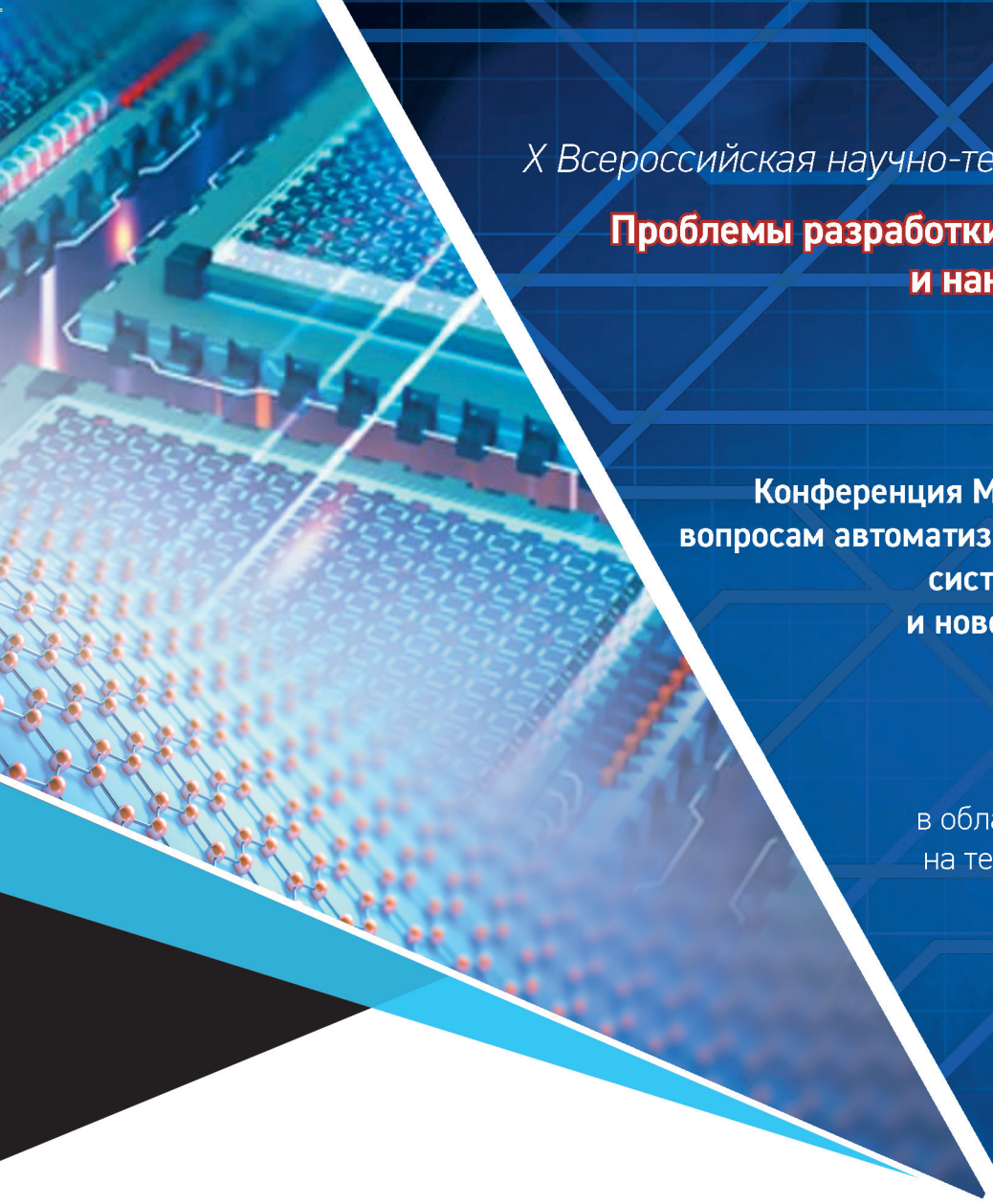
ISSN 2220-3397

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

ПрИн
2021
Том 12
3



X Всероссийская научно-техническая конференция

Проблемы разработки перспективных микро- и наноэлектронных систем – МЭС-2021

Конференция МЭС посвящена актуальным вопросам автоматизации проектирования МЭС, систем на кристалле, IP-блоков и новой элементной базы микро- и наноэлектроники

Конференция МЭС является крупнейшей конференцией в области САПР микроэлектроники на территории России и стран СНГ

В этом году конференция проходит в виде Интернет-форума с проведением заседаний научных секций разной тематической направленности в online-режиме. Конференция завершится проведением очного Пленарного заседания, на котором будут подведены ее итоги, а также секции «Презентации новых микроэлектронных проектов, САПР и готовых продуктов» на которой партнеры и спонсоры конференции представят доклады о своих разработках.

Прием докладов осуществляется **с 01 марта по 01 августа 2021 г.** Принятые доклады будут опубликованы на web-сайте конференции, а также в четырех выпусках трудов конференции, которые будут издаваться по мере поступления, рецензирования и редакционной подготовки текстов статей. Как и в 2020 году, будут проведены конкурсы на лучшие доклады с призами для победителей.

Участие в конференции МЭС-2021 бесплатное.

Более подробную информацию можно найти на web-сайте конференции МЭС-2021:

mes-conference.ru

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Грибова В. В., Москаленко Ф. М., Тимченко В. А., Шалфеева Е. А. Разработка решателей задач на основе управляющих графов для систем с базами знаний 115

Шелехов В. И. Методы трансформации и дедуктивной верификации программы инвертирования списков 127

Паринов С. И. Тематическое моделирование контекстов цитирований из научных публикаций: структура научного потребления автора 140

Галатенко А. В., Кузовихина В. А. Об одной модели безопасного функционирования компьютерных систем 150

Костенко К. И. Инварианты ядра фундаментальной модели интеллектуальной системы 157

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индекс по Объединенному каталогу "Пресса России" — **22765**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования и базу данных RSCI на платформе Web of Science.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 12
N 3
2021

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Gribova V. V., Moskalenko Ph. M., Timchenko V. A., Shalfeeva E. A. Control Graph Based Development of Problem Solvers for Systems with Knowledge Bases 115

Shelekhov V. I. Applying Program Transformations for Deductive Verification of the List Reverse Program 127

Parinov S. I. Topic Modeling of the Research Papers' Citation Contexts: a Structure of an Author's Research Consumption 140

Galatenko A. V., Kuzovikhina V. A. A Model of Secure Functioning of Computer Systems 150

Kostenko K. I. Core Invariants of the Mathematical Model of an Intelligent System 157

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.4'2 DOI: 10.17587/prin.12.115-126

В. В. Грибова, д-р техн. наук, ст. науч. сотр., зам. директора по научной работе, gribova@iacp.dvo.ru, **Ф. М. Москаленко**, канд. техн. наук, ст. науч. сотр., philipmm@iacp.dvo.ru, **В. А. Тимченко**, канд. техн. наук, ст. науч. сотр., vadim@iacp.dvo.ru, **Е. А. Шалфеева**, канд. техн. наук, ст. науч. сотр., shalf@iacp.dvo.ru, Федеральное государственное бюджетное учреждение науки Институт автоматики и процессов управления Дальневосточного отделения Российской академии наук, Владивосток

Разработка решателей задач на основе управляющих графов для систем с базами знаний

Предлагается методология использования декларативно задаваемых управляющих графовых структур для разработки решателей задач систем с базами знаний. Цель ее создания — снижение трудоемкости создания решателей, повышение их прозрачности, а также поддержка возможности распараллеливания процесса решения по доступным вычислительным узлам.

Ключевые слова: *база знаний, решатель, управляющий граф, программная инженерия, методология разработки программ, операции над информацией*

Введение

Снижение трудоемкости разработки программных систем (ПС), решающих комплексные задачи науки и практики, обеспечение их жизнеспособности[1] являются одними из актуальнейших задач программной инженерии [1]. Известные способы их решения состоят в разделении ПС на модули, использовании высокоуровневых (по возможности, декларативных) языков программирования, повторном использовании ранее разработанных компонентов.

Для систем с базами знаний (СБЗ), как отмечается во многих литературных источниках, например, в работе [2] перечисленные проблемы стоят наиболее остро, несмотря на то, что к настоящему времени удалось достичь существенного продвижения в их решении. Были предложены такие подходы, как выделение базы знаний (БЗ) в отдельный компонент; явное разделение предметных знаний и знаний о решении задачи [3, 4]; представление их в декларативном виде (с использованием онтологического подхода), который обеспечивает (особенно при наличии редакторов) понятное формирование и последующее модифицирование БЗ с привлечением экспертов предметной области [5, 6]. Тем не менее снижение трудоемкости создания решателей задач СБЗ и повышение их прозрачности (как необходимого критерия жизнеспособности [7]) все еще остается актуальной задачей. Особенно остро эта проблема стоит для тех СБЗ, в которых одним из требований является ограничение по времени, отводимому на принятие решения. Учитывая, что методы решения, применяемые во многих СБЗ, относятся к классу переборных, имеющих большую вычислительную сложность [8, 9], для уменьшения времени работы требуется распараллеливание. Это, в свою очередь, делает реализацию еще более трудоемкой, а сам решатель — менее прозрачным для внесения в него изменений в процессе жизненного цикла.

В работе предлагается методология разработки решателей задач СБЗ на основе декларативно задаваемых управляющих графовых структур. Цель ее создания — снижение трудоемкости разработки, повышение прозрачности решателей, поддержка возможности распараллеливания процесса решения и его выполнения на доступных вычислительных узлах разной архитектуры.

1. Архитектурные решения для комплексных и композитных ПС

В программной инженерии практикуются архитектурные модели и принципы управления, обеспечивающие анализируемость, тестируемость, сопровождаемость и повторную используемость программных модулей. К ним относятся: модель процедурной декомпозиции, модель объектной декомпозиции и др.; модель вызова-возврата, модель диспетчера (менеджера), модель передачи сообщений, модель событийного управления. Известен принцип вертикального структурирования, который рекомендует управляющую логику расположить "наверху", а собственно вычисления и обмен данными — "внизу" [10]. Модели вызова и диспетчера предлагают верхнеуровневую

[1] Под жизнеспособностью ПС понимается сохранение ее работоспособности и эволюционируемость в течение жизненного цикла с наименьшей стоимостью и поддержкой архитектурной целостности.

логику решаемой задачи прописывать в главном модуле (программной единице), из которого будет передаваться управление модулям, выполняющим вычисления. Запускаются они последовательно, в цикле или по некоторому условию, возможна параллельная обработка. Логика главного модуля редко повторно используема, в отличие от активируемых им (разнотипных) программных единиц. Некоторые из них типичны для класса задач, другие — для обрабатываемых информационных ресурсов[1], специфичных для предметной области.

Традиционно при создании интеллектуальной ПС базы знаний составляются из правил, в посылках которых факты реального мира сопоставляются со знаниями предметной области. При этом получение результата задачи проводится универсальной машиной вывода, которая выявляет соответствие извлеченных фактов и сведений, содержащихся в БЗ.

Если для знаний предметной области построена иерархия классов сущностей и их отношений, то "правила могут быть заданы в терминах онтологии" [11], а получением решения задачи занимается менее универсальный, чем машина вывода решатель (рассуждатель). Для установления истинности посылок в правилах он не просто использует факты, а сопоставляет их со структурными элементами сущностей предметной области ("используются введенные в онтологиях понятия и отношения в посылках и заключениях продукционных правил" [12]). То есть он ориентирован на онтологии объектов предметной области (пример: интерпретатор продукционных правил в системе Semp-TAO [13]).

Если для знаний предметной области построена онтология, не сводящаяся к иерархии сущностей, но позволяющая абстрагировать разные виды связей между сущностями, то решатель еще более ориентирован на онтологии. Это уже алгоритм, проводящий обход БЗ в соответствии с явно заданной онтологией. Для обеспечения его понимаемости, проверяемости и сопровождаемости требуются решения из программной инженерии, усовершенствованные с учетом особенностей обрабатываемой информации.

При разработке технологии построения решателей задач для СБЗ, опираясь на упомянутые модели, полезно учесть существующие технические и технологические решения. Точнее — решения, используемые для разработки комплексных ПС различного назначения, а также так называемых композитных приложений[2], как наиболее близких по архитектуре и сложности решаемых задач.

К первой группе (комплексные ПС), например, можно отнести системы управления (бизнес-) процессами (*workflow management system*, WFMS). Они предоставляют языковые средства и программную инфраструктуру для настройки, выполнения и мониторинга так называемых WF-приложений (*workflow application* или *process-oriented application*). Такое приложение представляет собой описание, как правило, в виде ориентированного ациклического графа (*directed acyclic graph*, DAG), выполнения последовательности (набора) операций (работ, действий). Вершины графа представляют операции, а дуги моделируют связи между операциями — передачу данных и управляющих условий перехода. То есть одной из важных функций WFMS является "оркестровка" операций.

Для создания WF-приложений разработано множество языков описания (XPDL, YAWL, SCUFL), ряд из которых поддерживает как графическую, так и основанную на XML нотацию. Теоретической основой таких языков является математический аппарат сетей Петри, а также их специализированные разновидности и расширения: *workflow nets*, *reset net*, *reset workflow nets* и т. п. В языках реализованы конструкции, позволяющие специфицировать различного рода зависимости по управлению (*control flow dependencie*s) между выполнением операций в WF-приложении: выполнение операций в заданной последовательности, в любом порядке или параллельно, синхронизация выполнения двух или более операций, взаимоисключающее выполнение, прекращение выполнения при наступлении определенного события и т. д.

Сегодняшние технологии облачных вычислений (второго поколения) реализуют перспективную модель облачных вычислений AaaS (*Application as a Service*), ориентированную на предоставление услуг по разработке и использованию композитных приложений (вторая рассматриваемая группа).

Ярким примером реализации данного подхода является многопрофильная инструментально-технологическая платформа облачных вычислений CLAVIRE [14, 15]. Она используется для создания композитных приложений, которые специфицируются в форме потока заданий как в текстовой (язык EasyFlow), так и в графической нотации. Программной единицей композитного приложения является так называемый вычислительный пакет, решающий некоторую прикладную задачу или набор задач. Платформенное описание пакета создается с помощью специального языка EasyPackage и высокоуровневого средства PackageManager. Язык EasyFlow основан на модели WF, представляющей собой DAG, в котором определяются множества входных параметров; выходных параметров; вычислительных, обрабатывающих данных, узлов, на которых запускаются пакеты; зависимостей по данным и множество зависимостей по управлению. Основные конструкции, используемые в языке EasyFlow: параллельная композиция — параллельный запуск двух и более пакетов в композитном приложении; последовательная композиция — управление последовательностью запуска пакетов в композитном приложении; циклический перебор значений из заданного множества — параллельный запуск пакетов с циклическим перебором входных параметров. Типичный скрипт на EasyFlow устроен следующим образом. Вначале указывается информация об исходных данных с по-

[1] Информационный ресурс (здесь и далее) – единица хранения, описывающая знания, данные, онтологию.

