тела и степени тяжести осмотра пациента.

Таким образом, для каждого наблюдения ставится балл за сатурацию при ее наличии или ожидаемые баллы при отсутствии.

4.3.5. Формирование выборок

Таким образом, с учетом описанной подготовки данных, по источникам муниципальных данных был сформирован набор данных с 299’792 наблюдениями. Распределение целевого поля “результат КТ” представлено на рис. 1.

Общий анализ крови заполнен на 176’354 наблюдениях (будем понимать наличие значения признака “Гематокрит”). Биохимический анализ крови заполнен на 176’866 наблюдениях (будем понимать наличие значения “С-реактивный белок”). Общий анализ крови и биохимический анализ одновременно заполнен на 151’532 наблюдениях.

После формирования исходных выборок для классификаторов КТ 01-234 и КТ 012-34, производится заполнение пропусков по медианному значению признака на исходных выборках. Также, производится балансировка классов: меньший класс размера N входит в выборку полностью, а из большего класса выбираются произвольным образом N наблюдений.

На основе полученного набора данных формируются выборки для обучения, валидации и тестирования моделей.

Для классификатора КТ 01-234 общее количество наблюдений составило 67–648 с пропорцией наблюдений в обучающей, валидационной и тестовой выборках: 54–118/6’765/6’765.

Для классификатора КТ 012-34 общее количество наблюдений составило 12’576 с пропорцией наблюдений в обучающей, валидационной и тестовой выборках: 10’062/1’257/1’257.

Также, в работе рассматривается 2 подхода формирования тестовых, валидационных и обучающих выборок. В случае рандомизированного разбиения, в обучающую выборку случайным образом определяются 80% наблюдений оригинальной выборки, а в тестовой и валидационной выборках оставшиеся наблюдения распределены в равных долях.

Таблица 3. Таблица результатов классификации КТ 01-234 и КТ 012-34

	Рандомизированная выборка		Выборка по времени	
	КТ 01-234	КТ 012-34	КТ 01-234	КТ 012-34
NN на лабораторных признаках	0.7954	0.8401	0.7899	0.7984
NN на анализах крови + RF + Калибровка Платта	0.8842	0.9227	0.8317	0.8737
Ансамбль NN	0.9153	0.9327	0.8729	0.8904
LGBM без пропусков + калибровка Платта	**0.9173**	**0.9455**	**0.8743**	**0.8988**
LGBM с пропусками + калибровка Платта	0.9170	0.9453	0.8742	0.8987

В случае разбиения по времени, в тестовую выборку определяется первые 10% наблюдений за весь период с максимальным временем проведения КТ, а в валидационную выборку последующие 10% наблюдений. Остальные наблюдения относятся к обучающей выборке.

4.4. Постановка экспериментов

Первоначально проводится предобработка набора данных, формирование признаковых пространств и целевых переменных для задач классификации КТ 01-234 и КТ 012-34.

Далее, проводится обработка признаков, заполнение пропусков на основе медианы по выборке, балансировка классов. Порождаются тестовые, валидационные и обучающие выборки на основе сформированных наборов данных.

На обучающей выборке с подбором параметров на валидационной выборке производится обучение бинарных классификаторов: нейронная сеть (NN), нейронная сеть со случайным лесом (NN+RF), ансамбль NN, LGBM с пропусками, LGBM без пропусков (пропуски заполняются медианным значением по исходной выборке классификатора).

Для построенных моделей рассчитывается прогноз на тестовой выборке, после чего проводится оценка качества прогнозирования по метрике ROC AUC.

4.5. Таблицы результатов

В табл. 3 представлена информация о полученных значениях метрики ROC AUC на рандомизированной и временной тестовой выборке для задач классификации КТ 01-234 и КТ 012-34.

Исходя из таблицы результатов, лучшие значения метрики ROC AUC на двух задачах классификации по рандомизированной и временной выборке показала модель LGBM без пропусков с калибровкой Платта.

Также, проводится тестирование модели на наборе данных городской больницы. На задаче классификации КТ 01-234 получен ROC AUC 0.8805, а на задаче классификации КТ 012-34 ROC AUC 0.9311. ROC кривые классификации изображены на рис. 3.

5. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ

На основе результатов теоретических и экспериментальных исследований был разработан вебсервис “Калькулятор КТ”, предоставляющий пользователю возможность экспресс-оценки изменений легочной ткани при COVID-19 без применения компьютерной томографии органов

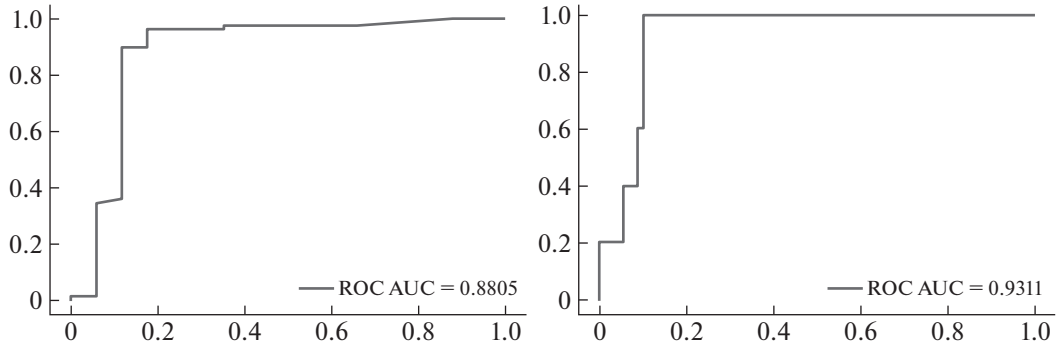


Рис. 3. ROC-кривые классификации КТ 01-234 и КТ 012-34 тестового набора данных городской больницы.

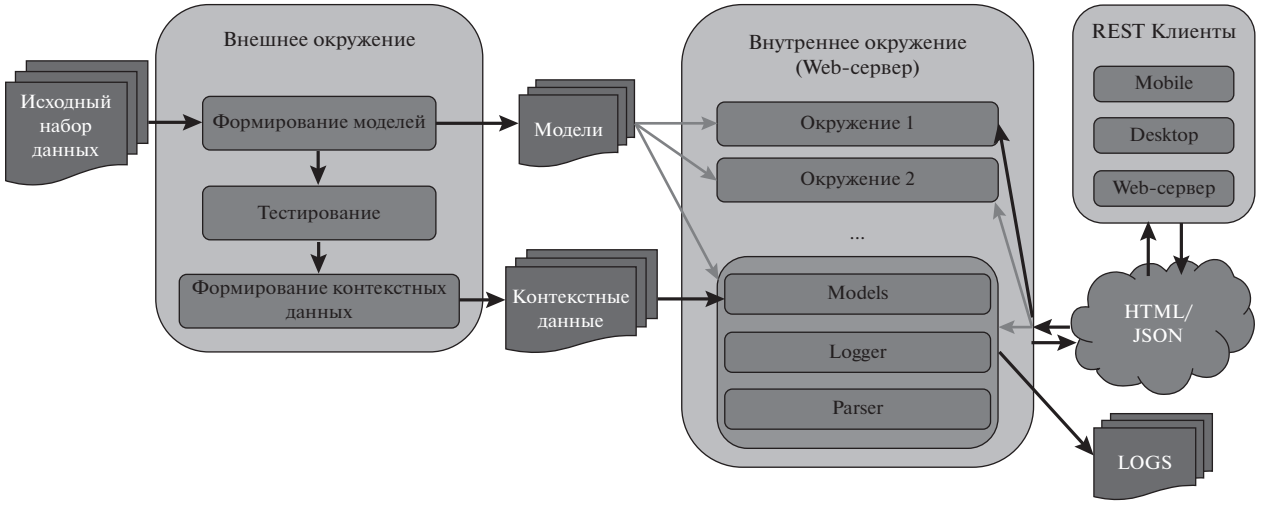


Рис. 4. Архитектура сервиса Калькулятор КТ.

грудной клетки на основе физикальных и лабораторных анализов пациента. Сервис позволяет получить прогноз вероятности легкой и тяжелой степеней поражения легких.

5.1. Архитектура реализации

Программная реализация сервиса "Калькулятор КТ" основана на двух независимых частях: внешнем и внутреннем окружении (полная архитектура представлена на рис. 4).

Внешнее окружение предназначено для детальной настройки прогнозных моделей, принимает на вход исходный набор данных, проводит предобработку признаков и формирует обучающие и тестовые выборки. На основе сформированных выборок производится обучение и выбор лучшей прогнозной модели на основе метрики ROC AUC. Тестовая выборка используется для формирования отчетов о прогнозной модели и расчета пороговых значений. Наконец, выводом внешнего окружения является файл лучшей прогнозной модели, а также контекстная информация (медианные значения по набору данных и коэффициенты стандартизации признаков).

Внутреннее окружение предназначено для запуска Web-сервиса и принимает на вход файлы, порожденные внешним окружением, проводит загрузку и инициализацию моделей. Каждый используемый вид прогнозной модели порождает окружение, состоящее из модели, функции-обработчика признаков наблюдений и логгера. В запросе пользователя возможно указание конкретного окружения прогноза.

Во внутреннем окружении также находится WSGI-сервер [24] на основе Waitress, который руководит работой Web-приложения. Web-приложение обрабатывает поступающие запросы и выполняет предсказания на основе ранее обученной модели.

При внедрении сервиса во внешнюю среду, необходимо передать только программный код внутреннего окружения, а также файлы, порожденные внешним окружением. Данный подход требует малое количество оперативной памяти, так как исходный набор данных обрабатывается перед запуском сервиса, а также не требует передачи набора данных во внешнюю среду, следуя требованиям безопасности.

После применения модели прогнозирования пользователю сервиса возвращается уникальный идентификатор запроса. При получении ошибочного результата пользователь может сообщить разработчику идентификатор запроса и разработчиком будет проведен анализ логированных данных о совершенном запросе (в частности, проверка на противоречивость). Если обращение корректно, событие размечается и вносится в тестовый набор.

Также, логируемые события используются для формирования статистики работы сервиса "Калькулятор КТ". Статистика была использована для исследования входных данных и освещена в статье на Официальном сайте Мэра Москвы[3].

5.2. Методы взаимодействия

Для взаимодействия с сервисом "Калькулятор КТ" используется архитектурный стиль REST [25], функционирующий поверх протокола HTTP. Для каждой операции сопоставляется свой собственный HTTP метод: GET – получение данных;

[3] Российские врачи 10 тысяч раз воспользовались КТ-калькулятором для диагностики COVID-19: https://www.mos.ru/news/item/86015073/

POST – создание новых данных; PUT – обновление, модификация данных; DELETE – удаление данных.

В качестве пакета данных отправляется JSON [26] массив на указанный адрес сервиса. Со стороны сервиса “Калькулятор КТ” срабатывает функция-обработчик, а в зависимости от отправленных данных и текущего запроса возвращается прогноз в определенном формате.

5.3. Интеграция сервиса во внешнюю среду с помощью docker

Практическая реализация позволяет пользователю сервиса предсказать степень поражения легких на основе протоколов течения болезни. Для получения прогноза достаточно использовать веб-форму или инструменты REST API.

Однако, в таком случае, запросы отправляются в единый сервис и логи запросов консолидируются в единое хранилище. Для поддержки нескольких независимых сервисов с локальными хранилищами, в данной работе используется программное обеспечение Docker [27, 28].

ПО Docker предназначено для автоматизации развертывания и управления приложениями в средах с поддержкой контейнеризации приложений, а также для более эффективного использования системы и ресурсов, быстрого развертывания программных продуктов, их масштабирования и переноса в другие среды с гарантированным сохранением стабильной работы.

Основной принцип работы Docker – контейнеризация приложений. Этот тип виртуализации позволяет упаковывать программное обеспечение по изолированным средам-контейнерам. Каждая из сред содержит все нужные элементы для работы приложения. Это дает возможность одновременного запуска большого количества контейнеров на одном источнике.

ПО Docker состоит из нескольких компонентов. Во-первых, сервер, выполняющий инициализацию демона (фоновой программы), который применяется для управления и модификации контейнеров, образов и томов. Демон управляет Docker-объектами (сети, хранилища, образы и контейнеры) и может связываться с другими демонами для управления сервисами Docker. Во-вторых, клиент, позволяющий пользователю взаимодействовать с сервером при помощи команд. В-третьих, механизм REST API, отвечающий за организацию взаимодействия Docker-клиента и Docker-демона.

Docker характеризуется достаточно простым синтаксисом, а также совместимо со всеми версиями операционных систем Linux и Windows.

Таким образом, с помощью ПО Docker сервис Калькулятора КТ может быть упакован в контейнер и интегрирован во внешнюю среду. Доступ к такому контейнеру также предоставляется через REST API.

В декабре 2020 года “КТ-калькулятор” был встроен в городскую медицинскую информационную систему, бесплатный доступ к нему получили врачи из любых регионов, а также обычные пользователи.

6. ЗАКЛЮЧЕНИЕ

В данной работе была рассмотрена задача прогнозирования степени поражения легких пациента с диагностированным COVID-19.

Результаты компьютерной томографии являются важным фактором для определения дальнейшей стратегии лечения пациента. При легкой степени поражения КТ 0-1 пациент не требует госпитализации и может проходить лечение на дому, а при тяжелой степени поражения КТ 3-4 пациента небходимо госпитализировать в стационар для проведения интенсивной терапии без промежуточного посещения КТ-центра или поликлиники.

Массовое применение КТ имеет и недостатки, например, риск создания искусственных эпидемических очагов, нерациональная работа бригад скорой помощи и экономические затраты.

В качестве альтернативного диагностического инструмента в рамках данной работы разработаны прогнозные модели оценки легкой (КТ 0-1) и тяжелой (КТ 3-4) степени поражения легких пациента при COVID-19 на основе осмотровых, клинических и биохимических признаков.

Важной особенностью данной работы является использование реальных медицинских данных, имеющих несколько значимых проблем, таких как: сложность сбора данных из нескольких источников, ограниченность сбора данных, несбалансированность заполненности признаков, а также наличие ошибок ввода и противоречий.

В данной работе рассматриваются как базовые модели машинного обучения: метод случайного леса, нейронные сети, градиентный бустинг, так и ансамбли базовых моделей, позволяющие обработать признаки с разной заполненностью. Для сведения откликов прогнозных моделей на вероятностную шкалу используется калибровка Платта.

По результатам экспериментального исследования лучшее качество по метрике ROC AUC показал метод LGBM с заполнением пропусков.

Предложенные прогнозные модели были реализованы в виде веб-сервиса “КТ-калькулятор”, и в декабре 2020 года сервис был встроен в городскую медицинскую информационную систему. Использование сервиса помогает в оперативности принятия врачебных решений и позволяет сократить количество исследований для пациента и

уменьшить облучение. Также, использование сервиса сокращает нагрузку на оборудование КТ-центров и может применяться в регионах, в которых доступ к компьютерному томографу ограничен, либо этого оборудования нет.

СПИСОК ЛИТЕРАТУРЫ

1. *García-Cremades S., Morales-García J., Hernández-Sanjaime R., Martínez-España R., Bueno-Crespo A., Hernáandez-Orallo E., López-Espín J.J., Cecilia J.M.* Improving prediction of Covid-19 evolution by fusing epidemiological and mobility data // Scientific Reports. 2021. V. 11. № 1. P. 1–16.
2. *Elaziz M.A., Hosny K.M., Salah A., Darwish M.M., Lu S., Sahlol A.T.* New machine learning method for image-based diagnosis of covid-19 // Plos One. 2020. V. 15. № 6. P. e0235187.
3. *Levy T.J., Richardson S., Coppa K., Barnaby D.P., McGinn T., Becker L.B., Davidson K.W., Cohen S.L., Hirsch J.S., Zanos T.P. et al.* Development and validation of a survival calculator for hospitalized patients with Covid-19. MedRxiv, 2020.
4. *Jin J., Agarwala N., Kundu P., Harvey B., Zhang Y., Wallace E., Chatterjee N.* Individual and community-level risk for Covid-19 mortality in the united states // Nature Medicine.2021. V. 27. № 2. P. 264–269.
5. *Yadaw A.S., Li Y.-C., Bose S., Iyengar R., Bunyavanich S., Pandey G.* Clinical features of Covid-19 mortality: development and validation of a clinical prediction model // Lancet Digital Health. 2020. V. 2. № 10. P. e516–e525.
6. *Shah S., Majmudar K., Stein A., Gupta N., Suppes S., Karamanis M., Capannari J., Sethi S., Patte C.* Novel use of home pulse oximetry monitoring in Covid-19 patients discharged from the emergency department identifies need for hospitalization // Academic Emergency Medicine. 2020. V. 27. № 8. P. 681–692.
7. *Bishop C.M.* Pattern recognition // Machine Learning. 2006. V. 128. № 9.
8. *Ripley B.D.* Pattern recognition and neural networks. Cambridge University Press, 2007.
9. *Breiman L.* Random forests // Machine Learning. 2001. V. 45. № 1. P. 5–32.
10. *Breiman L., Friedman J.H., Olshen R.A., Stone C.J.* Classification and regression trees. Routledge, 2017.
11. *Rumelhart D.E., Hinton G.E., Williams R.J.* Learning representations by backpropagating errors // Nature. 1986. V. 323. № 6088. P. 533–536.
12. *Minsky M.L., Papert S.A.* Perceptrons: expanded edition, 1988.
13. *Hunt K.J., Sbarbaro D., Żbikowski R., Gawthrop P.J.* Neural networks for control systems – a survey // Automatica. 1992. V. 28. № 6. P. 1083–1112.
14. *Breiman L.* Randomizing outputs to increase prediction accuracy // Machine Learning. 2000. V. 40. № 3. P. 229–242.
15. *Friedman J.H.* Greedy function approximation: a gradient boosting machine // Annals of Statistics. 2001. P. 1189–1232.
16. *Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., Liu T.-Y.* Lightgbm: A highly efficient gradient boosting decision tree // Advances in Neural Information Processing Systems. 2017. V. 30. P. 3146–3154.
17. *Dreiseitl S., Ohno-Machado L.* Logistic regression and artificial neural network classification models: a methodology review // Journal of Biomedical Informatics. 2002. V. 35. № 5–6. P. 352–359.
18. *Rosen B.E.* Ensemble learning using decorrelated neural networks // Connection Science. 1996. V. 8. № 3–4. P. 373–384.
19. *DeGroot M.H., Fienberg S.E.* The comparison and evaluation of forecasters // Journal of the Royal Statistical Society: Series D (The Statistician). 1983. V. 32. № 1–2. P. 12–22.
20. *Niculescu-Mizil A., Caruana R.* Obtaining calibrated probabilities from boosting / In: UAI. 2005. V. 5. P. 413–420.
21. *Platt J. et al.* Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods // Advances in Large Margin Classifiers. 1999. V. 10. № 3. P. 61–74.
22. *Hand D.J., Till R.J.* A simple generalisation of the area under the roc curve for multiple class classification problems // Machine Learning. 2001. V. 45. № 2. P. 171–186.
23. *Smith G.B., Redfern O.C., Pimentel M.A., Gerry S., Collins G.S., Malycha J., Prytherch D., Schmidt P.E., Watkinson P.J.* The national early warning score 2 (news2) // Clinical Medicine. Journal of the Royal College of Physicians of London. 2019. V. 19. № 3.
24. *Gardner J.* The web server gateway interface (wsgi) // The Definitive Guide to Pylons. 2009. P. 369–388.
25. *Ong S.P., Cholia S., Jain A., Brafman M., Gunter D., Ceder G., Persson K.A.* The materials application programming interface (api): A simple, flexible and efficient api for materials data based on representational state transfer (rest) principles // Computational Materials Science. 2015. V. 97. P. 209–215.
26. *Crockford D.* The application/json media type for javascript object notation (json) // RFC 4627, 2006.
27. *Anderson C.* Docker [software engineering] // IEEE Software. 2015. V. 32. № 3. P. 102_c3.
28. *Boettiger C.* An introduction to docker for reproducible research // ACM SIGOPS Operating Systems Review. 2015. V. 49. № 1. P. 71–79.

EDN: RLOCLC
ПРОГРАММИРОВАНИЕ, 2022, № 4, с. 17–26

ПРОГРАММНАЯ ИНЖЕНЕРИЯ, ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ ПРОГРАММ

УДК 681.3.06

АНАЛИЗ СТРУКТУРНОГО ПОКРЫТИЯ ТОЧЕК ВХОДА И ВЫХОДА, НЕОБХОДИМЫЙ ДЛЯ ДОСТИЖЕНИЯ ЦЕЛЕЙ, ОПРЕДЕЛЕННЫХ В DO-178C

© 2022 г. В. П. Козырев[a,b,*]

[a] *Национальный исследовательский ядерный университет "МИФИ", 115409 Москва, Каширское ш., 31, Россия*

[b] *ООО "ЛаБС" (Advalange), 123001 Москва, ул. Садовая-Кудринская, д. 25, к. 4, Россия*

*****E-mail: vkozyrev@list.ru, Vladimir.Kozyrev@advalange.com*

Поступила в редакцию 18.10.2021 г.
После доработки 11.01.2022 г.
Принята к публикации 23.01.2022 г.

В число целей процесса верификации, определенных в документе RTCA DO-178C, входит анализ структурного покрытия программного обеспечения (ПО), включающий анализ покрытия структурных элементов исходного кода программ в соответствии с критериями SC, DC и MC/DC, а также анализ связности компонентов ПО по данным и по управлению. Критерии покрытия структурных элементов используются уже много лет (документ RTCA DO-178B опубликован в 1992 г.), однако их определение в DO-178B/C не является однозначным. В частности, для критерия DC не определены понятия точек входа, выхода и их покрытия, и разработчики инструментов сбора и анализа структурного покрытия (ИССП) определяют их по своему усмотрению. В статье сделана попытка устранения этой неоднозначности для программ, написанных на языках C/C++, и предложены решения, реализация которых в ИССП, по мнению автора, необходима с точки зрения достижения целей анализа структурного покрытия, определенных в DO-178C.

DOI: 10.31857/S0132347422040033

1. ВВЕДЕНИЕ

Анализ покрытия исходного кода является одним из методов верификации, применяемых при разработке ПО, требующего высокой надежности, в частности – авиационного ПО, разработка которого регламентируется документом RTCA DO-178C [1]. В этом документе указано, что "анализ структурного покрытия определяет, какая структура кода, включая интерфейсы между компонентами, не была выполнена тестовыми процедурами, основанными на требованиях", а также определены (в разделе 6.4.4.2) задачи и цели анализа структурного покрытия, в которые входят:

– анализ информации о структурном покрытии, собранной во время тестирования на основе требований, для подтверждения, что степень структурного покрытия соответствует уровню ПО, и

– анализ, подтверждающий, что при тестировании на основе требований была проверена связность по данным и по управлению между компонентами кода.

В [4] первая задача была названа анализом покрытия структурных элементов программы, а вторая – анализом покрытия связей, которая, в свою очередь, включает задачи анализа связей по управлению и анализа связей по данным. Будем использовать такую терминологию и в этой статье.

Из определения задач анализа структурного покрытия следует, что этот анализ должен выполняться на основе информации о прохождении потоков управления и данных в ПО при выполнении тестов, написанных по требованиям к нему. Сбор такой информации называется сбором структурного покрытия (или просто сбором покрытия), и обычно он выполняется с помощью специальных инструментов сбора структурного покрытия (ИССП), которые, кроме того, выполняют первичный анализ собранного покрытия на предмет его полноты в соответствии с актуальным критерием и формируют отчет о покрытии, анализируемый затем уже человеком. Разными производителями выпущено довольно много таких инструментов, предназначенных для достижения целей DO-178C в части анализа структур-

ного покрытия. Но поскольку DO-178C, как и его предшественник – DO-178B [2], не содержат ни исчерпывающих определений критериев структурного покрытия, ни их интерпретаций применимых для конкретных языков программирования, используемых при разработке ПО, эти критерии и их интерпретации доопределяются разработчиками инструментов, вследствие чего характеристики и функциональность существующих ИИСП в той или иной степени различны.

В DO-178C определено три критерия покрытия структурных элементов, применяющихся при анализе покрытия ПО в соответствии с его уровнем. Эти критерии применяются при анализе покрытия ПО уровней C, B и A соответственно:

– SC (Statement Coverage) – покрытие операторов,

– DC (Decision Coverage) – покрытие решений и

– MC/DC (Modified Condition/Decision Coverage) – модифицированное покрытие условий и решений.

Однако, пожалуй, лишь определение критерия покрытия SC ("*каждый оператор программы был вызван хотя бы один раз*") не вызывает особых вопросов, связанных с его реализацией и использованием в ИССП, чего нельзя сказать о критериях DC ("*каждая точка входа и выхода в программе пройдена хотя бы раз, а также каждое решение в программе приняло все возможные значения хотя бы один раз*") и MC/DC ("*каждая точка входа и выхода в программе пройдена хотя бы раз, каждое условие в решении в программе было выбрано при всех возможных исходах хотя бы раз, каждое решение в программе приняло все возможные значения хотя бы один раз, а также показано, что каждое условие в решении независимо влияет на исход данного решения*"), хотя они и используются в процессах верификации, проводимых в соответствии с DO-178B/C, уже много лет. Ни в DO-178C, ни в сопутствующих документах нет определений понятий точек входа и выхода, которые входят в определения критериев покрытия DC и MC/DC. Что же касается критериев покрытия связности – по управлению и, в особенности, по данным, то в DO-178C нет достаточно четкого определения методов их анализа, обеспечивающих достижение целей, объявленных в этом документе. Разъяснения относительно целей анализа покрытия связности представлены в документе RTCA DO-248C [3], однако вопрос о том, какую именно информацию необходимо иметь для достижения целей анализа покрытия связности, остается открытым. Другими словами, эти документы не определяют, что именно должен делать ИССП.

Попытка определения функциональности ИССП, необходимой для достижения целей, определенных в DO-178C, была предпринята в статье [4], где были рассмотрены проблемы анализа структурного покрытия исходного кода программ, написанных на языках C/C++, и фактически были определены основные требования к ИССП, которые необходимо рассматривать в контексте достижения этих целей. В [4] были определены три группы задач анализа структурного покрытия, обозначенных как: анализ покрытия структурных элементов – в соответствии с критериями SC, DC и MC/DC, анализ покрытия связей по управлению (Control Coupling) и анализ покрытия связей по данным (Data Coupling). Наиболее полно в статье [4] были рассмотрены проблемы анализа покрытия структурных элементов, однако некоторые проблемы, относящиеся к этой теме, в ней не были рассмотрены.

Одной из таких проблем является упомянутая выше проблема интерпретации понятий точек входа и выхода, входящих в определения критериев DC и MC/DC, которая была лишь обозначена в [4], и автор решил привести здесь свои соображения на эту тему (которые можно рассматривать в качестве дополнения к [4]). Автор попытался выделить точки входа и выхода в конструкциях языков C/C++, определить критерии их покрытия и предложить вариант представления результатов их покрытия пользователю ИССП. При этом некоторые из приводимых здесь выводов и рекомендаций, касающихся представления результатов покрытия, не могут быть строго аргументированы и подтверждены, например, ссылками на какие-либо регламентирующие документы, но делаются на основе личного опыта автора и его представлениях о целесообразности того или иного решения для достижения целей анализа покрытия, исходя из того, что интерфейс ИССП должен быть адекватен потребностям пользователя, обеспечивая получение пользователем всей необходимой для анализа покрытия информацией и минимизируя его затраты на проведение этого анализа (этот тезис поясняется ниже).

Так как способы представления данных о покрытии нигде не регламентируются, они определяются разработчиками инструментов, и на их решения могут влиять различные факторы. Среди них могут быть стереотипы, сложившиеся у разработчиков и пользователей инструментов за время применения практик анализа структурного покрытия (документ DO-178B был принят в 1992 г.), представления разработчиков о процессе сбора и анализа покрытия, включающие, в частности, аспекты, связанные с управлением этим процессом, или какие-то иные, в том числе субъективные, соображения. Вследствие этого информация, выдаваемая ИССП, может оказаться не адекватной потребностям пользователя. Например, некоторые из существующих ИССП выдают информацию о процентном соотношении покрытых и не покрытых структурных элементов программ. Возможно, данные о процентах по-

крытия могут иметь какое-то применение, но вот при планировании процесса анализа покрытия они, возможно, будут бесполезными, поскольку для оценки необходимых затрат ресурсов на анализ обнаруженных “дыр” в покрытии более важно знать их абсолютное, а не относительное количество.

При проектировании интерфейса следует учитывать, что в DO-178C для инструментов верификации, к которым относятся ИССП, предусмотрена их квалификация, целью которой является обеспечение гарантии того, что инструмент не пропустит ошибку в верифицируем объекте. Для ИССП это требование следует понимать так, что инструмент должен обнаруживать все не покрытые структуры ПО и сообщать о них пользователю. А поскольку анализ покрытия проводится человеком на основе данных, предоставляемых ему ИССП, следует иметь в виду, что избыточность в данных, выдаваемых инструментом, может как упростить, так и усложнить этот анализ. Например, если инструмент явно не сообщает о результате покрытия некоторого структурного элемента, но этот результат может быть вычислен по данным о покрытии каких-то других элементов, это придется делать пользователю, а если в отчете о покрытии будет слишком много данных о покрытых элементах, это может затруднить поиск данных о не покрытых элементах, и при неудачно выбранном способе представления данных пользователь может в этом случае просто их не заметить.