[2] Композитное приложение (*composite application*) — предназначенное для решения некоторой задачи программное приложение, созданное путем объединения, как правило, уже существующих сервисов, взаимодействующих в распределенной среде.

мощью соответствующей директивы. После этого могут быть заданы атрибуты, которые определяют режим взаимодействия скрипта с пакетами и приоритет исполнения. Затем приводится описание так называемых шагов, каждый из которых определяет вызов пакета с описанием требуемых для него исходных данных.

Помимо CLAVIRE необходимо также упомянуть такие системы управления композитными приложениями в распределенных вычислительных средах, как Taverna [16], Pegasus [17], Weka4WS [18], DVega [19] и др. [20]. В основном это приложения поддержки научной деятельности, дающие возможность моделировать различные комплексные эксперименты. Главные отличия различных систем управления связаны с выразительными средствами языков описания WF-приложений; средствами мониторинга выполнения WF-приложений и механизмами обработки ошибок; типами используемых программных единиц (пакетные приложения, web-сервисы, локальное и удаленное выполнение команд) и механизмами передачи данных между ними; типами поддерживаемых вычислительных ресурсов (web-сервисы, грид, облако); механизмами встраивания существующего программного обеспечения.

Таким образом, для обеспечения понимаемости, проверяемости и сопровождаемости СБЗ на основе онтологий могут быть использованы описанные решения из программной инженерии с их адаптацией к особенностям СБЗ.

2. Архитектура жизнеспособных систем с базами знаний

В процессе эксплуатации ПС любого типа, в том числе СБЗ, появляется потребность в исправлении ошибок, добавлении пользовательских функций, совершенствовании метода решения и пользовательского интерфейса, а также, что характерно для СБЗ, изменении хранимых знаний. Дополнительным современным требованием к таким системам является включение экспертов предметной области в процесс разработки БЗ, а также ее верификация и корректировка, если она была построена с использованием методов машинного обучения. Это требует представления базы знаний в форме, понятной таким экспертам, что достигается через семантическую модель представления, формирование знаний на основе онтологии, а также через реализацию удобных редакторов БЗ, управляемых онтологиями [21—24].

Онтология задает типы утверждений (позволяющих решать задачи в предметной области) и ограничения на интерпретацию смысла понятий (терминов). Явное представление (и группирование) в онтологии всех структурных типов утверждений (и отделение онтологии от базы знаний) ведет к замене традиционных решателей-рассуждателей, интерпретирующих продукционные правила, на специализированные алгоритмы, основанные на онтологиях [25, 26].

Задача онтолого-ориентированного алгоритма — обход БЗ, сопоставление ее утверждений каждого типа (тип задается онтологией) с входной информацией (об объекте) для подтверждения или опровержения гипотез о решении.

Таким образом, в архитектуру жизнеспособной СБЗ входят следующие компоненты: онтология знаний, сформированная по ней БЗ, база фактов, онтолого-ориентированный решатель задачи, представляющий собой реализацию специализированного алгоритма (вместо универсальной машины логического вывода), а также пользовательский интерфейс.

Коллективом авторов разработана платформа IACPaaS, реализующая данный подход к созданию жизнеспособных СБЗ [21]. Согласно технологии разработки [27, 28] решатель создается из декларативно-процедурных модулей (агентов), выполняющих обработку информации в соответствии с конкретными типами причинно-следственных и других связей, характерными для решаемой задачи. В спецификации (декларативном описании) решателя задач указывается агент, начинающий работу. Связь агентов (передача управления от одного к другому путем посылки сообщений) задается процедурно в явном виде или с помощью дополнительно вводимых разработчиками механизмов.

За более чем десятилетний опыт использования платформы на основе данной технологии (и ее расширений) реализован большой спектр интеллектуальных систем в различных предметных областях — медицина, подводная робототехника, педагогическая психология, математика, сельское хозяйство и др. Опыт их создания и обратная связь с разработчиками платформы обозначили направления дальнейшего совершенствования методов разработки онтолого-ориентированных решателей интеллектуальных систем для обеспечения большей их прозрачности, повторной используемости компонентов решателя, а также в целях расширения средств управления распараллеливанием.

3. Методология разработки решателей СБЗ на основе информационно-управляющих графов

3.1. Общие сведения

Ориентируясь на описанные в разд. 1 методы и подходы к созданию композитных приложений, как наиболее близких по архитектуре и сложности к СБЗ, и к повышению их сопровождаемости, авторами предложена методология разработки решателей СБЗ, повышающая их прозрачность и основанная на декларативном формировании структуры, названной информационно-управляющим графом (ИУГ). Особое внимание уделяется обработке фрагментов БЗ и возможности размещения архитектурных компонентов решателя на разных вычислительных устройствах.

Как отмечено выше, выделение базы с предметными знаниями в отдельный компонент и представление в декларативном виде обеспечивают их понятное формирование и модифицирование. Преследуя ту же цель, формировать решатели (процедурные знания) также необходимо в как можно более

декларативной форме. Процедуру решения важно целиком представлять наглядно, создавать из прозрачных компонентов, реализующих требуемую обработку связей понятий предметной области. Предпочтительны повторно используемые компоненты, реализующие обработку групп или цепочек связей понятий, применяемых при решении нескольких задач, например, диагностики и прогноза.

Для обеспечения прозрачности (процедурных) компонентов решателя их описание выполняется в терминах информации, необходимой и достаточной для их функционирования, в том числе входных и выходных параметров, и способа передачи им управления. Для прозрачности интеграции процедурных элементов в рамках одного решателя необходимо определение множеств параметров и обрабатываемых данных, зависимостей по данным и управлению и, если требуется, то и вычислительных узлов, на которых они размещаются и работают.

В структуру ИУГ входят связанные конструкции (операторы), одна из которых помечена как "начало". Эти конструкции могут быть разделены на два основных типа: конструкции управления вычислениями (см. разд. 3.2) и информационно-вычислительные конструкции (см. разд. 3.3, 3.4). При этом, как и во всех современных языках программирования, в ИУГ могут быть заданы не только отдельные операторы, но и так называемые блоки, которые группируют операторы для многократного вызова, в том числе при организации рекурсии. Помимо этого, в структуру ИУГ входит описание данных (см. разд. 3.5).

Предполагается, что работу решателя обеспечивает интерпретатор ИУГ, работающий в рамках некоторой среды исполнения (платформы).

3.2. Конструкции управления вычислениями

Для организации обработки нескольких информационных ресурсов[1] и их фрагментов вводятся следующие управляющие конструкции:

- *последовательное выполнение действий* над указанными информационными ресурсами или их фрагментами;
- *циклическое выполнение действия* (*действий*), при котором последовательно обрабатываются элементы указанного конечного множества структурно одинаково устроенных фрагментов указанного информационного ресурса;
- *параллельное выполнение действия* (*действий*), при котором параллельно обрабатываются элементы указанного конечного множества структурно одинаково устроенных фрагментов указанного информационного ресурса (распараллеливание по данным);
- *выполнение действий по условиям*, при котором вычисляется значение условия (условий) или выражения и выполняется действие, поставленное в соответствие выполнившемуся условию или полученному значению выражения.

Здесь "действие" — отдельная конструкция (оператор), в том числе типа *последовательное выполнение действий* и *вызов блока*.

Для конструкции типа *параллельное выполнение действия* в качестве действия (тела цикла) применяется *вызов блока* — для обеспечения разделения по данным параллельно работающих обработчиков (см. разд. 3.5). Их максимальное число задается разработчиком и ограничено сверху средой исполнения.

Для конструкции типа *выполнение действий по условиям* доступно два варианта: IF-ELSEIF-...-ELSE и SWITCH-CASE-...-DEFAULT. В первом случае определение истинности в каждой ветке IF, ELSEIF задается оператором *вычислитель значения предиката* с логическим результатом. Во втором случае сначала вычисляется значение выражения, задаваемого оператором типа *вычислитель значения*, а потом оно сравнивается с константами некоторого единого типа. Для каждого варианта присутствует факультативная "альтернативная" ветка (ELSE и DEFAULT), не имеющая условия и выполняющаяся при невыполнении всех других условий.

3.3. Информационно-вычислительные конструкции

К информационно-вычислительным конструкциям относятся следующие:

- *конструкции вызова*, подразделяемые, в свою очередь, на следующие:
  - *вызов процедурного кода* — может исполняться либо непосредственно на том же вычислительном узле, где работает интерпретатор ИУГ, либо на некотором ином; во втором случае задается (вместе с передаваемыми данными) доступное вычислительное устройство (GPU, внешний сервер, узел вычислительного кластера и т. п.), а исполняемый код для него либо заранее прикрепляется в конструкцию, либо помещается на само устройство исполнения заранее (чтобы в момент запуска избежать передачи кода из конструкции на это устройство, если это возможно);
  - *вызов блока*, который содержит:
    - либо непосредственно определение блока (если такой блок не был определен ранее), включающее описание входных и выходных формальных параметров (имена и типы — примитивные, составные, ссылки в базы знаний), а также набор конструкций, одна из которых помечена как "начало" (подобие ИУГ),
    - либо ссылку на блок, чье определение дано ранее (в ином вызове), и в обоих случаях включает описание правил формирования фактических значений входных и сохранения значений выходных параметров;
- *вычислитель значения* (в том числе *вычислитель значения предиката*) — вычисляет значение некоторого типа (число, строка, множество и т. п.) на основе оперативных данных (см. разд. 3.5) и/или содержимого БЗ (используется только в составе других операторов — *присвоение значения переменной*,

[1] Моделью представления информационных ресурсов является особого вида граф понятий [29, 30].

выполнение действий по условиям, *циклическое выполнение*, *параллельное выполнение*);

- *присвоение значения переменной* — содержит ссылку на переменную (см. разд. 3.5) и ссылку на конструкцию *вычислитель значения*, либо на *интерфейсную операцию*;
- *операция над информационным ресурсом* (см. разд. 3.4) — выполняет манипуляцию информационным ресурсом или обработку его фрагмента с учетом значений параметров заданных типов (результат такой обработки — либо вычисленные значения, типы которых указываются в сигнатуре, либо модифицированное содержимое информационного ресурса);
- *интерфейсная операция*:
  - аппаратно-интерфейсная операция — предназначена для получения/передачи данных через внешние интерфейсы;
  - человеко-интерфейсная операция — предназначена для получения/передачи данных от/к человеку (выполняется через аппаратные средства человеко-машинного ввода/вывода).

С помощью операций последнего типа, с одной стороны, пользователю сообщается необходимая информация о ходе решения задачи, а с другой стороны, дается возможность непосредственно влиять на этот процесс. Человеко-интерфейсные операции можно разделить на следующие типы: а) ввод информации пользователем с помощью интерфейсных управляющих элементов; б) вывод (отображение) информации пользователю в заданной форме; в) смешанный тип (выбор).

3.4. Операции над информационным ресурсом

Основной класс конструкций, наличие которых способствует созданию прозрачных решателей интеллектуальных задач в виде СБЗ, это операции над (онтологическим) информационным ресурсом (ресурсами). По характеру обработки они делятся на те, которые:

— управляют существованием информационного ресурса:

- *создание пустого информационного ресурса* (в соответствии с выбранной онтологией);
- *удаление информационного ресурса*;

— манипулируют содержимым информационного ресурса (в соответствии с общепринятыми типами операций CRUD[1]):

- *создание фрагмента информационного ресурса* по некоторому адресу согласно заданному описанию (записать либо новые значения/фрагменты — Create, либо заменить имеющиеся — Update) — может сопровождать операцию создания информационного ресурса, задавая его начальное состояние;
- *получени*е данных, расположенных по некоторому адресу в информационном ресурсе — Read;
- удалени*е данных*, расположенных по некоторому адресу в информационном ресурсе — Delete.

[1] CRUD — Create, Read, Update, Delete.

Ввиду того, что СБЗ ведут обработку информационных ресурсов, имеющих сложную графовую структуру, к общепринятым обрабатываемым типам данных (строки, числа, логические значения и т. п.) добавляется тип данных "фрагмент информационного ресурса" ("подграф", "ссылка на вершину"). Соответственно, операции типа *вычислитель значения* и *вычислитель значения предиката* должны поддерживать обработку данных этого типа (чтение меток вершин, навигация по иерархии вершин, сопоставление фрагментов информационного ресурса с образцом[2] и т. п.).

Особым видом использования операции *создание фрагмента информационного ресурса* является *трансформирование фрагмента информационного ресурса* (графа)[3]. При вызове данной операции должно быть указано следующее:

- вершина, являющаяся *начальной* для применяемого блока иерархически организованных правил, т. е. вершина, прямыми потомками которой должны стать вершины, возникающие при интерпретации спецификации первого яруса правил (это могут быть как вершины, созданные непосредственно в процессе интерпретации этих правил, так и вершины, созданные ранее, и в этом случае в процессе интерпретации создаются только входящие в эти вершины дуги);
- вершина "начало пути" — вершина в информационном ресурсе, начиная с которой ведется поиск вершин по специфицированным в правилах метапутям (данная вершина и начальная вершина для блока правил могут совпадать);
- корневая вершина структуры, представляющая собой спецификацию иерархически организованных порождающих правил.

Структура для спецификации блоков иерархически организованных порождающих правил представлена на рис. 1. Нотация разметки изображенных на рисунке вершин и дуг совпадает с нотацией, используемой в работах [29, 30].

Метапуть представляет собой последовательность ссылок на вершины в графе, являющемся онтологией обрабатываемого информационного ресурса. Интерпретация правила, описывающего создание множества дуг к существующим вершинам, состоит в следующем. Необходимо пройти в формируемом информационном ресурсе (используя специфицированный путь в орграфе метаинформации) от вершины, указанной в качестве начала пути, до вершины, являющейся экземпляром вершины, представляющей *предпоследний* элемент метапути. Затем нужно создать дуги, входящие в множество вершин, являющихся экземплярами вершины, представляющей *последний* элемент метапути.

При интерпретации правила, описывающего создание новой вершины, прямыми потомками последней должны стать вершины, созданные в описаниях *непосредственно* вложенных правил (она передается во все такие правила, если таковые имеются). Если

[2] С другим фрагментом этого же или другого информационного ресурса, созданного по этой же онтологии.

[3] Раздел 2.2 в работе [31].

Рис. 1. Структура для спецификации блоков иерархически организованных порождающих правил

метка новой создаваемой вершины в правиле не задана, то в качестве метки ей присваивается метка ее метавершины.

3.5. Информационная составляющая ИУГ

Важным аспектом реализации механизма ИУГ является обеспечение передачи информации между конструкциями.

Передаваемая информация могла бы храниться лишь в обрабатываемых БЗ и ее совместное использование обеспечивалось бы адресацией нужного фрагмента. Однако это может снизить повторную используемость таких компонент решателя, усложнить разработку ввиду их загромождения такими фрагментами (возникает необходимость создания специальных разделов для хранения промежуточных данных для каждой задачи, решаемой с их применением).

Более удобным механизмом является отделение такой информации от БЗ и данных путем организации отдельной специальной "базы переменных". Ее структура задается как часть ИУГ, описывающего решатель, а сама база формируется в момент запуска решателя. Описание данных для удобства разработчиков делится на входные, выходные, временные, и фрагменты этих разделов являются то входными, то выходными для вызываемых конструкций:

- в структуру конструкций (формируемую в рамках структуры ИУГ) включено описание входных/выходных данных;
- при использовании конструкции в формируемом ИУГ решателя необходимо каждому такому данному установить ссылку на вершину в "базе переменных" (в частности, для конструкций *циклическое* и *параллельное выполнение действий* такая ссылка устанавливается для переменной цикла);
- при обработке конструкции данные для ее исполнения будут извлечены интерпретатором из "базы переменных", а после исполнения — помещены в нее.

Такая же база данных создается и каждый раз в момент начала работы конструкции типа *вызов блока*. Таким образом, каждый блок работает в изолированном хранилище переменных, что позволяет, в частности, при параллельной работе блоков избежать конфликта по используемым данным.

4. Применение методологии к созданию решателей задач

Продемонстрируем применение предложенных решений на двух задачах — медицинской диагностики и интерактивного построения доказательств теорем.

4.1. Медицинская диагностика

Диагностика — один из типов интеллектуальных задач. Обратимся к такой ее постановке, при которой рассматривается изменение значений признаков исследуемого субъекта во времени, а значения признаков — вещественные числа. Модель онтологии наблюдений следующая[1]:

```
онтология наблюдений {
   ~setmm ~new группа признаков {
      ~setmm ~new -> группа признаков;
      ~setmm ~new признак {
         ~new значения {
            ~new нижняя граница [real]
            ~new верхняя граница [real]
            ~copy норма {
               ~one ~new -> верхняя граница;
               ~one ~new -> нижняя граница;
} } } } } }
```

Для описания субъекта используется следующая модель онтологии:

```
онтология истории неисправности {
   ~copy ~new наблюдения {
      ~seq осмотр {
         ~one ~new дата [date]
         ~set ~clone признак {
            ~one ~new значение [real]
} } }
~copymm ~new диагноз ~ALT {
      ~one ~new ["норма"]
      ~set ~ref -> онтология базы знаний
      о неисправностях$/неисправность;
      ~one ~new ["неизвестная неисправ-
      ность"]
} }
```

То есть в течение истории наблюдений выполняется несколько осмотров субъекта в определенные моменты времени и при каждом осмотре формируется имя признака (как зависимый клон [30] от имени признака из базы наблюдений) и под ним — его

[1] Здесь и далее используется конкретный текстовый синтаксис языка формального представления модели онтологии, описанной в работах [29, 30].

наблюдаемое значение. По окончании диагностики либо формируется вершина "норма", либо непустое множество ссылок на вершины базы знаний о неисправностях, либо вершина "неизвестная неисправность".

База знаний о неисправных состояниях включает имена неисправностей, и для каждой из них — список значимых признаков, вариантов их развития, а для каждого варианта — периоды динамики с областью значений и длительностью.

ИУГ решателя начинается с оператора (конструкции) типа *выполнение действий по условиям*. Он состоит из одного условия с соответствующим ему действием и альтернативного действия. Условие состоит в проверке гипотезы о том, что субъект находится в нормальном состоянии. Соответствующее ему действие — выдача этого результата в выходной информационный ресурс (сформировать вершину "норма"). Альтернативное действие — выполнить проверку гипотез о наличии неисправностей и сохранить ее результаты. Перечень неисправных состояний извлекается из БЗ, они исследуется и подтвержденные добавляются к результату. Если таковых нет, то формируется вершина "неизвестная неисправность".

Исходные данные берутся из значений переменных (часть оперативных данных решателя):

- "База наблюдений" — ссылка на некоторый информационный ресурс, содержащий названия наблюдений (признаков) и их значений (включая нормальные);
- "База знаний о неисправностях" — ссылка на некоторый информационный ресурс, содержащий знания о неисправностях;
- "База истории неисправности" — ссылка на некоторый информационный ресурс, содержащий описание неисправности субъекта.

Эти информационные ресурсы сформированы по описанным выше моделям онтологий и подаются на вход интерпретатору ИУГ при запуске. В этот момент эти переменные получают конкретные значения (устанавливаются ссылки). Временные данные в решателе не используются в данном примере.

На рис. 2 изображен фрагмент ИУГ, представляющего решатель данной задачи. Прямоугольниками со сплошной рамкой обозначены конструкции ИУГ. Для упрощения отображения параметры конструкций описаны внутри данных прямоугольников (а не отдельными вершинами ИУГ), сплошные стрелки используются лишь для связи целостных конструкций друг с другом.

Проверка гипотезы о нормальном состоянии (условие) специфицируется с использованием структуры, применяемой для задания логических формул. С ее помощью формируется конструкция типа *вычислитель значения предиката*, которая описывает правило пребывания субъекта в нормальном состоянии. Ее суть заключается в следующем: для каждого наблюдаемого признака каждое его наблюдаемое значение принадлежит области его нормальных значений. В терминах онтологий баз наблюдений и истории неисправности можно описать это условие так: для любого наблюдаемого признака и его значения в истории неисправности данное значение больше или равно минимальному нормальному значению этого признака (заданному в базе наблюдений) и меньше или равно максимальному нормальному значению. Этот предикат задается вершиной типа *квантор всеобщности*, под которой присутствуют вершины следующих типов:

- *переменная* — имеет имя "признак из истории" с областью значений, формируемой из вершин, присутствующих в информационном ресурсе, представленном переменной "База истории неисправности", и доступных по метапути, *начало* которого есть "наблюдения", а *продолжение* — "осмотр" / "признак";
- *формула* — корень подграфа, при интерпретации которого выполняется: а) выбор вершины, представляющей наблюдаемое значение признака "признак из истории", с последующим взятием метки этой вершины (числа); б) взятие оригинала вершины "признак из истории" — вершины, описывающей признак в базе наблюдений, после чего — переход через вершины "значения" и "норма" к вершинам, сформированным по метавершинам "верхняя граница" и "нижняя граница", и взятие их меток (назовем их "верхняя граница нормы" и "нижняя граница нормы"); в) проверка выполнения условия "(метка $\geqslant$ $\geqslant$ нижняя граница нормы) & (метка $\leqslant$ верхняя граница нормы)".

При истинном значении предиката должно выполниться сохранение результата "норма". Для этого используется конструкция типа *создание фрагмента информационного ресурса*. Информационным ресурсом является значение переменной "База истории неисправности", а адресом является метапуть "диагноз" / "норма" — обе эти вершины будут порождены путем копирования [30].

При ложном значении предиката выполняется альтернативное действие, представленное конструкцией типа *последовательное выполнение действий*. Шаг 1 — в (параллельном) цикле проверить гипотезы о диагнозе (рассматривая все неисправности). Шаг 2 — проверить, подтверждены ли какие-то гипотезы, в противном случае сохранить результат "неизвестная неисправность".

Выполнение цикла для проверки гипотез о диагнозе (наличие некоторой неисправности) проходит в параллельном режиме. Переменной цикла назначается "текущая неисправность", а ее значениями становятся вершины, доступные по ссылке на информационный ресурс, хранимой в переменной "База знаний о неисправностях" по метапути "неисправность". В качестве тела цикла выступает *вызов блока*, где блок есть конструкция типа *вызов процедурного кода*. При ее интерпретации будет вызван процедурный код, выполняющийся на том же узле, что и интерпретатор ИУГ, с передачей ему ссылки в БЗ на исследуемую неисправность. При подтверждении исследуемой гипотезы этот код выполнит сохранение этой неисправности как части результата. По окончании работы параллельного цикла в истории неисправности либо уже будут сохранены ссылки на подтвержденные неисправности, либо будет отсутствовать вершина "диагноз". Для обработки второй ситуации на шаге 2 присутствует конструкция типа

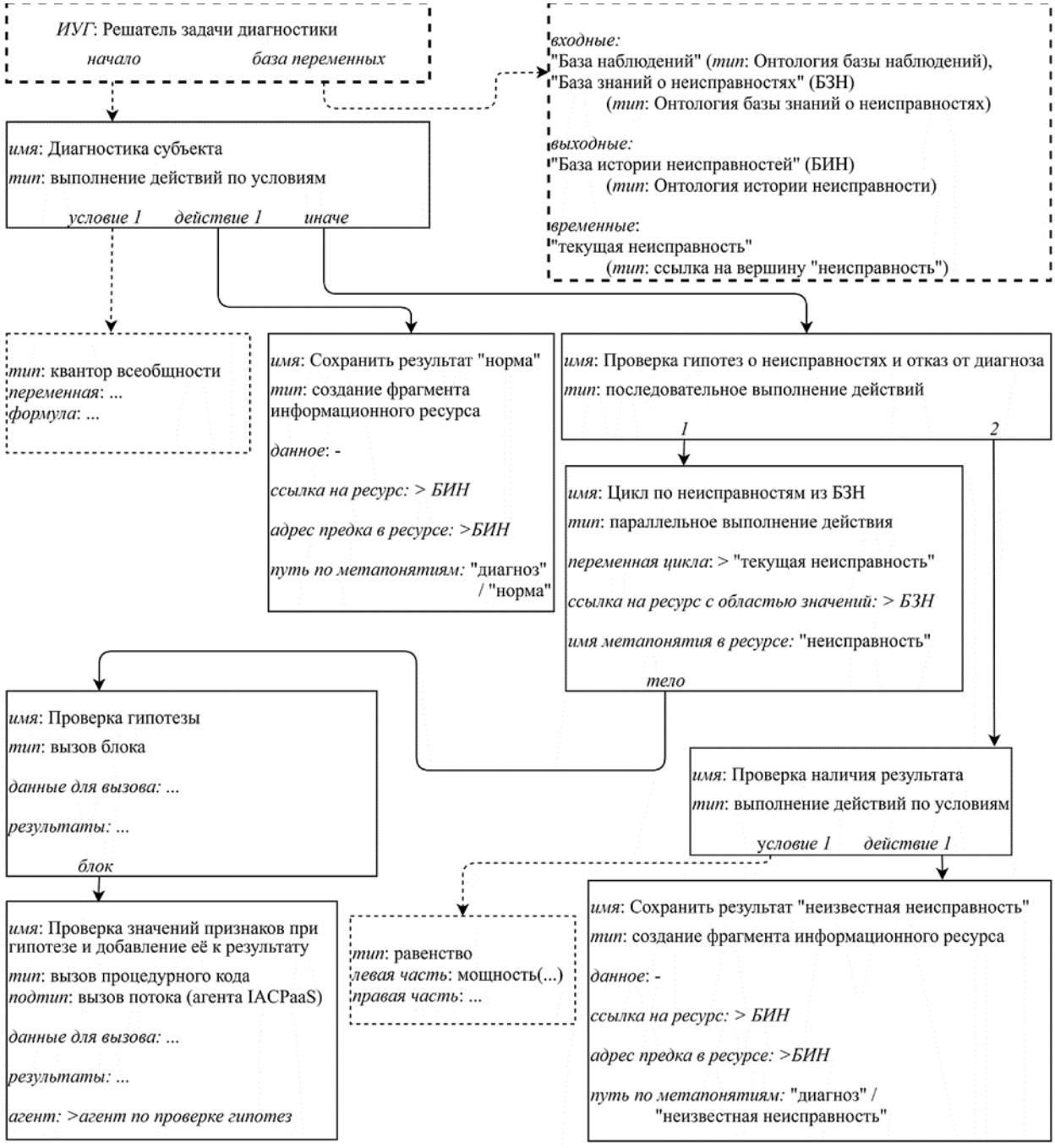


Рис. 2. Решатель задачи диагностики

выполнение действий по условиям, где присутствует одна пара условие-действие. В первом элементе описывается условие на отсутствие сохраненных результатов — логическая формула вида "вершина диагноз не существует". Действие — конструкция типа *создание фрагмента информационного ресурса*, где ресурс — значение выходной переменной "база истории неисправности"; адрес предка — корневая вершина этого ресурса; создаваемый фрагмент информационного ресурса — метапуть "диагноз"/"неизвестная неисправность".

Конструкция "Проверка значений признаков при гипотезе и добавление ее к результату" в предложенном методе организации вычислений имеет тип *вызов процедурного кода* (агента). При иных реализациях здесь мог бы быть описан запуск некоторого процедурного решателя на указанном пользователем внешнем вычислительном устройстве (графическом процессоре, мемристорном массиве, узле вычислительного кластера), куда загружается процедурный код, предварительно сформированный и настроенный на работу по определенному набору знаний о неисправности, и наблюдаемые данные. Исполняемый код может храниться в данной вершине и загружаться в вычислительное устройство до начала решения задачи (при формировании решателя) или в момент интерпретации данной вершины вместе с данными наблюдения.

4.2. Верификация интуитивного математического доказательства

Рассмотрим решение задачи верификации интуитивного доказательства математического утверждения (теоремы), принадлежащего базе математических знаний, путем расширения этого доказательства до полного, выполненного в рамках формальной системы, описанной в работе [32]. В рамках решения этой задачи выполняется пошаговое построение информационного ресурса — графа полного формального доказательства, который представляет собой множество связанных определенным образом доказываемых *целей*. Цель представляется собственно *математическим утверждением* (*предложением*) и *списком предположений* (возможно, пустым) — множеством математических утверждений, из справедливости которых следует справедливость данного утверждения. Информационный ресурс формируется в соответствии с моделью онтологии полных доказательств [33]. Процесс заканчивается, когда достигнуто состояние, в котором нет недоказанных целей. Ссылка на граф полного формального доказательства добавляется во множество ссылок на доказательства у выбранного из базы математических знаний утверждения. В результате проверки условия завершения процесса конструирования доказательства пополняется используемая база знаний.

Одним из методов доказательства цели является применение правила доказательства импликации (естественного вывода). Это правило позволяет свести доказательство цели, *математическое утверждение* (*предложение*) которой имеет форму импликации $s = (v_1{:}\ t_1)\ \ldots\ (v_n{:}\ t_n)\ f_1\ \&\ \ldots\ \&\ f_m \Rightarrow f$, а *список предположений* есть $p_1, \ldots, p_k$ ($k \geqslant 0$), к доказательству цели, *математическое утверждение* которой есть заключение этой импликации — $s = (v_1{:}\ t_1)\ldots(v_n{:}\ t_n)\ f$, а *список предположений* есть $p_1, \ldots, p_k, f_1, \ldots, f_m$. Здесь s, $p_1, \ldots, p_k$ — математические утверждения; $f, f_1, \ldots, f_m$ — математические формулы утверждения s, содержащие вхождения предметных переменных $v_1, \ldots, v_n$; $(v_i; t_i)(i = 0, \ldots, n)$ — описание предметной переменной v_i; t_i — математический терм, значением которого является множество (область возможных значений переменной v_i).