Однако в целом проблема проектирования интерфейсов ИССП является открытой. Автор считает, что ее исследование и решение позволили бы стандартизировать и за счет этого упростить для пользователя интерпретацию выдаваемых инструментами данных о покрытии и, кроме того, упростить процесс квалификации ИИСП, требуемый DO-178C.

Исследования проблем анализа структурного покрытия были начаты автором несколько лет назад совместно с М.А. Сабуровым, памяти которого автор хотел бы посвятить эту статью.

2. ПОКРЫТИЕ ТОЧЕК ВХОДА И ВЫХОДА

В определения критериев DC и MC/DC (включающего в себя критерий DC) входят понятия точек входа и выхода, однако определений этих понятий документ [1] не содержит. Краткое пояснение смысла этих понятий содержалось в документе DO-248B [5]: “... entry and exit points include low level entry and exits for decision and control constructs (e.g., IF-THEN-ELSE, DO WHILE, CASE) and such things as implicit ELSE conditions, CASE defaults, and implicit jumps”. Однако оно исчезло из следующей редакции этого документа – DO-248C [3], и для того, чтобы говорить о реализации сбора покрытия по критериям DC и MC/DC, необходимо определить, какие элементы в программах на языках C/C++, рассматриваемых в этой статье, следует считать точками входа и выхода.

Достаточно общим можно считать подход к определению понятий точек входа и выхода, связанный с организацией передачи управления в программе. В соответствии с этим подходом под точкой входа следует понимать точку в программе, в которую управление может быть передано в результате нарушения последовательного выполнения элементов исходного кода программы, в качестве которых в языках C/C++ могут рассматриваться операторы и выражения. Соответственно, точкой выхода следует считать точку, из которой выполняется такой переход. При этом следует рассматривать как переходы между функциями (к этой категории будем относить и методы классов в C++), выполняемые при вызовах функций и возвратах из них, так и переходы внутри функций, для организации которых в языках C/C++ могут использоваться, например, условный оператор, операторы цикла, оператор безусловного перехода и некоторые другие языковые конструкции. При таком определении покрытием точек входа/выхода следует считать прохождение потока управления через них, связанное с передачей управления, нарушающей последовательное выполнение элементов программы, при этом точку входа будем считать покрытой, если на нее было передано управление, а точку выхода – если при ее выполнении была выполнена такая передача управления.

В приведенном определении критерия DC можно выделить две части, на которые далее по тексту будем ссылаться как на первую и вторую соответственно. Первая часть определяет, что каждая точка входа и выхода в программе должна быть пройдена хотя бы раз (покрытие точек входа и выхода), а вторая – что каждое решение в программе приняло все возможные значения хотя бы один раз (покрытие логики). Далее по тексту понятие “покрытие решения” будет использоваться в контексте второй части критерия DC, при этом будем считать решение покрытым полностью (или просто покрытым), если в ходе тестирования оно приняло оба своих значения, покрытым частично, если оно приняло только одно значение – истина или ложь, и не покрытым, если оно ни разу не выполнилось.

Целесообразность выделения точек входа и выхода в тех или иных языковых конструкциях определяется целями анализа структурного покрытия, которые для ПО уровней A и B включают, в частности, оценку того, все ли операторы и ветвления в программе были выполнены в ходе ее

тестирования. Рассматривая в этом контексте первую часть определения критерия DC, можно обнаружить, что в языках C/C++ для управляющих конструкций, в которых ветвление выполняется на основе значений некоторого решения (логического выражения), покрытие этого решения, обеспечивает и покрытие соответствующих ветвей в этих конструкциях. Поэтому в таких конструкциях, например, в операторе if, нет смысла отдельно анализировать покрытие точек входа и выхода, соответствующих передачам управления по результату вычисления его условия: для оценки полноты тестов такой анализ не даст ничего нового по сравнению с анализом того, что каждое решение в программе приняло все возможные значения хотя бы один раз.

Аналогичный вывод можно сделать и относительно нецелесообразности отдельного рассмотрения точек выхода и входа, образующихся при использовании в программе операторов вызовов функций, поскольку анализ их покрытия обеспечивается анализом покрытия операторов по критерию SC: покрытие оператора вызова функции и оператора, следующего за ним, фактически означает покрытие соответствующих точек выхода и входа в вызывающей функции. Однако при вхождении вызовов функций в выражения, вычисляемые при выполнении программы, например, в логические выражения, которые вычисляются в языках C/C++ по короткой схеме, такой вывод уже сделать нельзя. Убедиться в выполнении всех присутствующих в программе вызовов функций можно было бы при анализе покрытия связности по управлению, однако при таком анализе должны рассматриваться связи между компонентами, каждый из которых в общем случае может включать множество функций, а передачи управления между функциями, входящими в один и тот же компонент, могут и не рассматриваться. Кроме того, поскольку сбор покрытия связности, как по данным, так и по управлению, должен проводиться при выполнении только интеграционных тестов, его вообще не следует рассматривать как возможный источник информации о покрытии структурных элементов, которое может собираться как по интеграционным, так и по модульным тестам. Следовательно, при наличии вызовов функций в логических выражениях необходимо убедиться, что каждый такой вызов был выполнен хотя бы однажды, а после его выполнения управление хотя бы однажды было возвращено из вызванной функции, и сообщать пользователю ИССП обо всех случаях невыполнения этого условия. Фактически здесь речь идет об уточнении определения критерия DC.

Таким образом, если прохождение через точку входа или выхода может быть вызвано ветвлением, которое не обнаруживается анализом покрытия операторов и логики – в соответствии со второй частью определения критерия DC, то анализ покрытия таких точек необходим. Например, такой анализ необходим для ветвей оператора switch, при использовании безусловных переходов по меткам, или когда выход из функции производится на оператор, не следующий сразу за оператором ее вызова, что возможно, например, при использовании механизмов setjump/longjump и try/catch. Однако явное выделение точек входа или выхода может оказаться целесообразным и в случаях, когда их покрытие может быть определено из покрытия операторов и логики, но дополнительная (избыточная) информация позволяет упростить пользователю анализ результатов покрытия, выдаваемых инструментом. Понятие простоты анализа будем рассматривать здесь неформально, оценивая полноту информации, необходимой для анализа покрытия, выводимой ИССП обычно в некотором отчете.

На основе приведенных рассуждений рассмотрим далее более подробно конструкции языков C/C++ на предмет необходимости или целесообразности выделения в них точек входа и выхода и отображения их в отчете о покрытии, выдаваемом инструментом. Не рассматривая конкретных вариантов возможных форм представления отчета о покрытии, будем считать, что в отчете должна содержаться информация о покрытии структурных элементов исходного кода, актуальная для выбранного критерия покрытия. И поскольку речь идет об анализе покрытия исходного кода, выводимая в отчете информация должна быть так или иначе ассоциирована с конкретными элементами или фрагментами этого кода. В связи с этим, при рассмотрении языковых конструкций будут предлагаться и возможные варианты “привязки” выделяемых в них точек входа и выхода к исходному коду, необходимой для выполнения анализа его покрытия. Разумеется, способы представления информации о покрытии могут быть различными, и при рассмотрении предлагаемых вариантов следует учитывать, что рассмотрение проблем организации интерфейсов ИССП не входит в цели настоящей статьи.

2.1. Функции

В контексте анализа покрытия точек входа и выхода специфика функций как структурных элементов программы состоит в том, что передачи управления возможны как в пределах одной функции, так и между разными функциями. Заметим, что при анализе покрытия по критериям DC и MC/DC имеет значение только сам факт передачи управления и не имеет значения, откуда или куда при этом управление передается. Поэтому здесь способы передачи управления между функциями нас будут интересовать в контексте определения элементов программы (операторов,

выражений), которые следует рассматривать как точки входа и выхода.

В языках C/C++ для передачи управления между функциями могут использоваться разные способы, основным из которых можно считать выполнение вызова функции и возврата из нее при выполнении оператора return или по достижению конца тела void-функции. Будем назвать такой способ, а также соответствующие ему точки входа и выхода функции регулярными. Неявные вызовы функций конструкторов, деструкторов и переопределений операторов в языке C++ также отнесем к регулярным. Кроме регулярных возможны и другие (нерегулярные) способы передачи управления между функциями, когда управление передается не в начало тела функции или когда возврат управления из функции выполняется в точку, расположенную не сразу после точки ее вызова. В этом разделе рассматриваются регулярные способы передачи управления, а нерегулярные – в следующем.

Чтобы обеспечить возможность анализа покрытия точек входа и выхода необходимо, прежде всего, выявить их при синтаксическом анализе программы, а затем по результатам сбора покрытия предоставить пользователю инструмента отчет об обнаруженных в программе точках входа/выхода и их покрытии. Выше отмечено, что для сокращения объема информации, которую придется обрабатывать пользователю при анализе покрытия, некоторые точки могут явно не выделяться в отчете. Например, рассмотрим следующий фрагмент кода:

int F(int x) { int y; y = f1(x); return y; }.

В этом примере анализ покрытия точек выхода и входа, соответствующих вызову функции f1(x), может быть выполнен на основе данных о покрытии операторов, причем результат покрытия точки выхода (т.е. покрыта она или нет) будет соответствовать результату покрытия оператора y = f1(x), а результат покрытия точки входа – результату покрытия следующего за ним оператора return, и явное указание в отчете результатов покрытия точек входа и выхода в этом случае можно считать излишним, так как их покрытие определяется покрытием операторов.

Усложним немного этот пример, добавив в оператор присваивания вызов другой функции:

int F(int x) { int y; y = f1(x) + f2(x); return y; }.

В этом примере покрытие обоих операторов в функции F будет означать и покрытие точек входа и выхода, соответствующих вызовам обеих функций – f1(x) и f2(x). Если же оператор присваивания окажется покрытым, а оператор return – нет, из этого факта будет следовать только то, что вызов f1(x) был выполнен и соответствующая ему точка выхода была покрыта, а точка входа для вызова f2(x) покрыта не была, так как оператор return может оказаться не покрытым, если не будет выполнен регулярный выход из любой из двух функций – f1 или f2.

Внесем еще одно изменение в рассматриваемый пример, сделав выражение в операторе присваивания логическим:

int F(int x, int y) {return x && (y || f(x)); }.

Вызов функции f(x) в выражении x && (y || f(x)) будет выполнен, если значение параметра x функции F будет не равно нулю, а значение параметра y окажется нулевым, однако для покрытия этого выражения по критерию DC достаточно двух наборов параметров функции F: {x = 0} и {x = 1; y = 1}, и в этом случае вызов функции f(x) не будет выполнен, хотя решение x && (y || f(x)) будет покрыто.

В языке C++ кроме явных вызовов возможно выполнение и неявных вызовов функций – конструкторов, деструкторов и переопределений операторов. Ниже приведен пример программы, в которой выполняются такие неявные вызовы. Строки кода, в которых происходят неявные вызовы функций, помечены соответствующими комментариями.

```
1 class A
2 {
3   int x;
4 public:
5   A(int i = 0) { x = i; }                  // Конструктор по умолчанию
6   A(const A& z) { x = z.x + 10; } // Конструктор копирования
7   ~A()       { x = 0; }                    // Деструктор
8   const A operator+(A& a)
9     { return A(const x + a.x);}     // вызов конструктора по умолчанию
10  A operator-()
11     { return A(-x); }                    // вызов конструктора по умолчанию
12 };
13
```

```
14 A a1;                    // вызов конструктора A(0)
15
16 void F(const A& x)
17 {
18   A a2(1);               // вызов конструктора A(2)
19   A a3 = -a2;            // вызов функции перегрузки унарного минуса
20   A* a4 = new A;         // вызов конструктора A(0)
21   A a5 = x;              // вызов конструктора копирования
22   A a6 = *a4;            // вызов конструктора копирования
23   delete a4;             // вызов деструктора для переменной a3
24   A a7 = a1 + 1;         // вызов конструктора A(1), перегрузки бинарного
25                          // плюса и деструктора для константы A(1)
26 }                        // вызов деструктора для переменных a7, a6, a5, a3, a2
27
28 int main()
29 {
30   F(5);                  // вызов конструктора константы A(5), функции F и
31                          // деструктора для константы A(5)
32 }                        // вызов деструктора для переменной a1
```

Даже в таком, весьма простом, примере выполняется довольно много неявных передач управления, и можно предположить, что в более сложных программах неявных передач управления будет намного больше, причем некоторые из них могут оказаться неочевидными с точки зрения человека, выполняющего анализ покрытия. Например, в строках 19 и 24 выполняется вызов функций переопределяющих операторы “–” и “+”, из которых, в свою очередь, вызываются конструкторы по умолчанию, а непосредственно в строках 19 и 24 вызовов конструкторов не происходит (такие результаты были получены при выполнении этого примера в среде MS Visual Studio).

Из рассмотрения приведенных примеров можно сделать вывод о том, что в случаях, когда точки входа или выхода, соответствующие явным или неявным вызовам функций, оказываются не покрытыми, отображение этих фактов в отчете о покрытии необходимо. Если же точки входа или выхода оказываются покрытыми, явное представление их в отчете о покрытии представляется избыточным, поскольку основной целью анализа структурного покрытия следует считать доказательство отсутствия не покрытых структурных элементов, а не их перечисление. Но в любом случае, покрыта точка или нет, возникнет вопрос о том, каким образом представить ее в отчете. Например, в приведенном выше фрагменте кода точки входа и выхода, соответствующие явным вызовам функций и перегруженных операторов, можно ассоциировать непосредственно с их представлениями в коде, в частности, с символами “F”, “–” и “+” в строках 30, 19 и 24, а неявные вызовы конструкторов и деструкторов – со строками кода, отмеченными соответствующими комментариями. Однако решение этого вопроса не входит в цели настоящей статьи.

Идентификация точек входа в тело функции и выхода из нее представляется более простой задачей, чем идентификация точек входа и выхода ее вызовов. В качестве регулярной точки входа в функцию естественно рассматривать какую-то точку в ней, выполняемую перед первым исполняемым оператором в теле функции. А так как объектный код, выполняемый при входе в функцию, при анализе покрытия исходного кода нас не интересует, точку входа, в принципе, можно ассоциировать с любым структурным элементом определения функции, расположенным текстуально выше первого исполняемого оператора ее тела, и наиболее подходящим, в плане универсальности, представляется отнесение этой точки к фигурной скобке, задающей начало тела функции. Соответственно, при выводе в отчете информации о ее покрытии будут использоваться координаты этой скобки.

Так как регулярные выходы из функции производятся при выполнении операторов return, а для void-функции – и при завершении выполнения ее тела, соответствующие точки выхода естественно ассоциировать с операторами return и фигурной скобкой, завершающей тело функции. При этом следует учитывать, что если в конце тела void-функции, непосредственно перед завершающей ее тело фигурной скобкой, расположен оператор return, то нет смысла рассматривать эту скобку в качестве отдельной точки выхода. В этом случае, вероятно, будет целесообразно объединить ее с предшествующим ей оператором return.

Аналогичное решение можно предложить и для функций, возвращающих значение, поскольку некоторые компиляторы допускают отсутствие операторов return или наличие исполняемых операторов, отличных от return, перед завершающей фигурной скобкой в таких функциях.

Конечно, предлагаемое сопоставление точек входа и выхода с фигурными скобками, ограничивающими тело функции, основывается на субъективном представлении автора о наглядности разметки текста, но в его пользу можно привести такой аргумент, что образующаяся при этом симметрия может упростить восприятие пользователем информации об их покрытии.

2.2. Нерегулярные способы передачи управления

Нерегулярная передача управления может выполняться как внутри одной функции, так и между разными функциями. В частности, это возможно при использовании библиотечных функций setjmp, longjmp и exit, определенных в стандартах языка C. В программах на языке C++ нерегулярные способы передачи управления могут реализовываться и при организации обработчиков исключений с использованием операторов try, catch и throw, причем в этом случае управление на обработчик catch может передаваться как из блока try, так и из любой функции, вызванной из него или находящейся в цепочке вызовов, начинающейся в нем. Кроме того, с использованием ассемблера могут быть реализованы и другие способы передачи управления, однако ограничимся здесь рассмотрением только тех методов, которые определены в стандартах языков C/C++.

Рассматривая функции setjmp, longjmp и exit в контексте достижения целей, определенных в [1], можно заметить, что хотя эти функции и определены в стандарте языка C, они обычно реализуются в библиотеках времени выполнения, которые в случае их использования в бортовом ПО должны рассматриваться как его части, и для их верификации должны применяться общие методы, определенные в [1]. Тем не менее, вызовы этих функций целесообразно рассматривать как точки входа (setjmp) и выхода (longjmp и exit), поскольку на уровне исходного кода программ на C/C++, в которых используются эти вызовы, они представляют собой расширение языка, реализующее соответствующие передачи управления, что делает необходимым регистрацию их покрытия с предоставлением пользователю (в отчете о покрытии) соответствующей информации.

Использование операторов try/catch вносит свою специфику в определение связанных с ними точек выхода, поскольку переход на блок catch может происходить как при выполнении операторов throw, так и при возникновении исключений при выполнении программы внутри блока try, например, при делении на ноль. При использовании операторов throw их естественно рассматривать как точки выхода из блока try, но точки, в которых может возникнуть исключение, вызывающее переход к блоку catch, обнаружить при синтаксическом анализе программы, как правило, невозможно. Да и рассматривать все потенциальные точки выхода из блока try в качестве предмета для анализа их покрытия вряд ли целесообразно, поскольку при тестировании программы вряд ли будет целесообразным (а в общем случае и вряд ли возможным) проверять обработку исключений для каждой такой точки. Скорее всего, при тестировании будет достаточным обеспечить проверку переходов к обработчикам исключений catch из некоторых точек в блоке try или просто проверить, что переходы на блоки catch выполняются.

Руководствуясь приведенными рассуждениями, можно предложить ассоциировать все потенциальные точки возникновения исключений с одной точкой выхода из соответствующего блока try, связав ее, например, с концом блока try – закрывающей фигурной скобкой, но отличая их от регулярной точки выхода, выполняемой по завершению блока try. Но если “срабатывание” каких-либо конкретных точек выхода в программе при возникновении исключений предусмотрено тестами, то анализ их покрытия следует считать необходимым, и для этого в ИССП необходимо предусмотреть возможность их идентификации. Например, это можно сделать путем аннотирования анализируемого исходного кода. Инструмент же в этом случае должен будет отслеживать возникновение исключений в каждой из таких точек по отдельности. Хотя такое решение можно считать субъективным, доводом в его пользу является то, что альтернативой ему будет рассмотрение всех операторов и выражений в блоке try в качестве потенциальных точек выхода, но большинство из них могут оказаться не покрытыми при выполнении тестов, что усложнит анализ полученного покрытия, поскольку пользователю придется анализировать и описывать причины отсутствия покрытия для каждой такой точки.

Что же касается точек входа, связываемых с передачами управления из блока try, то естественно считать такими точками входы в блоки catch (соответствующие открывающие фигурные скобки), а также первый оператор после последнего блока catch группы try/catch, а если его нет, то фигурную скобку, завершающую блок, в который входит try. Соответствующим образом операторы, обеспечивающие нерегулярные передачи управления, могут отображаться и в отчетах о покрытии, выдаваемых пользователю.

2.3. Безусловные переходы по меткам

Безусловные переходы по меткам организуются с использованием операторов goto и меток, при этом оператор goto соответствует точке выхода, а метка – точке входа. Поскольку выполнение операторов goto и, соответственно, меток, по которым они передают управление, может быть обнаружено при анализе покрытия операторов (SC), их выделение в качестве точек входа/выхода в отчетах о покрытии не является обязательным. Однако если не выделять в качестве точек входа метки, анализ их покрытия усложнится, так как потребуется просмотр всего кода функции для поиска операторов goto, обращающихся к ним. Поэтому целесообразно отображать покрытие меток как точек входа в отчетах о покрытии по критериям DC и MC/DC, а операторы goto отображать и как операторы (statements), и как точки выхода.

2.4. Условные операторы

При выполнении операторов if и условной операции (“?”) управление может передаваться на начала их ветвей (true и false), а также на следующий оператор в коде программы – по завершении ветви true при наличии ветви false, или при отсутствии ветви false (некоторые компиляторы позволяют использовать т.н. бинарные условные операции, в которых нет ветви false). Как отмечено выше (в 2), явное выделение точек входа и выхода, формируемых условными операторами, не является обязательным, поскольку их покрытие полностью определяется покрытием логики. Исходя из этого, можно считать не обязательным и отображение их в отчетах о покрытии.

Однако поскольку выбор интерфейсов ИССП остается, все же, за разработчиками инструментов, возможен и альтернативный подход. Например, в инструменте может быть предусмотрена выдача статистики по всем точкам входа и выхода, и потребуется явное их указание для того, чтобы пользователь смог проверить выданные в отчете данные об их покрытии. Возможна также и реализация в инструменте обоих подходов с выбором пользователем нужного ему представления данных о покрытии.

2.5. Оператор switch

В операторе switch имеет смысл рассматривать в качестве точек входа метки case и default, в которые управление попадает после вычисления значения выражения в заголовке оператора switch. Условно можно считать, что соответствующие точки выхода реализуются в заголовке. В случаях, когда метка default отсутствует, управление из заголовка может передаваться на оператор, следующий за switch, а поскольку такого оператора может и не оказаться, в таких случаях точку входа можно отнести к завершению тела оператора switch, которым может являться скобка “}”, если тело оператора является составным оператором (блоком), или разделитель “;” – в противном случае (стандарт языка С допускает использование в качестве тела switch оператора, не являющегося составным). Несмотря на некоторую искусственность такого решения, оно позволяет избежать проблем сопоставления точек входа с конкретными элементами исходного кода при выводе отчета о покрытии, что может облегчить его восприятие пользователем ИССП.

В качестве точки выхода в операторе switch может выступать оператор break, передающий управление оператору, следующему за switch. Поскольку его покрытие определяется покрытием операторов (SC), его представление в качестве точки выхода не является обязательным, однако, как и для условных операторов (2.4), возможна реализация в ИССП и альтернативного подхода.

Рассматривать же точки выхода, реализуемые в заголовке оператора switch нет смысла, поскольку, во-первых, их невозможно выделить в коде, а во-вторых, каждая такая точка выхода и соответствующая ей точка входа в операторе switch покрываются одновременно. Таким образом, в операторах switch целесообразно отображать в качестве точек выхода операторы break, а в качестве точек входа – операторы case и default, а также скобку “}” или разделитель “;” – в зависимости от конкретного формата оператора switch. В частности, символы “}” или “;” имеет смысл представлять как точки входа только в тех случаях, когда в операторе switch отсутствует метка default. В противном случае эти символы, (так же, как и символ “{“ в начале составного оператора) нет смысла отображать в отчете о покрытии как точки входа/выхода. Исключение составляет случай, когда телом switch является пустой составной оператор (“{}”). В этом случае его целесообразно представлять как непокрытый оператор.

2.6. Операторы цикла

В операторах цикла целесообразно рассматривать неявные точки входа и выхода, связанные с ветвлением программы, которое выполняется для организации цикла (переход на начало тела цикла при выполнении его очередного шага, переход на изменение параметров цикла или проверку условия после очередного выполнения тела цикла, выход из цикла при невыполнении условия цикла). Кроме них в теле оператора цикла могут присутствовать явные точки выхода, порождаемые операторами безусловного перехода – как контекстно связанными с ним (break и continue), так и нет (return, throw, goto и др.). При рассмотрении циклов будем учитывать из них только опе-

раторы break и continue (остальные же операторы рассматриваются в других разделах). Поскольку покрытие этих явных точек выхода определяется покрытием операторов (SC), может быть, например, реализовано их опциональное представление – как точек выхода или просто как операторов. Рассмотрим специфику операторов цикла, определенных в языках C/C++.

2.6.1. Оператор while

В операторе while точки входа реализуются в условии цикла и в начале тела цикла. Точки выхода реализуются в условии и в конце тела цикла, а также могут явно определяться в теле оператора цикла операторами перехода. На основе рассуждений, приведенных в 2, можно сделать вывод о том, что:

– отображать точки входа и выхода в заголовке цикла не нужно;

– для тела цикла, являющегося составным оператором, целесообразно отображать в качестве точек входа и выхода обрамляющие его скобки “{“ и ”}” соответственно;

– если телом цикла является одиночный оператор, целесообразно отображать в качестве точки выхода завершающий его разделитель “;”.

2.6.2. Оператор do

В операторе do (do-while), также, как и в операторе while, точки входа реализуются в условии цикла и в начале тела цикла. Точки выхода реализуются в условии и в конце тела цикла, а также могут явно определяться в теле цикла операторами break и continue. На основе рассуждений, приведенных в 2, так же можно сделать вывод о том, что:

– отображать точки входа и выхода в заголовке цикла не нужно;

– для тела цикла, являющегося составным оператором, целесообразно отображать в качестве точек входа и выхода обрамляющие его скобки “{“ и ”}” соответственно;

– если телом цикла является одиночный оператор, целесообразно отображать в качестве точек входа и выхода ключевые слова do и while соответственно.

2.6.3. Оператор for

Оператор for может быть представлен в “классическом” формате, определенном в стандарте языка C, или же в формате “с диапазоном” (range-based), который может использоваться в языке C++. В классическом варианте оператор имеет вид **for** (⟨инициализация⟩; ⟨условие⟩; ⟨модификация⟩) ⟨тело цикла⟩. В формате с диапазоном оператор выглядит следующим образом: **for** (⟨параметр цикла⟩:⟨диапазон⟩) ⟨тело цикла⟩.

Рассмотрим точки входа и выхода, которые ассоциируются с возможными ветвлениями потока управления, происходящими при выполнении оператора. Точки входа реализуются в заголовке, в выражениях ⟨условие⟩, ⟨модификация⟩ и ⟨диапазон⟩, а также в начале тела цикла. Точки выхода реализуются в заголовке – в выражениях ⟨условие⟩, ⟨модификация⟩ и ⟨диапазон⟩, в конце тела цикла, а также могут явно определяться в теле цикла операторами break или continue.

Отображение в отчетах о покрытии точек входа, реализуемых в заголовках циклов, не представляется целесообразным, поскольку их покрытие определяется покрытием логики и операторов соответственно. Точкой выхода для выражения ⟨модификация⟩ всегда является переход на ⟨условие⟩, поэтому ее покрытие также определяется покрытием операторов. Точками выхода для выражений ⟨условие⟩ и ⟨диапазон⟩ являются переходы к телу цикла или выход из цикла. Покрытие точек выхода из условия в цикле for “классического” формата также определяется покрытием решения ⟨условие⟩: если решение не покрыто полностью, то соответствующие точки выхода (true и/или false) следует считать не покрытыми.

Циклы с диапазоном (range-based) с точки зрения логики передач управления для организации цикла можно считать аналогичными циклам while, где в роли условия выступает ⟨диапазон⟩. Хотя ⟨диапазон⟩ и не является логическим выражением, при представлении результатов его покрытия по критериям DC и MC/DC целесообразно рассматривать его как некий аналог решения и считать его:

– не покрытым, если множество, определяемое выражением ⟨диапазон⟩, пустое и тело цикла не выполняется ни разу;

– покрытым полностью, если цикл завершается по исчерпанию множества, определяемого выражением ⟨диапазон⟩;

– покрытым частично, если тело цикла выполняется, но при его выполнении на некоторой итерации происходит выход из цикла.

При этом покрытие точек входа, соответствующих условию цикла, началу тела цикла и завершению цикла, а также точек выхода, соответствующих переходам на эти точки входа, будет определяться покрытием выражения ⟨диапазон⟩.

Для отображения точки входа в тело цикла и точки выхода, из которой управление попадает на проверку условия, в случаях, когда тело цикла является составным оператором, наиболее очевидным способом представляется представление в качестве точки входа в тело цикла его начальную скобку “{“, а в качестве точки выхода – завершающую скобку “}”. Если же тело цикла является одиночным (не составным) оператором, то в качестве представления точки выхода можно использовать завершающий его символ “;”. В отображении же точки входа, которой является сам

этот оператор, нет необходимости, поскольку ее покрытие определяется покрытием условия цикла.

3. ЗАКЛЮЧЕНИЕ

Приведенные в статье соображения позволяют, по мнению автора, более точно понять условия, необходимые для достижения целей, определенных в DO-178C в части анализа покрытия структурных элементов (в терминологии [4]), и сформулировать требования, которые должны учитываться при разработке ИССП, обеспечивающих возможность достижения этих целей. На основе соображений, приведенных в [4] и в настоящей статье, была сделана попытка определить и реализовать такие требования в ИССП, получившем название COVERest [6]. В ходе работ по созданию этого инструмента были продолжены исследования проблем анализа покрытия связности по управлению и данным, однако эти исследования пока не закончены, и автор надеется, что у него будет возможность опубликовать результаты этих работ по их завершению.