Операция *трансформирование фрагмента информационного ресурса* в данном случае используется как средство реализации применения описанного правила. Для ее использования в ИУГ решателя верификатора необходимо в описании вызова операции сформировать блок порождающих правил, ссылаясь на вершины графа, представляющего фрагмент онтологии полных доказательств (рис. 3). Штриховыми линиями обведены прямоугольники, обозначающие вершины связанного с ним графа, представляющего онтологию базы математических знаний. Блок правил формируется в соответствии с приведенной на рис. 1 структурой.

Также в вызове операции в качестве *начальной* для применяемого блока правил вершины указывается вершина в графе доказательства, представляющая текущую доказываемую цель (вершина-экземпляр вершины "Цель" на рис. 3). Эта же вершина указывается в качестве вершины "начало пути".

Результатом интерпретации структуры, представляющей собой описание блока порождающих правил, является выполнение следующих действий по созданию подграфа в формируемом графе доказательства:

- создание вершины-экземпляра вершины "Доказательство импликации", а в качестве ее прямого потомка — вершины-экземпляра вершины "Цель" — *goal*;
- создание вершины-экземпляра вершины "Предложение" — *s′* как прямого потомка вершины *goal*, а из вершины *s′* создание
  - множества дуг к вершинам, представляющим описания переменных $(v_1{:}\ t_1),\ldots,(v_n{:}\ t_n)$ в математическом утверждении *s*;
  - дуги к вершине, представляющей заключение импликации в математическом утверждении *s* — формулу *f*;
- создание вершины-экземпляра вершины "Список предположений" — *props* как прямого потомка вершины *goal*, а у вершины *props* создание

Рис. 3. Фрагмент онтологии полных доказательств. Доказательство импликации

- множества дуг к вершинам, представляющим предположения из списка предположений текущей доказываемой цели — p_1, ..., p_k (если $k > 0$);
- m (по числу конъюнктов в условии импликации) прямых потомков — вершин-экземпляров вершины "Предложение" — s_1, ..., s_m, а из вершины s_i ($i = 1, ..., m$) создание
- множества дуг к вершинам, представляющим описания переменных (v_1: t_1), ..., (v_n: t_n) в математическом утверждении s;
- дуги к вершине, представляющей соответствующий i-й конъюнкт в условии математического утверждения s — формулу f_i.

Заключение

В работе предложена методология разработки решателей задач для СБЗ, предназначенная для снижения трудоемкости их разработки и повышения прозрачности. С учетом большой вычислительной сложности систем данного класса важной задачей, решению которой также уделено внимание в работе, является обеспечение возможности распараллеливания обработки БЗ.

Указанные свойства обеспечиваются за счет использования декларативных средств спецификации компонентов, а также специализированных операторов, управляющих процессом вычисления, в том числе с возможностью параллельного выполнения действий для обработки множества структурно одинаково устроенных фрагментов информационного ресурса. Декларативная спецификация решателя в виде ИУГ обеспечивает возможность специалистам, не имеющим навыков программирования, участвовать в разработке решателя. Они указывают в вершинах графа основные подзадачи (процедуры) решения основной задачи и специфицируют правила и условия их выполнения через набор операций. Специфика предложенного решения и его принципиальное отличие от других решений — ориентированность на обработку информационных ресурсов, прежде всего, БЗ, сформированных на основе онтологии и имеющих семантическое представление.

Предложенные методы также позволяют задавать вызовы процедурного кода на разных вычислительных устройствах, доступных в среде исполнения.

Демонстрация предложенных решений иллюстрируется на двух принципиально разных интеллектуальных задачах. Это задача медицинской диагностики, которая на основе истории болезни пациента и базе знаний по диагностике заболеваний формирует выходной информационный ресурс — детализированное объяснение гипотез о возможных заболеваниях. Второй является задача интерактивного построения доказательств математических теорем, в ходе выполнения которого пополняется база математических знаний.

Работа выполнена при частичной финансовой поддержке грантов РФФИ № 19-07-00244 и № 20-07-00670.

Список литературы

1. **Черников Б. В.** Управление качеством программного обеспечения. — М.: ООО "Научно-издательский центр ИНФРА-М", 2020. 240 с.

2. **Kobbacy K. A. H.** Artificial intelligence in maintenance // Complex system maintenance handbook. — London, Springer. 2008. — P. 209—231.

3. **Gribova V. V., Kleshchev A. S., Shalfeeva E. A.** Control of Intelligent Systems // Journal of Computer and Systems Sciences International. — 2010. — Vol. 49, No. 6. — P. 952—966.

4. **Голенков В. В., Гулякина Н. А., Давыденко И. Т., Шункевич Д. В.** Семантические технологии проектирования интеллектуальных систем и семантические ассоциативные компьютеры // Доклады БГУИР. — 2019. — Т. 3. — С. 42—50.

5. **Musen M. A.** The Protégé project: A look back and a look forward // AI Matters. 2015. — Vol. 1, No. 4. — P. 4—12.

6. **Грибова В. В., Шалфеева Е. А.** Обеспечение жизнеспособности систем, основанных на знаниях // Информационные технологии. — 2019. — Т. 25, № 12. — С. 738—746.

7. **Dehaghani S. M. H., Hajrahimi N.** Which factors affect software projects maintenance cost more? // Acta Informatica Medica. — 2013. — Vol. 21, No. 1. — P. 63—66.

8. **Gholamian M. R., Fatemi Ghomi S. M. T., Ghazanfari M.** A Hybrid System for Multiobjective Problems:A case study in NP-hard problems // Knowledge-Based Systems. — 2007. — Vol. 20, No. 4. — P. 426—436.

9. **Chen J.-H., Wu Y.-S.** Dispatching Rule in a Fuzzy Rule Knowledge-based System for Automated Manufacturing // International Journal of the Computer, the Internet and Management. — 2008. — Vol. 16, No. 1. — P. 53—60.

10. **Pressman R. S.** Software Engineering: A practitioner's approach, Seventh edition. — McGraw-Hill, 2009. — 928 p.

11. **Загорулько Ю. А., Загорулько Г. Б.** Онтологии и их практическое применение в системах, основанных на знаниях // Всероссийская конференция с международным участием "Знания — Онтологии — Теории" (ЗОНТ-2011). Новосибирск, Институт математики им. С. Л. Соболева СО РАН. — 2011. — Т. 1. — С. 132—141.

12. **Загорулько Ю. А.** Семантические модели и технологии разработки информационных и интеллектуальных систем, ориентированные на экспертов // Информационные и математические технологии в науке и управлении. — 2014. — № 3. — С. 131—138.

13. **Загорулько Ю. А.** Онтологический подход к разработке системы поддержки принятия решений на нефтегазодобывающем предприятии // Вестник Новосибирского государственного университета. Серия: Информационные технологии. — 2012. — Т. 10, вып. 1. — С. 121—128.

14. **Smirnov P. A., Kovalchuk S. V., Boukhanovsky A. V.** Knowledge-based support for complex systems exploration in distributed problem solving environments // Communications in Computer and Information Science. — 2013. — Vol. 394. — P. 147—161.

15. **Мельник М. А., Насонов Д. А., Бухановский А. В.** Технология интеллектуальной организации процесса выполнения неоднородных композитных приложений в распределенной вычислительной среде // Известия высших учебных заведений. Приборостроение. — 2020. — Т. 63, № 2. — С. 191—193.

16. **Oinn T., Addis M., Ferris J.** et al. Taverna: a tool for the composition and enactment of bioinformatics workflows // Bioinformatics. — 2004. — Vol. 20, No. 17. — P. 3045—3054.

17. **Deelman E., Singh G., Su M. H.** et al. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems // Scientific Programming. — 2005. — Vol. 13, No. 3. — P. 219—237.

18. **Lackovic M., Talia D., Trunfio P.** A framework for composing knowledge discovery workflows in grids // Foundations of Computational, Intelligence. — Berlin, Springer Heidelberg. — 2009. — Vol. 6. — P. 345—369.

19. **Tolosana-Calasanz R., Bañares J. A., Álvarez P.** et al. An uncoordinated asynchronous checkpointing model for hierarchical scientific workflows // Journal of Computer and System Sciences. — 2010. — Vol. 76, No. 6. — P. 403—415.

20. **Talia D.** Workflow Systems for Science: Concepts and Tools // International Scholarly Research Notices. — 2013. — Vol. 2013. — Article ID 404525.

21. **Грибова В. В., Москаленко Ф. М., Тимченко В. А., Шалфеева Е. А.** Создание жизнеспособных интеллектуаль-

ных систем с управляемыми декларативными компонентами // Информационные и математические технологии в науке и управлении. — 2018. — № 3 (11). — С. 6—17.

22. **Golenkov V., Shunkevich D., Davydenko I.** Principles of Organization and Automation of the Semantic Computer Systems Development // Open semantic technologies for intelligent systems. — 2019. — Is. 3. — P. 53—90.

23. **Atta-ur-Rahman.** Knowledge Representation: A Semantic Network Approach // Handbook of Research on Computational Intelligence Applications in Bioinformatics. — IGI Global. — 2016. — P. 55—74.

24. **Costa R., Lima C., Sarraipa J., Jardim-Gonçalves R.** Facilitating knowledge sharing and reuse in building and construction domain: an ontology-based approach // Journal of Intelligent Manufacturing. — 2016. — Vol. 27, No. 1. — P. 263—282.

25. **Vrandecic D.** Ontology evaluation // Handbook on Ontologies. — Berlin, Springer Heidelberg. — 2009. — P. 293—313.

26. **Mulder D.** Explanation and Ontological Reasoning (1996). Dissertations. 3675. URL: https://ecommons.luc.edu/luc_diss/3675

27. **Грибова В. В., Клещев А. С., Крылов Д. А.** и др. Базовая технология разработки интеллектуальных сервисов на облачной платформе IACPaaS. Часть 1. Разработка базы знаний и решателя задач // Программная инженерия. — 2015. — № 12. — С. 3—11.

28. **Грибова В. В., Клещев А. С., Крылов Д. А.** и др. Базовая технология разработки интеллектуальных сервисов на облачной платформе IACPaaS. Часть 2. Разработка агентов и шаблонов сообщений // Программная инженерия. — 2016. — Т. 7, № 1. — С. 14—20.

29. **Gribova V. V., Kleshchev A. S., Moskalenko F. M., Timchenko V. A.** A Two-level Model of Information Units with Complex Structure that Correspond to the Questioning Metaphor // Automatic Documentation and Mathematical Linguistics. — 2015. — Vol. 49, No. 5. — P. 172—181.

30. **Gribova V. V., Kleshchev A. S., Moskalenko F. M., Timchenko V. A.** A Model for Generation of Directed Graphs of Information by the Directed Graph of Meta-information for a Two-Level Model of Information Units with a Complex Structure // Automatic Documentation and Mathematical Linguistics. — 2015. — Vol. 49, No. 6. — P. 221—231.

31. **Andries M., Engels G., Habel A.** et al. Graph transformation for specification and programming // Science of Computer Programming. — 1999. — Vol. 34, No. 1. — P. 1—54.

32. **Клещев А. С., Тимченко В. А.** Теоретические основы оболочки для интерактивных систем верификации интуитивных математических доказательств // Онтология проектирования. — 2018. — Т. 8, № 2 (28). — С. 219—239.

33. **Клещев А. С., Тимченко В. А.** Реализация оболочки и портала знаний по верификации математических доказательств на платформе IACPaaS // Онтология проектирования. — 2018. — Т. 8, № 3 (29). — С. 427—448.

Control Graph Based Development of Problem Solvers for Systems with Knowledge Bases

V. V. Gribova, gribova@iacp.dvo.ru, **Ph. M. Moskalenko,** philipmm@iacp.dvo.ru, **V. A. Timchenko,** vadim@dvo.ru, **E. A. Shalfeeva,** shalf@iacp.dvo.ru, Institute of Automation and Control Processes Far Eastern Branch of the Russian Academy of Sciences, Vladivostok, 690041, Russian Federation

Corresponding author:

Moskalenko Philip M., PhD, Senior Researcher, Institute of Automation and Control Processes Far Eastern Branch of the Russian Academy of Sciences, Vladivostok, 690041, Russian Federation
E-mail: philipmm@iacp.dvo.ru

Received on November 13, 2020
Accepted on December 01, 2020

Reducing the complexity of developing software systems that solve complex problems of science and practice, as well as ensuring their viability are among urgent tasks of software engineering. For systems with knowledge bases, as noted in many literature sources, this problem is most acute, despite the fact that significant results have been achieved in its solving. This problem is especially critical for those systems in which one of the requirements is a time limit for decision-making. Given that the used solution methods belong to the class of exhaustive ones that have a large computational complexity, parallelization is required to reduce the execution time.

A methodology for using declaratively defined control graph structures for developing problem solvers for systems with knowledge bases is proposed. Special attention is paid to the processing of knowledge base fragments and the possibility of placing architectural components of the solver on different computing devices.

The structure of such an information-and-control graph includes related constructs (operators), one of which is marked as "initial". These structures can be divided into two main types: computing control structures and information processing structures. The developer can group individual operators into blocks — for their repeated usage (including organization of recursive calculations). The structure of the graph also includes a description of the data.

Keywords: *knowledge base, solver, control graph, software engineering, program development methodology, information processing*

Acknowledgements: *This work was partially supported by the Russian Foundation for Basic Research, project nos. 19-07-00244 and 20-07-00670.*

For citation:

Gribova V. V., Moskalenko Ph. M., Timchenko V. A., Shalfeeva E. A. Control Graph Based Development of Problem Solvers for Systems with Knowledge Bases, *Programmnaya Ingeneria,* 2021, vol. 12, no. 3, pp. 115—126

DOI: 10.17587/prin.12.115-126

References

1. **Chernikov B. V.** *Software quality management,* Moscow, INFRA-M Publishing House. 240 p. (in Russian).
2. **Kobbacy K. A. H.** Artificial intelligence in maintenance, *Complex system maintenance handbook*, Springer, London, 2008, pp. 209—231.
3. **Gribova V. V., Kleshchev A. S., Shalfeeva E. A.** Control of Intelligent Systems, *Journal of Computer and Systems Sciences International*, 2010, vol. 49, no. 6, pp. 952—966.
4. **Golenkov V. V., Gulyakina N. A., Davydenko I. T., Shunkevich D. V.** Semantic technologies of intelligent systems design and semantic associative computers, *Doklady BGUIR*, 2019, vol. 3, pp. 42—50.
5. **Musen M. A.** The Protégé project: A look back and a look forward, *AI Matters*, 2015, vol. 1, no. 4, pp. 4—12.
6. **Gribova V. V., Shalfeeva E. A.** Ensuring of Viability of Systems Based on Knowledge, *Information Technologies*, 2019, vol. 25, no. 12, pp. 738—746.
7. **Dehaghani S. M. H., Hajrahimi N.** Which factors affect software projects maintenance cost more? *Acta Informatica Medica*, 2013, vol. 21, no. 1, pp. 63—66.
8. **Gholamian M. R., Fatemi Ghomi S. M. T., Ghazanfari M.** A Hybrid System for Multiobjective Problems — A case study in NP-hard problems, *Knowledge-Based Systems*, 2007, vol. 20, no. 4, pp. 426—436.
9. **Chen J.-H., Wu Y.-S.** Dispatching Rule in a Fuzzy Rule Knowledge-based System for Automated Manufacturing, *International Journal of the Computer, the Internet and Management*, 2008, vol. 16, no. 1, pp. 53—60.
10. **Pressman R. S.** *Software Engineering: A Practitioner's Approach*, Seventh edition, McGraw-Hill, 2009, 928 p.
11. **Zagorul`ko Yu. A., Zagorul`ko G. B.** Ontologies and their practical application in knowledge-based systems, *Vserossijskaya konferenciya s mezhdunarodnym uchastiem "Znaniya — Ontologii — Teorii" (ZONT-2011)*, Novosibirsk, Sobolev Institute of mathematics SB RAS, 2011, vol. 1, pp. 132—141 (in Russian).
12. **Zagorul`ko Yu. A.** Semantic models and technologies for development of information and intelligent expert-oriented systems, *Informacionnye i matematicheskie tekhnologii v nauke i upravlenii*, 2014, vol. 3, pp. 131—138 (in Russian).
13. **Zagorul`ko Yu. A.** Ontological approach to the development of a decision support system for an oil and gas production enterprise, *Vestnik Novosibirskogo gosudarstvennogo universiteta. Seriya: Informacionnye tekhnologii*, 2012, vol. 10, is. 1, pp. 121—128 (in Russian).
14. **Smirnov P. A., Kovalchuk S. V., Boukhanovsky A. V.** Knowledge-based support for complex systems exploration in distributed problem solving environments, *Communications in Computer and Information Science*, 2013, vol. 394, pp. 147—161.
15. **Mel'nik M. A., Nasonov D. A., Bukhanovskij A. V.** Technology for intelligent organization of the process of executing heterogeneous composite applications in a distributed computing environment, *Izvestiya vysshikh uchebnykh zavedenij. Priborostroenie*, 2020, vol. 63, no. 2, pp. 191—193 (in Russian).
16. **Oinn T., Addis M., Ferris J.** et al. Taverna: a tool for the composition and enactment of bioinformatics workflows, *Bioinformatics*, 2004, vol. 20, no. 17, pp. 3045—3054.
17. **Deelman E., Singh G., Su M. H.** et al. Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems, *Scientific Programming*, 2005, vol. 13, no. 3, pp. 219—237.
18. **Lackovic M., Talia D., Trunfio P.** A framework for composing knowledge discovery workflows in grids, *Foundations of Computational, Intelligence*. Springer, Berlin, Heidelberg, 2009, vol. 6, pp. 345—369.
19. **Tolosana-Calasanz R., Bañares J. A., Álvarez P.** et al. An uncoordinated asynchronous checkpointing model for hierarchical scientific workflows, *Journal of Computer and System Sciences*, 2010, vol. 76, no. 6, pp. 403—415.
20. **Talia D.** Workflow Systems for Science: Concepts and Tools, *International Scholarly Research Notices*, 2013, vol. 2013, article ID 404525.
21. **Gribova V. V., Moskalenko P. M., Timchenko V. A., Shalfeeva E. A.** Viable intelligent systems development with controlled declarative components, *Information and Mathematical Technologies in Science and Management*, 2018, no. 3 (11), pp. 6—17 (in Russian).
22. **Golenkov V., Shunkevich D., Davydenko I.** Principles of Organization and Automation of the Semantic Computer Systems Development, *Open semantic technologies for intelligent systems*, 2019, no. 3, pp. 53—90.
23. **Atta-ur-Rahman**. Knowledge Representation: A Semantic Network Approach, *Handbook of Research on Computational Intelligence Applications in Bioinformatics*, IGI Global, 2016, pp. 55—74.
24. **Costa R., Lima C., Sarraipa J., Jardim-Gonçalves R.** Facilitating knowledge sharing and reuse in building and construction domain: an ontology-based approach, *Journal of Intelligent Manufacturing*, 2016, vol. 27, no. 1, pp. 263—282.
25. **Vrandecic D.** Ontology evaluation, *Handbook on Ontologies*, Springer Berlin Heidelberg, 2009, pp. 293—313.
26. **Mulder D.** Explanation and Ontological Reasoning (1996). Dissertations. 3675, available at: https://ecommons.luc.edu/luc_diss/3675 (access date: 27.11.2020).
27. **Gribova V. V., Kleschev A. S., Krylov D. A., Moskalenko Ph. M., Timchenko V. A., Shalfeyeva E. A.** A Base Technology for Development of Intelligent Services with the Use of IACPaaS Cloud Platform. Part 1. A Development of Knowledge Base and Problem Solver, *Programmnaya Ingeneria,* 2015, no. 12, pp. 3—11 (in Russian).
28. **Gribova V. V., Kleschev A. S., Krylov D. A., Moskalenko Ph. M., Timchenko V. A., Shalfeyeva E. A.** The Base Technology for Intelligent Services Development with the Use of IACPaaS Cloud Platform. Part 2. Development of Agents and Message Templates, *Programmnaya Ingeneria,* 2016, vol. 7, no. 1, pp. 14—20 (in Russian).
29. **Gribova V. V., Kleshchev A. S., Moskalenko F. M., Timchenko V. A.** A Two-level Model of Information Units with Complex Structure that Correspond to the Questioning Metaphor, *Automatic Documentation and Mathematical Linguistics*, 2015, vol. 49, no. 5, pp. 172—181.
30. **Gribova V. V., Kleshchev A. S., Moskalenko F. M., Timchenko V. A.** A Model for Generation of Directed Graphs of Information by the Directed Graph of Meta-information for a Two-Level Model of Information Units with a Complex Structure, *Automatic Documentation and Mathematical Linguistics*, 2015, vol. 49, no. 6, pp. 221—231.
31. **Andries M., Engels G., Habel A.** et al. Graph transformation for specification and programming, *Science of Computer Programming*, 1999, vol. 34, no. 1, pp. 1—54.
32. **Kleschev A. S., Timchenko V. A.** Theoretical foundations of the shell for interactive systems of intuitive mathematical proofs verification, *Ontology of designing*, 2018, vol. 8, no. 2, pp. 219—239 (in Russian).
33. **Kleschev A. S., Timchenko V. A.** Implementation of the shell and knowledge portal for mathematical proofs verification on the IACPaaS platform, *Ontology of designing*, 2018, vol. 8, no. 3, pp. 427—448 (in Russian).

УДК 004.05 DOI: 10.17587/prin.12.127-139

В. И. Шелехов, канд. тех. наук, зав. лаб., vshel@iis.nsk.su, Институт систем информатики им. А. П. Ершова СО РАН, Новосибирский государственный университет

Методы трансформации и дедуктивной верификации программы инвертирования списков

На примере программы инвертирования односвязных списков исследуются методы трансформации и дедуктивной верификации функций библиотеки ОС Linux. Исходная программа трансформируется с языка Си на язык cP без указателей. Далее программа преобразуется на язык функционального программирования WhyML. Рассмотрено три варианта дедуктивной верификации программы инвертирования односвязных списков в рамках системы автоматического доказательства Why3.

Ключевые слова: *дедуктивная верификация, трансформации программ, функциональное программирование, предикатное программирование, алгебраический тип данных, односвязный список, операционная система Linux*

Введение

Три года назад была проведена дедуктивная верификация 26 библиотечных функций [1] ядра операционной системы (ОС) Linux с помощью разработанного инструмента AstraVer [2, 3]. Эта верификация являлась частью работ по моделированию и верификации политик безопасности управления доступом в ОС [4] в интересах разработки отечественной ОС Astra Linux, в настоящее время активно внедряемой в различных отраслях и регионах России.

Сложность и трудоемкость дедуктивной верификации в рамках существующих инструментов верификации вынуждают искать другие решения. В настоящей работе рассматривается задача трансляции программ на языке Си с рекурсивными структурами данных на язык функционального программирования WhyML [5] через промежуточный язык cP с устранением указателей. Язык cP не содержит указателей, но максимально близок к языку Си по типам данных и языковым конструкциям. Итоговая программа на языке WhyML проще исходной программы на языке Си, что существенно упрощает дедуктивную верификацию.

Разработка языка функционального программирования cP и методов трансляции на него с языка Си осуществляется с апробацией на наборе библиотечных программ из ядра ОС Linux. Для части программ этого набора ранее проводилась дедуктивная верификация [6—8] в технологии предикатного программирования [9—12]. Почти все программы из этого набора обрабатывают массивы. Поэтому язык cP в основном ориентирован на массивы. Для списков и деревьев есть особенности, которые требуют дополнительных языковых конструкций в языке cP.

Операции с указателями заменяются при трансляции эквивалентными действиями без указателей применением специального набора *трансформаций*. Трансляция на языки cP и WhyML при этом должна быть автоматической. В результате трансляции дополнительно формируется *трансформационная программа*, фиксирующая модификации программы. Пользователь может изменить трансформационную программу и запустить трансляцию повторно.

Методы трансформации и верификации программ, оперирующие со списками и деревьями, рассматриваются в настоящей работе на примере программы инвертирования односвязных списков llist_reverse_order, входящей в модуль llist библиотеки ядра ОС Linux для работы со списками без блокировки (*lock-free*).

Во разд. 1 настоящей статьи представлен обзор работ, связанных с устранением указателей. Описан опыт дедуктивной верификации программы инвертирования односвязных списков. Проведено сравнение методов верификации различных библиотек программ для ОС. В разд. 2 представлен метод трансформации программы на языке Си, работающей со списками или деревьями, с получением программы без указателей на языке cP. В разд. 3 представлена трансформация исходной программы llist_reverse_order в эквивалентную программу на языке cP. Далее описано преобразование программы с языка cP на язык WhyML. Рассмотрены три варианта дедуктивной верификации программы инвертирования односвязных списков в рамках системы автоматического доказательства Why3 [5] и проведено их сравнение. В Заключении перечислены ключевые особенности технологии трансформации программ, оценены результаты дедуктивной верификации и определены планы дальнейших работ.

1. Смежные работы по трансформации и дедуктивной верификации программ

1.1. Устранение указателей

Императивное программирование немыслимо без указателей для разработки эффективной программы. Указатели неразрывно вплетаются в логику алгоритма создаваемой программы. Идея того, что указатели

могут быть внесены в программу независимо как результат оптимизирующей трансформации, появилась только в предикатном программировании [9—12]. Определена оптимизирующая трансформация кодирования списков с помощью указателей [13]. Показано, что именно внесение указателей дает критический прирост сложности трансформируемой программы [8].

Для упрощения дедуктивной верификации программ из библиотеки ядра ОС Linux применялась дедуктивная верификация эквивалентных предикатных программ [6, 7] с опорой на гипотезу, что для всякой императивной программы можно построить эквивалентную предикатную программу, из которой императивная программа может быть получена применением некоторой системы оптимизирующих трансформаций. Было обнаружено, что оптимизирующие трансформации обратимы, и можно вести процесс трансформации в обратном направлении от языка Си к языку предикатного программирования. Новая технология оказалось более успешной и была применена для дедуктивной верификации программ sort, memweight и kstrtoull из библиотеки ядра ОС Linux.

Следующим шагом на этом направлении стала реализация автоматической трансляции с языка Си, устраняющей указатели, на некоторый промежуточный язык.

Обратные трансформации — новое направление исследований. Наиболее близкими к обратной трансформации являются работы по анализу видов структур данных (*shape analysis*) [14—20] и обратной трансляции [21—24] на язык более высокого уровня. Причем в этих работах не ставится задача устранения указателей.

Отметим, что имеет место и противоположная тенденция — попытка внести указатели в языки функционального программирования, например, в языки Coq [25] и WhyML [5]. В язык WhyML включены циклы и модифицируемые переменные, а также указатели.

1.2. Верификация программы инвертирования списков

Дедуктивная верификация программы инвертирования односвязных списков около 15 лет назад была объявлена как задача высокой сложности (*verification grand challenge*). Причина такой сложности кроется в сложности спецификации и верификации операций с указателями. Классический метод дедуктивной верификации Хоара [26] не применим при наличии в программе указателей. Большинство методов дедуктивной верификации программ с указателями базируются на логике разделения (*separation logic*) [27]. Постоянно появляются новые методы верификации программ с указателями, из отечественных работ следует отметить работу [3].

Было предпринято много попыток проведения дедуктивной верификации программы инвертирования односвязных списков, в частности, описанных в работах [28, 29]. Спецификации длинные и сложные, в частности, потому что в них дополнительно представлены условия корректности структур памяти, проверка которых описана в подразд. 3.3 как часть потокового анализа гнезд объектов. Дедуктивная верификация сложна и трудоемка.

В работе [30] описана верификации предикатной программы [9, 10] инвертирования односвязных списков применением метода дедуктивной верификации [12]. Предикатная программа похожа на рекурсивную программу, полученную как результат трансформации исходной программы llist_reverse_order, см. подразд. 3.5. Сгенерированные по методу [12] формулы корректности доказывались в системе интерактивного доказательства PVS [31]. Доказательство было быстрым и простым. При доказательстве потребовалось ввести пять дополнительных простых лемм. Эффективная императивная программа, эквивалент исходной программы llist_reverse_order, была получена применением системы оптимизирующих трансформаций к исходной предикатной программе. Одна из трансформаций кодирования с применением указателей оказалась весьма сложной. Примечательно, что в работе [32] используется такая же спецификация на списках и аналогичная идея обобщения задачи, что и для программы reverseG [30]. В целях улучшения понимания императивных программ с указателями используются некоторая абстрактная модель односвязных списков и логика разделения.

1.3. Дедуктивная верификация библиотек программ

Программа ОС содержит вызовы библиотечных программ для операций со строками, списками, деревьями и другими объектами. Наличие ошибки в библиотечной программе может стать причиной уязвимости самой ОС. Дедуктивная верификация библиотек, гарантирующая отсутствие ошибок, становится насущной задачей особенно для ОС с высокими требованиями к защищенности от нежелательного внешнего воздействия.

В работе [34] представлена дедуктивная верификация программ библиотеки для односвязных списков, интенсивно используемой в ОС Contiki для интернета вещей. Требование защищенности от вредоносного проникновения — одно из важнейших в ОС Contiki. Для верификации использовался инструмент FRAMA-C/WP с выходом на SMT-решатель Alt-Ergo и интерактивную систему доказательства Coq [25].

Авторами сделана попытка добиться максимальной автоматизации доказательства формул корректности, генерируемых инструментом для программ библиотеки относительно их спецификаций на языке ACSL. Чтобы избежать использования индуктивных предикатов, затрудняющих автоматическое доказательство, выбрана достаточно сложная модель спецификации списков. Спецификации программ, написанные авторами для этой модели, оказались длинными, сложными и трудно проверяемыми, а также потребовали модификации исходного кода из-за введения теневых (*ghost*) переменных. Модифицирован также код для доступа к элементу списка с заменой универсального типа `void*` на `int`.