СПИСОК ЛИТЕРАТУРЫ

1. RTCA/DO-178C. Software Considerations in Airborne Systems and Equipment Certification. RTCA Inc., 2011.
2. RTCA/DO-178B. Software Considerations in Airborne Systems and Equipment Certification. RTCA Inc., 1992.
3. RTCA/DO-248C. Supporting Information for DO-178C and DO-278A. RTCA Inc., 2011.
4. *Kozyrev V.P., Saburov M.A.* Satisfying DO-178C Structural Coverage Objectives // Programming and Computer Software. 2018. V. 44. № 1. P. 43–50.
5. RTCA/DO-248B. Final report for clarification of DO-178B. RTCA Inc., 2001.
6. COVERest: инструмент автоматизации анализа структурного покрытия кода. URL: https://gosniias.ru/coverest.html (дата обращения: 18.10.2021).

EDN: WEMCXC
ПРОГРАММИРОВАНИЕ, 2022, № 4, с. 27–49

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

УДК 004.43

КОНТРОЛЬ ИНФОРМАЦИОННЫХ ПОТОКОВ В ПРОГРАММНЫХ БЛОКАХ БАЗ ДАННЫХ НА ОСНОВЕ ФОРМАЛЬНОЙ ВЕРИФИКАЦИИ

© 2022 г. А. А. Тимаков[a],*

[a] *МИРЭА – Российский технологический университет, 119454 Москва, Проспект Вернадского, д. 78, Россия*

******E-mail: timakov@mirea.ru*

Поступила в редакцию 08.06.2021 г.
После доработки 13.09.2021 г.
Принята к публикации 21.12.2021 г.

К настоящему времени проведено большое количество исследований в области формальных моделей безопасности компьютерных систем. В работе не рассматриваются криптографические методы защиты информации, которые, применительно к информационным системам, обеспечивают безопасность при реализации ими функций хранения и передачи данных. Основное внимание уделяется моделям, построенным на основе управления доступом и контроля информационных потоков и обеспечивающим безопасность при обработке данных. Модели, основанные на управлении доступом, нашли широкое применение при реализации механизмов защиты уровня операционных систем (ОС) и систем управления базами данных (СУБД). Контроль информационных потоков (КИП) сегодня активно внедряется в языковые платформы, предназначенные для создания как системного так и прикладного программного обеспечения. Однако, несмотря на все усилия, на практике при построении автоматизированных информационных систем промышленного уровня доминирует подход, ориентированный исключительно на реализацию общесистемных механизмов управления доступом. КИП на уровне приложений зачастую рассматривается без привязки к требованиям глобальной политики безопасности. В работе делается попытка обосновать необходимость дополнения глобальных систем управления доступом механизмами КИП в программном обеспечении. Приводится алгоритм внедрения КИП на уровне программных блоков баз данных (БД) на основе реализованного на системном уровне ролевого управления доступом.

DOI: 10.31857/S0132347422040057

1. ВВЕДЕНИЕ

Формальные методы обеспечения безопасности информационных систем можно разделить на две категории: криптографические и управления доступом/контроля информационных потоков. Криптографические механизмы защиты информации (КМЗИ) направлены на поддержание конфиденциальности и целостности информации в процессе ее хранения и передачи. Механизмы управления доступом/контроля информационных потоков защищают данные при их обработке.

В сложившейся практике область применения механизмов второй группы ограничена системным уровнем (ОС и СУБД). При проектировании безопасных систем предпочтение отдается комбинированным подходам, таким как мандатно-ролевое управление доступом. В настоящее время все больше специалистов признает необходимость расширения формальных моделей безопасности, в первую очередь, построенных на основе КИП, до вершины информационного стека – уровня приложений. Как будет показано далее, исследования активно продолжаются на протяжении последних 20 лет, в настоящее время появились первые языковые платформы, в которых реализованы полноценные механизмы контроля информационных потоков: Joana [1, 2], JIF [3], Paragon [4], Flow Caml [5]. Надо признать, что теоретические исследования в области формальных моделей безопасности приложений значительно опережают практику. При этом сложность современных систем и соответствующих угроз [6] не позволяет отказаться от идеи полноценного внедрения этих моделей на прикладном уровне и в целом на уровне специального программного обеспечения. Приведем ряд дополнительных аргументов в поддержку последнего утверждения.

В мире формальных моделей безопасности уровня ОС считается вполне обоснованным предположение о разделении всех процессов на доверенные и недоверенные. При этом доверенные процессы реализуют функции безопасности и не вступают в кооперацию с недоверенными при реализации запрещенных информационных

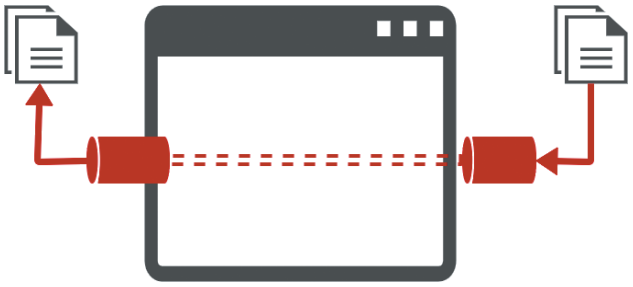


Рис. 1. Среда передачи данных.

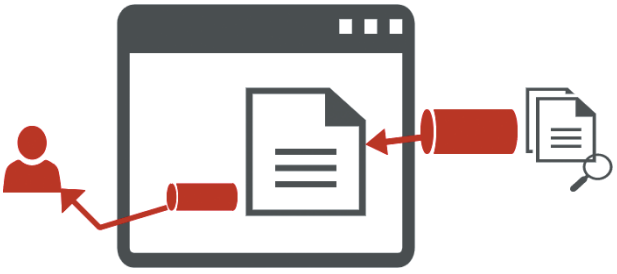


Рис. 2. Приложение – “витрина”.

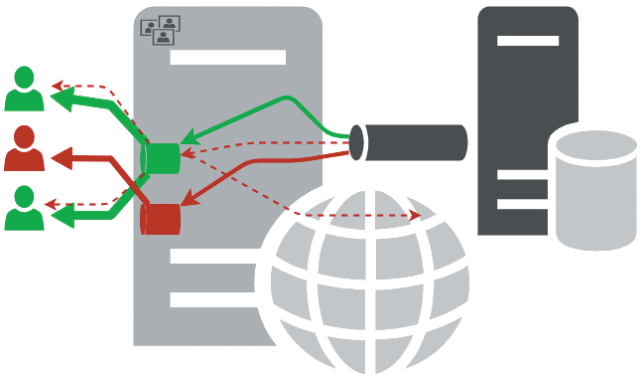


Рис. 3. Многопользовательские приложения.

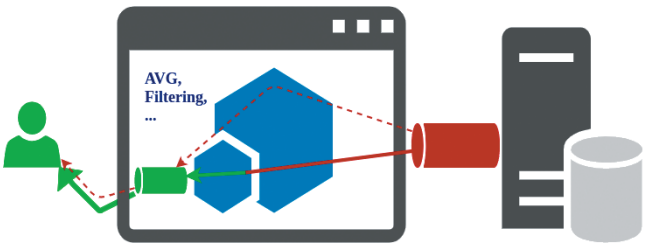


Рис. 4. Предобработка данных.

потоков. Возможность нарушения целостности программ, инициирующих доверенные процессы исключается. Мы утверждаем, что даже при этих существенных допущениях самодостаточными механизмы управления доступом системного уровня могут считаться только в случае, когда пользовательские приложения выступают в роли тривиальной среды передачи данных от источников к получателям или ”витрин”, то есть их функции ограничены представлением данных пользователям без искажения смысла (рис. 1, 2).

В случае, когда приложения осуществляют существенные преобразования данных, приводящие, например, к деклассификации (рис. 4), или в случае, когда допускается многопользовательский режим работы, управление доступом на уровне ОС или СУБД в строгом смысле не может рассматриваться как самодостаточный механизм (рис. 3).

В соответствии с некоторыми рекомендациями в случаях, показанных на рис. 4, допускается (по аналогии с административными функциями) выносить выполнение критичных вычислений в отдельные доверенные процессы. Наличие подобных доверенных процессов в системе видится недопустимым послаблением. Очевидно, все модули программного обеспечения информационной системы создаются, как правило, одной командой разработчиков. Кроме того, подобное разделение на процессы существенно повышает сложность самой разработки.

Известный опыт внедрения мандатно-ролевого управления доступом на уровне ОС и СУБД одновременно свидетельствует о практической сложности достижения требуемого уровня детализации политики безопасности. В частности, говоря об объектах БД, разграничение доступа может быть эффективно реализовано на уровне пользовательских схем. Дальнейшая детализация осложняется большим числом различных объектов (представления, хранимые процедуры, синонимы, последовательности и др.) и скрытых информационных каналов.

Наконец, полезно провести аналогию с криптографическими методами защиты информации, которые нашли практическое применение на всем стеке информационных технологий: физическом, сетевом, системном и прикладном (Таблица 1).

Таким образом специальное программное обеспечение (пользовательские приложения, несистемные службы, программные блоки баз данных и др.) целесообразно воспринимать как нетривиальную среду передачи данных, анализ информационных каналов в которой представляет собой отдельную задачу. Широко распространенные формальные методы статического анализа применительно к этой задаче обладают рядом существенных ограничений. В первую очередь, описание политики безопасности информационных потоков (в отличие, например, от описания правил безопасной работы с памятью) является

Таблица 1. Реализация механизмов управления доступом и КМЗИ

	Канальный	Сетевой	Системный	Прикладной
Управление доступом КМЗИ	Port Security PPTP	фильтрация трафика IPSec (VPN)	управление доступом EFS (BitLocker)	**КИП – ?** Application Layer Encryption

Таблица 2. КИП в приложениях – исторический аспект

	Описание политики	Безопасность семантики	Механизм проверки	Реализация
Модель Белла–Лападулы [9]	+	–	±	–
Модель Деннинг (решетки) [10]	+	–	+	–
Система типов Волпано [11]	+	±	+	–
Flow Caml [5]	+	±	+	±
JIF [3]	+	–	+	+
Paragon [4]	+	+	+	+

специфичным для конкретной системы и требует значительного инструментирования кода: добавления комментариев, аннотаций, расширенных модификаторов доступа и т.д. Кроме того, с учетом сложности семантики информационных потоков, возникающих в программном обеспечении (явных, неявных, вероятностных, временных, случайных и т.д.), значительно возрастают затраты на выявление “ложных” срабатываний.

Далее в п. 2 кратко рассматривается исторический аспект развития методов КИП в программном обеспечении. Последовательно излагаются результаты некоторых наиболее значимых исследований в этой области, описываются направления развития и приводятся отдельные проекты, которые можно считать прототипами первых безопасных (в смысле информационного невлияния [7]) расширений языков программирования. В конце раздела формулируется идея разделения функций безопасной разработки и анализа с переносом механизма КИП из конкретной языковой среды в среду формальной верификации. Мотивирующий пример (см. п. 3) облегчит восприятие следующего раздела (см. п. 4), в котором более подробно описан подход, ориентированный на реализацию безопасных вычислений в среде, максимально приближенной к данным. Руководствуясь желанием сохранить в фокусе главную идею предложенного подхода, в данной работе ограничимся общими рассуждениями о корректности и применимости предложенного механизма контроля.

2. РЕТРОСПЕКТИВА

Читателям, знакомым с общей теорией и историей развития механизмов КИП, рекомендуется перейти к п. 3. Как уже отмечалось ранее, в последние годы проблеме КИП в программном обеспечении уделяется значительное внимание. Общее определение и классификация информационных потоков в программном обеспечении приводятся в [8]. Современные исследователи различают четыре аспекта проблемы: описание политики безопасности – языковая часть механизма КИП, условия безопасности вычислений (безопасность семантики), механизм проверки условий безопасности и доказательство его корректности, реализация. Полноценное внедрение КИП в информационную систему предполагает охват всех четырех аспектов.

В таблице 2 приведены наиболее значимые для развития теории КИП исследования, сделана попытка оценить их результаты применительно к перечисленным направлениям.

История механизмов КИП берет свое начало с первых формальных моделей безопасности информационных систем. Среди них особое место занимает модель Белла–Лападулы [9]. Авторы впервые предложили использовать для описания политики безопасности множество упорядоченных меток – уровней безопасности, соответствующих стандартной классификации данных по уровню конфиденциальности: “несекретно” ≤ “для служебного пользования” ≤ “секретно” ≤ “совершенно секретно”. Несмотря на то, что данная модель остается актуальной и в наше время, в работе не дано строгого определения условий безопасности.

В основу современных механизмов КИП положено понятие безопасности вычислений на основе информационного невлияния [7, 12]. В общем виде применительно к программным средам условие безопасности можно сформулировать

как отсутствие влияния элементов среды вычислений (переменных, входных-выходных потоков, исключений и т.д.) с заданным уровнем конфиденциальности (если используется решетка уровней конфиденциальности) на элементы с более низким уровнем конфиденциальности.

Следующей вехой развития теории КИП стали работы Д. Деннинг [10, 13]. Основными достижениями считаются: обобщение упорядоченного множества меток модели Белла–Лападулы до алгебраических решеток, предложения по реализации механизма проверки условий безопасности программ на основе статического анализа, ввод понятия неявного информационного потока (и контекста безопасности), который возникает в операторах ветвления и циклов при использовании конфиденциальных данных в условиях:

1 if high = 1 then low = 0; else low = 1; end if;

Дать определение безопасности вычислений на основе информационного невлияния и доказать корректность механизма проверки (в смысле информационного невлияния), предложенного Д. Деннинг, спустя почти двадцать лет удалось Волпано и др. [11]. Они создали первый прототип системы безопасных типов для простого императивного языка, объединив, таким образом, первые три аспекта КИП.

Первой реализацией КИП на уровне реального языка программирования стал проект *Flow Caml* [5] – безопасное расширение функционального языка *ML*. Авторами был реализован механизм контроля информационных потоков на основе строгой схемы информационного невлияния. Данный факт стал бесспорным преимуществом проекта и одновременно причиной его ограниченного распространения – проект получил популярность лишь в академической среде и стал отправной точкой для дальнейших исследований.

Можно сказать, что проект *FlowCaml* завершил начальный этап становления теории КИП. В дальнейшем исследования продолжались по всем четырем направлениям. Несколько более подробная информация по ним приводится далее. Среди наиболее значимых проектов, доведенных до практической реализации в промышленных языках программирования, выделяются *JIF* [3] и *Paragon* [4].

Описание политики. По сути, речь идет о языковой части механизма КИП. Грамматика языка описания политики может быть основана на таких понятиях, как: субъект, объект, право доступа, роль, уровень доступа, метка безопасности и т.д. Политика безопасности информационных потоков, реализуемая на уровне специального программного обеспечения, должна строго соответствовать более общей политике управления доступом, реализуемой на системном уровне. С учетом общепринятого подхода к управлению информационными потоками на основе алгебраических решеток наиболее гладко трансформация требований общей политики в метки элементов среды вычислений может осуществляться в системах с мандатным управлением доступом.

В значимой части известных механизмов КИП за основу принимается решетка уровней конфиденциальности данных [10] – $(SC, \sqsubseteq)$, где SC – конечное множество уровней конфиденциальности, $\sqsubseteq$ – отношение частичного порядка. В научных трудах для упрощения системы доказательств чаще всего принимается двухуровневая решетка: $\{H, L\}$, где $L \sqsubseteq H$.

Еще одним известным способом описания политики безопасности информационных потоков является децентрализованная модель на основе меток (ДММ) [3]. В основу положены двухкомпонентные метки, определяющие владельцев и "читателей" информации. Например, метка $\{A : B, C\}$ будет означать, что A является владельцем данных, а B и C предоставлено право чтения. В общем случае, метка переменной (или элемента среды вычислений иного типа) может включать несколько владельцев, каждый из которых может иметь свой список "читателей". В [3] задано отношение частичного порядка на множестве меток, минимальный и максимальный элементы, таким образом, данное множество также образует решетку, и для систем, использующих ДММ, применим подход [10]. Можно сказать, ДММ соответствует дискреционному принципу управления доступом.

В последнее время распространение получил принципиально новый перспективный язык *Flow Locks* [4, 14, 15]. Математической основой выражений языка являются конъюнкции определенных дизъюнктов Хорна вида: $\{L_1 \Rightarrow A_1, \ldots L_n \Rightarrow A_n\}$, где L_n – множество блокировок, которые должны быть открыты для того, чтобы данные могли быть прочитаны соответствующим актером (пользователем). В отличие от статических политик безопасности, политики, описываемые *Flow Locks*, могут быть динамическими, то есть зависеть от состояния некоторых элементов (блокировок) среды вычислений выражений. Например, политика $P \triangleq \{High, TExpired \Rightarrow Low\}$ означает, что соответствующее значение может быть всегда прочитано актером $High$, и актером Low – при условии открытой блокировки $TExpired$.

Расширение этого языка *Paralocks* позволяет описывать параметрические блокировки, обладает высокой выразительной способностью и позволяет, например, создавать политики управления информационными потоками на основе логических ролей приложений, в том числе, в предположении о том, что пользователи могут переключать роли в текущих сеансах.

Безопасность семантики. Отвечает на вопрос, какая система считается безопасной или описывает условия безопасности. Важно отметить, что понятие безопасности вычислений не должно определять алгоритм или какой-либо порядок проверки свойств системы. Например, для дискреционных моделей управления доступом часто принимается, что система является безопасной, если в ней отсутствует возможность получения субъектом нового права доступа к какому-либо объекту без участия владельца. Безусловно, это требование является недостаточно строгим, поскольку отсутствие доступа не исключает наличие запрещенных информационных потоков между соответствующими сущностями. Обязательное требование мандатного разграничения доступа в критичных системах во многом продиктовано стремлением исключить нисходящие информационные потоки между сущностями с разными уровнями доступа (уровнями конфиденциальности).

Как отмечалось ранее, в основу формального определения безопасности систем с контролем информационных потоков положено понятие информационного невлияния. Введем обозначения и определения по аналогии с [16]. Пусть C – множество выражений некоторого языка, $\Lambda \triangleq \Upsilon \times \Gamma \times \Theta$ – множество состояний среды вычислений, где Υ – множество состояний среды переменных, Γ – множество состояний среды входных потоков, Θ – множество состояний среды выходных потоков, $\langle c, S\rangle$ – контекст вычисления, $\langle c, S\rangle \downarrow o$ – отношение вычисления, которое обозначает, что при вычислении выражения c в состоянии $S \in \Lambda$ наблюдается поведение o. Под наблюдаемым поведением шага вычислений следует понимать значение, которое становится доступно внешнему обозревателю после его завершения (значение, сохраненное в переменную или выходной поток, временная задержка, информация об исключении и т.д.). Для простоты воспользуемся двухуровневой решеткой уровней конфиденциальности и будем полагать, что состояния S_1 и S_2 среды вычислений находятся в отношении низкой эквивалентности: $S_1 \approx S_2$, если значения их низкоуровневых элементов совпадают. Тогда:

Определение 2.1. *Программа P* удовлетворяет свойству информационного невлияния ИН(P), если для любых двух состояний S_1 и S_2 среды вычислений, состоящих в отношении низкой эквивалентности, успешное выполнение программы в первом (начальном) состоянии с наблюдаемым поведением o_1 гарантирует успешное выполнение программы во втором (начальном) состоянии с наблюдаемым поведением o_2, при этом o_1 и o_2 являются неразличимыми, то есть:

$$\text{ИН}(P) \triangleq \forall S_1, S_2, o_1, o_2 : S_1 \approx S_2 \wedge$$

$$\wedge \langle P, S_1\rangle \downarrow o_1 \wedge \langle P, S_2\rangle \downarrow o_2 \Rightarrow o_1 \approx o_2, \qquad (2.1)$$

где: $\approx$ – *отношение неразличимости поведений*.

В строгом смысле свойство информационного невлияния требует низкой эквивалентности итоговых состояний, то есть поведения считаются неразличимыми, если в итоговых состояниях S_1', S_2' полностью совпадают значения всех низкоуровневых элементов, включая значения соответствующих переменных. На практике такие ограничения считаются слишком строгими, кроме того, иногда допускается некоторое влияние высокоуровневых элементов на низкоуровневые. Например, приложение может использовать статистическую функцию *AVG* для расчета среднего значения стоимости контракта. Данное значение считается открытым, в то время как информация об отдельном контракте – конфиденциальной.

Для ослабления требования информационного невлияния введено понятие деклассификации данных (в случае контроля целостности – очистки данных). Под деклассификацией понимается контролируемое раскрытие информации с ограничениями по месту (*where*), времени (*when*), инициатору (*who*) и, конечно, содержимому (*what*). Требование информационного невлияния может быть еще больше ослаблено в зависимости от возможностей нарушителя, например, способности эксплуатировать скрытые информационные каналы, а также характера вычислений: пакетное выполнение команд, интерактивные вычисления. В последнем случае наблюдаемые значения могут генерироваться на каждом шаге вычислений. Двигаясь в сторону адаптации строгих требований безопасности к требованиям, реализуемым на практике – "допустимой некорректности", исследователи разработали дополнительные схемы информационного невлияния. Далее указаны некоторые из них в нисходящем порядке, от более строгих к менее строгим: информационное невлияние ИН, количественное информационное невлияние КИН, информационное невлияние с учетом деклассификации ИНД, терминальное-зависимое информационное невлияние ТЗИН, терминально-независимое информационное невлияние ТНЗИН, прогресс-зависимое информационное невлияние ПЗИН, прогресс-независимое информационное невлияние ПНЗИН и т.д. Подробное описание приведенных схем представлено в [16]. Отдельную ветвь составляют исследования, посвященные безопасности многопоточных программ и параллельных вычислений в целом. В работах [17–19] представлены схемы информационного невлияния, зависимые от вероятностных каналов и каналов по времени. Наибольший интерес представляет схе-

ма на основе принципа вероятностной бисимуляции [17].

В заключение хотелось бы отметить принципиально новый подход к определению безопасности семантики [4, 20]. Предпосылками к его появлению стали попытки адаптировать стандартное требование информационного невлияния к условиям, в которых допускается деклассификация данных. Предложенные модифицированные схемы ИН (для деклассификации) не отличались интуитивной простотой. Новый подход построен на основе эволюции знаний нарушителя, он хорошо подходит для анализа безопасности информационных потоков на основе динамических политик, то есть политик, требования которых могут изменяться на этапе выполнения (например, в случае успешной проверки условий деклассификации). Под знаниями нарушителя на шаге n выполнения программы P с соответствующей трассой наблюдаемых поведений $\vec{o}$, будем понимать множество начальных состояний среды вычислений, при которых выполнение P приводит к генерации $\vec{o}$. Дадим формальное определение безопасности программы на основе понятия статичности знаний нарушителя аналогично [21].

Определение 2.2. *Программа P удовлетворяет свойству статичности знаний нарушителя* СЗН(P), *если во время выполнения на нормальных шагах (не относящихся к деклассификации) знания нарушителя не меняются, то есть:*

$$\text{СЗН}(P) \triangleq \forall S, A, \vec{C}c' : K_A(S, P, \vec{C}c') = \\ = K_A(S, P, \vec{C}), \tag{2.2}$$

где: $K_A(S, P, \vec{C})$ *— знания нарушителя A после прохождения трассы $\vec{C}$ программы P из начального состояния S, c' — выражение, вычисляемое вне контекста деклассификации.*

Механизм проверки. Существует большое количество способов организации скрытых информационных каналов в программной среде. К таковым, например, относится измерение потребляемых ресурсов и времени выполнения. Эксплуатация таких каналов на практике сложна и маловероятна, часто они просто игнорируются. Проверка безопасности стандартных информационных потоков (явных и неявных), а также потоков, возникающих вследствие наблюдения за прогрессом вычислений, традиционно реализуется средствами статического и динамического анализа программ. В настоящее время превалируют статические методы анализа на основе безопасных систем типов. Как уже отмечалось ранее, катализатором этого направления послужили исследования Д. Деннинг [10, 13] и Д. Волпано [11]. Основным преимуществом статического анализа, очевидно, является полнота покрытия кода (при динамическом анализе проверке подвергаются все же отдельные траектории), а также отсутствие необходимости реализовывать дополнительные проверки на этапе выполнения программы. К преимуществам динамического анализа относят, главным образом, точность, что особенно важно для платформ, поддерживающих динамическую загрузку (генерацию) кода, таких как *JavaScript*. При этом, как доказано в [22], статический и динамический анализ позволяют в равной степени гарантировать безопасность программы в смысле ПНЗИН. Попытки объединить преимущества обоих подходов привели к появлению гибридных механизмов КИП [23]. Полный сравнительный анализ обозначенных подходов приведен в [16].

Значимая часть исследований посвящена внедрению механизмов контроля на уровне расширенных безопасных систем типов. Анализ на основе более полных программных моделей, например межпроцедурных графов потоков управления, является, безусловно, более точным, но и более сложным с точки зрения практической реализации (требуется построение самих моделей, отсутствует возможность композиционного анализа). Тем не менее, отдельные исследования направлены на сращивание и этих подходов [24].

Не угасает интерес к анализу информационных потоков при параллельных вычислениях. В проекте *Veronica* [25] предложена, по утверждению авторов, новая композиционная логика проверки, поддерживающая широкий спектр политик безопасности.

Методы формальной верификации несмотря на общий рост популярности для решения проблемы КИП в программном обеспечении применяются редко. Объясняется это достаточно просто. Проверка условий безопасности программного обеспечения должна быть комплексной, то есть применяться ко всем модулям системы. Значительный объем кода современных приложений не позволяет даже с учетом абстрагирования добиться приемлемого числа состояний для проверки инвариантных свойств программы, описанной в виде спецификации, с использованием инструментов проигрывания моделей типа *TLC*. Получение формальных доказательств является трудоемким процессом, и на практике может применяться лишь для выделенных компонент. Среди отдельных работ, где формальная верификация применяется для проверки условий безопасности информационных потоков, следует отметить: [26, 27]. Данные исследования, однако, посвящены анализу бизнес-процессов. В некоторых случаях формальная верификация используется для доказательства корректности современных моделей управления доступом в ОС и СУБД [28].

```
public void play{P1<-* meet P2<-*}(PrintStream[{}]
    {P1<-* meet P2<-*} output) throws
        (SecurityException,
         IllegalArgumentException{P1<-* meet P2<-*})
         where authority (P1, P2)
```

Рис. 5. Пример описания политики безопасности в JIF.

```
public !bottom void play
        (?bottom PrintStream<bottom> output)
         throws !bottom IllegalArgumentException
```

Рис. 6. Пример описания политики безопасности в Paragon.

Реализация. Наиболее значимые проекты были обозначены ранее. Отметим также, что в целом число инструментов и платформ, предназначенных для решения задачи КИП, достаточно велико, известны решения для проверки web и мобильных приложений [29–32] платформы для анализа безопасности информационных потоков в системах, построенных на основе баз данных [33–35]. К сожалению, ни один проект не получил широкого распространения. Отказ от строгих схем информационного невлияния, внедрение КИП в самые популярные языковые платформы не оправдали ожиданий. Примечательным является исследование [36], в котором автор делает попытку оценить возможность использования существующих решений для разработки полноценных безопасных Java-приложений. В работе были рассмотрены проекты *JIF* и *Paragon*. Сложность описания политик безопасности и необходимость значительного инструментирования кода привели к тому, что работа, по сути, была сведена к анализу известных демонстрационных примеров[1].

Причиной сложившейся ситуации видится следующее. Все реализации, несмотря на возможность описания гибких политик безопасности и проверки их требований, предполагают существенное изменение языковой грамматики в отсутствии расширения функциональных возможностей (см. рис. 5 и 6).

Безопасность всегда оставалась вторичной по отношению к функционалу, и ситуация не будет меняться. Вряд ли стоит ожидать широкого распространения безопасных языков программирования. Стоит также отметить, что за последние 25 лет (с момента публикации [11]) мир существенно изменился: расширились сферы применения языков программирования, значительно увеличились объемы кода в типовых проектах, наблюдается тенденция к упрощению языковых грамматик, делается акцент в сторону повышения производительности написания кода (rapid application development) и сокращению времени внедрения новых функций в существующее программное обеспечение. Активно развивается архитектура микросервисов и модульная разработка, происходит постепенный отказ от монолитного принципа написания программ. Все перечисленное плохо сочетается с трудоемкими процедурами КИП, лежащими в основе “языковой” безопасности (language based security). Решение видится в разделении функций разработки и анализа безопасности информационных потоков.

Предположение 2.1. *Смещение механизма проверки условий безопасности в область формальной верификации позволит решить проблему при выполнении следующих требований к реализации:*

– автоматическая генерация спецификаций с приемлемым уровнем абстракции (наличие утилиты типа C2TLA [37]);

– удобный интерфейс аналитика для описания политик безопасности элементов среды вычислений;

– приемлемые временные затраты на проверку инвариантных свойств безопасности (надежности);

– наглядное представление результатов анализа.