Чтобы подтвердить корректность написанных спецификаций, в частности, полноту постусловий, эти спецификации были использованы для верификации специально написанных тестов. Отметим, что это не дает полной гарантии корректности спецификаций.

Технология дедуктивной верификации иллюстрируется на примере программы list_push для вставки элемента в начало списка. Для программы list_push из семи строк написаны спецификации размером более 310 строк. Отметим, кстати, что операция вставки элемента в начало списка применяется в программе инвертирования списков. Поэтому программа llist_reverse_order существенно сложнее list_push.

В работах [1, 35] описывается дедуктивная верификация 26 библиотечных функций (в основном для операций со строками) ядра ОС Linux с помощью инструмента верификации FRAMA-C/AstraVer [2, 3]. Плагин AstraVer, разработанный авторами на основе плагина Jessie, транслирует исходную программу на язык WhyML [5], дополняя ее моделью памяти для корректного отображения операций с указателями и моделью операций с числами. В плагине максимально учтена специфика кода программ ОС Linux. Инструмент Why3 генерирует формулы корректности для программы на языке WhyML и преобразует их во входные задания для автоматических решателей (Alt-Ergo, CVC3, CVC4, Z3 и десятка других) и интерактивных систем доказательства (Coq, Isabelle/HOL, PVS).

Была успешно доказана корректность 26 функций ядра Linux. Лишь для трех функций проведены незначительные изменения исходного кода.

Приведем соотношение размера кода и спецификации (в строках), представленное авторами трех разных работ:

- библиотека для операций со списками — 10, т. е. 10 строк спецификации на одну строку кода;
- 26 функций для операций со строками из библиотеки ОС Linux — 2,6;
- 5 программ из библиотеки ОС Linux после трансформации на язык cP — 0,9.

Данный показатель косвенно отражает сложность проведения дедуктивной верификации для разных подходов.

2. Трансформация рекурсивных структур

Язык cP получается из языка Си устранением в нем указателей. Любое вхождение указателя в программе заменяется объектом, являющимся значением указателя. Операции с указателями заменяются эквивалентными действиями без указателей. Остальные конструкции языка Си — типы данных, наборы операторов и операций — максимально сохраняются в языке cP. Некоторые конструкции взяты из языка WhyML [5], что упростит последующий перевод на WhyML. Есть конструкции, заимствованные из языка предикатного программирования P [9]. Здесь описывается часть языка cP, ориентированная на трансформацию рекурсивных структур данных.

Трансляция с устранением указателей реализуется применением набора трансформаций разного вида. *Трансформация* `A --> B` определяет замену конструкции `A`, содержащей указатели, на эквивалентную ей конструкцию `B` без указателей, возможно при соблюдении определенных условий. Дополнительным результатом трансляции является *трансформационная программа*, содержащая модификации программы, в частности, модифицированные описания типов и заголовков функций. Реализация автоматической трансляции нетривиальна. У пользователя должна быть возможность изменить трансформационную программу и запустить трансляцию повторно, в частности, поменять введенные при трансформации имена типов и переменных.

Рекурсивные структуры программы на языке Си, в частности, списки и деревья, трансформируются в структуры типа объединения **union**$\{K_1; \ldots; K_m\}$. Тип объединения является алгебраическим типом в стиле функционального программирования и определяет набор конструкторов $K_1, \ldots, K_m$. Конструктор содержит имя конструктора и возможно пустой набор полей как у структуры (записи). Тип **union** языка Си преобразуется в тип объединения языка cP. Для корректности такого преобразования необходимо гарантировать отсутствие конфликтов, связанных с наложением полей.

Нулевой указатель **NULL** заменяется стандартным конструктором `Empty`. Допустим, в программе имеется некоторая позиция указателя `T*` на структуру типа `T` и в этой позиции допускается значение **NULL**. После трансформации данная позиция будет иметь тип **union** `{ Empty; Node(T) }`, где `Node` — стандартный конструктор с одним полем типа `T`. Если `T` — тип структуры (записи), то поля структуры становятся полями конструктора `Node`.

В языке cP значением переменной `x` алгебраического типа при исполнении оператора присваивания `x = y` станет копия объекта — значения переменной `y`. Копирование проводится рекурсивно по всей иерархии объекта `y`. В исходной программе переменной `x` типа "указатель" присваивается не само значение, а его адрес. При трансляции на язык cP замена адреса на значение в операторе присваивания не является эквивалентным преобразованием в случае дальнейшей модификации одной копии и использования другой копии значения.

Чтобы обеспечить эквивалентность трансформаций, в языке cP наряду с оператором присваивания используется *оператор присоединения* `x <- y`, при исполнении которого копия объекта не создается. Значением переменной `x` становится тот же объект, что и для переменной `y`. Для оператора присоединения вида `x.next <- y` объект, значение переменной `y`, становится также значением поля `next` структуры `x` без копирования объекта `y`. Таким образом, объект `y` становится *подобъектом* объекта `x`. Дальнейшая модификация объекта `y` приведет также к модификации объекта `x`. Оператор присоединения вида `x <- x.next` (*оператор сканирования*) реализует перемещение переменной `x` на объект по полю `next` исходной структуры `x` без копирования объекта `x.next`.

Здесь не описываются другие конструкции языка cP, обеспечивающие трансформацию подстановки указателя аргументом функции, операций с двойными указателями[1] и некоторых операций с рекурсивными структурами в языке Си.

Допустим, тип `N` является типом структуры (записи), определяющей вершину (элемент списка, вершину дерева) некоторой рекурсивной структуры. Для вхождений типа указателя на вершину действует следующее правило трансформации:

```
N* --> recT
```

Здесь `recT` — стандартное имя в языке cP, определяемое описанием:

```
type recT = union { Empty; Node(N) };
```

Определим трансформации операций с рекурсивными структурами. Пусть `x` — переменная типа `N*`; `C` — выражение, отличное от `NULL`; `next` — одно из полей структуры типа `N`.

Нулевой указатель:	**NULL** --> Empty
Доступ к полю:	x -> next --> x.next
Присваивание указателю:	x = C --> x <- C
Присваивание полю:	x -> next = C --> x.next <- C
Сканирование:	x = x -> next --> x <- x.next

После трансформации переменная `x` и выражение `C` будут иметь тип `recT`.

Трансляция программы с языка Си на язык cP включает следующие этапы: потоковый анализ программы, применение трансформаций, устраняющих указатели, и замену операторов присоединения эквивалентными операторами присваивания.

Потоковый анализ программы определяет информационные связи между переменными программы. Строится картина памяти в виде набора независимых гнезд объектов после каждого оператора. С этой целью применяется символьное исполнение программы, используемое иногда при анализе видов структур (*shape analysis*) [14—20].

Мы рассматриваем *объекты* как объекты памяти исполняемой программы. Это простые переменные, массивы, структуры — записи в виде набора полей, списки, деревья и другие рекурсивные структуры. Выделяются *подобъекты*: элементы массивов, поля структур, вершины списка и т. д. *Гнездо объекта* определяет объект, его структуру и набор переменных, для которых объект или его компоненты являются значениями переменных.

Потоковый анализ реализует также аппроксимацию значений переменных-указателей. Если имеются две переменные-указатели одинакового типа, являющиеся входными параметрами программы, то строится два варианта картины памяти: первый вариант для случая, когда переменные указывают на разные объекты, и второй, когда переменные указывают на один и тот же объект.

Аналогичным образом отслеживается возможность совпадения значений указателей внутри рекурсивной структуры, являющейся входным параметром программы. Объект, на который ссылается указатель следующего элемента в некотором элементе односвязного списка, является новым объектом, отличным от всех существующих элементов списка. Однако нельзя исключать возможность, что этот объект является одним из уже существующих предыдущих элементов списка. Тогда получаем список с переходом в кольцо. Такой список состоит из двух частей: обычного правильного списка и следующего за ним кольца. В языке cP будем использовать описатель **ring** для определения рекурсивных типов с кольцами; при отсутствии такого описателя тип `recT` определяет правильные рекурсивные структуры без колец.

В процессе потокового анализа реализуется также контроль корректности структур данных. Проверяется возможность потери структур (утечки памяти), т. е. ситуации, когда гнездо объекта становится пустым. Появление кольца в большинстве случаев является следствием ошибки. Разумеется, структура в виде кольца может также оказаться осознанным экзотическим решением программиста.

3. Трансформация программы инвертирования односвязных списков

В данном разделе сначала представлен код и описание программы инвертирования списков. Проводится потоковый анализ программы. С помощью символьного исполнения программы строится картина памяти в виде набора независимых гнезд объектов. К исходной программе применяются описанные выше трансформации устранения указателей с заменой операторов присваивания операторами присоединения. Далее операторы присоединения заменяются операторами присваивания. Эквивалентность замен доказывается. В итоге получена рекурсивная программа на языке cP.

3.1. Программа инвертирования списков

Рассмотрим программу инвертирования односвязных списков llist_reverse_order, входящей в модуль llist библиотеки ядра ОС Linux для работы со списками без блокировки (*lock-free*). Программа llist_reverse_order вызывается в семи разных модулях ядра ОС Linux. Односвязный список определяется описанием типа:

```
struct llist _ node {
   struct llist _ node *next;
};
```

Указатель **NULL** кодирует пустой список. В последнем элементе списка поле `next` равно **NULL**. В приведенном описании нет поля для собственно элемента списка, т. е. данных, хранящихся в элемен-

[1] Например, в функции `list _ sort` из библиотеки ядра ОС Linux.

те списка. В соответствии со стилем, принятым в ОС Linux, элемент списка, т. е. структура llist _ node является частью другой структуры, доступ к которой реализуется стандартным макросом container _ of, применяемым к указателю next элемента списка внутри этой структуры.

Ниже представлен код программы llist _ reverse _ order, он, а также описание доступны на сайте: https://elixir.bootlin.com/linux/v5.9/source/lib/llist.c#L79.

```
struct llist _ node *llist _ reverse _ order
(struct llist _ node *head)
{
   struct llist _ node *new _ head = NULL;

   while (head! = NULL) {
          struct llist _ node *tmp = head;
          head = head->next;
          tmp->next = new _ head;
          new _ head = tmp;
   }
   return new _ head;
}
```

Алгоритм классический. Имеются два списка. Исходный список является начальным значением параметра head. Новый список new _ head — результат инвертирования исходного списка по завершению работы программы. На очередном шаге цикла new _ head содержит некоторую начальную часть исходного списка в инвертированном виде. Оставшаяся часть исходного списка является текущим значением переменной head. Алгоритм очередного шага цикла: очередной элемент удаляется из начала списка head и вставляется в начало списка new _ head.

Программа применима также для списка с продолжением в кольцо. Инвертируется только кольцо. Такое применение вряд ли может быть практически полезным.

3.2. Построение и анализ гнезд объектов

Сначала проводится предварительный анализ программы, в ходе которого фиксируются все позиции указателей в типах, переменных и выражениях, а также вхождения нулевых указателей NULL. Тип llist _ node является рекурсивным, поскольку поле next является указателем на этот же тип llist _ node. Кроме того, в качестве значения переменной head, а также его поля next допускается NULL. Поэтому тип llist _ node отображается в стандартный рекурсивный тип recT с возможным продолжением в кольцо:

```
type ring recT = union { Empty; Node(recT next) };
```

При этом единственное поле next структуры llist _ node становится полем стандартного конструктора Node.

Построение гнезд объектов для программы llist _ reverse _ order осуществляется применением символьного исполнения программы.

В начальный момент исполнения имеется единственная переменная head. Пусть ее значением является объект Head, что будем изображать в виде head: Head. Объект Head определяет независимое гнездо, в котором находится переменная head. В результате исполнения начального оператора new _ head = NULL возникает новое независимое гнездо, определяемое объектом NULL. В этом гнезде будет находиться переменная new _ head. Таким образом, имеем пару гнезд, что записывается в виде <head: Head, new _ head: NULL>. Принципиально, что Head и NULL — разные объекты.

Необходимо рассмотреть две специальные точки программы: точку (*) сразу после while в начале цикла и точку (**) непосредственно после завершения цикла. Гнезда объектов в точке (*) определяются как результат стабилизации в процессе циклического символьного исполнения. Поскольку переменная new _ head модифицируется в цикле, необходимо обобщить значение new _ head в точке (*) введением нового объекта New _ head.

Сначала рассмотрим случай, когда объект Head в точке (*) не принадлежит кольцевой части исходного списка и объекты Head и New _ head — независимы, т. е. недостижимы друг из друга по указателям. Таким образом, картина памяти в начале цикла следующая:

```
(*) <head: Head, new _ head: New _ head>
```

Состояние символьного исполнения в конце тела цикла унифицируется с состоянием (*). В частности, проверяется, что два объекта остаются независимыми.

Исполнение условия head!= NULL в заголовке цикла модифицирует первое гнездо следующим образом. Объект Head конкретизируется в начале тела цикла в виде структуры {Head1} типа llist _ node, где Head1 — объект по полю next в структуре объекта Head. По завершению цикла, в состоянии (**) Head заменяется на NULL. Отметим, что объект Head1 отличен от объекта Head, поскольку Head находится вне кольца. Кроме того, объект Head недостижим из Head1.

В результате исполнения оператора tmp = head переменная tmp помещается в первое гнездо, что будем изображать в виде head, tmp: {Head1}. В гнезде две переменные head и tmp. При исполнении оператора head = head->next переменная head теряет прежнюю позицию в гнезде и перемещается на объект Head1 в том же гнезде. Гнездо принимает вид: tmp: {head:Head1}.

Исполнение оператора tmp->next = new _ head разрывает прежнюю связь объекта Head1, находящегося по полю next, со структурой Head. Теперь по полю next находится объект New _ head, к которому осталась прикрепленной переменная new _ head. После данного оператора объект Head1, к которому прикреплена переменная head, становится автономным независимым объектом, поскольку являлся ранее подобъектом Head, который по исходному предположению в начале цикла не

зависел от New _ head. Получаем следующую картину памяти:

```
<tmp: {new _ head: New _ head}, head: Head1>
```

Оператор tmp->next = new _ head является корректным, поскольку в результате получены правильные объекты и не один из объектов не потерян. Ошибочное присваивание (или присоединение) по полю next потенциально может привести к образованию незапланированного кольцевого списка или к потере объекта, ранее прикрепленного по полю next.

После исполнения оператора new _ head = tmp переменная new _ head отцепляется от объекта New _ head и устанавливается на головную вершину Head вместе с tmp. Оператор корректен, так как потери объекта New _ head не происходит.