Вынося механизм проверки за пределы языковой среды, мы, в определенном смысле, теряем гарантии безопасности, поскольку, соответствие программы заданным свойствам, в конечном счете, определяется соответствием программы проверенной модели (спецификации). Тем не менее, с учетом описанных выше тенденций развития индустрии программного обеспечения, частичный отказ от таких гарантий в пользу декомпози-

[1] В основе обоих проектов лежат безопасные системы типов, по этой причине разметке подвергаются не только входные и выходные значения, но и заголовки методов. Для них могут задаваться метки формальных параметров, эффектов записи (writing effects), выходных значений, генерируемых исключений. Описание политики для систем, основанных на использовании межпроцедурного ГПУ, например Joana [2], выглядит несколько проще.

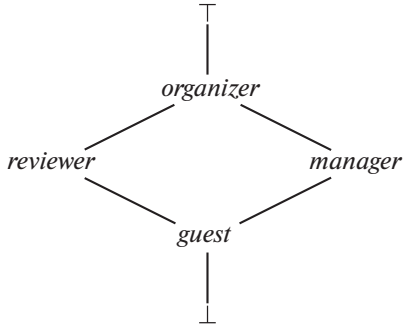


Рис. 7. Решетка ролей.

ции функций вовлеченных в процесс разработки специалистов и упрощения работы программистов (от них требуется лишь внесение конкретных проверок, сформированных по результатам тестирования спецификации) видится обоснованным. Основные положения предлагаемого подхода представлены в п. 4.

3. МОТИВИРУЮЩИЙ ПРИМЕР

Для удобства воспользуемся известным примером описания системы управления конференциями. Она позволяет участникам заявлять доклады, организаторам – осуществлять их рецензирование и распределение по секциям.

Глобальная политика безопасности, заданная в терминах ролевой модели управления доступом (рис. 7), диктует ряд ограничений, например: рецензент имеет доступ по чтению к содержимому статей, но не может ассоциировать работы с конкретными авторами. Статус доклада (одобрен или нет) доступен по чтению менеджеру, который занимается составлением программы конференции и распределением докладов по секциям, и другим пользователям по истечении определенного временного интервала. Программа конференции доступна по чтению всем пользователям. Схема данных состоит из следующих отношений: *Papers* – доклады, *Submissions* – заявки на участие с докладом, *Conferences* – конференции, *Sections* – секции, *Allocations* – распределение докладов по секциям, *Logs* – журнал ошибок.

Функции системы, связанные с обработкой данных, реализуются посредством хранимых процедур и функций БД, полный состав которых можно найти в [38]. Ключевыми для нас хранимыми программными блоками являются: *p_submit_paper* – добавляет строку в таблицу *Submissions* (регистрирует заявку на участие с докладом), *p_chahge_status* – обновляет значение поля *Status* для заданной строки в таблице *Submissions* после рецензирования соответствующего доклада, *p_allocate* – добавляет сведения о докладе в таблицу *Allocations* после осуществления ряда проверок, *f_getsection_program* – возвращает программу заданной секции, для формирования отчета обращается к таблицам: *Papers* и *Allocations*.

Назначение привилегий в соответствии с заданными ограничениями политики безопасности может быть представлено как:

```
1 grant execute on p_submit_paper to guest;
2 grant execute on p_chahge_status to reviewer;
3 grant execute on p_allocate to manager;
4 grant execute on f_getsection_program to guest;
```

Требование политики по предоставлению доступа к атрибуту *Status* доклада (менеджеры или **иные пользователи по истечению тайм-аута**) является примером использования процедуры деклассификации данных. Корректность реализации этой процедуры определяется, в том числе, и “доверенным” сервисом, который формирует данные о программах конференций. На рис. 8 показан пример нарушения заданного требования (правый фрагмент кода) через неявный информационный поток, и пример правильной проверки условий вставки данных в таблицу *Allocations* (статус 2 указывает на доклады приглашенных специалистов, их рецензирование не требуется).

Отсутствие утечек информации о принадлежности докладов конкретным авторам на этапе рецензирования также зависит от свойств соответствующих программных модулей, и может не гарантироваться системой управления доступом уровня БД.

4. КОНТРОЛЬ ИНФОРМАЦИОННЫХ ПОТОКОВ В ПРОГРАММНЫХ БЛОКАХ БД

Выбор **программной среды** для внедрения механизма КИП – хранимые программные блоки БД, объясняется следующими особенностями: близость к данным, малый объем кода – в среднем размер хранимой процедуры (функции) не превышает 60 строк, отсутствие вспомогательного кода – PL/SQL код фактически не реализует вспомогательные функции (сфокусирован на основном алгоритме), наличие развитых средств анализа зависимостей – позволяет эффективно выделять релевантное относительно чувствительной информации подмножество хранимых программных блоков, сериализация транзакций с использованием различных стратегий управления блокировками – позволяет сократить количество состояний модели, описывающей параллельные вычисления.

Для **описания политик безопасности** в исследовании используется язык *Paralocks*. Он обладает простой математической интерпретацией – основан на выражениях логики предикатов первого порядка, отличается высокой гибкостью и выразительной способностью – позволяет описывать правила мандатной, ролевой моделей управления

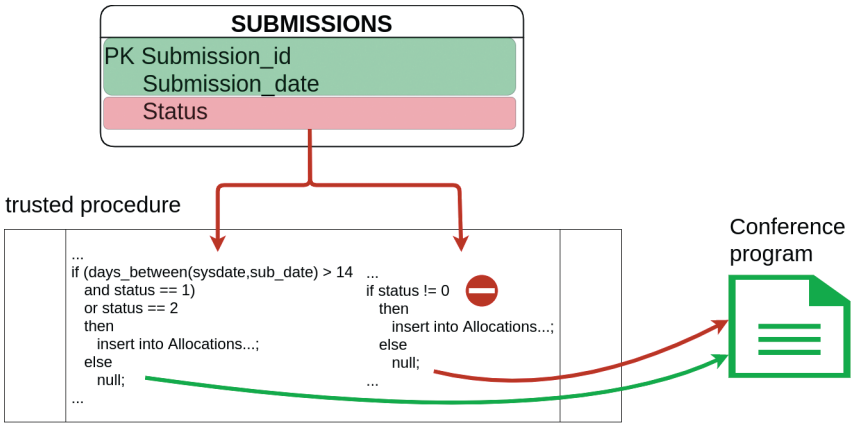


Рис. 8. Пример нарушения политики безопасности.

доступом, децентрализованной модели на основе меток, позволяет задавать правила деклассификации (очистки) данных – ослабляющие условия являются частью самих выражений политики безопасности. Спецификация *Paralocks* [38] содержит *TLA+* представление примитивов, необходимых для определения политики. В данной работе, однако, воспользуемся упрощенной решеткой меток конфиденциальности (рис. 9) (для описания дуальной политики целостности может использоваться аналогичная решетка). Введем следующие обозначения ролей, которые могут быть заданы в сеансе пользователя: g – *guest*, r – *reviewer*, m – *manager*, o – *organizer*. Метка конфиденциальности определяется как множество вида: $L_k = \{R_1, \ldots R_n\}$, здесь $R_1, \ldots R_n$ – условия предоставления доступа. Для чтения данных необходимо выполнение одного из условий. С учетом требования преемственности политики базовой ролевой модели положим:

$$\forall R_n, R_m, L_k : R_n \le R_m \wedge \\ \wedge\ R_n \in L_k \Rightarrow R_m \in L_k \qquad (4.1)$$

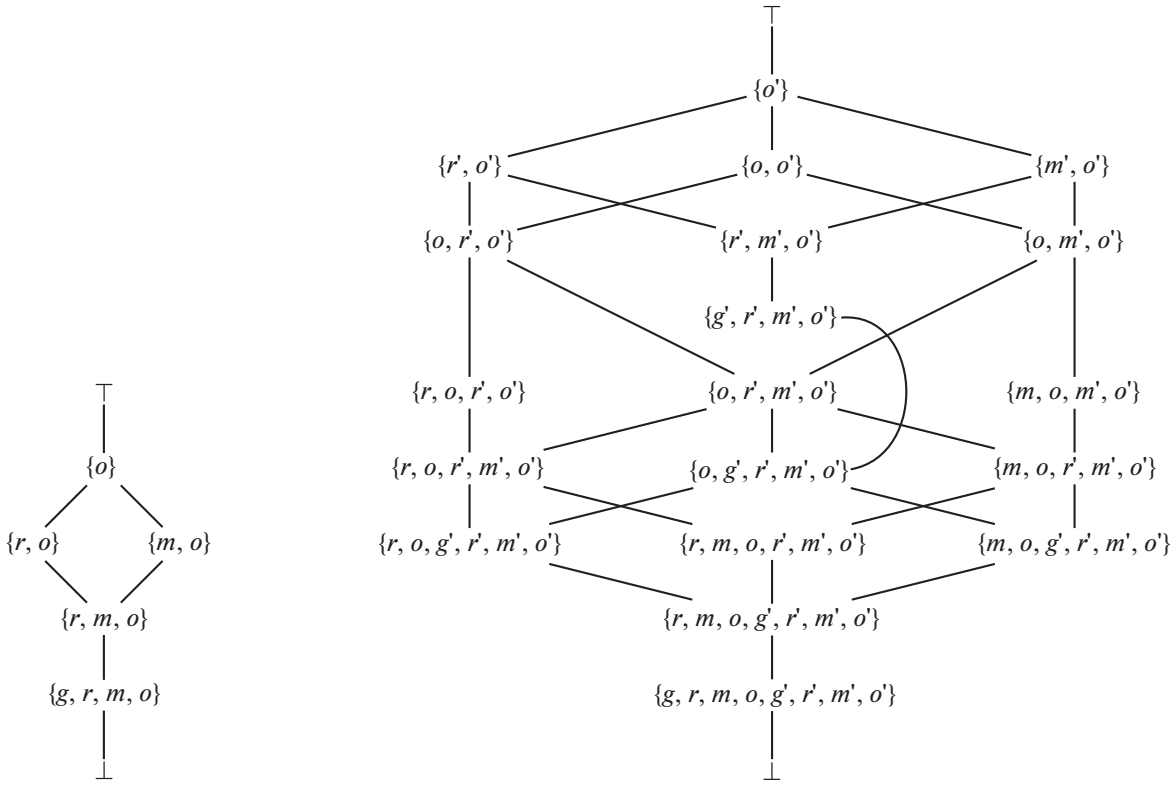
Тогда метка данных, доступ на чтение к которым предоставлен g, будет иметь вид: $\{g, r, m, o\}$. На рис. 9а представлена решетка меток конфиденциальности на основе иерархии ролей (рис. 7) без учета дополнительных ограничений. Отношение частичного порядка $\sqsubseteq$ может быть задано как: $L_1 \sqsubseteq L_2$ тогда и только тогда, когда для любого $R_n \in L_2$ имеем: $R_n \in L_1$. Оператор пересечения (нахождения наибольшей нижней грани) определяется как объединение условий (ролей) соответствующих меток конфиденциальности: $L_1 \sqcap L_2 = L_1 \cup L_2$, оператор соединения (нахождения наименьшей верхней грани) – как пересечение условий: $L_1 \sqcup L_2 = L_1 \cap L_2$.

В случае, если в политике используется дополнительное ограничение, например “time expired”, решетка меток конфиденциальности задается несколько сложнее. Пусть R' означает, что чтение возможно, если в текущем сеансе установлена роль R и выполняется условие “time expired”. Будем также полагать, что на множестве $\{R'_1, \ldots R'_n\}$ сохраняется отношение частичного порядка, то есть: $R_1 \le R_2 \Rightarrow R'_1 \le R'_2$, и условие R' всегда является более строгим, чем R, то есть, если для чтения данных достаточно R, то достаточно и R' или:

$$\forall R_n, L_k : R_n \in L_k \Rightarrow R'_n \in L_k \qquad (4.2)$$

С учетом выражений 4.1 и 4.2 метка данных, доступ на чтение к которым предоставлен менеджерам – m или обычным пользователям при истечении заданного интервала времени – g', будет иметь вид: $\{g', r', m', o', m, o\}$. Отношение частичного порядка, операторы пересечения и соединения задаются также, как и для решетки на рис. 9а. Графически решетка меток конфиденциальности на основе иерархии ролей (рис. 7) и дополнительного ограничения “time expired” показана на рис. 9б.

Безопасность семантики в данной работе рассматривается в контексте прогресс-зависимого информационного невлияния. В отличие от стандартной схемы (Определение 2.1), в данном случае проверка условий безопасности информационных потоков выполняется после каждого шага вычислений, при этом сами условия ослабляются (контролируется только вывод данных в выходные потоки). Описанная далее инструментированная абстрактная семантика безопасных информационных потоков (далее просто – абстрактная семантика) положена в основу механизма генерации спецификаций программных блоков и предполагает монотонное изменение меток безопасности внутренних глобальных (далее – глобальных) переменных среды (таблиц) на этапе выполнения. Под внутренними будем понимать объекты, недоступные для прямого просмотра. Пусть $M_n : X \to \{L_1, \ldots L_k\}$ – отображе-



(а) на основе ролей (б) на основе ролей и ограничения – “time expired”

Рис. 9. Решетка меток безопасности.

ние множества переменных на множество меток безопасности на шаге вычислений n.

Определение 4.1. *Программа P удовлетворяет свойству прогресс-зависимого информационного невлияния для начального и конечного отображений множества переменных на множество меток безопасности M_1 и M_2 – ПЗИН$(P)_{M_1,M_2}$, если для любых двух состояний S_1 и S_2 среды вычислений, состоящих в отношении низкой эквивалентности относительно некоторого уровня конфиденциальности L при отображении M_1*: а) *каждый шаг вычислений сопровождается генерацией одинаковых наблюдаемых значений относительно уровня L или приводит к “расхождению” для обоих состояний*”; б) *соответствующие финальные состояния – S_1' и S_2' находятся в отношении низкой эквивалентности относительно уровня L при отображении M_2*:

$$\text{ИЗИН}(P)_{M_1,M_2} \stackrel{\Delta}{=} \forall L, S_1, S_2, o_1, o_2 : S_1 \approx_{M_1,L} S_2 \wedge$$
$$\wedge P(S_1) \downarrow \langle S_1', o_1\rangle \wedge P(S_2) \downarrow \langle S_2', o_2\rangle \Rightarrow \qquad (4.3)$$
$$\Rightarrow o_1 \approx_L o_2 \wedge S_1' \approx_{M_2,L} S_2'$$

Условие б) в данном определении позволяет применять композиционный подход при проверке заданных свойств программного обеспечения [8]. Это важно, поскольку в общем случае сеанс взаимодействия пользовательского приложения с БД неограничен по количеству вызываемых процедур (функций), то есть модель параллельных вычислений, соответствующая такому взаимодействию, имеет бесконечное число состояний. В текущей версии проекта с помощью model checker проверяется основной инвариант безопасности (соответствует условию а) для системы, имитирующей однократное выполнение программного блока в каждом сеансе. С помощью отдельного предиката на последнем шаге каждой возможной траектории сеанса проверяется соответствие начальных и конечных меток безопасности всех глобальных переменных. В случае расхождения, требуется повторный прогон модели, при котором в начальном состоянии для глобальных переменных задаются метки безопасности последнего шага предыдущего прогона. Алгоритм является конечным, поскольку решетка меток безопасности имеет фиксированный размер, и все правила абстрактной инструментированной семантики являются монотонными в отношении глобальных переменных.

Учет скрытых вероятностных информационных каналов и каналов по времени потребует дополнительных ограничений, например, запрета

синхронизации для программных блоков, в которых возникают явные или неявные информационные потоки, связанные с чувствительными данными. Учитывая сложность эксплуатации таких каналов на практике, особенно в условиях многопользовательского, удаленного режима работы, на данном этапе они игнорируются.

Как уже отмечалось в п. 2, **проверку условий безопасности информационных потоков** в специальном программном обеспечении предложено вынести в область формальной верификации. В качестве инструмента создания спецификаций предлагается использовать язык *TLA+* [39]. Основными преимуществами, предопределившими его выбор среди множества альтернатив, являются: открытость, хорошая документированность и сопровождение, удобство интегрированной среды разработки *TLA Toolbox*, улучшенная поддержка свойств “живучести” (liveness), позволяющих более точно описывать поведение системы. Для непосредственной проверки свойств используются инструменты типа model checker: *TLC*, *Apalache* [40].

Пользовательские сеансы взаимодействия с БД посредством хранимых программных блоков описываются как параллельные процессы. Абстрактную семантику последовательных вычислений в рамках выделенного сеанса можно описать системой переходов состояний вида: $\langle PC, c, M \rangle$, где: PC – метка безопасности счетчика команд (program counter) в текущем сеансе; c – исполняемая команда текущего сеанса; M – абстрактное состояние среды, которое задает отображение локальных и глобальных переменных, входных и выходных потоков на соответствующие им метки безопасности. Пусть $D = \{d_1, \ldots d_n\}$ – множество активных сеансов, $O : D \times C$ – функция отображения множества сеансов на множество команд. Тогда глобальная семантика, описывающая переходы между параллельными процессами (сеансами), задается правилами GLOBAL RULES (Приложение 2).

С учетом выбранной позиции относительно скрытых вероятностных информационных каналов и каналов по времени, использование равномерного планировщика выглядит обоснованным.

Состояние системы, представленной как набор спецификаций [38], описывается более сложной структурой, включающей, в том числе, состояние памяти (значения локальных и глобальных переменных, параметров и т.д.), множество эксклюзивных табличных блокировок, множество открытых блокировок политики *Paralocks*, трассу вычислений и другие сущности. Кроме того, в проекте за основу принята расширенная схема ПЗИН, допускающая деклассификацию данных.

Опишем подробнее ключевые правила абстрактной семантики (малого шага) последовательных вычислений. Для простоты воспользуемся подмножеством языка *PL/SQL*, BNF-грамматика которого представлена в табл. 3.

Представленное подмножество PL/SQL не включает динамических запросов и команд, параметрических циклов и циклов с постусловием, а также некоторых иных конструкций. Тем не менее, полагаем, большая часть возможностей стандартного PL/SQL остается доступной. Исчерпывающие правила абстрактной семантики информационных потоков приведены в Приложении 2. Условия *inv* соответствуют инварианту безопасности.

Метка арифметического выражения (E-OPER) вычисляется как минимальная верхняя грань меток составляющих его операндов, это же утверждение справедливо для условных выражений (E-C-OPER) и выражений в списках SELECT курсоров (E-L-OPER). Метка записи (E-VAR-REC) определяется метками соответствующих полей. Метку таблицы было бы логично представить как минимальную верхнюю грань меток ее строк и значения, определяющего ее текущий размер, по аналогии с коллекциями (E-VAR-ARR), однако, с целью упростить *TLA+* спецификации, описывающие информационные потоки в программной среде БД, поднимем уровень абстракции при вычислении меток до столбцов (E-TAB-COL). Правила (C-INS), (C-UPD), (C-DEL) гарантируют корректность изменения этих меток. Значения меток столбцов монотонно возрастают в соответствии с метками изменяемых данных, условиями и контекстом вызова DML-операторов. При вычислении меток курсоров (E-CUR) учитываются метки условных выражений во фразе WHERE. Атрибуты курсоров (E-CUR-FND) наследуют метки курсоров. Заданная языковая грамматика допускает вызов сторонних процедур и функций (далее – просто сторонних функций), распространение информации в которых не контролируется. Будем различать доверенные и чистые функции. Полагаем, что при вызове доверенных функций $x_f(x_1 \to e_1, \ldots x_m \to e_m)$ запрещенных сторонних эффектов (side effects) не возникает, метка возвращаемого значения задается явно аналитиком (E-TR-LIB). Условие *inv* ограничивает метки фактических параметров и проверяет соответствие метки счетчика команд метке эффекта записи p_w (writing effect) вызываемой функции. Если “разметка” вызываемой сторонней функции не произведена, полагаем, что формальным параметрам и эффекту записи присваивается минимальная метка $\perp$, возвращаемому значению – максимальная метка $\top$. Под чистыми понимаются функции, в которых отсутствует эффект записи, то есть данные не сохраняются в

Таблица 3. Грамматика подмножества *PL/SQL*

(*values*)	$\nu ::=$	$n \mid b$
(*declarations*)	$d ::=$	x **number** $\mid x_1\ x_2 \mid d_1; d_n \mid$
		type x_r **is record** $(x_{f_1}\ x_1, \ldots x_{f_n}\ x_n) \mid$
		type x_t **is table of** $x \mid$ **exception** $x \mid$
		cursor x **is select** $e_1, \ldots e_n$
		from $x_{t_1}, \ldots x_{t_n}$ **where** $cnd \mid$
		procedure $x_p(x_1, \ldots x_n)$ **as** d
		begin c_1 **[exception]**c_2 **end**;$\mid$
		function $x_{fn}(x_1, \ldots x_n)$ **return** x_{rt} **as** d
		begin c_1**[exception** c_2**] end**;
(*expressions*)	$e ::=$	$\nu \mid x \mid x[e] \mid x_1.x_2 \mid e_1 \odot e_2 \mid x(e_1, \ldots e_2) \mid$
		$x\%$ **found** $\mid$
(*conditions*)	$cnd ::=$	$e_1 * e_2 \mid cnd_1 \star cnd_2$
(*statements*)	$c ::=$	$x := e \mid c_1; c_2 \mid x_f(x_1 \to e_1, \ldots x_n \to e_n) \mid$
		if e **then** c_1 **else** $c_2 \mid$ **while** e **do** $c \mid$
		endif $\mid$ **endwhile** $\mid c_1 \vee c_2 \mid$
		select $e_1, \ldots e_n$ **into** $x_1, \ldots x_n$ **from** $x_{t_1}, \ldots x_{t_n}$
		where $cnd \mid$
		insert into $x_t(x_1, \ldots x_n)$ $(e_1, \ldots e_n) \mid$
		update x_t **set** $x_1 = e_1, \ldots x_n = e_n$ **where** $cnd \mid$
		delete from x_t **where** $cnd \mid$
		open $x_{cur} \mid$ **fetch** x_{cur} **into** $x \mid$ **close** $x_{cur} \mid$
		throw $x_{exc} \mid$ **when** x_{exc} **then** $c \mid$ **endexc** $\mid$
		commit $\mid$ **rollback** $\mid$ **null** $\mid$ **return**$(x_{out} \to e) \mid$
(*program*)	$p ::=$	**declare** d **begin** c_1 **[exception** c_2**] end**;

глобальные переменные и не выводятся в выходные потоки, возвращаемое значение полностью зависит от фактических параметров (E-PUR-LIB). Как уже отмечалось выше, метки внутренних переменных могут изменяться на этапе выполнения. Правила, в названиях которых присутствует “EXT”, применяются в тех случаях, когда метка изменяемого объекта является статической, например, если к таблице кому-либо предоставлен прямой доступ (не через программные блоки PL/SQL). Еще одним источником неточности процедуры формальной верификации могут выступать правила (C-IF), (C-WHILE), как показано, они не предполагают проверку истинности условий. Применение этих правил добавляет в метку счетчика команд, представленную кортежем вида $\langle pc_1, \ldots pc_n \rangle$, новый элемент, соответствующий метке условия. В [41] показан вариант реализации запрещенного вероятностного информационного потока в многопоточном приложении через глобальные переменные в условиях *while*. Ограничения вида $e - local$ легко проверяются в ходе ручного анализа кода и позволяют гарантировать отсутствие описанных информационных потоков. Через $\to_p$ в (C-OR) обозначается вероятностный переход. Появление недетерминизма в данном случае не вызывает опасений, поскольку верификация модели с использованием model checker предполагает прогон всех ее состояний, при этом вероятностные каналы в соответствии с принятой моделью угроз игнорируются. Правила (C-EXT-PRC) и (C-EXT-RET) применяются для процедур, привилегия на выполнение которых (execute) предоставлена не пустому множеству ролей. Для таких процедур статические метки формальных параметров и возвращаемого

значения, представленного через x_{out}, определяются как $\{r_1, \ldots, r_n\}$, где $r_1, \ldots, r_n$ – роли, обладающие привилегией execute. Правила для обработки исключений и выхода из блока обработки исключений: (C-EXC) и (C-END-EXC), задаются аналогично правилам для условного оператора и цикла while. Оставшиеся правила не требуют пояснений.

Докажем ряд утверждений, чтобы показать, что модель параллельных вычислений, прошедшая проверку с использованием описанной ниже процедуры, является безопасной в соответствии с Определением 4.1. Введем обозначения:

$\mathbb{M}$ – множество возможных абстрактных состояний среды вычислений, т.е. функций, задающих отображение множества элементов среды вычислений на множество меток безопасности;

$\mathbb{C}$ – множество возможных глобальных шагов вычислений вида $\langle c_1, \ldots c_n \rangle$ для n сеансов (глобальный шаг определяется текущими состояниями счетчиков команд в активных сеансах). Если k – общее количество хранимых программных блоков, то: $\mathbb{C} = \bigcup_{i=1}^{k^n} C_i^1 \times \ldots C_i^n$, где C_i^m (при $m \in [1, n]$) – множество возможных шагов вычислений конкретной процедуры, выполняемой в сеансе m при i-м размещении (с повторами) k процедур по n сеансам. Отметим, что ситуацию, когда в сеансе завершено выполнение одной процедуры, и еще не начато выполнение другой, можно описать состоянием, в котором этому сеансу ставится в соответствие так называемый load шаг очередной процедуры.

$\mathbb{O} = \mathbb{C} \times \mathbb{M}$ – множество глобальных состояний модели.

Введем отношение частичного порядка на множестве абстрактных состояний среды вычислений: $M_1 \le M_2 \Leftrightarrow Dom(M_1) = Dom(M_2)$ и $\forall x \in Dom(M_1) : M_1[x] \sqsubseteq M_2[x]$, где $Dom(M_n)$ – область определения функции M_n, x – переменная среды вычислений.

Предположение 4.1. *Заданные в виде предикатов состояний инварианты безопасности гарантируют неразличимость эффектов соответствующих команд и выражений в низкоэквивалентных контекстах* (*Условие а*). *Справедливо также и обратное утверждение. Например*, ПЗИН $(x := e)$ *является истинным для начального отображения M множества элементов среды вычислений на множество меток тогда и только тогда*, *когда истинным является инвариант правила* (*C-EXT-ASSIGN*).

Предположение является обоснованным, так как правила безопасных вычислений, взятые за основу в представленной абстрактной семантике, соответствуют известным общепринятым правилам для простого императивного языка [10], для которых, в свою очередь, в [11] была предложена классическая схема ИН.

Лемма 4.1. *Если* $M_1 \le M_2$, *то для* $\forall c_i$ *из* $\langle c_1, \ldots c_i \ldots c_n \rangle \in \mathbb{C}$, *такого*, *что предикат* ПЗИН (c_i) *является истинным при начальном отображении множества переменных на множество меток* M_2 *и конечном отображении* M_2' *будем иметь*, *что предикат* ПЗИН (c_i) *является также истинным для начального отображения* M_1 *и конечного* – M_1'.

Доказательство. Истинность условия б) обеспечивается потокочувствительностью семантики – значения меток безопасности переменных изменяются в процессе вычислений. Докажем истинность условия а). В соответствии с правилами абстрактной семантики внешними эффектами обладают следующие команды и выражения: (E-TR-LIB), (C-EXT-ASSIGN), (C-EXT-SEL), (C-EXT-INS), (C-EXT-UPD), (C-EXT-DEL), (C-EXT-PRC), (C-EXT-RET) и (C-EXT-FETCH). Проведем доказательство для правила (C-EXT-ASSIGN).

Если для вычисления выражения e используется правило (E-CONST), то e обладает меткой $\bot$. Из истинности ПЗИН (c_i) для начального отображения M_2 следует, что $p \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$ (по Предположению 4.1), откуда имеем: $pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$, тогда также справедливо и $\bot \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$, т.е. Условие а) справедливо для любого начального отображения, в том числе и M_1.

Если для вычисления выражения e используется правило (E-VAR), то из $M_1 \le M_2$ следует, что $p^1 \sqsubseteq p^2$, где $p^1 = M_1[x]$, $p^2 = M_2[x]$ (1). Тогда из истинности ПЗИН (c_i) для начального отображения M_2 следует, что $p^2 \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$ (2) (по Предположению 4.1). Из 1 и 2 имеем: $p^1 \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$, т.е. Условие а) справедливо для начального отображения M_1.

Теперь если последним правилом является (E-OPER), докажем по индукции на основе правил вычисления выражений, что если $\langle e, M_1 \rangle \Downarrow p^1$ и $\langle e, M_2 \rangle \Downarrow p^2$, то $p^1 \sqsubseteq p^2$ (1). Начальный шаг, когда e представляет собой константу или переменную, очевиден. По предположению индукции для e_1 и e_2 примем: $p_1^1 \sqsubseteq p_1^2$ и $p_2^1 \sqsubseteq p_2^2$, тогда $p_1^1 \sqcup p_2^1 \le p_1^2 \sqcup p_2^2$, то есть $p^1 \sqsubseteq p^2$. Индуктивный шаг доказан. Из $p^2 \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$ – следует из истинности ПЗИН (c_i) для начального отображения M_2, и 1 имеем: $p^1 \sqcup pc_1 \sqcup \ldots pc_n \sqsubseteq p_x$, то есть Условие а) справедливо для начального отображения M_1.

Для остальных правил доказательство выглядит аналогично.

Докажем утверждение, определяющие условие безопасной последовательной композиции вычислений в многопоточных системах.

Лемма 4.2. *Если для* c_i *из* $\langle c_1,\dots c_n\rangle_1 \in \mathbb{C}$, c_j *из* $\langle c_1,\dots c_n\rangle_2 \in \mathbb{C}$ *таких, что существуют переходы*: $\langle\langle c_1,\dots c_n\rangle_1, M_1\rangle \to^i_{1/n} \langle\langle c_1,\dots c_n\rangle_2, M_2\rangle$ *и* $\langle\langle c_1,\dots,c_n\rangle_2, M_2\rangle \to^j_{1/n} \langle\langle c_1,\dots,c_n\rangle_3, M_3\rangle$ *выполняется*: ПЗИН $(c_i)_{M_1,M_2}$ *и* ПЗИН $(c_j)_{M_2,M_3}$, *то выполняется также и* ПЗИН $(c_i;c_j)_{M_1,M_3}$.

Доказательство. Положим для некоторых состояний среды вычислений S_1 и S_2 справедливо: $S_1 \approx_{M_1,L} S_2$ (1), то есть состояния являются низкоэквивалентными относительно уровня конфиденциальности L для отображения M_1. Пусть $\langle\langle c_i;c_j\rangle, S_1\rangle \downarrow \langle S_1'', o_1\rangle$ – при последовательном выполнении двух шагов из начального состояния S_1 система переходит в состояние S_1'', переход сопровождается наблюдаемым поведением o_1 относительно уровня L, а также аналогично: $\langle\langle c_i;c_j\rangle, S_2\rangle \downarrow \langle S_2'', o_2\rangle$. Тогда существуют такие S_1', o_1' и o_1'', что $\langle c_i, S_1\rangle \downarrow \langle S_1', o_1'\rangle$ (2), $\langle c_j, S_1'\rangle \downarrow \langle S_1'', o_1''\rangle$ (3), $o_1 = \langle o_1', o_1''\rangle$ (4). Аналогично существуют такие S_2', o_2' и o_2'', что $\langle c_i, S_2\rangle \downarrow \langle S_2', o_2'\rangle$ (5), $\langle c_j, S_2'\rangle \downarrow \langle S_2'', o_2''\rangle$ (6), $o_2 = \langle o_2', o_2''\rangle$ (7). Из истинности ПЗИН $(c_i)_{M_1,M_2}$, 1, 2, 5 следует, что $S_1' \approx_{M_2,L} S_1'$ (8) и $o_1' \approx_L o_2'$ (9). Из истинности ПЗИН $(c_j)_{M_2,M_3}$, 8, 3, 6 следует, что $S_1'' \approx_{M_3,L} S_2''$ (10) и $o_1'' \approx_L o_2''$ (11). Наконец, из 1, 10, 9, 11 и 4 имеем ПЗИН $(c_i;c_j)_{M_1,M_3}$.

4.1. Этапы проверки выполнения условий безопасности информационных потоков

В общем виде распространение действующей системы управления доступом в программную среду БД для верификации информационных потоков, возникающих в хранимых процедурах и функциях, с точки зрения конфиденциальности (для целостности процедура выглядит аналогично), предлагается осуществлять в три этапа.

4.1.1. Трансляция требований существующей политики безопасности в метки элементов среды вычислений

На этом этапе требуется задать метки безопасности для входных значений, формальных параметров и возвращаемых значений процедур и функций, которые могут явно вызываться пользователями (пользовательскими приложениями). Очевидно, выбор меток должен соответствовать существующей политике управления доступом. Например, если привилегия execute на вызов некоторой процедуры предоставлена ролям: r_1, r_2, при этом существует такая подрешетка $(R_1, \sqsubseteq) : R_1 = \{r_1\}\uparrow$, в которой r_1 является нижней гранью, а T – верхней гранью и подрешетка $(R_2, \sqsubseteq) : R_2 = \{r_2\}\uparrow$, то формальным параметрам процедуры и ее возвращаемому значению присваивается метка $R_1 \cup R_2$. Предполагается, что параметрам и возвращаемым значениям “неразмеченных” программных блоков по умолчанию назначается наименее строгий элемент решетки меток безопасности в предположении, что доступ к таким программным блокам неограничен. Например, в соответствии с расширенной решеткой меток безопасности (рис. 9б) и требованиями политики:

– вызов функции *f_getsubmissions* возможен, если в сеансе установлена роль *manager* или выполнено условие ”time_expired” (в этом случае допускается вызов функции в сеансе любого пользователя);

– вызов процедуры *p_submit_paper* может осуществляться любым пользователем;

– вызов процедуры *p_change_status* может осуществляться пользователем, в сеансе которого установлена роль *reviewer*;

разметка соответствующих программных блоков выглядит, как показано в табл. 4.

Метки входных значений определяются аналитиком исходя из понимания их фактического уровня конфиденциальности. Для доверенных сторонних программных блоков в явном виде задаются метки: формальных параметров, выходного значения и эффекта записи (writing effect). Чистые функции разметки не требуют. Для простоты будем полагать, что внешние программные блоки не выбрасывают не обработанных исключений (достигается универсальным обработчиком WHEN OTHERS null;).

Аналогично задаются метки безопасности для иных элементов среды вычислений: атрибутов отношений, доступ к которым предоставлен явно некоторым пользователям, исключений проверяемых (не сторонних) программных блоков, переменных, выводимых в выходные потоки.

4.1.2. Построение и проверка модели параллельных вычислений с использованием линейной темпоральной логики действий

Далее с использованием *TLA+* спецификаций строится модель выполнения *PL/SQL* блоков в параллельных сеансах пользователей. Модель описывает все возможные переходы системы,

Таблица 4. Пример разметки программных блоков БД

	Входные значения	Формальные параметры	Выходное значение
f_getsubmissions	f_gs_i: $\bot$	f_gs_p1:	f_gs_r:
		$\{m, o, g', r', m', o'\}$	$\{m, o, g', r', m', o'\}$
p_submit_paper	p_sp_i1: $\bot$,	p_sp_p1: $\bot$,	
	p_sp_i2: $\bot$,	p_sp_p2: $\bot$,	
	p_sp_i3: $\bot$,	p_sp_p3: $\bot$,	
	p_sp_i4: $\bot$	p_sp_p4:	
p_change_status	p_cs_i1: $\bot$,	p_cs_p1: $\{r, o, r', o'\}$,	
	p_cs_i2: $\bot$ $\{m, o, g', r', m', o'\}$	p_sp_p2: $\{r, o, r', o'\}$	

описанной в п. 4, для множества начальных состояний $D \times C_{init} \to \{M_{init}\}$ и множества всех состояний $\mathbb{O}$ при условии, что в каждом из n сеансов (задается как параметр модели) выполняется только один программный блок. Заданное условие упрощает модель, при этом за счет композиционной природы свойства ПЗИН (Лемма 4.2) результат успешной верификации можно обобщить до системы, в которой в каждом сеансе пользователя выполняется произвольное количество хранимых процедур и функций. Здесь C_{init} – множество начальных шагов вычислений имеющихся процедур, M_{init} – начальное абстрактное состояние среды вычислений, в котором отображение глобальных переменных и выходных потоков на метки безопасности задается явно, а локальным переменным присваивается минимальная метка $\bot$.

Алгоритм проверки циклический.

Предварительный этап.

Модель инициализируется значениями констант. Они определяют количество параллельных сеансов, множество пользователей, алфавит меток безопасности. В соответствии с имеющимися привилегиями ролей и дополнительными требованиями политики безопасности осуществляется разметка элементов модели, ассоциируемых с проверяемыми программными блоками, а также используемыми в вычислениях доверенными сторонними процедурами. Задаются начальные состояния элементам модели, соответствующим глобальным переменным (таблицам) среды вычислений. Элементам, соответствующим локальным переменным, присваивается минимальная метка безопасности.

Шаг 1. Осуществляется прогон модели. Если нарушений инварианта безопасности не выявлено, осуществляется сопоставление начального и конечного отображения глобальных переменных на метки безопасности G_{init} и G_{fin}. Если начальное и конечное отображение совпадают, алгоритм завершается, рекомендаций по инструментированию кода или изменению привилегий системы разграничения доступа не требуется. В противном случае начальное отображение M_{init} изменяется следующим образом:

$$M_{init}(x) = \begin{cases} G_{fin}(x), & \text{если} \quad x \in Dom(G_{fin}) \\ M_{init}(x), & \text{в противном случае} \end{cases}$$

и шаг 1 повторяется. Если для некоторого состояния выявлено нарушение инварианта, происходит переход к шагу 2.

Шаг 2. Производится анализ трассы, приведшей к возникновению ошибки, в спецификацию вносятся соответствующие изменения, в список рекомендаций добавляется новая рекомендация, производится переход к шагу 1.

Как уже отмечалось, метки глобальных переменных возрастают монотонно, решетка меток конечна, поэтому алгоритм также конечен.

Утверждение 4.1. *Успешное завершение алгоритма гарантирует безопасное функционирование системы в смысле ПЗИН при неограниченном количестве программных блоков, выполняемых в каждом сеансе.*

Доказательство. Положим алгоритм проверки завершен успешно, для некоторого отображения множества глобальных переменных на множество меток безопасности G'. С учетом ранее сформулированной стратегии действий при “простаивании” сеанса после завершения выполнения программного блока (очередным действием сеанса становится load – операция следующего программного блока), а также, принимая во внимание сброс меток безопасности локальных переменных при новом запуске процедур и возможность их инициализации только значениями, извлекаемыми из глобальных переменных, для каждого возможного состояния системы $\langle C_k, M_k \rangle \in \mathbb{O}$, где $C_k = \langle c_1, \ldots c_n \rangle_k$ будем иметь: для $\forall i \in \{1, \ldots n\}$ справедливо, что при переходе $\langle \langle c_1, \ldots c_n \rangle_k, M_k \rangle \to^i_{1/n} \langle \langle c_1, \ldots c_n \rangle_l, M_l \rangle$ инвариант безопасности не нарушается, то есть в соответствии с Предположением 4.1 для c_i выполняет-

ся условие а) Определения 4.1. Истинность условия б) обеспечивается потокочувствительностью абстрактной семантики. Таким образом, для любого контекста $\langle c_i, M_k \rangle$ выполняется условие ПЗИН $(c_i)_{M_k, M_l}$. Истинность утверждения устанавливается по структурной индукции на основе правил переходов системы (см. п. 4). Истинность индуктивного шага следует из Леммы 4.2.

Также отметим, в соответствии с Леммой 4.1 приведение меток таблиц, доступ к которым явно не предоставлен ни одной роли, в соответствие с некоторым конечным отображением множества глобальных переменных на множество меток безопасности G' не требуется.

Справедливость выполнения условий Определения 4.1 в части "расхождения" вычислений для проверенных в соответствии с алгоритмом программ может быть обеспечена исключением самой возможности бесконечных вычислений в рамках выделенного сеанса путем использования так называемых ресурсных ограничений на уровне СУБД.

В данной работе подробно не рассматривается процедура трансформации *PL/SQL* кода в формулы линейной темпоральной логики действий, однако, в Приложении 1 представлен результат трансформации процедуры:

```
1 p_chahge_status (s_id number, stat number)
2 IS
3 BEGIN
4 UPDATE SUBMISSIONS
5 SET STATUS = stat
6 WHERE SUBMISSION_ID = s_id
7 END p_chahge_status;
```

Для генерации спецификаций используется абстрактная семантика (Приложение 2). Проверка выполнения условий прогресс-зависимого информационного невлияния на каждом шаге вычислений осуществляется с помощью инварианта безопасности вида:

$$NonInt\_Inv \triangleq p_{expr} \sqcup p_{pc} \sqsubseteq p_{out}$$

Здесь p_{expr} – метка выражения, которое выводится в выходной поток (присваивается глобальной переменной), p_{pc} – метка текущего счетчика команд, pc_{out} – метка выходного потока.

4.1.3. Выработка рекомендаций по инструментированию кода и изменению общей политики

При проверке модели с помощью model checker могут возникать нарушения инварианта безопасности. В таких случаях с целью устранения запрещенных информационных потоков аналитик исправляет спецификацию, корректируя метки размеченных программных блоков (элементов среды вычислений, доступных внешнему наблюдателю) или используя операторы декласс-ификации. На основе внесенных исправлений формируются рекомендации по инструментированию кода хранимых процедур и функций или изменению правил разграничения доступа.

Пример обработки нарушения инварианта

Пусть в системе реализуется политика безопасности, описанная выше. Исключением является требование по ограничению доступа к функции *f_getsubmissions*. Допустим, в соответствии с политикой вызывать функцию *f_getsubmissions* могут любые пользователи, если выполняется условие "time_expired". Таким образом, для настройки привилегий могут применяться следующие команды:

```
1 grant execute on f_getsubmissions to PUBLIC;
2 grant execute on p_submit_paper_to PUBLIC;
3 grant execute on p_change_status to reviewer;
```

Разметка программных блоков на этапе трансляции требований политики безопасности в метки элементов среды вычислений выполняется, как описано в табл. 4, за исключением формального

Таблица 5. Варианты корректировки модели

	Исправление спецификации	Практические рекомендации
Вариант 1 (*корректировка политики*)	f_gs_p1: $\{m, o\}$,	grant execute on
	f_gs_r: $\{m, o\}$	f_getsubmissions to managers
Вариант 2 (*деклассификация*)	f_gs_p1: $\{g', r', m', o'\}$,	if time_expired(c_id) then ...
	f_gs_r: $\{g', r', m', o'\}$	return v_submissions; else null;
Вариант 3 (*деклассификация, корректировка политики*)	f_gs_p1: $\{m, o, g', r', m', o'\}$	if time_expired(c_id) or session_role.exists('manager') or
	f_gs_r: $\{m, o, g', r', m', o'\}$	session_role.exist('organizer') then ... return v_submissions; else null;
Вариант 4 (*ложное срабатывание – игнорирование*)	(C-EXT-RET): Inv – ignore	

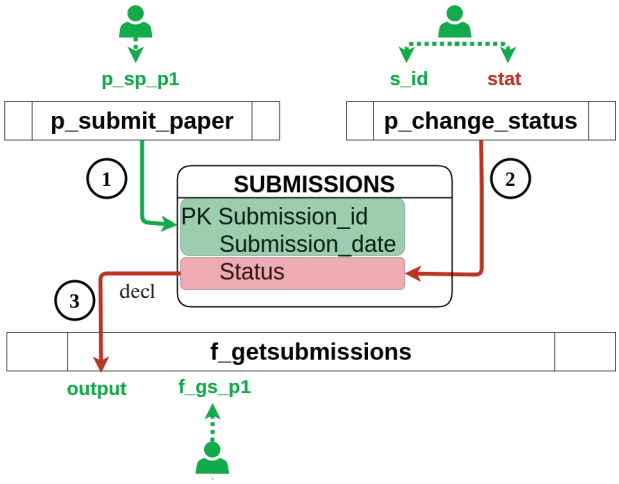


Рис. 10. Нарушение инварианта информационного невлияния.

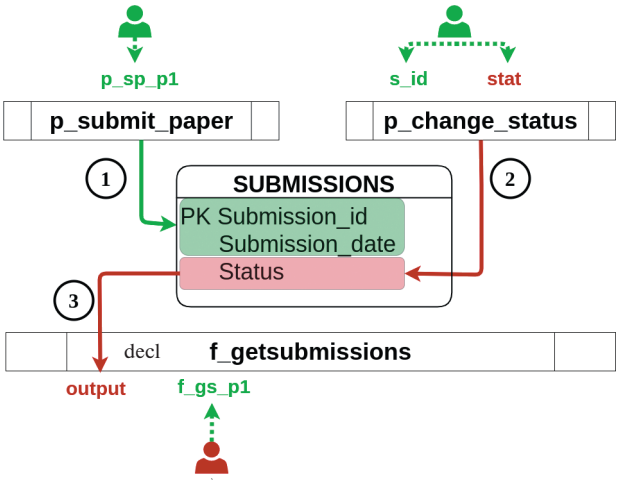


Рис. 11. Деклассификация – рекомендации по изменению кода.

параметра и возвращаемого значения функции *f_getsubmissions*, для них выбирается минимальная метка $\perp$. В этом случае при прогоне модели возникает нарушение инварианта. Трасса переходов, предшествующих ошибке выглядит так:

```
FATAL InvariantViolation:
    Error validating NonInt_Inv:
    {m,o,g',r',m',o'} is greater than bot
    bob → p_f_getsubmissions9
    bob → p_f_getsubmissions6
    mattew → p_change_status_exit
    mattew → p_change_status4
    bob → p_f_getsubmissions_load
    mattew → p_change_status_load
    alex → p_submit_paper_exit
    alex → p_submit_paper4
    alex → p_submit_paper_load
```

Запрещенный информационный поток возникает вследствие переноса входного значения *stat* (p_cs_i2) функции *f_getsubmissions*, обладающего меткой $\{m,o,g',r',m',o'\}$, в выходную переменную *out* (f_gs_p1), обладающую меткой $\perp$ (рис. 10).

В табл. 5 приведены возможные варианты исправления модели и выработки рекомендаций для разработчиков (администраторов безопасности).

Первый вариант предполагает ужесточение глобальной политики управления доступом – рис. 11, второй вариант основан на процедуре деклассификации данных, которая может быть реализована путем внесения в соответствующие участки кода проверок выполнения условий деклассификации – рис. 12. Третий – наиболее гибкий вариант, предполагает внесение минимальных ограничений, обеспечивающих устранение запрещенного информационного потока. Поскольку при построении модели приходится прибегать к абстрагированию, точность анализа теряется, и в определенных случаях нарушение инварианта может классифицироваться как ложное срабатывание (Вариант 4). Например, в описанной абстрактной семантике источником неточности, как отмечалось выше, могут выступать правила: (C-IF) и (C-WHILE).

5. ЗАКЛЮЧЕНИЕ

Основной вклад данной работы в развитие теории КИП заключается в разработке подхода к разделению функций написания кода и описания политики безопасности прикладного уровня. Формализована процедура распространения требований ролевого разграничения доступа в про-

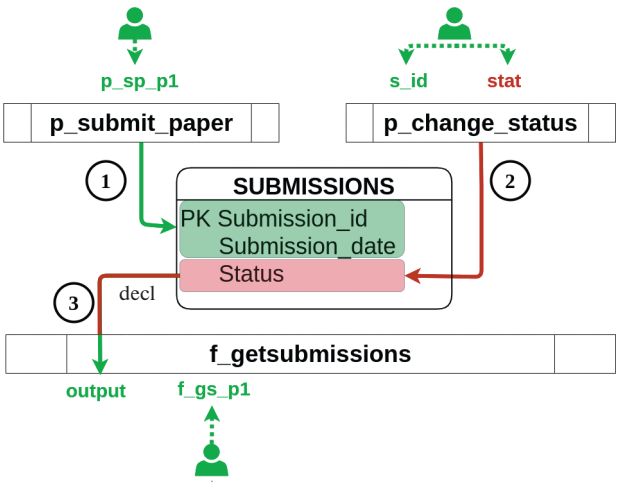


Рис. 12. Изменение глобальной политики.

граммную среду БД, предложен алгоритм проверки модели вычислений и подготовки рекомендаций по инструментированию кода (изменению политики). Сформулированы условия миграции механизма проверки условий безопасности вычислений из языковой среды в область формальной верификации.

Очевидно, реализация КИП на языковом уровне не имеет альтернативы с точки зрения гарантий безопасности. Объективно, в силу влияния человеческого фактора на процесс написания (исправления) кода, предложенный подход не может полностью исключить появления в программном обеспечении запрещенных информационных потоков. Кроме того, абстрагирование, которое применяется для сокращения числа состояний до приемлемого, понижает точность анализа и повышает критичность задачи выявления "ложных" срабатываний. Тем не менее предложенный подход на практике может оказаться более жизнеспособным с учетом тенденций развития отрасли.

К перспективным направлениям дальнейших исследований целесообразно отнести: оптимизацию разработанной абстрактной модели параллельного выполнения хранимых программных блоков, разработку утилиты разметки и трансформации *PL/SQL* кода в спецификации *TLA+*, выработку рекомендаций по встраиванию вызовов проверенных процедур и функций в прикладное ПО.

ПРИЛОЖЕНИЕ 1.
ПРИМЕР ТРАНСФОРМАЦИИ PL/SQL2TLA+

```
p_change_status_load(id) ≜
 IF XLocks = Undef
    THEN
    ∧ XLocks' = id
    ∧ Sessions' = [Sessions EXCEPT ! [id ]["SessionM"] = Sessions[id]["SessionM"] ∘
                  ⟨min; {⟨u1; ⟨[t_expire ↦ {NONE}]; [guest ↦ {NONE};
                        reviewer ↦ {NONE}; manager ↦ {u1}; organizer ↦ {NONE}⟩⟩;
                  ⟨u1; ⟨[t_expire ↦ {}]; [guest ↦ {NONE}; reviewer ↦ {NONE};
                        manager ↦ {NONE}; organizer ↦ {NONE}⟩⟩}⟩]
    ∧ New2Old' = ⟨⟨p_cs_p_s_id(id):policy; p_cs_p_stat(id).policy⟩,
                  ⟨min; {⟨u1; ⟨[t_expire ↦ {NONE}]; [guest ↦ {NONE};
                        reviewer ↦ {NONE}; manager ↦ {u1}; organizer ↦ {NONE}]⟩⟩;
                  ⟨u1; ⟨[t expire ↦ {}]; [guest ↦ {NONE}; reviewer ↦ {NONE};
                        manager ↦ {NONE}; organizer ↦ {NONE}]⟩⟩}⟩⟩
    ∧ Ignore' = 1
    ∧ StateE' = SLocks'[id]
    ∧ UNCHANGED ⟨VPol; SLocks⟩
 ELSE UNCHANGED vars

p_change_status4(id) ≜
    ∧ update(id; ⟨"col_submissions_status"⟩, ⟨load(id; p_cs_p_stat(id))⟩;
              LUB(VPol["col_submissions_submission_id"].policy, load(id; p_cs_p_s_id(id)));
                   ⟨"p_change_status"; "exit"⟩
    ∧ Trace' = Append(Trace; ⟨id; "p_change_status4"⟩)
    ∧ Ignore' = 0
    ∧ StateE' = SLocks'[id]
    ∧ UNCHANGED ⟨XLocks; SLocks⟩

p_change_status_exit(id) ≜
    ∧ IF Len(Sessions[id]["StateRegs"]) = 1
       THEN XLocks' = Undef
       ELSE XLocks' = XLocks
    ∧ Sessions' =
      [Sessions EXCEPT
```

! [*id*][“StageRegs”] = Tail (*Sessions*[*id*][“StateRegs”]) ∘ ⟨⟩;
! [*id*][“SessionM”] = *SubSeq*(*Sessions*[*id*][“SessionM”]; 1; *Len*(*Sessions*[*id*][“SessionM”]) − 2)]
∧ *Trace'* = *Append*(*Trace*; ⟨*id* ; “p_change_status_exit”⟩)
∧ *Ignore'* = 1
∧ *StateE'* = *SLocks'*[*id*]
∧ UNCHANGED ⟨*New2Old; VPol; SLocks*⟩

p_change_status(*id*; *st*) ≜
CASE *Head*(*st*):*pc* [2] = *“lbl_*4” → *p_change_status*4(*id*)
□ *Head*(*st*):*pc* [2] = *“exit”* → *p_change_status_exit*(*id*)
□ OTHER → UNCHANGED *vars*

ПРИЛОЖЕНИЕ 2.
АБСТРАКТНАЯ СЕМАНТИКА ИНФОРМАЦИОННЫХ ПОТОКОВ

EXPRESSIONS:

$$\textbf{(E-CONST)}\frac{}{\langle v, M\rangle \Downarrow \bot} \quad \textbf{(E-VAR)}\frac{}{\langle x, M\rangle \Downarrow M[x]} \quad \textbf{(E-OPER)}\frac{\langle e_1, M\rangle \Downarrow p_1 \quad \langle e_2, M\rangle \Downarrow p_2}{\langle e_1 \odot e_2, M\rangle \Downarrow p_1 \sqcup p_2}$$

$$\textbf{(E-REC-FLD)}\frac{\langle x, M\rangle \Downarrow p}{\langle x_r.x, M\rangle \Downarrow p} \quad \textbf{(E-VAR-REC)}\frac{\langle x_r.x_1, M\rangle \Downarrow p_1 \ldots \langle x_r.x_n, M\rangle \Downarrow p_n}{\langle x_r, M\rangle \Downarrow p_1 \sqcup p_2 \ldots \sqcup p_n}$$

$$x_{cur} \triangleq \begin{array}{l}\textbf{select } e_1,\ldots,e_n \\ \textbf{from } x_{t_1},\ldots,x_{t_n}\end{array} \textbf{where } cnd$$

$$\textbf{(E-TAB-COL)}\frac{}{\langle x_t.c, M\rangle \Downarrow M[x_t.c]} \quad \textbf{(E-CUR)}\frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle x_n, M\rangle \Downarrow p_n \quad \langle cnd, M\rangle \Downarrow p_{cnd}}{\langle x_{cur}, M\rangle \Downarrow p_1 \sqcup p_2 \ldots \sqcup p_n \sqcup p_{cnd}}$$

$$\textbf{(E-VAR-ARR)}\frac{\langle x_a[1], M\rangle \Downarrow p_{x_a[1]} \ldots \langle x_a[n], M\rangle \Downarrow p_{x_a[n]} \quad \langle n, M\rangle \Downarrow p_{len}}{\langle x_a, M\rangle \Downarrow p_{x_a[1]} \sqcup p_{x_a[2]} \ldots \sqcup p_{x_a[n]} \sqcup p_{len}}$$

$$\textbf{(E-CUR-FND)}\frac{\langle x, M\rangle \Downarrow p_x}{\langle x\%\ \text{found}, M\rangle \Downarrow p_x} \quad \textbf{(E-PUR-LIB)}\frac{x_f - pure \quad \langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_n, M\rangle \Downarrow p_n}{\langle PC, x_f(x_1 \to e_1, \ldots x_n \to e_n), M\rangle \Downarrow p_1 \sqcup \ldots p_n}$$

$$\textbf{(E-TR-LIB)}\frac{\begin{array}{r}x_f - trusted \quad \langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \langle f_{out}, M\rangle \Downarrow p_{out} \quad x_f \rightsquigarrow p_w \\ \langle x_1, M\rangle \Downarrow p_{x_1} \ldots \langle x_m, M\rangle \Downarrow p_{x_m} \quad \textbf{inv} : \wedge p_1 \sqsubseteq p_{x_1} \ldots p_m \sqsubseteq p_{x_m} \\ \wedge pc_1 \sqcup pc_2 \ldots pc_n \sqsubseteq p_w\end{array}}{\langle PC, x_f(x_1 \to e_1, \ldots x_m \to e_m), M\rangle \Downarrow p_{out}}$$

$$\textbf{(E-C-OPER)}\frac{\langle e_1, M\rangle \Downarrow p_1 \quad \langle e_2, M\rangle \Downarrow p_2}{\langle e_1 \times e_2, M\rangle \Downarrow p_1 \sqcup p_2} \quad \textbf{(E-L-OPER)}\frac{\langle cnd_1, M\rangle \Downarrow p_1 \quad \langle cnd_2, M\rangle \Downarrow p_2}{\langle cnd_1 \star cnd_2, M\rangle \Downarrow p_1 \sqcup p_2}$$