Представим картину памяти по всем операторам программы:

```
struct llist _ node *llist _ reverse _ order (struct llist _ node *head){ // <head: Head>
{    struct llist _ node *new _ head = NULL; // <head: Head, new _ head: NULL>
     while                          // <head: Head, new _ head: New _ head>              (*)
       (head! = NULL) {             // <head: {Head1}, new _ head: New _ head>
        struct llist _ node *tmp = head; //<head, tmp: {Head1}, new _ head: New _ head>
        head = head->next;          // <tmp: {head: Head1}, new _ head: New _ head>
        tmp->next = new _ head;     //<tmp: {new _ head: New _ head}, head: Head1>
        new _ head = tmp;           // <tmp, new _ head: {New _ head}, head: Head1>
     }                              // <head: NULL, new _ head: New _ head>              (**)
     return new _ head;
}
```

Гнезда, полученные в конце тела цикла, унифицируются с гнездами начального состояния (*). Второе гнездо в конце тела цикла унифицируемо с первым гнездом в состоянии (*), а первое гнездо унифицируемо со вторым гнездом из состояния (*). Унификация обеспечивает стабилизацию процесса символьного исполнения: на следующих итерациях вне кольцевой части представленная картина памяти не изменится.

Далее рассмотрим случай, когда на очередной итерации цикла объект Head оказывается начальной вершиной кольцевой части списка. В состоянии (*) имеем два независимых гнезда. При исполнении оператора tmp->next = new _ head получим только одно независимое гнездо следующего вида:

```
                < head: Head1>, где
Head1 == {.. tmp: {new _ head: New _ head}.. }
```

Объект Head был модифицирован, однако остался достижимым из кольца, которому также принадлежит объект Head1. Поэтому объект Head достижим из Head1 по цепочке указателей.

Унификация одного гнезда в конце тела цикла с двумя гнездами в состоянии (*) невозможна. Поэтому символьное исполнение продолжается со следующей итерации цикла, причем в состоянии (*) останется только одно гнездо. После прохождения кольцевой части программа переходит к инвертированной начальной части. В результате работы программы будет инвертирована кольцевая часть списка, а начальная часть списка, дважды инвертированная, остается неизменной.

Проведенный анализ подтверждает корректность всех операций со структурами данных. Кроме того, что если параметр head определяет правильный список, то инвертированный список new _ head также будет правильным.

3.3. Устранение указателей

К исходной программе llist_reverse_order применяются правила трансформации, описанные в разд. 2. Результат трансформации представлен ниже в виде программы на языке cP.

```
type ring recT = union {Empty; Node
(recT next) };
recT llist _ reverse _ order(recT head)
{
   recT new _ head = Empty;
   while (head! = Empty) {
        recT tmp;
        tmp <- head;
        head <- head.next;
        tmp.next <- new _ head;
        new _ head <- tmp;
   }
   return new _ head;
}
```

Дополнительным результатом трансляции является трансформационная программа, содержащая модифицированные описания типов и заголовков функций. Трансформационная программа приведена ниже:

```
type ring recT = union { Empty; Node(recT next)};
recT llist _ reverse _ order(recT head)
```

Вследствие отсутствия собственно элемента списка в структуре типа recT задача инвертирования

и ее верификация становятся бессмысленными, поскольку инвертированный список совпадает с исходным. Необходимо явным образом определить дополнительное поле для данных, хранящихся в элементе списка. С этой целью модифицируем трансформационную программу следующим образом:

```
type T;
type slist = union { Nul, Node(T data; slist next)};
slist llist _ reverse _ order(slist head)
```

Вводится произвольный тип `T` — тип данных, хранящихся в элементе списка. Данные элемента списка представлены дополнительным полем `data`. Имя типа `recT` здесь заменяется именем `slist`. Вместо `Empty` используется имя `Nul`.

При описании типа `slist` опущен описатель **ring**, что определяет тип `slist` как правильный список без колец. Вместо этого можно при трансляции на WhyML моделировать список с продолжением в кольцо двумя правильными списками и затем вставить предусловие: кольцевая часть списка пуста.

Повторный запуск трансляции исходной программы llist_reverse_order вместе с модифицированной трансформационной программой дает следующую программу на языке cP:

```
slist llist _ reverse _ order(slist head){
   slist new _ head = Nul;
   while (head! = Nul) {
      slist tmp;
      tmp <- head;
      head <- head.next;
      tmp.next <- new _ head;
      new _ head <- tmp;
   }
   return new _ head;
}
```

3.4. Замена операторов присоединения операторами присваивания

Замена оператора присоединения `x <- y` оператором присваивания `x = y` порождает при исполнении копию объекта, значения переменной `y`. В общем случае, если одна из двух копий объекта модифицируется, то для сохранения эквивалентности программы достаточно гарантировать, что другая копия далее не используется.

Замена операторов присоединения операторами присваивания дает следующую версию программы llist_reverse_order:

```
slist llist _ reverse _ order(slist head){
   slist new _ head = Nul;
   while (head! = Nul) {
      slist tmp = head;
      head = head.next;
      tmp.next = new _ head;
      new _ head = tmp;
   }
   return new _ head;
}
```

Эквивалентность программ, новой и предыдущей, можно установить с помощью построения картины памяти после каждого оператора, отслеживая разные копии объектов. В начале тела цикла после прохождения через условие `head != Nul` получим картину памяти:

```
<head: Head, new _ head: New _ head>,
```

где `Head = Node(D, Head1)`, `D` и `Head1` обозначают новые объекты в позиции полей конструктора `Node`. Объекты `Head` и `Head1` различны, поскольку параметр `head` определяет правильный список вне кольцевой части.

Оператор `tmp = head` порождает копию объекта `Head` в качестве значения переменной `tmp`. Получим три гнезда:

```
<head: Head, tmp: Head, new _ head: New _ head>
```

После исполнения оператора `head = head.next` первая копия `Head` становится недоступной. Появляется вторая копия объекта `Head1`; первая копия `Head1` находится внутри второй копии `Head`, получим:

```
<tmp: Head, head: Head1, new _ head: New _ head>
```

Оператор `tmp.next = new _ head` заменяет в объекте `Head` компонент `Head1` на `New _ head`. Замененный компонент `Head1` становится недоступным. Изменился объект `Head`. Теперь `Head = Node(D, New _ head)`. Его обновленный компонент по полю `next` — копия объекта `New _ head`.

После исполнения оператора `new _ head = tmp` получим следующую картину памяти:

```
<tmp: Head, head: Head1, new _ head: Head>
```

Исходная копия `New _ head`, которая ранее была значением `new _ head`, становится недоступной. Однако появляется другая копия внутри объекта `Head` при копировании объекта `Head`.

Поскольку переменная `tmp` — локальная переменная тела цикла, при переходе в начало тела цикла первый объект `Head` становится недоступным вместе с находящейся в нем копией `New _ head`. Таким образом, в конце тела цикла остается действующей только одна копия каждого объекта. Гнезда объектов в конце тела цикла унифицируются с состоянием памяти в начале цикла.

Конфликта между копиями объектов нет. Новая программа, результат замены присоединения на копирование и предыдущая программа эквивалентны.

3.5. Трансформация в рекурсивную программу

Модификация поля `next` оператором `tmp.next = new _ head` является модификацией объекта, значения переменной `tmp`. В языках функционального программирования такую модификацию принято представлять явно через конструктор `Node`. Перепишем программу в следующем виде:

```
slist llist _ reverse _ order(slist head){
      slist new _ head = Nul;
      while (head != Nul) {
            slist tmp = head;
            head = head.next;
            tmp = Node(data: tmp.data, next:
            new _ head);
            new _ head = tmp;
      }
      return new _ head;
}
```

Листинг 1

Данная программа приведена к стандартному виду, где все переменные независимы, и изменение одной переменной не приведет к скрытому изменению других переменных.

Оператор `head = head.next` можно переместить в конец цикла, после этого появляется возможность устранить переменную `tmp`:

```
slist llist _ reverse _ order(slist head){
      slist new _ head = Nul;
      while (head != Nul) {
      new _ head = Node(data: head.data,
      next: new _ head);
      head = head.next;
      }
      return new _ head;
}
```

Приведенные преобразования, устраняющие переменную `tmp`, не являются обязательными. Рекурсивная программа также может быть построена и для предыдущей версии программы.

Далее по данной программе построим эквивалентную рекурсивную программу. Цикл заменяется рекурсивной функцией. Назовем ее `reverseC`. Ее параметры: `new _ head` и `head`, поскольку они модифицируются в конце тела цикла. Эти перевычисления заменяются рекурсивным вызовом. Отметим, что преобразование в рекурсивную программу нетрудно провести автоматически.

Построим также спецификацию рекурсивной программы в виде предусловий и постусловий. Чтобы упростить верификацию, спецификация ориентирована на язык why3. Вводится функция `tolist`, отображающая список типа `slist` в стандартный список типа `list` языка why3 с конструкторами `Nil` и `Cons`. Функция `reverse` для реверсирования стандартного списка и операция конкатенация списков `++` определены в стандартной библиотеке системы Why3 [5].

Здесь **`ensures`** предшествует утверждению, определяющему постусловие текущей определяемой функции. Стандартная переменная **`result`** соответствует возвращаемому результату текущей функции. Предусловия отсутствуют, поскольку допускаются произвольные списки.

Итак, результатом трансформации исходной программы на языке Си являются две программы на языке cP: программа из листинга 1 в виде цикла **`while`** и рекурсивная программа из листинга 2.

4. Верификация программы в системе Why3

Рассмотрим три варианта дедуктивной верификации программы инвертирования списков в системе автоматического доказательства Why3 [5] и проведем их сравнение. Используется система Why3 версии 1.3.1 с SMT-решателями Z3 4.8.6, CVC3 2.4.1, CVC4 1.7.

Система Why3 базируется на языке функционального программирования WhyML. Его частью является язык спецификаций why3. Для дедуктивной верификации применяется классический метод Хоара [26] с генерацией формул корректности на языке why3. Для доказательства формул корректности к системе Why3 может быть подключено более 20 разнообразных инструментов автоматического доказательства теорем. Это автоматические решатели, SMT-решатели и интерактивные инструменты, такие как PVS [31], HOL и Coq [25], в которых пользователь строит доказательство, применяя команды инструмента.

Система Why3 имеет собственные средства интерактивного доказательства. Ранее этими средствами в комбинации с SMT-решателями иногда не удавалось завершить доказательство. Некоторые ветви доказательства приходилось переводить в систему Coq, где доказательство часто было длительным и сложным [7]. Средства интерактивного доказательства системы Why3 были расширены в последней версии 1.3.1, что позволило полностью проводить доказательство в системе Why3 без перехода в систему Coq.

```
type listD = list(T);
function tolist(slist x): listD = if x = Nul then Nil else Cons(x.data, tolist(x.next));

slist llist _ reverse _ order(slist head)
  ensures tolist(result) = reverse(tolist(head))
{   return reverseC(Nul, head); }

slist reverseC(slist new _ head, head)
  ensures tolist(result) = reverse(tolist(head)) ++ tolist(new _ head)
{
  if head = Nul then return new _ head
  else return reverseC(Node(head.data, new _ head), head.next)

}
```

Листинг 2

4.1. Верификация рекурсивной программы

Рекурсивную программу из листинга 2 перепишем на язык WhyML:

```
theory Slist
   use int.Int, list.List, list.Reverse, list.Append
   type data
   type listD = list data
   type slist = Nul | Node data slist

let rec function tolist(x: slist): listD =
   match x with
     | Nul -> Nil
     | Node d sl -> Cons d (tolist sl)
   end

let rec function reverseC(new _ head head: slist): slist
   ensures{ tolist result = reverse (tolist
head) ++ tolist new _ head }
=
   match head with
     | Nul -> new _ head
     | Node d sn -> reverseC (Node d new _ head) sn
   end
let function llist _ reverse _ order(head: slist): slist
ensures{ tolist result = reverse (tolist head) }
=
   reverseC Nul head
end (*Slist*)
```

В языке WhyML у полей конструкторов нет имен. Поэтому доступ к полям возможен только через оператор **match**.

По данной программе сгенерированы две формулы корректности. Они были доказаны в системе Why3 автоматически запуском простейшей стратегии **auto level 0**.

4.2. Верификация предикатной программы

Рекурсивная программа из листинга 2 была переписана на язык предикатного программирования P [9, 10]. В соответствии с методом дедуктивной верификации предикатных программ [6, 7, 11, 12] сгенерированы формулы корректности. Они представлены ниже в виде теории на языке why3:

```
theory SlistPred
   use int.Int, list.List, list.Reverse, list.Append, list.Length
   type tT
   type listD = list tT
   type slist = Nul | Node tT slist
   let rec function tolist(x: slist): listD =
      match x with
        |Nul -> Nil
        |Node d sl -> Cons d (tolist sl)
      end
   predicate qRev(head new _ head: slist) = tolist new _ head = reverse (tolist head)
   predicate reverseG(new _ head head new _ head1: slist) =
      tolist new _ head1 = reverse (tolist head) ++ tolist new _ head
goal RB1: forall head new _ head: slist. reverseG Nul head new _ head -> qRev head new _ head
goal COR: forall head new _ head new _ head1: slist.
          head = Nul /\ new _ head1 = new _ head -> reverseG new _ head head new _ head1
goal RB2: forall head: slist.
       match head with
         |Nul -> true
         |Node d sn -> length (tolist sn) < length (tolist head)
       end
goal RB3: forall head new _ head new _ head1: slist.
       match head with
         |Nul -> true
         |Node d sn -> reverseG (Node d new _ head) sn new _ head1 ->
              reverseG new _ head head new _ head1
       end
end (*SlistPred*)
```

Четыре цели, соответствующие сгенерированным формулам корректности, были доказаны в системе Why3 автоматически запуском простейшей стратегии **auto level 0**.

4.3. Сравнение двух методов дедуктивной верификации

Интересно сопоставить формулы корректности, полученные разными методами дедуктивной верификации. Формула RB1 соответствует второй цели:

```
goal llist _ reverse _ order'vc:
  forall head:slist.
   let o = Nul in
   let result = reverseC o head in
   tolist result = (reverse (tolist head) ++
      tolist o) ->
   tolist result = reverse (tolist head)
```

Принципиальное отличие формул в том, что в RB1 вместо вызова функции reverseC используется постусловие reverseG. В остальном формулы эквивалентны.

Для формулы RB2 нет эквивалента. Аналогичная формула была бы сгенерирована по спецификации:

```
variant { length (tolist head)}
```

Система Why3 сумела статически определить, что функция reverseC всегда завершается, поэтому генерации формулы корректности не потребовалась.

Конъюнкция формул COR и RB3 эквивалентна первой цели:

```
goal reverseC'vc:
  forall new _ head:slist, head:slist.
  forall result:slist.
   match head with
   | Nul -> result = new _ head
   | Node d sn ->
   tolist result = (reverse (tolist sn) ++
      tolist (Node d new _ head))
   end -> tolist result = (reverse (tolist head) ++
      tolist new _ head)
```

В целом следует констатировать, что системы формул корректности, сгенерированные двумя разными методами дедуктивной верификации, оказались эквивалентными, несмотря на множество различий в оформлении.