STATEMENTS:

$$\textbf{(C-ASSIGN)}\frac{\langle e, M\rangle \Downarrow p}{\langle PC, x := e, M\rangle \to \langle PC, \textbf{null}, M[x \mapsto p \sqcup pc_1 \ldots pc_n]\rangle}$$

$$\textbf{(C-CLOSE)}\frac{}{\langle PC, \textbf{close } x, M\rangle \to \langle PC, \textbf{null}, M\rangle} \quad \textbf{(C-OPEN)}\frac{}{\langle PC, \textbf{open } x, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-EXT-ASSIGN)}\frac{\langle e, M\rangle \Downarrow p \quad \langle x, M\rangle \Downarrow p_x \quad \textbf{inv}{:}p \sqcup pc_1 \ldots pc_n \sqsubseteq p_x}{\langle PC, x := e, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-IF)}\frac{\langle e, M\rangle \Downarrow p \quad \textbf{inv}{:}e - local}{\langle PC, \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2, M\rangle \to \langle p :: PC, c_1 \vee c_2, M\rangle}$$

$$\textbf{(C-END-IF)}\ \frac{}{\langle p :: PC, \textbf{endif}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-WHILE)}\ \frac{\langle e, M\rangle \Downarrow p \quad \textbf{inv}{:}e - local}{\langle PC, \textbf{while}\ e\ \textbf{do}\ c, M\rangle \to \langle p :: PC, c \vee \textbf{null}, M\rangle}$$

$$\textbf{(C-COMMIT)}\ \frac{}{\langle PC, \textbf{commit}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-END-WHILE)}\ \frac{}{\langle p :: PC, \textbf{endwhile}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-ROLLBACK)}\ \frac{}{\langle PC, \textbf{rollback}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-SEL)}\ \frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \langle cnd, M\rangle \Downarrow p_{cnd}}{\left\langle PC, \begin{array}{c}\textbf{select}\ e_1,\ldots,e_m\ \textbf{into}\ x_1,\ldots,x_m \\ \textbf{from}\ x_{t_1},\ldots,x_{t_k}\ \textbf{where}\ cnd\end{array}, M\right\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_1 \mapsto p_1 \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_m \mapsto p_m \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-SEL)}\ \frac{\begin{array}{l}\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \langle cnd, M\rangle \Downarrow p_{cnd} \quad \textbf{inv}{:} \wedge\, p_1 \sqcup p_{cnd} \sqcup pc_1 \ldots px_1 \sqsubseteq p_{x_1} \wedge \ldots \\ \langle x_1, M\rangle \Downarrow p_{x_1} \ldots \langle e_m, M\rangle \Downarrow p_{x_m} \qquad\qquad \wedge\, p_m \sqcup p_{cnd} \sqcup pc_1 \ldots px_m\end{array}}{\left\langle PC, \begin{array}{c}\textbf{select}\ e_1,\ldots,e_m\ \textbf{into}\ x_1,\ldots,x_m \\ \textbf{from}\ x_{t_1},\ldots,x_{t_k}\ \textbf{where}\ cnd\end{array}, M\right\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-INS)}\ \frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m}{\left\langle PC, \begin{array}{c}\textbf{insert into}\ x_t(c_1,\ldots,c_m) \\ \textbf{values}\ (e_1,\ldots,e_m)\end{array}, M\right\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_t.c_1 \mapsto M[x_t.c_1] \sqcup p_1 \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_t.c_m \mapsto M[x_t.c_m] \sqcup p_m \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-INS)}\ \frac{\begin{array}{l}\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \textbf{inv}{:} \wedge\, p_1 \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_1] \wedge \ldots \\ \qquad\qquad \wedge p_m \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_m]\end{array}}{\left\langle PC, \begin{array}{c}\textbf{insert into}\ x_t(x_t.c_1,\ldots,x_t.c_m) \\ \textbf{values}\ (e_1,\ldots,e_m)\end{array}, M\right\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-UPD)}\ \frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \langle cnd, M\rangle \Downarrow p_{cnd}}{\left\langle \begin{array}{c}\textbf{update}\ x_t \\ PC, \textbf{set}\ x_t.c_1 = e_1,\ldots, M \\ \textbf{where}\ cnd\end{array}\right\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_t.c_1 \mapsto M[x_t.c_1] \sqcup p_1 \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_t.c_m \mapsto M[x_t.c_m] \sqcup p_m \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-UPD)}\ \frac{\begin{array}{l}\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m \quad \textbf{inv}{:} \wedge\, p_1 \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_1] \wedge \ldots \\ \langle cnd, M\rangle \Downarrow p_{cnd} \qquad\qquad \wedge p_m \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_m]\end{array}}{\left\langle PC, \begin{array}{c}\textbf{update}\ x_t \\ \textbf{set}\ x_t.c_1 = e_1,\ldots\textbf{where}\ cnd\end{array}, M\right\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-DEL)}\ \frac{}{\left\langle PC, \begin{array}{c}\textbf{delete from}\ x_t \\ \textbf{where}\ cnd\end{array}, M\right\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_t.c_1 \mapsto M[x_t.c_1] \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_t.c_m \mapsto M[x_t.c_m] \sqcup p_{cnd} \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-DEL)}\frac{\begin{array}{c}\textbf{inv}: \wedge\ p_{cnd} \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_1] \wedge \ldots \\ \wedge\ p_{cnd} \sqcup pc_1 \ldots pc_n \sqsubseteq M[x_t.c_m]\end{array}}{\left\langle PC, \begin{array}{c}\textbf{delete from}\ x_t \\ \textbf{where}\ cnd\end{array}, M\right\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-PROC)}\frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_m, M\rangle \Downarrow p_m}{\langle PC, x_f(x_1 \to e_1, \ldots x_m \to e_m), M\rangle \to \left\langle PC, \textbf{null}, M\begin{bmatrix} x_1 \mapsto p_1 \sqcup pc_1 \ldots pc_n, \\ \ldots \\ x_m \mapsto p_m \sqcup pc_1 \ldots pc_n \end{bmatrix}\right\rangle}$$

$$\textbf{(C-EXT-PRC)}\frac{\langle e_1, M\rangle \Downarrow p_1 \ldots \langle e_n, M\rangle \Downarrow p_n \quad \langle x_1, M\rangle \Downarrow p_{x_1} \ldots \langle x_n, M\rangle \Downarrow p_{x_n} \quad \textbf{inv}{:}p_1 \sqsubseteq p_{x_1} \ldots p_n \sqsubseteq p_{x_n}}{\langle PC, x_f(x_1 \to e_1, \ldots x_n \to e_n), M\rangle \to \langle PC, \textbf{null}, M[x_1 \mapsto p_1, \ldots x_n \mapsto p_n]\rangle}$$

$$\textbf{(C-RET)}\frac{\langle e, M\rangle \Downarrow p_1}{\langle PC, \textbf{return}(x_{out} \to e), M\rangle \to \langle PC, \textbf{null}, M[x_{out} \mapsto p_1 \sqcup pc_1 \ldots pc_n]\rangle}$$

$$\textbf{(C-EXT-RET)}\frac{\langle e, M\rangle \Downarrow p_1 \quad \langle x_{out}, M\rangle \Downarrow p_2 \quad \textbf{inv}{:}p_1 \sqcup pc_1 \ldots pc_n \sqsubseteq p_2}{\langle PC, \textbf{return}(x_{out} \to e), M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-FETCH)}\frac{\langle x_{cur}, M\rangle \Downarrow p}{\langle PC, \textbf{fetch}\ x_{cur}\ \textbf{into}\ x, M\rangle \to \langle PC, \textbf{null}, M[x \mapsto p \sqcup pc_1 \ldots pc_n]\rangle}$$

$$\textbf{(C-EXT-FETCH)}\frac{\langle x_{cur}, M\rangle \Downarrow p \quad \langle x, M\rangle \Downarrow p_x \quad \textbf{inv}{:}p \sqcup pc_1 \ldots pc_n \sqsubseteq p_x}{\langle PC, \textbf{fetch}\ x_{cur}\ \textbf{into}\ x, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-OR)}\frac{i \in 1,2}{\langle PC, c_1 \vee c_2\rangle \to_{1/2} \langle PC, c_i, M\rangle}$$

$$\textbf{(C-EXC)}\frac{\langle x_{exc}, M\rangle \Downarrow p}{\langle PC, \textbf{when}\ x_{exc}\ \textbf{then}\ c, M\rangle \to \langle p :: PC, c, M\rangle} \qquad \textbf{(C-NULL)}\frac{}{\langle PC, \textbf{null}, M\rangle \to \langle PC, M\rangle}$$

$$\textbf{(C-END-EXC)}\frac{}{\langle p :: PC, \textbf{endexc}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-THROW)}\frac{}{\langle PC, \textbf{throw}\ x_{exc}, M\rangle \to \langle PC, \textbf{null}, M\rangle}$$

$$\textbf{(C-SEQ-1)}\frac{\langle PC, c_1, M\rangle \to \langle PC, c_1', M\rangle}{\langle PC, c_1; c_2, M\rangle \to \langle PC, c_1'; c_2, M\rangle} \qquad \textbf{(C-SEC-2)}\frac{\langle PC, c_1, M\rangle \to \langle PC, M\rangle}{\langle PC, c_1; c_2, M\rangle \to \langle PC, c_2, M\rangle}$$

GLOBAL RULES:

$$\textbf{(GLOB-1)}\frac{O(d) = c_i \quad \langle PC_i, c_i, M\rangle \to \langle PC_k, c_k, M'\rangle}{\langle\langle PC_1, c_1\rangle \ldots \langle PC_n, c_n\rangle\rangle, M\rangle \to^i_{1/n} \langle\langle\langle PC_1, c_1\rangle \ldots \langle PC_{i-1}, c_{i-1}\rangle\langle PC_k, c_k\rangle \ldots \langle PC_n, c_n\rangle\rangle, M'\rangle}$$

$$\textbf{(GLOB-2)}\frac{O(d) = c_i \quad \langle PC_i, c_i, M\rangle \to \langle PC_i, M\rangle}{\langle\langle PC_1, c_1\rangle \ldots \langle PC_n, c_n\rangle\rangle, M\rangle \to^i_{1/n} \langle\langle\langle PC_1, c_1\rangle \ldots \langle PC_{i-1}, c_{i-1}\rangle\langle PC_{i+1}, c_{i+1}\rangle \ldots \langle PC_n, c_n\rangle\rangle, M\rangle}$$

СПИСОК ЛИТЕРАТУРЫ

1. *Fischer P., Katkalov K., Stenzel K., Reif W.* Formal Verification of Information Flow Secure Systems with IFlow, 2012
2. *Graf J., Hecker M., Mohr M., Snelting G.* Checking Applications using Security APIs with JOANA, 2015. http://www.dsi.unive.it/ focardi/ASA8/.
3. *Myers C.A., Liskov B.* A decentralized model for information flow control // ACM SIGOPS Operating Systems Review. 1997. № 5. P. 129–142.
4. *Broberg N., Sands D.* Paralocks: Role-based information flow control and beyond // Conference Record of the Annual ACM Symposium on Principles of Programming Languages, 2010. P. 431–444.
5. *Pottier F., Simonet V.* Information Flow Inference for ML // ACM Transactions on Programming Languages and Systems, 2003. P. 117–158.
6. *Лесько С.А.* Модели и сценарии реализации угроз для интернет-ресурсов // Российский технологический журнал. 2020. Т. 8. № 6. С. 9–33.
7. *Goguen A.J., Meseguer J.* Security policies and security models // 1982 IEEE Symposium on Security and Privacy. IEEE, 1982. P. 11–11.
8. *Hedin D., Sabelfeld A.* A Perspective on Information-Flow Control // Software Safety and Security. 2012. P. 319–347.
9. *Bell D., LaPadula Lj J.* Secure computer systems: Mathematical foundations // Data Base. 1973. MTR-2547. V. I. P. 513–523.
10. *Denning E.D., Denning J.P.* Certification of Programs for Secure Information Flow // Communications of the ACM. 1977. № 7. P. 504–513.
11. *Volpano D.I., Cynthia S.G.* Sound type system for secure flow analysis // Journal of Computer Security. 1996. № 2–3. P. 167–187.
12. *Cohen S.E.* Information transmission in sequential programs // Foundations of Secure Computation. 1978. P. 297–335.
13. *Denning E.D.* A lattice model of secure information flow // Communications of the ACM. 1976. № 5. P. 236–243.
14. *Broberg N., Sands D.* Flow locks: Towards a core calculus for dynamic flow policies // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2006. P. 180–196.
15. *Bart V.D., Broberg N., Sands D.* A Datalog semantics for Paralocks // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2013. P. 305–320.
16. *Hedin D., Sabelfeld A.* A Perspective on Information-Flow Control // Software Safety and Security. 2012. P. 319–347.
17. *Sabelfeld A., Sands D.* Probabilistic noninterference for multi-threaded programs // Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13. IEEE, 2000. P. 200–214.
18. *Mantel H., Sabelfeld A.* A Generic Approach to the Security of Multi-Threaded Programs // CSFW. Citeseer, 2001. P. 126–142.
19. *Mantel H., Sabelfeld A.* A unifying approach to the security of distributed and multi-threaded programs // Journal of Computer Security. 2003. № 4. P. 615–676.
20. *Askarov A., Sabelfeld A.* Gradual release: Unifying declassification, encryption and key release policies // Proceedings – IEEE Symposium on Security and Privacy, 2007. P. 207–221.
21. *Broberg N.* Thesis for the Degree of Doctor of Engineering Practical, Flexible Programming with Information Flow Control. 2011.
22. *Sabelfeld A., Russo A.* From dynamic to static and back: Riding the roller coaster of information-flow control research // International Andrei Ershov Memorial Conference on Perspectives of System Informatics. Spinger, 2009. P. 352–365.
23. *Shroff P., Smith S., Thober M.* Dynamic dependency monitoring to secure information flow // 20th IEEE Computer Security Foundations Symposium (CSF'07). 2007. P. 203–217.
24. *Mantel H., Sudbrock H.* Types vs. pdgs in information flow analysis // International Symposium on Logic-Based Program Synthesis and Transformation. 2012. P. 106–121.
25. *Schoepe D., Murray T., Sabelfeld A.* VERONICA: expressive and precise concurrent information flow security (extended version with technical appendices) // arXiv preprint arXiv:2001.11142. 2020.
26. *Müller C., Seidl H., Z*ā linescu E. Inductive invariants for noninterference in multi-agent workflows // 2018 IEEE 31st Computer Security Foundations Symposium (CSF). 2018. P. 247–261.
27. *Finkbeiner B., Muller C., Seidl H., Z*ā linescu E. Verifying security policies in multi-agent workflows with loops // Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017. P. 633–645.
28. *Девянин П.Н., Леонова М.А.* Применение подтипов и тотальных функций формального метода Event-B для описания и верификации МРОСЛ ДП-модели // Программная инженерия. 2020. Т. 11. № 4. С. 230–241.
29. *Balliu M., Liebe B., Schoepe D., Sabelfeld A.* Jslinq: Building secure applications across tiers // Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. 2016. P. 307–318.
30. *Huang Y.-W., Yu F., Hang C., Tsai C.-H., Lee D.-T., Kuo S.-Y.* Securing web application code by static analysis and runtime protection // Proceedings of the 13th international conference on World Wide Web. 2004. P. 40–52.
31. *Chlipala A.* Ur/Web: A simple model for programming the web // Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 2015. P. 153–165.
32. *Arden O., George M.D., Liu J., Vikram K., Askarov A., Myers A.C.* Sharing mobile code securely with information flow control // 2012 IEEE Symposium on Security and Privacy. 2012. P. 191–205.
33. *Schoepe D., Hedin D., Sabelfeld A.* SeLINQ: tracking information across application-database boundaries // Proceedings of the 19th ACM SIGPLAN international conference on Functional programming. 2014. P. 25–38.

34. *Schultz D., Liskov B.* IFDB: decentralized information flow control for databases // Proceedings of the 8th ACM European Conference on Computer Systems. 2013. P. 43–56.

35. *Guarnieri M., Balliu M., Schoepe D., Basin D., Sabelfeld A.* Information-Flow Control for Database-backed Applications // 2019 IEEE European Symposium on Security and Privacy (EuroS&P). 2019. P. 79–94.

36. *Spearritt J.* The Applicability of Information Flow to Secure Java Development. 2017.

37. *Methni A., Lemerre M., Hedia B.B., Barkaoui K., Haddad S.* An Approach for Verifying Concurrent C Programs // 8th Junior Researcher Workshop on Real-Time Computing. 2014. P. 33–36.

38. *Timakov A.* PLIF. 2021. GitHub. https://github.com/timimin/plif.

39. *Lamport L.* Specifying systems. B.: Addison-Wesley, 2002. 388 p.

40. *Konnov I., Kukovec J., Tran T.-H.* TLA+ model checking made symbolic // Proceedings of the ACM on Programming Languages. 2019. V. 3. OOPSLA. P. 1–30.

41. *Smith G., Volpano D.* Secure information flow in a multi-threaded imperative language // Conference Record of the Annual ACM Symposium on Principles of Programming Languages. 1998. P. 355–364.

42. *Methni A., Lemerre M., Hedia B.B., Haddad S., Barkaoui K.* Specifying and verifying concurrent C programs with TLA+ // Communications in Computer and Information Science. 2015. V. 476. P. 206–222.

43. *Harrison M.A ., Ruzzo W.L., Ullman J.D.* Protection in operating systems // Communications of the ACM. 1976. V. 19. № 8. P. 461–471.

44. *Volpano D., Smith G.* Probabilistic noninterference in a concurrent language // Journal of Computer Security. 1999. V. 7. № 2. P. 231–253.

45. *Sabelfeld A., Sands D.* Probabilistic noninterference for multi-threaded programs // Proceedings 13th IEEE Computer Security Foundations Workshop. CSFW-13. 2000. P. 200–214.

EDN: PPSFHS
ПРОГРАММИРОВАНИЕ, 2022, № 4, с. 50–56

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 519.85

АЛГОРИТМ ВЫЧИСЛЕНИЯ РЕШЕНИЯ ЗАДАЧИ КОШИ ДЛЯ ДВУМЕРНОГО РАЗНОСТНОГО УРАВНЕНИЯ С НАЧАЛЬНЫМИ ДАННЫМИ, ЗАДАННЫМИ В “ПОЛОСЕ”

© 2022 г. М. С. Апанович[a],* (ORCID: 0000-0002-0889-2475),
А. П. Ляпин[b] (ORCID: 0000-0002-0149-7587), К. В. Шадрин[a],*** (ORCID: 0000-0002-8290-0904)**

[a]*Красноярский государственный медицинский университет имени профессора В.Ф. Войно-Ясенецкого, 660022 Красноярск, ул. Партизана Железняка, 1, Россия*

[b]*Сибирский федеральный университет, 660041 Красноярск, пр. Свободный, 79, Россия*

**E-mail: marina.apanovich@list.ru*

***E-mail: aplyapin@sfu-kras.ru*

****E-mail: kvsh_buffon@mail.ru*

Поступила в редакцию 16.07.2021 г.
После доработки 13.12.2021 г.
Принята к публикации 31.01.2022 г.

Представлен алгоритм вычисления решения задачи Коши для двумерного разностного уравнения с постоянными коэффициентами в точке по коэффициентам разностного уравнения и начальным данным задачи Коши, заданными в “полосе”, методами компьютерной алгебры. Для автоматизации процесса вычисления решения данной задачи был разработан алгоритм в среде MATLAB, где входными данными являются матрица коэффициентов двумерного полиномиального разностного уравнения, координаты матрицы начальных данных, регламентирующей структуру разностного уравнения, координаты точки, задающей размерность матрицы начальных данных; матрица начальных данных. Результатом работы алгоритма является решение задачи Коши, начальные данные которой заданы в “полосе”, для двумерного разностного уравнения, представляющее собой значение функции в искомой точке.

DOI: 10.31857/S0132347422040021

1. ВВЕДЕНИЕ

Линейные разностные уравнения возникают в различных областях математики, следовательно, поиск решений таких уравнений является одной из математических задач, имеющей многочисленные приложения в различных областях науки и техники. Например, разностные уравнения часто используются в моделях динамики с дискретным временем [1, 2], а также для приближенного решения дифференциальных уравнений [3], в комбинаторном анализе разностные уравнения в сочетании с методом производящих функций дают мощный аппарат исследования перечислительных задач (см., например, [4–6]). Алгоритм вычисления производящей функции решения задачи Коши для двумерного разностного уравнения с постоянными коэффициентами по коэффициентам разностного уравнения и начальным данным задачи Коши представлен в работе [7]. В работе [8] представлен алгоритм вычисления решения задачи Коши для двумерного разностного уравнения с постоянными коэффициентами в точке по коэффициентам разностного уравнения и начальным данным задачи Коши. В данной работе представлен алгоритм вычисления решения задачи Коши для двумерного разностного уравнения с постоянными коэффициентами по коэффициентам разностного уравнения и начальным данным задачи Коши, которые заданы в “полосе”.

2. ПОСТАНОВКА ЗАДАЧИ И ИЗВЕСТНЫЕ РЕЗУЛЬТАТЫ

Обозначим $\mathbb{Z}$ – множество целых чисел, $\mathbb{Z}^2 = \mathbb{Z}\times\mathbb{Z}$ – двумерная целочисленная решетка и $\mathbb{Z}_+^2$ – подмножество этой решетки, состоящее из точек с целыми неотрицательными координатами. Пусть δ_1 – оператор сдвига по переменной x, т.е. $\delta_1 f(x,y) = f(x+1,y)$, а δ_2 – оператор сдвига по переменной y, т.е. $\delta_2 f(x,y) = f(x,y+1)$. Зададим “полосу”, $\Pi = \{(x,y)\in\mathbb{Z}^2, 0\le x\le B, y\ge 0\}$ в положительном октанте целочисленной решетки, число $B+1$ будем называть шириной “поло-

сы” П. Рассмотрим разностный полиномиальный оператор с постоянными коэффициентами вида

$$P(\delta_1, \delta_2) = \sum_{j=0}^{m} \sum_{i=0}^{b} c_{ij} \delta_1^i \delta_2^j = \sum_{j=0}^{m} P_j(\delta_1) \delta_2^j, \qquad (1)$$

где m и b определяют размер схемы, $P_j(\delta_1) = \sum_{i=0}^{b} c_{ij} \delta_1^i$, $j = 0, 1, \ldots, m$.

Многочлен $P(z, w) = \sum_{j=0}^{m} \sum_{i=0}^{b} c_{ij} z^i w^j$ называется характеристическим. Степень m многочлена $P(z, w)$ по переменной w будем называть порядком разностного оператора $P(\delta_1, \delta_2)$ и предполагать, что $b < B$.

Зафиксируем $\beta = (x_\beta, m)$ такое, что $c_\beta \neq 0$, и рассмотрим множество $\Pi_\beta = \{(x, y) \in \mathbb{Z}_+^2: 0 \le x - x_\beta \le B - b, y > m - 1\}$. Обозначим $L_\beta = \Pi \backslash \Pi_\beta$ и сформулируем следующую задачу:

найти решение разностного уравнения

$$P(\delta_1, \delta_2) f(x, y) = g(x, y), \quad (x, y) \in \Pi, \qquad (2)$$

удовлетворяющее условию

$$f(x, y) = \varphi(x, y), \quad (x, y) \in L_\beta, \qquad (3)$$

где $g(x, y)$ и $\varphi(x, y)$ – заданные функции целочисленных аргументов.

Задачу (2)–(3) назовем задачей Коши для полиномиального разностного оператора (1) и приведем легко проверяемое условие ее разрешимости.

Известно (см. [9]), что задача (2)–(3) однозначно разрешима, если выполняется условие

$$|c_\beta| \ge \sum_{|\alpha|=0, \alpha \neq \beta}^{b} |c_\alpha|. \qquad (4)$$

Поставим задачу вычислить значение функции $f(x, y)$ в точке A с координатами (x_A, y_A), т.е. $f(x_A, y_A)$.

3. ОПИСАНИЕ ВХОДНЫХ ДАННЫХ

Решение задачи Коши (2)–(3) для двумерного разностного уравнения с постоянными коэффициентами в точке $z = (x_z, y_z)$ представляет собой значение функции $f(x, y)$ в этой точке.

Алгоритм вычисления значения функции $f(x, y)$ в заданной точке A с координатами (x_A, y_A) имеет рекурсивный характер и сводится к вычислению значений функции $f(x, y)$ на конечном подмножестве точек $(x, y) \in L_\beta = \Pi \backslash \Pi_\beta$.

Начальные данные (3) задаются матрицей F размерности $(B+1) \times (y_A + 1)$, содержащей конечное подмножество значений начальных данных задачи Коши.

Коэффициенты двумерного разностного уравнения задаются матрицей C, имеющей прямоугольный вид (размерность: $(m+1) \times (b+1)$).

Проиллюстрируем процедуру задания функции начальных данных на примере.

Для разностного уравнения

$$\begin{aligned} &c_{00} f(x, y) + c_{10} f(x+1, y) + c_{20} f(x+2, y) + \\ &\quad + c_{01} f(x, y+1) + c_{11} f(x+1, y+1) + \\ &\quad + c_{21} f(x+2, y+1) + c_{02} f(x, y+2) + \\ &+ c_{12} f(x+1, y+2) + c_{22} f(x+2, y+2) = 0 \end{aligned} \qquad (5)$$

и $\beta = (1, 2)$ матрица коэффициентов C имеет вид

$$C = \begin{pmatrix} c_{02} & c_{12} & c_{22} \\ c_{01} & c_{11} & c_{21} \\ c_{00} & c_{10} & c_{20} \end{pmatrix},$$

а матрица начальных данных F размерности, например, 4×5 будет иметь вид

$$F = \begin{pmatrix} \varphi(0, 3) & * & * & * & \varphi(4, 3) \\ \varphi(0, 2) & * & * & * & \varphi(4, 2) \\ \varphi(0, 1) & \varphi(1, 1) & \varphi(2, 1) & \varphi(3, 1) & \varphi(4, 1) \\ \varphi(0, 0) & \varphi(1, 0) & \varphi(2, 0) & \varphi(3, 0) & \varphi(4, 0) \end{pmatrix}.$$

Здесь элементы, обозначенные *, вычисляются при выполнении алгоритма. При этом, вычислить элемент $\varphi(1, 2)$ не представляется возможным без вычисления элемента $\varphi(2, 2)$. Таким образом, для нахождения неизвестных элементов необходимо решить систему линейных разностных уравнений вида (5) с использованием начальных данных $\varphi(i, j)$, где $i = 0, \ldots, 4$, $j = 0, 1$ и $\varphi(0, 2)$, а именно:

$$\begin{cases} c_{00}\varphi(0, 0) + c_{10}\varphi(1, 0) + c_{20}\varphi(2, 0) + \\ + c_{01}\varphi(0, 1) + c_{11}\varphi(1, 1) + c_{21}\varphi(2, 1) + \\ + c_{02}\varphi(0, 2) + c_{12} f(1, 2) + c_{22} f(2, 2) = 0, \\ c_{00}\varphi(1, 0) + c_{10}\varphi(2, 0) + c_{20}\varphi(3, 0) + \\ + c_{01}\varphi(1, 1) + c_{11}\varphi(2, 1) + c_{21}\varphi(3, 1) + \\ + c_{02} f(1, 2) + c_{12} f(2, 2) + c_{22} f(3, 2) = 0, \\ c_{00}\varphi(2, 0) + c_{10}\varphi(3, 0) + c_{20}\varphi(4, 0) + \\ + c_{01}\varphi(2, 1) + c_{11}\varphi(3, 1) + c_{21}\varphi(4, 1) + \\ + c_{02} f(2, 2) + c_{12} f(3, 2) + c_{22}\varphi(4, 2) = 0. \end{cases}$$

Все, что нам известно, перенесем вправо и получим константы:

$$\begin{cases} c_{12}f(1,2) + c_{22}f(2,2) = \text{const}_1, \\ c_{02}f(1,2) + c_{12}f(2,2) + c_{22}f(3,2) = \text{const}_2, \\ c_{02}f(2,2) + c_{12}f(3,2) = \text{const}_3. \end{cases}$$

Матрица полученной системы уравнений будет иметь вид:

$$\begin{pmatrix} c_{12} & c_{22} & 0 \\ c_{02} & c_{12} & c_{22} \\ 0 & c_{02} & c_{12} \end{pmatrix}.$$

Поскольку на всех диагоналях матрицы, параллельных главной диагонали и на самой главной диагонали, элементы матрицы одинаковые, то такая матрица является теплицевой [10].

Вернемся к входным данным. Входные данные конечны и имеют вид:

1. точка $\beta = (x_\beta, m)$;

2. прямоугольная марица коэффициентов $C = (c_\alpha)$, $\alpha = (\alpha_1, \alpha_2)$, $\alpha_1 = 0, \ldots, b$, $\alpha_2 = 0, \ldots, m$ размера $(m+1)\times(b+1)$ из коэффициентов c_α полиномиального разностного оператора (1);

3. точка A с координатами (x_A, y_A), определяющая координаты искомого значения функции $f(x, y)$ и размерность матрицы начальных данных F;

4. матрица начальных данных $F = (\varphi(x, y))$, размера $(B+1)\times(y_A+1)$, где $(x, y) \in L_\beta$, для всех остальных значений (x, y) значения $\varphi(x, y) = 0$.

Для технической реализации алгоритма целесообразно отразить матрицы коэффициентов C и начальных данных F зеркально относительно горизонтальной оси. Поскольку координаты элементов матрицы коэффициентов разностного оператора и матрицы начальных данных в декартовой системе координат (X, Y) не совпадают с их координатами в зеркально отраженной матрице (строка×столбец), то следует перейти из декартовой системы координат $(D(d_1, d_2))$ в "матричную" $(M(m_1, m_2))$ по правилу: $D(d_1, d_2) \to M(m_1, m_2)$, где $m_1 = 1 + d_2$, $m_2 = 1 + d_1$.

Например, элемент c_{00} матрицы коэффициентов C, имеющий в декартовой системе координат координаты $(d_1, d_2) = (0, 0)$, в "матричной" системе координат будет иметь координаты $(m_1, m_2) = (1, 1)$, а, например, элемент $\varphi(1, 0)$ матрицы начальных данных F в "матричной" системе координат будет иметь координаты (1, 2).

Далее необходимо будет проверить задачу Коши (2)–(3) на разрешимость, т.е. проверить, выполняется ли условие (4) для коэффициентов разностного оператора (1).

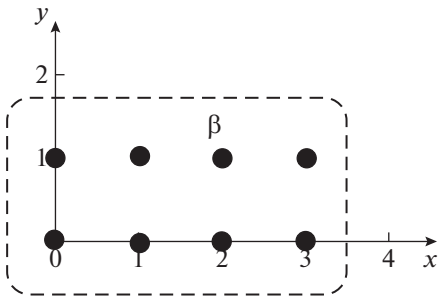


Рис. 1. Расположение элементов матрицы C в декартовой системе координат.

4. ПРИМЕР

Алгоритм был реализован в среде MatLab2014 32bit. Вычисления производились на машине Intel(R) Core(TM) i5-3330S CPU 2.70 GHz, 32bit, ОЗУ 4.00 Гб под управлением Windows 7 Корпоративная SP1. Время счета для приведенного примера составило менее 1 секунды.

В качестве примера рассмотрим задачу о распределении температуры в некотором образце. Требуется оценить распределение температуры в образце в течение некоторого времени, если заданы начальное распределение температуры в образце и изменение температуры на концах образца с течением времени. Для конкретизации задачи положим, что требуется определить температуру через четыре временных шага и четыре шага по координате.

Зафиксируем $\beta = (x_\beta, m)$, $x_\beta = 2$, $m = 1$, $b = 3$, $B = 5$.

Для полиномиального разностного оператора

$$\begin{aligned} P(\delta_1, \delta_2) = c_{00} + c_{10}\delta_1 + \\ + c_{20}\delta_1^2 + c_{30}\delta_1^3 + c_{01}\delta_2 + \\ + c_{11}\delta_1\delta_2 + c_{21}\delta_1^2\delta_2 + c_{31}\delta_1^3\delta_2 \end{aligned} \tag{6}$$

задача Коши будет иметь вид

$$\begin{aligned} c_{00}f(x, y) + c_{10}f(x+1, y) + \\ + c_{20}f(x+2, y) + c_{30}f(x+3, y) + \\ + c_{01}f(x, y+1) + c_{11}f(x+1, y+1) + \\ + c_{21}f(x+2, y+1) + \\ + c_{31}f(x+3, y+1) = 0, (x, y) \in \Pi, \\ f(x, y) = \varphi(x, y), (x, y) \in L_{(2,1)}, \end{aligned} \tag{7}$$

где $\Pi = \{(x, y) \in \mathbb{Z}^2, 0 \le x \le 5, y \ge 0\}$, $\Pi_{(2,\ 1)} = \{(x, y) \in \mathbb{Z}_+^2 : 2 \le x \le 4, y \ge 1\}$ и $L_{(2,1)} = \Pi \backslash \Pi_{(2,1)}$.

Зададим матрицу коэффициентов полиномиального разностного оператора (6)

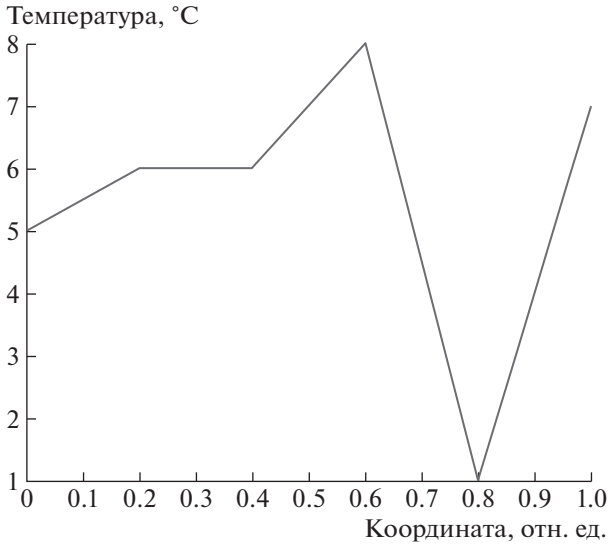


Рис. 2. Распределение температуры в образце в начальный момент времени.

$$C = \begin{pmatrix} c_{01} & c_{11} & c_{21} & c_{31} \\ c_{00} & c_{10} & c_{20} & c_{30} \end{pmatrix} = \begin{pmatrix} 1 & 3 & 10 & 1 \\ 2 & 4 & 2 & 0 \end{pmatrix}.$$

Расположение элементов матрицы C в декартовой системе координат представлено на рис. 1.

Поставим задачу: найти значения функции $f(x, y)$ в точке A с координатами (4, 4), т.е. $f(4, 4)$.

Зададим матрицу начальных данных:

$$F = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 3 \\ 2 & 3 & 0 & 0 & 0 & 4 \\ 3 & 4 & 0 & 0 & 0 & 5 \\ 4 & 5 & 0 & 0 & 0 & 6 \\ 5 & 6 & 6 & 8 & 1 & 7 \end{pmatrix}.$$

Распределение температуры в образце в начальный момент времени, т.е. последняя строка матрицы F, имеет форму, представленную на рис. 2.

Расположение элементов матрицы F в декартовой системе координат представлено на рис. 3.

1. Переход из декартовой системы координат в "матричную" осуществляется путем отражения

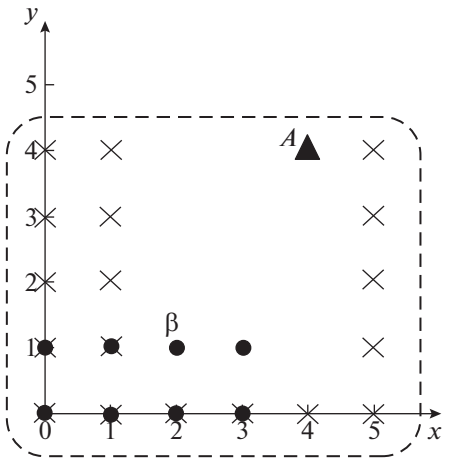


Рис. 3. Расположение элементов матрицы F в декартовой системе координат.

матриц начальных данных и коэффициентов зеркально относительно горизонтальной оси, поэтому точка β с координатами (2, 1) переходит в точку с координатами (2, 3) в "матричной" системе координат.

Пусть p – количество столбцов матрицы коэффициентов, а i – количество столбцов матрицы начальных данных. Тогда для рассматриваемого примера $p = 4$ и $i = 6$.

"Матричная" система координат для матрицы коэффициентов разностного оператора (6) имеет вид, представленный в табл. 1:

"Матричная" система координат для матрицы начальных данных F имеет вид, представленный в табл. 2:

В табл. 2 знак × – начальные данные. Точка A имеет координаты [5, 5] в "матричной" системе координат.

2. Проверка на разрешимость задачи Коши (7), т.е. выполняется ли условие (4) для коэффициентов полиномиального разностного оператора (6). Так как

$$|10| = |c_{21}| \geq |c_{01}| + |c_{11}| + |c_{31}| = 5,$$

то условие (4) выполняется и задача Коши разрешима.

Таблица 1

	1	2	3	4
1	•	•	•	•
2	•	•	β_0	•

Здесь • – элементы матрицы C.

Таблица 2

	1	2	3	4	5	6
1	×	×	×	×	×	×
2	×	×				×
3	×	×				×
4	×	×				×
5	×	×			A	×

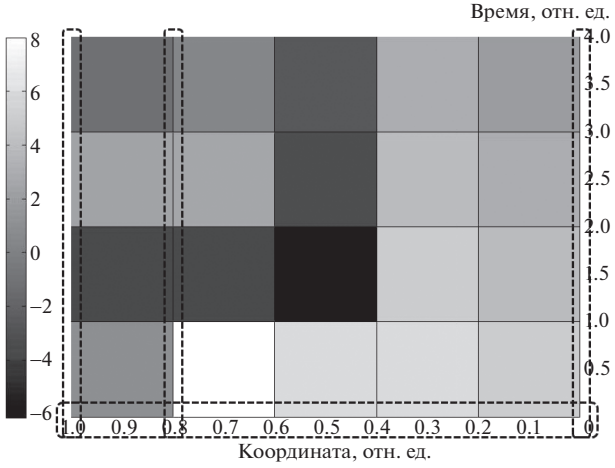


Рис. 4. Изменение температуры в образце в течение 4 временных шагов (пунктиром выделено множество начальных данных).

3. Зеркальное отражение относительно горизонтальной оси матрицы коэффициентов C и матрицы начальных данных F:

$$C_{work} = \begin{pmatrix} 2 & 4 & 2 & 0 \\ 1 & 3 & 10 & 1 \end{pmatrix},$$

$$F_{work} = \begin{pmatrix} 5 & 6 & 6 & 8 & 1 & 7 \\ 4 & 5 & 0 & 0 & 0 & 6 \\ 3 & 4 & 0 & 0 & 0 & 5 \\ 2 & 3 & 0 & 0 & 0 & 4 \\ 1 & 2 & 0 & 0 & 0 & 3 \end{pmatrix}.$$

4. Последняя строка в матрице C_{work}:

$$a = (c_{01}\ \ c_{11}\ \ c_{21}\ \ c_{31})^{\mathrm{T}} = (1\ 3\ 10\ 1)^{\mathrm{T}}$$

5. Получение общей Теплицевой матрицы [10, 11], размер которой зависит от искомого значения.

Для построения теплицевой матрицы необходимы два вектора $column_c$ и row_c. Пусть матрица $neizv_1$ – несимметричная Теплицева матрица размера $(i-p+1)\times(i-p+1)$ (в рассматриваемом примере 3 × 3), построенная на основе правила, что элементы вектора $column_c$ – ее первый столбец, а элементы вектора row_c – ее первая строка, причем первые элементы векторов $column_c$ и row_c совпадают. В случае, если количество элементов в векторах $column_c$ и row_c меньше $(i-p+1)$, то вектор меньшего размера дополняется нулями. Элементы векторов $column_c$ и row_c берутся из вектора a. В вектор $column_c$ попадают элементы, начиная с $c_\beta = c_{21}$ в порядке уменьшения номера элемента, т.е. $column_c = (c_{21}\ \ c_{11}\ \ c_{01})^{\mathrm{T}} = (10\ 3\ 1)^{\mathrm{T}}$, а в вектор row_c попадают элементы, начиная $c_\beta = c_{21}$ в порядке увеличения номера элемента, т.е. $row_c = (c_{21}\ \ c_{31}) = (10\ 1)$. Поскольку длина вектора row_c меньше $(i-p+1)$, то дополним его нулем: $row_c = (10\ 1\ 0)$. Таким образом, матрица $neizv_1$ имеет вид

$$neizv_1 = \begin{pmatrix} 10 & 1 & 0 \\ 3 & 10 & 1 \\ 1 & 3 & 10 \end{pmatrix}.$$

6. Решение системы линейных уравнений с целью нахождения значений $f(x,y)$, которые могут потребоваться для вычисления искомого значения в точке $A(x_A, y_A)$.

Количество неизвестных для каждого значения m (т.е. номера строки матрицы F в декартовой системе координат), показывающего номер “слоя”, на каждом шаге алгоритма одинаково. Поэтому размер матрицы Теплица остается постоянным. На каждом шаге алгоритма происходит запись найденных значений функции $f(x,y)$ в матрицу начальных данных F.

$$F = \begin{pmatrix} 5 & 6 & 6 & 8 & 1 & 7 \\ 4 & 5 & -6.15 & -3.50 & -3.53 & 6 \\ 3 & 4 & -3.32 & 2.51 & 2.42 & 5 \\ 2 & 3 & -2.70 & 0.35 & -1.15 & 4 \\ 1 & 2 & -1.83 & 0.68 & 0.19 & 3 \end{pmatrix}.$$

На рис. 4 представлено итоговое распределение температуры в образце в течение четырех временных шагов.

7. Переход к декартовой системе координат $f(4,4) = F[5, 5]$.

Output: значение функции $f(x,y)$ в искомой точке $A(4, 4)$ равно 0.19.

Входными данными для работы алгоритма в этом случае будут:

1. $\beta = (2, 1)$;

2. матрица коэффициентов

$$C = \begin{pmatrix} 1 & 3 & 10 & 1 \\ 2 & 4 & 2 & 0 \end{pmatrix};$$

3. $A = (4, 4)$;

4. матрица начальных данных

$$F = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 3 \\ 2 & 3 & 0 & 0 & 0 & 4 \\ 3 & 4 & 0 & 0 & 0 & 5 \\ 4 & 5 & 0 & 0 & 0 & 6 \\ 5 & 6 & 6 & 8 & 1 & 7 \end{pmatrix}.$$

Алгоритм 1: Пример оформления алгоритма

Input: Точка β, матрица коэффициентов C, точка A, матрица начальных данных F.

Output: Значение функции $f(x, y)$ в точке A с координатами (x_A, y_A).

begin

i := количество столбцов матрицы F

p := количество столбцов матрицы C

w := количество строк матрицы C

$neizv_1$:= теплицевая матрица размера $(i - p + 1) \times (i - p + 1)$

v := количество строк в матрице $neizv_1$

β_1 := координаты точки β в “матричной” системе координат

A_1 := координаты точки A в “матричной” системе координат

if $|C(\beta_1(1), \beta_1(2))| \le \sum(|C(\beta_1(1), :)|) - |C(\beta_1(1), \beta_1(2))|$

then

  return Ошибка ввода матрицы C

C_{work} := матрица C, зеркально отраженная относительно горизонтальной оси

F_{work} := матрица F, зеркально отраженная относительно горизонтальной оси

a := последняя строка в матрице C_{work}

$e := \beta_1(2)$

$column_c := zeros(i - (p - 1), 1)$

$col := 1$

while $e \ge 1$ **do**

  $column_c(col) := a(e)$

  $col := col + 1$

  $e := e - 1$

$row_c := zeros(l, i - (p - 1))$

$e := \beta_1(2)$

$r := 1$

for h *from* $\beta_1(2)$ *to* l **do**

  $row_c(r) = a(e)$

  $r := r + 1$

  $e := e + 1$

$neizv_1 := toeplitz(column_c, row_c)$

for k *from* $\beta_1(1)$ *to* $A_1(1)$ **do**

  $b := zeros(v, 1)$

  for i *from* 0 *to* $v - 1$ **do**

    $P := F_{work}((k - (w - 1)) : k, (1 + i) : i + p)$

    $P_{work} := P \circ C_{work}$

    $(b(i + 1) = -(sum(sum(P_{work})))$

  $elements := neizv^{-1}.b$

  for s *from* 1 *to* v **do**

    $F_{work}(k, e - 1 + s) := elements(s)$

  return $f(x_A, y_A)$

БЛАГОДАРНОСТИ

Работа поддержана Красноярским математическим центром, финансируемым Минобрнауки РФ в рамках мероприятий по созданию и развитию региональных НОМЦ (Соглашение 075-02-2022-876).

СПИСОК ЛИТЕРАТУРЫ

1. *Даджион Д., Мерсеро Р.* Цифровая обработка многомерных сигналов. М.: Мир, 1988.
2. *Изерман Р.* Цифровые системы управления. М.: Мир, 1984. 541 с.
3. *Рябенький В.С.* Введение в вычислительную математику: учеб. пособие, изд. 2-е, исправл. М. : ФИЗМАТЛИТ, 2000. 296 с.
4. *Стенли Р.* Перечислительная комбинаторика: Пер. с англ. М.: Мир, 1990. 440 с.
5. *Стенли Р.* Перечислительная комбинаторика. Деревья, производящие функции и симметрические функции: Пер. с англ. М.: Мир, 2005. 767 с.
6. *Ляпин А.П.* Последовательности Риордана и двумерные разностные уравнения // Журн. СФУ. Сер. Матем. и физ. 2009. Т. 2. Вып. 2. С. 210–220.
7. *Кытманов А.А., Ляпин А.П., Садыков Т.М.* Алгоритм вычисления рациональной производящей функции решения задачи Коши двумерного разностного уравнения с постоянными коэффициентами // Программирование. 2017. Вып. 2. С. 54–62.
8. *Апанович М.С., Ляпин А.П., Шадрин К.В.* Применение методов компьютерной алгебры для вычисления решения задачи Коши для двумерного разностного уравнения в точке // Программирование. 2021. Вып. 1. С. 1–5.
9. *Рогозина М.С.* Разрешимость разностной задачи Коши для многослойных неявных разностных схем // Вестник СибГАУ. Математика, механика, информатика. 2014. Т. 55. № 3. С. 126–130.
10. *Иохвидов И.С.* Ганкелевы и теплицевы матрицы. М.: Наука, 1974. 264 с.
11. *Апанович М.С., Ляпин А.П., Шадрин К.В.* Вычисление последовательности главных миноров теплицевой ленточной матрицы // Прикладная математика и физика. 2020. Т. 52. № 1. С. 5–10.

EDN: KVSZRU
ПРОГРАММИРОВАНИЕ, 2022, № 4, с. 57–64

КОМПЬЮТЕРНАЯ АЛГЕБРА

УДК 512.547.2:530.145.81

РАЗЛОЖЕНИЕ ЗАМКНУТОЙ КВАНТОВОЙ СИСТЕМЫ НА ПОДСИСТЕМЫ В КОНЕЧНОЙ КВАНТОВОЙ МЕХАНИКЕ

© 2022 г. В. В. Корняк[a],* (ORCID: 0000-0002-5712-2960)

[a] *Объединенный институт ядерных исследований, 141980 Дубна Московской области, Россия*

*****E-mail: vkornyak@gmail.com*

Поступила в редакцию 17.06.2021 г.
После доработки 29.08.2021 г.
Принята к публикации 11.09.2021 г.

Любое гильбертово пространство с составной размерностью можно представить в виде тензорного произведения гильбертовых пространств меньших размерностей. Такая факторизация дает возможность разложить квантовую систему на подсистемы. Мы предлагаем модель, основанную на конечной квантовой механике, для конструктивного изучения разложений изолированной квантовой системы на подсистемы. Для исследования поведения составных систем, получаемых в результате разложений, мы разрабатываем алгоритмы, основанные на методах компьютерной алгебры и вычислительной теории групп.

DOI: 10.31857/S0132347422020078

ВВЕДЕНИЕ

Мереологией называют изучение отношений части к целому и отношений между частями внутри целого. В *квантовой мереологии* целое – это замкнутая квантовая система (“Вселенная”)[1] в заданном чистом состоянии, претерпевающая заданную унитарную (шредингеровскую) эволюцию.

Квантовая мереология изучает взаимосвязи между выделенными подсистемами Вселенной (“наблюдаемая система”, “наблюдатель”, “измерительный прибор”, “окружающая среда” и т.д.), возникновение геометрии и даже времени (механизм Пейджа–Вуттерса [1]) из квантовой запутанности и другие фундаментальные вопросы квантовой механики [2–4].

Разделение целого на части по своей сути произвольно – оно зависит от используемых критериев разделения. Имеется два различных аспекта разделимости квантовых систем.

1. Квантовые системы отделены друг от друга, если энергия взаимодействия между ними мала. Это более зримый, материальный критерий, хорошо согласующийся с обычной концепцией локальности. Количественно энергию взаимодействия между подсистемами A и B можно представить в виде

$$\Delta E(A,B) = E(A\cup B) - E(A) - E(B). \qquad (1.1)$$

2. Подсистемы A и B разделены, если квантовые корреляции между ними малы. Этот более тонкий критерий, имеющий нелокальные проявления. Количественно квантовую запутанность между подсистемами можно описать, например, *взаимной информацией*

$$\mathcal{I}(A,B) = S(A) + S(B) - S(A\cup B), \qquad (1.2)$$

где S обозначает энтропию.

Между выражениями (1.1) и (1.2) имеется определенное структурное сходство. Однако они описывают совершенно разные типы связей между подсистемами.

Например, в модели эмерджентного времени Пейджа–Вуттерса предполагается, что вся вневременная Вселенная разделена на две подсистемы: “часы”, C, и остальную часть Вселенной, R. Гамильтониан Вселенной в этой модели имеет вид

$$H = H_C \otimes 1_R + 1_C \otimes H_R,$$

что предполагает нулевую энергию взаимодействия между C и R. С другой стороны, предполагается наличие нетривиальных квантовых корреляций между C и R.

Было бы интересно внимательнее взглянуть на взаимосвязи между этими двумя различными – энергетическим и информационным – аспектами квантовой разделимости.

[1] В точном смысле замкнутой системой может быть только Вселенная в целом, в противном случае понятие замкнутой системы является приближенным.

Мы разрабатываем и реализуем алгоритмы, основанные на методах компьютерной алгебры и вычислительной теории групп, для выполнения следующих задач. Изолированная квантовая система, построенная в рамках конечной квантовой механики, разлагается в тензорное произведение подсистем. С помощью редукции чистого квантового состояния “Вселенной”, мы получаем смешанные состояния для подсистем. В результате у нас появляется возможность изучать энергии взаимодействий и квантовые корреляции между подсистемами, а также эволюции этих величин во времени.

2. РАЗЛОЖЕНИЕ КВАНТОВОЙ СИСТЕМЫ

2.1. Тензорное произведение гильбертовых пространств

Глобальным гильбертовым пространством K-компонентной квантовой системы является тензорное произведение локальных гильбертовых пространств компонент:

$$\mathcal{H} = \bigotimes_{k=1}^{K} \mathcal{H}_k. \tag{2.1}$$

Если $\dim \mathcal{H} = \mathcal{N}$ и $\dim \mathcal{H}_k = d_k$, тогда $\mathcal{N} = \prod_{k=1}^{K} d_k$.

Для любого d-мерного гильбертова пространства i-й элемент ортонормального базиса будет обозначаться символом $|i\rangle$, т.е., $|0\rangle = (1, 0, \ldots)^\top$, $|1\rangle = (0, 1, 0, \ldots)^\top$, ..., $|d-1\rangle = (0, 0, \ldots, 1)^\top$.

Тензорные мономы базисных элементов локальных пространств образуют ортонормальный базис в глобальном гильбертовом пространстве:

$$|i\rangle = |i_1\rangle \otimes \cdots \otimes |i_k\rangle \otimes \cdots \otimes |i_K\rangle, \tag{2.2}$$

где $|i\rangle \in \mathcal{H}$, $|i_k\rangle \in \mathcal{H}_k$ и

$$i = \sum_{k=1}^{K} i_k \prod_{m=k+1}^{K} d_m. \tag{2.3}$$

2.2. Тензорная факторизация гильбертова пространства

Мы можем обратить процедуру, поскольку (2.2) – взаимно однозначное соответствие: последовательность $i_1, \ldots, i_K$ однозначно восстанавливается из i с помощью простого алгоритма:

$$\begin{aligned} &k \leftarrow K, \quad \tilde{i} \leftarrow i \\ &\textbf{while } k \geq 1 \textbf{ do} \\ &\quad i_k \leftarrow \tilde{i} \bmod d_k, \quad \tilde{i} \leftarrow \lfloor \tilde{i}/d_k \rfloor \\ &\quad k \leftarrow k-1 \end{aligned} \tag{2.4}$$

Выбрав ортонормальный базис в пространстве $\mathcal{H}$ и разложение его размерности на множители $\mathcal{N} = d_1 \cdots d_K$, мы можем построить конкретную биекцию вида (2.1).

Для построения произвольной биекции необходимо учесть свободу выбора базисов в гильбертовых пространствах: любой ортонормальный базис можно перевести в любой другой унитарным преобразованием.

Легко показать, что произвольный набор унитарных замен базисов во всех пространствах, фигурирующих в соотношении (2.1), эквивалентен единственной замене базиса в глобальном пространстве.

Более конкретно, любой вектор в глобальном пространстве можно представить в виде суммы тензорных произведений элементов локальных пространств

$$|\psi\rangle = \sum_{\ell} \bigotimes_{k=1}^{K} |\psi_k^\ell\rangle, \quad |\psi_k^\ell\rangle \in \mathcal{H}_k, \quad |\psi\rangle \in \mathcal{H}. \tag{2.5}$$

Применив унитарные преобразования ко всем векторам в (2.5) и воспользовавшись свойствами тензорного произведения, мы имеем

$$U|\psi\rangle = \sum_{\ell} \bigotimes_{k=1}^{K} U_k |\psi_k^\ell\rangle = \bigotimes_{k=1}^{K} U_k \sum_{\ell} \bigotimes_{k=1}^{K} |\psi_k^\ell\rangle$$

$$\Downarrow$$

$$U'|\psi\rangle = \sum_{\ell} \bigotimes_{k=1}^{K} |\psi_k^\ell\rangle, \quad \text{где} \quad U' = \left(\bigotimes_{k=1}^{K} U_k \right)^{-1} U.$$

Таким образом, тензорная факторизация гильбертова пространства $\mathcal{H}$ полностью характеризуется следующими двумя установками[2]:

1. *разложением размерности* на множители $\dim \mathcal{H} = d_1 \cdots d_K$,

2. и *унитарным преобразованием* U, фиксирующим базис в глобальном гильбертовом пространстве $\mathcal{H}$.

2.3. Декомпозиция чистого квантового состояния

Любое смешанное состояние квантовой системы можно получить с помощью *разложения Шмидта* [7] чистого состояния в гильбертовом пространстве большей размерности, которое можно сконструировать с помощью процедуры называемой расширением до *чистого состояния*. Естественно предположить, что при фундаментальном подходе состояние изолированной системы должно быть чистым[3].

[2] Другой подход, в котором факторизация гильбертова пространства задается алгеброй наблюдаемых, был предложен в [5, 6].

Для данной факторизации $\mathcal{H} = \mathcal{H}_1 \otimes \cdots \otimes \mathcal{H}_K$ мы вводим множество индексов (которые удобно представлять себе как *"(пре)геометрические точки"*)

$$X = \{1, \ldots, K\}.$$

Подсистемы квантовой системы отождествляются с подмножествами $A \subseteq X$. Матрица плотности чистого состояния $|\psi\rangle \in \mathcal{H}$ глобальной системы представляет собой проектор ранга один вида

$$\rho_X = \frac{|\psi\rangle\langle\psi|}{\langle\psi \mid \psi\rangle}.$$

В соответствии с законами квантовой механики статистическое поведение подсистемы A правильно описывается *редуцированной матрицей плотности* $\rho_A = \mathrm{tr}_{X\setminus A}\rho_X$, которая вычисляется из глобальной матрицы плотности взятием частичного следа по дополнению к A.

Более подробно, вычисление редуцированной матрицы плотности сводится к следующему. Согласно (2.2) базис глобального гильбертова пространства можно представить как декартово произведение локальных базисов

$$B_X = \prod_{k\in X} B_k.$$

Аналогичным образом мы вводим множества

$$B_A = \prod_{k\in A} B_k \quad \text{и} \quad B_{X\setminus A} = \prod_{k\in X\setminus A} B_k.$$

В компонентах глобальную матрицу плотности можно написать в виде

$$\rho_X = (\rho_X)_{i_X j_X} |i_X\rangle\langle j_X|,$$

где $i_X \simeq \{i_1, \ldots, i_K\}$, $j_X \simeq \{j_1, \ldots, j_K\} \in B_X$, а эквивалентность $\simeq$ обеспечивается формулой (2.3) и алгоритмом (2.4). Процедуру вычисления редуцированной матрицы плотности легко реализовать, запрограммировав формулу

$$(\rho_A)_{i_A j_A} = \sum_{\substack{m_{X\setminus A}\in B_{X\setminus A} \\ i_X = i_A \sqcup m_{X\setminus A} \\ j_X = j_A \sqcup m_{X\setminus A}}} (\rho_X)_{i_X j_X}, \quad \text{где} \quad i_A, j_A \in B_A.$$

3. КОНЕЧНАЯ КВАНТОВАЯ МЕХАНИКА

Мы используем версию квантовой теории [9–11] в которой группы Ли унитарных эволюций заменяются линейными представлениями конечных групп, а поле комплексных чисел заменяется его плотными конструктивными подполями, которые естественным образом строятся из натуральных чисел и корней из единицы. Такая модификация квантового формализма "*can accurately reproduce all of the results of conventional quantum mechanics*" [12].

3.1. Перестановочное гильбертово пространство

Любое линейное представление конечной группы – в силу простого математического факта такое представление всегда унитарно – является подпредставлением некоторого перестановочного представления. Из этого следует, что формализм квантовой механики можно полностью[4] воспроизвести исходя из перестановок некоторого множества

$$\Omega = \{e_1, \ldots, e_{\mathcal{N}}\} \cong \{1, \ldots, \mathcal{N}\} \tag{3.1}$$

первичных (*"онтических"*) объектов, на котором действует некоторая подгруппа G симметрической группы $\mathrm{S}_{\mathcal{N}}$.

Гильбертово пространство на множестве Ω, необходимое для вычислений в квантовой теории, можно наиболее экономным образом построить используя только два примитивных понятия:

1. *натуральные числа* $\mathbb{N} = \{0, 1, \ldots\}$ – абстракция *счета*,

2. *корни из единицы* – абстракция *периодичности*.

Поле $\mathcal{F}$, достаточное для всех потребностей квантового формализма – в частности для расщепления любого представления любой подгруппы группы G на неприводимые компоненты – можно построить, добавив к натуральным числам примитивный корень из единицы ℓ-й степени ζ_ℓ, где ℓ – наименьшее общее кратное периодов (порядков) всех элементов группы G, называемое *экспонентой* группы.

Примитивный корень из единицы ζ_ℓ является *алгебраическим целым*, поскольку он является корнем полинома $\Phi_\ell(x)$ с целыми коэффициентами и единичным старшим коэффициентом. $\Phi_\ell(x)$ называется ℓ-м *циклотомическим полиномом* (или ℓ-м *многочленом деления круга*). Алгебраическое целое ζ_ℓ можно записать в виде комплексного числа: $\zeta_\ell = e^{2\pi\mathrm{i}k/\ell}$, где натуральное число $k < \ell$ является взаимно простым с ℓ. Мы будем всегда подразумевать, что $k = 1$, а соответствующий примитивный корень из единицы $\zeta_\ell = e^{2\pi\mathrm{i}/\ell}$ называть *основным*[5].

[3] Это убеждение – проявление бритвы Оккама, выражаемое метафорой "Церковь Большего Гильбертова Пространства" (J.A. Smolin), позволяет получить вероятности всех типов, возникающих в квантовой теории, из единственной фундаментальной вероятности, которая фигурирует в теореме Глисона [8] и соответствует правилу Борна.

[4] С точностью до эмпирически несущественных элементов традиционного формализма, главным образом разного рода бесконечностей.

[5] Это название мотивировано физическим термином *основное состояние* квантовомеханической системы.

Добавив ζ_ℓ к натуральным числам, мы расширяем полукольцо $\mathbb{N}$ в кольцо $\mathbb{N}[\zeta_\ell]$ (исключив тривиальный случай $\ell = 1$). Далее, построив *поле частных* кольца $\mathbb{N}[\zeta_\ell]$, мы получаем циклотомическое расширение поля рациональных чисел $\mathcal{F} = \mathbb{Q}(\zeta_\ell)$. При $\ell > 2$ поле $\mathcal{F}$, будучи плотным подполем комплексного поля $\mathbb{C}$, эмпирически неотличимо от $\mathbb{C}$.

Интерпретируя множество Ω как базис, мы приходим к $\mathcal{N}$-мерному гильбертову пространству $\mathcal{H}_\mathcal{N}$ над полем $\mathcal{F}$. Действие G на Ω определяет *перестановочное представление* $\mathcal{P}$ в пространстве $\mathcal{H}_\mathcal{N}$ матрицами вида

$$\mathcal{P}(g)_{i,j} = \delta_{ig,j}, \tag{3.2}$$

где ig обозначает действие (справа) элемента группы $g \in G$ на элемент множества $i \in \Omega$.

3.2. Разложение перестановочного представления

Перестановочное представление любой группы имеет *тривиальное* одномерное подпредставление в пространстве, состоящем из векторов с равными компонентами. В качестве базисного элемента тривиального подпространства мы будем использовать вектор

$$|\omega\rangle = (\underbrace{1,1,\ldots,1}_{\mathcal{N}})^{\mathrm{T}}.$$

Дополнение к тривиальному подпредставлению называется *стандартным представлением*. Оператор проектирования в $(\mathcal{N}-1)$-одномерное *стандартное пространство* $\mathcal{H}_\star$ имеет вид

$$\mathrm{P}_\star = 1_\mathcal{N} - \frac{|\omega\rangle\langle\omega|}{\mathcal{N}}.$$

Квантовомеханическое поведение (интерференция и т.д.) проявляется именно в стандартном пространстве $\mathcal{H}_\star$. Том Бэнкс сделал глубокое наблюдение [12], что проекция классических перестановочных эволюций во всем пространстве $\mathcal{H}_\mathcal{N}$ приводит к истинно квантовым эволюциям в его подпространстве $\mathcal{H}_\star < \mathcal{H}_\mathcal{N}$. Бэнкс также показал, что наиболее естественным выбором является полная группа всех перестановок онтических элементов $G = \mathrm{S}_\mathcal{N}$, где $\mathcal{N}$ – число фундаментальных (планковских) элементов реальности[6].

Для пояснения соответствия между конечной квантовой механикой и традиционной теорией, основанной на непрерывных унитарных группах, Бэнкс указал на наличие связи между симметрической группой на $\mathcal{N}$ элементах и унитарной группой в размерности $\mathcal{N}-1$. А именно, для достаточно большого[7] числа $\mathcal{N}$ наиболее общая конечная подгруппа G группы $\mathrm{SU}(\mathcal{N}-1)$ имеет структуру полупрямого произведения абелевой группы A и симметрической группы $\mathrm{S}_\mathcal{N}$

$$G = A \rtimes \mathrm{S}_\mathcal{N} < \mathrm{SU}(\mathcal{N}-1).$$

3.3. Онтические состояния

$\mathrm{S}_\mathcal{N}$ – *рационально представляемая* группа, т.е., любое ее неприводимое представление (в частности, стандартное) можно реализовать над $\mathbb{Q}$ (или, эквивалетно, над $\mathbb{Z}$). Это означает, что для описания квантовых состояний в $\mathcal{H}_\star$ достаточно только векторов с рациональными компонентами[8].

Легко показать, что любое квантовое состояние в $\mathcal{H}_\star$ можно получить как проекцию целочисленного вектора из неотрицательного ортанта $\mathcal{H}^+_\mathcal{N} \subset \mathcal{H}_\mathcal{N}$. Пусть $|x\rangle = (x_1, \ldots, x_\mathcal{N})^{\mathrm{T}} \in \mathcal{H}^+_\mathcal{N}$ – вектор с неотрицательными рациональными компонентами. Тогда его проекция на $\mathcal{H}_\star$ является $(\mathcal{N}-1)$-мерным вектором вида

$$|y\rangle = (y_1, \ldots, y_{\mathcal{N}-1})^{\mathrm{T}} = \mathrm{P}_\star |x\rangle. \tag{3.3}$$

Удобным для наших целей базисом в $\mathcal{H}_\star$ является множество

$$\{|0\rangle - |\mathcal{N}-1\rangle, \ldots, |\mathcal{N}-2\rangle - |\mathcal{N}-1\rangle\}, \tag{3.4}$$

где $\{|0\rangle, \ldots, |\mathcal{N}-1\rangle\}$ – базис в $\mathcal{H}_\mathcal{N}$. В базисе (3.4), соотношение (3.3) в компонентах имеет вид

$$y_1 = x_1 - x_\mathcal{N},$$
$$\vdots$$
$$y_{\mathcal{N}-1} = x_{\mathcal{N}-1} - x_\mathcal{N}.$$

Очевидно, что *любой* набор величин $y_1, \ldots, y_{\mathcal{N}-1}$ можно получить, используя только *неотрицательные* величины $x_1, \ldots, x_\mathcal{N}$.

Поскольку квантовые состояния являются лучами в гильбертовом пространстве, мы можем заменить неотрицательные рациональные векторы $|x\rangle$ натуральными векторами

$$|n\rangle = (n_1, \ldots, n_\mathcal{N})^{\mathrm{T}} \in \mathbb{N}^\mathcal{N} \subset \mathcal{H}^+_\mathcal{N}.$$

[6] По текущим космологическим данным $\mathcal{N} \sim \mathrm{Exp}(\mathrm{Exp}(20))$ для 1 см^3 вещества и $\mathcal{N} \sim \mathrm{Exp}(\mathrm{Exp}(123))$ для всей Вселенной.

[7] В [13] приведена точная нижняя граница $\mathcal{N} \geq 72$.

[8] Потребность в комплексных числах – нетривиальных элементах циклотомических расширений – может возникнуть только в задачах, где фигурируют некоторые собственные подгруппы симметрической группы $\mathrm{S}_\mathcal{N}$. Типичный пример – циклические группы, неприводимые представления которых (за исключением $\mathrm{Z}_2 \simeq \mathrm{S}_2$) невозможно получить без комплексных чисел.

Чтобы строить конструктивные модели, необходимо выбрать конечное подмножество в $\mathbb{N}^{\mathcal{N}}$. Простейшим выбором являются векторы с координатами из множества {0, 1}, т.е., битовые строки длины $\mathcal{N}$, которые мы будем называть *онтическими векторами* или *онтическими состояниями*. Эти состояния привлекательны как по онтологическим, так и по вычислительным причинам.

Интерпретируя онтическое состояние $|q\rangle$ как характеристическую функцию, мы можем отождествить его с подмножеством $q \subset \Omega$ или, эквивалентно, с разбиением онтического множества (3.1) на два нетривиальных подмножества

$$\Omega = q \sqcup \sim q, \quad \sim q = \Omega \backslash q,$$

где $\sim$ обозначает *теоретико-множественное дополнение* (или *побитовую инверсию*). Полное множество онтических состояний имеет вид

$$Q = 2^{\Omega} \backslash \{\emptyset, \Omega\}.$$

Число онтических состояний экспоненциально зависит от $\mathcal{N}$: $|Q| = 2^{\mathcal{N}} - 2$. Поэтому для больших $\mathcal{N}$ они порождают достаточно большие множества квантовых состояний в стандартном пространстве и его подпространствах.

Операция дополнения, примененная к онтическому состоянию, вызывает изменение знака соответствующего квантового состояния в стандартном пространстве:

$$|\psi\rangle = \mathrm{P}_\star |q\rangle \Rightarrow |\psi\rangle = \mathrm{P}_\star |\sim q\rangle.$$

Скалярное произведение нормализованных проекций онтических векторов $|q\rangle$ и $|r\rangle$ в пространство $\mathcal{H}_\star$ имеет вид

$$S(q,r) \equiv \frac{\langle q|\mathrm{P}_\star|r\rangle}{\sqrt{\langle q|\mathrm{P}_\star|q\rangle\langle r|\mathrm{P}_\star|r\rangle}} = \frac{\mathcal{N}\langle q \,\&\, r\rangle - \langle q\rangle\langle r\rangle}{\sqrt{\langle q\rangle\langle \sim q\rangle\langle r\rangle\langle \sim r\rangle}},$$

где & – побитовое И для битовых строк, а $\langle\cdot\rangle$ обозначает *вес Хэмминга* (*число заполнения*).

Из очевидных тождеств $\langle \sim a\rangle = \mathcal{N} - \langle a\rangle$ и $\langle a \,\&\, b\rangle + \langle a \,\&\, \sim b\rangle = \langle a\rangle$ легко выводятся следующие симметрии относительно операций дополнения, примененных к онтическим состояниям

$$S(q,r) = -S(\sim q, r) = -S(q, \sim r) = S(\sim q, \sim r).$$

4. ОНТИЧЕСКИЙ И ЭНЕРГЕТИЧЕСКИЙ БАЗИСЫ

4.1. Онтический базис

Исходный перестановочный базис в пространстве $\mathcal{H}_{\mathcal{N}}$, т.е. множество Ω, мы будем называть *онтическим базисом*. В этом базисе матрица плотности онтического состояния $|q\rangle \in \mathcal{H}_{\mathcal{N}}$ в пространстве $\mathcal{H}_\star$ имеет вид

$$\rho_q^o = \frac{1}{\mathcal{N}} \frac{(|q\rangle - \alpha|\omega\rangle)(\langle q| - \alpha\langle\omega|)}{\alpha(1-\alpha)}, \tag{4.1}$$

где $\alpha = \langle q\rangle / \mathcal{N}$ – *плотность заполнения*.

Имеется очевидная двойственность: выражение для матрицы плотности дополнения к онтическому состоянию $|q\rangle$ получается из (4.1) заменами $q \to \sim q$ и $\alpha \to 1 - \alpha$

$$\rho_q^o \xrightarrow[\alpha \to 1-\alpha]{q \to \sim q} \rho_{\sim q}^o.$$

Заметим, что онтический базис не зависит ни от состояния, ни от эволюции квантовой системы, а полностью определяется числом $\mathcal{N}$. Поэтому его можно рассматривать как прообраз, из которого любой другой базис можно получить подходящим унитарным преобразованием.

4.2. Энергетический базис

В континуальной квантовой механике эволюция изолированной квантовой системы описывается однопараметрической унитарной группой $U_t = e^{-\mathrm{i}Ht}$, порождаемой гамильтонианом H. Собственные значения гамильтониана называются *энергиями*.

В конечной квантовой механике эволюция описывается циклической группой $U(g)^t$, порождаемой элементом $U(g) \in \mathcal{P}(G)$, где $\mathcal{P}(G)$ – перестановочное представление группы G, определяемое формулой (3.2), а время t – целочисленный параметр. Мы называем *энергетическим базисом* ортонормальный базис в пространстве $\mathcal{H}_{\mathcal{N}}$, в котором матрица $U(g)$ диагональна. Этот базис зависит от заданной эволюции.

Формула Планка $E = h\nu$ выражает связь энергии E с частотой ν – величиной, обратной периоду соответствующего циклического процесса.

Любую перестановку можно представить как произведение непересекающихся циклов. Поучительно посмотреть, как часто циклы разной длины встречаются в группе всех перестановок $\mathrm{S}_{\mathcal{N}}$. Простое комбинаторное вычисление показывает, что общее число циклов длины ℓ во всей симметрической группе $\mathrm{S}_{\mathcal{N}}$ равно $\mathcal{N}!/\ell$, и, следовательно, ожидаемое число ℓ-циклов в одной перестановке равно $1/\ell$. То есть в нашей модели Вселенной, основанной на перестановках, преобладают эволюции с высокими энергиями[9].

Матрица цикла длины ℓ имеет вид

[9] Конечно, более адекватным было бы вычислить распределение энергий для данной индивидуальной перестановочной эволюции, но это более трудная комбинаторная задача.

$$C_\ell = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix},$$

т.е. $(C_\ell)_{ij} = \delta_{i-j+1(\mathrm{mod}\,\ell)}$. Эта матрица имеет следующую диагональную форму

$$F_\ell C_\ell F_\ell^{-1} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & \zeta_\ell & 0 & \cdots & 0 \\ 0 & 0 & \zeta_\ell^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \zeta_\ell^{\ell-1} \end{pmatrix},$$

где $\zeta_\ell = e^{2\pi \mathrm{i}/\ell}$ – ℓ-й основной примитивный корень из единицы, а

$$F_\ell = \frac{1}{\sqrt{\ell}} \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \zeta_\ell^{-1} & \zeta_\ell^{-2} & \ldots & \zeta_\ell^{-(\ell-1)} \\ 1 & \zeta_\ell^{-2} & \zeta_\ell^{-4} & \ldots & \zeta_\ell^{-2(\ell-1)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \zeta_\ell^{-(\ell-1)} & \zeta_\ell^{-2(\ell-1)} & \ldots & \zeta_\ell^{-(\ell-1)(\ell-1)} \end{pmatrix}$$

– матрица *преобразования Фурье*. Матрица F_ℓ является одновременно унитарной и симметрической, что позволяет легко написать обратную матрицу

$$F_\ell^{-1} = F_\ell^\dagger = F_\ell^* \Rightarrow (F_\ell^{-1})_{ij} = \frac{1}{\sqrt{\ell}} \zeta_\ell^{(i-1)(j-1)}.$$

В общем случае матрица перестановочного представления элемента $g \in \mathrm{S}_\mathcal{N}$ является прямой суммой циклических матриц $U(g) = \bigoplus_{m=1}^{M} C_{\ell_m}$, а соответствующая диагонализирующая матрица – матрица перехода от онтического к энергетическому базису – имеет вид $F = \bigoplus_{m=1}^{M} F_{\ell_m}$.

Матрицу плотности онтического состояния $|q\rangle$ в энергетическом базисе можно вычислить из (4.1) по формуле

$$\rho_q^\varepsilon = F \rho_q^o F^*.$$

4.3. Замечание о базисах

Как показано в разделе 2.2, выбор базиса в гильбертовом пространстве замкнутой квантовой системы имеет решающее значение для выделения подсистем в рассматриваемой системе.

Наиболее универсальным является онтический базис, не требующий никаких данных, характеризующих квантовую систему, кроме числа онтических элементов. Для построения разложения замкнутой системы в этом базисе достаточно задать только ее квантовое состояние.

Если данные, описывающие систему, помимо состояния, включают унитарную эволюцию, то наиболее адекватным с точки зрения физики является энергетический базис, связанный с разложением оператора эволюции на неприводимые компоненты.

В [12] в качестве универсального энергетического базиса, сопряженного к онтическому, предлагается использовать базис, диагонализирующий цикл максимальной возможной длины. Энергия основного состояния такого цикла является минимально возможной для Вселенной, состоящей из $\mathcal{N}$ онтичеких элементов.

5. МЕРЫ КВАНТОВОЙ ЗАПУТАННОСТИ

Количественно квантовые корреляции описываются *мерами запутанности*, в основе которых лежит понятие энтропии. Чаще всего в физике используется *энтропия фон Неймана*

$$S_1(\rho) = -\mathrm{tr}(\rho \log \rho). \qquad (5.1)$$

Широко используются также энтропии из *семейства Реньи* [14]

$$S_\alpha(\rho) = \frac{1}{1-\alpha} \log \mathrm{tr}(\rho^\alpha), \quad \alpha \geq 0, \quad \alpha \neq 1. \qquad (5.2)$$

Общим свойством для энтропий фон Неймана и Реньи является аддитивность на комбинациях вероятностных распределений, задаваемых собственными значениями матриц плотности, если эти распределения независимы.

Энтропию фон Неймана предпочитают потому, что она дополнительно имеет более сильное свойство – цепное правило для условных энтропий. В принципе, семейство Реньи можно пополнить энтропией фон Неймана с помощью предельного перехода $\alpha \to 1$.

В наших вычислениях мы используем 2-ю энтропию Реньи (называемую также *энтропией столкновений*)

$$S_2(\rho) = -\log \mathrm{tr}(\rho^2)$$

по следующим причинам:

• Ее легко вычислять. Для матрицы плотности размера $n \times n$ мы имеем

$$S_2(\rho) = -\log\left(\sum_{i=1}^{n} \rho_{ii}^2 + 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} |\rho_{ij}|^2 \right).$$

• $\mathrm{tr}(\rho^2)$ – это величина, называемая *чистотой* квантового состояния ρ. Чистота лежит в интервале $[1/n, 1]$, достигая верхнего значения 1 только для чистых состояний (отсюда название).

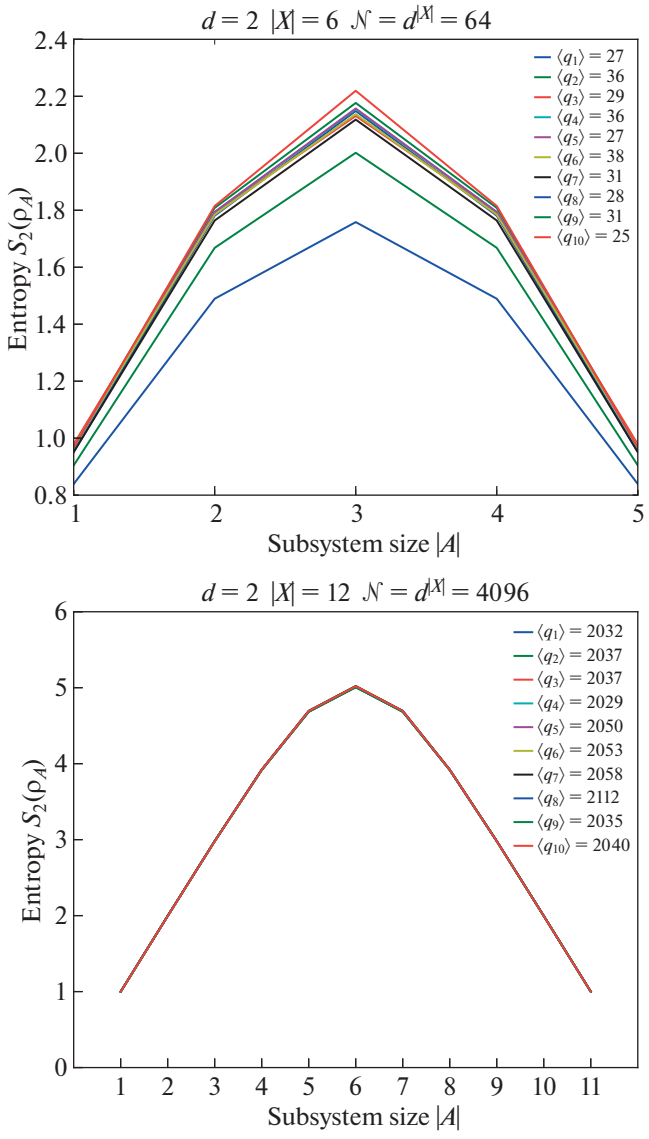


Рис. 1. Энтропии подсистем.

$\mathrm{tr}(\rho^2)$ совпадает с борновской вероятностью: “система наблюдает саму себя”.

• $\mathrm{tr}(\rho^2) \equiv \langle \rho \rangle_\rho$ – *ожидаемое значение наблюдаемой* ρ в состоянии ρ.

• $\mathrm{tr}(\rho^2) \equiv \|\rho\|_{\mathrm{F}}^2$ – квадрат *нормы Фробениуса* (*Гильберта–Шмидта*) матрицы плотности.

Скалярное произведение Фробениуса для двух матриц (или *скалярное произведение Гильберта–Шмидта* для двух операторов) a и b определяется как $\langle a \mid b \rangle_{\mathrm{F}} = \mathrm{tr}(a^\dagger b)$, а соответствующая норма имеет вид $\|a\|_{\mathrm{F}} = \sqrt{\langle a \mid a \rangle_{\mathrm{F}}}$.

Можно показать, что любые конструкции, используемые при изучении квантовых корреляций и основанные на энтропии фон Неймана, можно переформулировать в терминах матричных метрик.

Например, в моделях эмерджентной геометрии [3, 15, 16] расстояния между подсистемами A и B моделируются функциями взаимной информации

$$\mathcal{F}(A,B) = S_1(\rho_A) + S_1(\rho_B) - S_1(\rho_{A\cup B}). \qquad (5.3)$$

Заменив в (5.3) энтропию фон Неймана 2-й энтропией Реньи, мы получаем выражение

$$\mathcal{F}_2(A,B) = S_2(\rho_A) + S_2(\rho_B) - S_2(\rho_{A\cup B}) \qquad (5.4)$$

экспонента которого имеет вид

$$\frac{\mathrm{tr}(\rho_{A\cup B}^2)}{\mathrm{tr}(\rho_A^2)\mathrm{tr}(\rho_B^2)} \equiv \frac{\|\rho_{A\cup B}\|_{\mathrm{F}}^2}{\|\rho_A \otimes \rho_B\|_{\mathrm{F}}^2}. \qquad (5.5)$$

Очевидно, оба выражения (5.4) и (5.5) хорошо описывают отклонение от сепарабельности вызываемое запутанностью – хотя (5.4) и не имеет той вероятностной интерпретации, которая присуща выражению (5.3).

6. НЕКОТОРЫЕ ВЫЧИСЛИТЕЛЬНЫЕ НАБЛЮДЕНИЯ

Алгоритмы для построения тензорных разложений и вычисления квантовых корреляций реализованы в виде программы на языке Си.

Для иллюстрации вычислений с помощью этой программы рассмотрим однородную квантовую систему, т.е., систему, гильбертово пространство которой разлагается в тензорное произведение локальных гильбертовых пространств одинаковой размерности

$$\mathcal{H} = \bigotimes_{x\in X} \mathcal{H}_x,$$

где $X = \{1, \ldots, |X|\}$ – множество “геометрических точек”, $\dim \mathcal{H}_x = d$ для всех $x \in X$.

Предварительные вычисления указывают на то, что разложения с меньшими локальными размерностями демонстрируют более интересное поведение. Рассмотрим пример с $d = 2$ и $|X| = 23$. В этом случае размерность глобального пространства $\dim \mathcal{H} = 8388608$. В качестве меры расстояния между точками воспользуемся аналогом взаимной информации (5.4). Вычисления (5.4) на всех ребрах $(x, y) \in X \times X$ полного графа на вершинах X дают разброс величин в два порядка. Типичный результат вычислений

$$\mathcal{I}_2(x,y) \in [5.3\times10^{-8}, 7\times10^{-6}].$$

Большой разброс расстояний между точками можно рассматривать как признак нетривиальности предполагаемой эмерджентной геометрии в данном примере.

В примере с несколько большей локальной размерностью, $d = 7$ и $|X| = 7$ (следовательно $\dim \mathcal{H} = 823543$), аналогичное вычисление дает

$$\mathcal{I}_2(x, y) \in [0.0037, 0.0041].$$

То есть, геометрия близка к тривиальной – все точки почти эквидистантны и их можно интерпретировать как вершины близкого к правильному симплекса в 6-мерном евклидовом пространстве.

Более детальные вычисления показывают, что основной вклад в нетривиальность геометрии вносят локальные размерности d, а не число точек $|X|$.

На рис. 1 приведены результаты вычисления энтропий подсистем всех возможных размеров для квантовых систем с локальной размерностью $d = 2$, но с двумя разными количествами геометрических точек: $|X| = 6$ и $|X| = 12$. В обоих случаях использовалось по 10 случайно сгенерированных онтических состояний, в легендах указаны их веса Хэмминга. Данные, представленные на рисунке, демонстрируют следующие особенности:

• Тенденция к универсальности (слабой зависимости от квантовых состояний) с ростом $|X|$: визуально все графики почти идентичны в "большем" случае $|X| = 12$.

• Симметрия $S_2(\rho_A) = S_2(\rho_{X\setminus A})$ – это проявление свойств *разложения Шмидта* [7] чистого состояния: обе матрицы ρ_A и $\rho_{X\setminus A}$ имеют идентичные наборы ненулевых собственных значений.

• Для подсистем, размеры которых $|A|$ заметно меньше чем $|X|/2$, редуцированные состояния близки к *максимально смешанным* состояниям: $S_2(\rho_A) \approx |A| \log d$. Напомним, что максимально смешанным называют состояние, матрица плотности которого описывает равномерное распределение вероятностей, т.е., все ее собственные значения равны.

СПИСОК ЛИТЕРАТУРЫ

1. *Page D.N., Wootters W.K.* Evolution without evolution: Dynamics described by stationary observables // Phys. Rev. D. 1983. V. 27. P. 2885–2892. URL: https://link.aps.org/doi/10.1103/PhysRevD.27.2885.
2. *Carroll S.M.* Singh Ashmeet. Quantum mereology: Factorizing Hilbert space into subsystems with quasi-classical dynamics // Physical Review A. 2021. V. 103. № 2. URL: https://doi.org/10.1103/PhysRevA.103.022213
3. *Cao ChunJun, Carroll Sean M., Michalakis Spyridon*. Space from Hilbert space: Recovering geometry from bulk entanglement // Physical Review D. 2017. V. 95. № 2. URL: https://doi.org/10.1103/PhysRevD.95.024031
4. *Woods Mischa*. The Page-Wootters mechanism 36 years on: a consistent formulation which accounts for interacting systems // Quantum Views. 2019. V. 3. P. 16. URL: https://doi.org/10.22331/qv-2019-07-21-16.
5. *Zanardi Paolo*. Virtual Quantum Subsystems // Physical Review Letters. 2001. V. 87. № 7. URL: https://doi.org/10.1103/PhysRevLett.87.077901
6. *Zanardi Paolo, Lidar Daniel A., Lloyd Seth*. Quantum Tensor Product Structures are Observable Induced // Physical Review Letters. 2004. V. 92. № 6. URL: https://doi.org/10.1103/PhysRevLett.92.060402
7. *Nielsen M.A., Chuang I.L.* Quantum Computation and Quantum Information (10th Anniversary edition). USA: Cambridge University Press, 2016. ISBN: 978-1-10-700217-3.
8. *Gleason A.M.* Measures on the closed subspaces of a Hilbert space // Journal of Mathematics and Mechanics. 1957. V. 6. № 6. P. 885–893. URL: http://www.jstor.org/stable/24900629.
9. *Kornyak V.V.* Quantum models based on finite groups // Journal of Physics: Conference Series. 2018. V. 965. P. 012023. URL: https://doi.org/10.1088/1742-6596/965/1/012023.
10. *Kornyak V.V.* Modeling Quantum Behavior in the Framework of Permutation Groups // EPJWeb of Conferences. 2018. V. 173. P. 01007. URL: https://doi.org/10.1051/epjconf/201817301007.
11. *Kornyak V.V.* Mathematical Modeling of Finite Quantum Systems // Lect. Notes Comput. Sci. 2012. V. 7125. P. 79–93. 1107.5675.
12. *Banks T.* Finite Deformations of Quantum Mechanics. 2020. 2001.07662.
13. *Collins M.J.* On Jordan's theorem for complex linear groups // Journal of Group Theory. 2007. V. 10. № 4. P. 411–423. URL: https://doi.org/10.1515/JGT.2007.032
14. *Rényi A.* On measures of entropy and information / Proc. 4th Berkeley Symp. Math. Stat. Probab. 1961. V. 1. P. 547–561.
15. *Van Raamsdonk M.* Building up spacetime with quantum entanglement // Gen. Rel. Grav. 2010. V. 42. P. 2323–2329. 1005.3035.
16. *Maldacena J., Susskind L.* Cool horizons for entangled black holes // Fortschritte der Physik. 2013. V. 61. № 9. P. 781–811. URL: https://doi.org/10.1002/prop.201300020