4.4. Верификация императивной программы

Императивную программу из листинга 1 перепишем на язык WhyML. Модифицируемые переменные необходимо снабжать описателем **ref**. Модификация значений таких переменных возможна лишь операцией <-, определенной в библиотеке системы Why3. Оператора присваивания в языке WhyML нет.

Первоначально инвариант цикла был получен из постусловия программы reverseC. Однако при этом итоговая формула корректности не доказывалась. Правильный инвариант удалось найти не сразу.

Система Why3 потребовала вставить спецификацию **variant** для доказательства того, что цикл всегда завершается. Используя эту информацию, Why3 включила условие завершения цикла в итоговую формулу корректности.

Система Why3 сгенерировала одну формулу корректности. В действительности итоговая формула объединяет пять разных формул корректности, получаемых из итоговой применением команды **split _ vc**. Эти формулы, кроме последней, были автоматически доказаны применением стратегии **auto level 0**. При доказательстве последней формулы потребовалось применить команду **destruct** для декомпозиции одной из посылок исходной формулы.

Перевод императивной программы из листинга 1 на язык WhyML и доказательство формул корректности оказались значительно сложнее, чем для рекурсивной программы из листинга 2.

Программа на языке WhyML приведена ниже.

Заключение

В рамках проекта по верификации библиотеки ядра ОС Linux разработана технология трансляции программ с языка Си на промежуточный язык cP с устранением указателей и упрощением программы. Полученная программа конвертируется на язык WhyML, специфицируется и верифицируется в рамках инструмента Why3 [5].

Разработана универсальная система трансформаций, устраняющих указатели, для операций с массивами и рекурсивными структурами данных. Методы трансформации отрабатывались на десятке нетривиальных программ из библиотеки ядра ОС Linux. Планируется разработка транслятора с языка Си на язык WhyML с перспективой его дальнейшей формальной верификации на базе формальной операционной семантики.

Трансляция с языка Си на язык WhyML, реализуемая при дедуктивной верификации программ по данной технологии, проводится пока вручную. Но даже такая технология вполне работоспособна. Применяемые трансформации детально документируются. Проверка их корректности оценщиком, сертифицирующим итоги дедуктивной верификации, вполне осуществима и не сложнее, например, проверки корректности спецификации программы.

В настоящей статье представлены методы трансформации и дедуктивной верификации программ с рекурсивными структурами данных на языке Си на примере программы инвертирования односвязных списков. Оператор присоединения в языке cP позволяет адекватно трансформировать присваивания указателям. Дальнейшая замена присоединения на присваивание требует дополнительного нетривиального анализа программы.

```
theory SlistImp
   use int.Int, list.List, list.Length, list.Reverse, list.Append
   type data
   type listD = list data
   type slist = Nul | Node data slist

   let rec function tolist(x: slist): listD =
      match x with
        | Nul -> Nil
        | Node d sl -> Cons d (tolist sl)
       end

   let function llist _ reverse _ order(head0: slist): slist
     ensures{ reverse(tolist result) = tolist head0 }
   =
   let ref new _ head = Nul in
   let ref head = head0 in
   while match head with | Nul -> false | Node _ _ -> true end do
     invariant { reverse (tolist head0) =
                    reverse (tolist head) ++ tolist new _ head }
     variant { length (tolist head)}
     match head with
       | Nul -> absurd
       | Node d ne ->
         let ref tmp = head in
         head <- ne;
         tmp <- Node d new _ head;
         new _ head <- tmp
     end
   done;
   new _ head
end (*SlistImp *)
```

Дедуктивная верификация двух рекурсивных программ, полученных в результате трансформации, прошла автоматически без дополнительных лемм, что подтверждает преимущество предлагаемых методов и высокую эффективность последней версии системы автоматического доказательства Why3. Методы дедуктивной верификации, классический [26] и разработанный в предикатном программировании [12], оказались практически равноценными. Дедуктивная верификация полученной при трансформации императивной программы из листинга 1 оказалась существенно проще, чем в работе [30], где потребовалось введение пяти дополнительных лемм, однако при этом значительно сложнее, чем дедуктивная верификация рекурсивной программы из листинга 2.

Метод трансляции на язык cP с устранением указателей для рекурсивных структур был проверен на более сложных программах, а именно — сортировки двусвязных списков `list _ sort` из библиотеки ядра ОС Linux и нерекурсивного алгоритма вставки в сбалансированное по высоте двоичное дерево (АВЛ-дерево). Данный алгоритм рассматривался ранее в работе [33]. Для этих программ предстоит проверить разработанный метод замены присоединения на присваивание.

Планируется развитие предлагаемых методов трансформации в целях дедуктивной верификации библиотеки ядра ОС Linux для следующих объектов с рекурсивной структурой: односвязных списков (модуль `llist`), двусвязных списков (`list`), односвязных списков с асинхронным доступом (`klist`), красно-черных деревьев (`rbtree`).

Список литературы

1. **Ефремов Д.В., Мандрыкин М. У.** Формальная верификация библиотечных функций ядра Linux // Труды ИСП РАН. — 2017. — Т. 29, № 6. — С. 49—76.

2. **AstraVer Toolset:** инструменты дедуктивной верификации моделей и механизмов защиты ОС. ИСП РАН. URL: http://linuxtesting.org/astraver (дата обращения 12.11.2020).

3. **Mandrykin M. U., Khoroshilov A. V.** Region analysis for deductive verification of C programs // Programming and Computer Software. — 2016. — Vol. 42, Issue 5. — P. 257—278.

4. **Девянин П. Н., Ефремов Д. В., Кулямин В. В., Петренко А. К., Хорошилов А. В., Щепетков И. В.** Моделирование и верификация политик безопасности управления доступом в операционных системах. — М.: Горячая линия — Телеком, 2019. — 214 с. URL: http://www.ispras.ru/publications/security_policy_modeling_and_verification.pdf (дата обращения 12.11.2020).

5. **Why3.** Where Programs Meet Provers. URL: http://why3.lri.fr (дата обращения 12.11.2020).

6. **Шелехов В. И.** Верификация предикатной программы бинарного поиска объекта произвольного типа // Системная информатика. — 2019. — № 15. — С. 45—64.

7. **Шелехов В. И.** Верификация предикатной программы пирамидальной сортировки с применением обратных трансформаций // Системная информатика. — 2020. — № 16. — С. 75—102.

8. **Шелехов В. И.** Дедуктивная верификация программы конкатенации строк с применением обратной трансформации // Знания—Онтологии—Теории (ЗОНТ-19). — Новосибирск, ИМ СО РАН, 2019. — С. 369—378. URL: http://persons.iis.nsk.su/files/persons/pages/logcflc1.pdf (дата обращения 14.12.2020).

9. **Карнаухов Н. С., Першин Д. Ю., Шелехов В. И.** Язык предикатного программирования Р. — Новосибирск: ИСИ СО РАН, 2018. — 45 с. URL: http://persons.iis.nsk.su/files/persons/pages/plang14.pdf (дата обращения: 14.12.2020).

10. **Шелехов В. И.** Семантика языка предикатного программирования // Знания—Онтологии—Теории (ЗОНТ-15). — Новосибирск, 2015. — 13 с. URL: http://persons.iis.nsk.su/files/persons/pages/semZont1.pdf (дата обращения: 14.12.2020).

11. **Шелехов В. И.** Доказательное построение, верификация и синтез предикатных программ // Знания—Онтологии—Теории (ЗОНТ-2017). Том 2. — Новосибирск: Институт Математики СО РАН, 2018. — С. 156—165. URL: http://persons.iis.nsk.su/files/persons/pages/lbase.pdf (дата обращения: 14.12.2020).

12. **Шелехов В. И., Чушкин М. С.** Верификация программы быстрой сортировки с двумя опорными элементами // Научный сервис в сети Интернет: Труды XX Всероссийской научной конференции. — М.: ИПМ им. М. В. Келдыша, 2018. — С. 511—524.

13. **Каблуков И. В., Шелехов В. И.** Реализация оптимизирующих трансформаций в системе предикатного программирования // Системная информатика. — 2017. — № 11. — С. 21—48.

14. **Boockmann J. H., Lüttgen G., Mühlberg J. T.** Generating Inductive Shape Predicates for Runtime Checking and Formal Verification // ISoLA 2018: Leveraging Applications of Formal Methods, Verification and Validation. Verification. — 2018. — LNCS 11245. — P. 64—74.

15. **Jung C., Clark N. DDT:** Design and evaluation of a dynamic program analysis for optimizing data structure usage // 42nd Int. Symposium on Microarchitecture (MICRO 42). — NY, 2009. — P. 56—66.

16. **White D., Rupprecht T., Luttgen G.** DSI: An Evidence-based Approach to Identify Dynamic Data Structures in C Programs // Intl. Symposium on Software Testing and Analysis (ISSTA 2016). ACM, 2016. — P. 259—269.

17. **Haller I., Slowinska A., Bos H.** Scalable data structure detection and classification for C/C + + binaries // Empirical Software Engineering. — 2016. — Vol. 21, Issue 3. — P. 778—810.

18. **Dudka K., Holík L., Peringer P., Trtík M., Vojnar T.** From Low-Level Pointers to High-Level Containers // Verification, Model Checking, and Abstract Interpretation. — 2016. — LNCS 9583. — P. 431—452.

19. **White D. H., Lüttgen G.** Identifying Dynamic Data Structures by Learning Evolving Patterns in Memory // Tools and Algorithms for the Construction and Analysis of Systems. — 2013. — LNCS 7795. — P. 354—369.

20. **Holík L., Lengál O., Rogalewicz A., Šimáček J., Vojnar T.** Fully Automated Shape Analysis Based on Forest Automata // Computer Aided Verification. CAV 2013. — LNCS 8044. — P. 740—755.

21. **Plaisted D.** Source-to-Source Translation and Software Engineering // J. of Software Engineering and Applications. — 2013. — Vol. 6, No. 4A. — P. 30—40.

22. **Trudel M., Furia C. A., Nordio M., Meyer B., Oriol M.** C to O-O Translation: Beyond the Easy Stuff // 19th Working Conference on Reverse Engineering, 2012. — P. 19—28.

23. **Martin J., Müller H. A.** Strategies for migration from C to Java. CSMR // IEEE Computer Society, 2001. — P. 200—210.

24. **Novosoft C2J:** a C to Java translator. 2001. URL: http://www.novosoft-us.com/solutions/product_c2j.shtml (дата обращения: 24.02.2021).

25. **The Coq** Proof Assistant. URL: http://coq.inria.fr (дата обращения: 14.12.2020).

26. **Hoare C. A. R.** An axiomatic basis for computer programming // Communications of the ACM. — 1969. — Vol. 12, No. 10. — P. 576—585.

27. **Reynolds J. C.** Separation Logic: A Logic for Shared Mutable Data Structures // LICS '02 Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science. IEEE, 2002. — P. 55—74.

28. **Leino K. R. M.** Specification and verification of object-oriented software.- Marktoberdorf International Summer School, 2008. — 36 p.

29. **Meyer B.** Towards a Calculus of Object Programs // Patterns, Programming and Everything, Judith Bishop Festschrift. — Springer-Verlag, 2012. — P. 91—128.

30. **Шелехов В. И.** Дедуктивная верификация и реализация предикатной программы инвертирования односвязных списков // Информационные и математические технологии в науке и управлении. — 2018. — № 3(11). — С. 136—146. URL: http://persons.iis.nsk.su/files/persons/pages/listinvert.pdf (дата обращения: 14.12.2020).

31. **Owre S., Rushby J. M., Shankar N.** PVS: Specification and Verification System // CADE-11: Automated Deduction. LNCS. — 1992. — Vol. 607. — P. 748—752.

32. **Jones C. B., Yatapanage N.** Reasoning about Separation Using Abstraction and Reification // SEFM 2015. — 2015. — LNCS 9276. — P. 3—19.

33. **Шелехов В. И.** Предикатная программа вставки в АВЛ-дерево // Системная информатика. — 2017. — № 9. — С. 23—42.

34. **Blanchard A., Kosmatov N., Loulergue F.** Ghosts for Lists: A Critical Module of Contiki Verified in Frama-C // Tenth NASA Formal Methods. — 2018. — LNCS 10811. — P. 37—53. URL: https://hal.inria.fr/hal-01720401/document (дата обращения 12.01.2021).

35. **Volkov G., Mandrykin M., Efremov D.** Lemma functions for Frama-C: C programs as proofs // Proceedings of the 2018 Ivannikov ISPRAS Open Conference (ISPRAS-2018), 2018. — P. 31—38.

Applying Program Transformations for Deductive Verification of the List Reverse Program

V. I. Shelekhov, vshel@iis.nsk.su, A. P. Ershov Institute of Informatics Systems, Novosibirsk, 630090, Russian Federation

Corresponding author:

Shelekhov Vlaimir I., Head of Laboratory. A. P. Ershov Institute of Informatics Systems, Novosibirsk, 630090, Russian Federation
E-mail: vshel@iis.nsk.su

Received on December 16, 2020
Accepted on February 01, 2021

The program transformation methods to simplify the deductive verification of programs with recursive data types are investigated. The list reversion program is considered as an example. A source program in the C language is translated to the cP functional language which includes no pointers. The resulting program is translated further to the WhyML language to perform deductive verification of the program. The cP language includes the same constructs of the C language except pointers. In the C program, all actions that include pointers are replaced by the equivalent fragments without pointers. These replacement are performed by the special transformations using the results of the program dataflow analysis.

Three variants of deductive verification of the transformed list reverse program in the Why3 verification platform with SMT solvers (Z3 4.8.6, CVC3 2.4.1, CVC4 1.7) are performed. First, the recursive WhyML program supplied with specifications was automatically verified successfully using only SMT solvers. Second, the recursive program was translated to the P predicate language. Correctness formulae were constructed for the P program and translated further to the why3 specification language. The formulae proving correctness were easy like the first variant. But correctness formulae for the first and second variants were different. Third, the "imperative" WhyML program that included while loop with additional invariant specifications was verified. The proving was easy but not automatic. So, for deductive verification, recursive program variant appears to be more preferable against imperative program variant.

Keywords: *deductive verification, program transformation, functional program, predicate program, algebraic data type, linked list, Linux operating system*

For citation:

Shelekhov V. I. Applying Program Transformations for Deductive Verification of the List Reverse Program, *Programmnaya Ingeneria,* 2021, vol. 12, no. 3, pp. 127—139

DOI: 10.17587/prin.12.127-139

References

1. **Efremov D., Mandrykin M.** Formal verification of Linux kernel library functions, *Trudy ISP RAN*, 2017, vol. 29, no. 6, pp. 49—76 (in Russian).

2. **AstraVer Toolset:** deductive verification tools for OS protection models and mechanisms, ISP RAN, available at: http://linuxtesting.org/astraver

3. **Mandrykin M. U., Khoroshilov A. V.** Region analysis for deductive verification of C programs, *Programming and Computer Software*, 2016, vol. 42, no. 5, pp. 257—278.

4. **Devyanin P., Efremov D., Kulyamin V., Petrenko A., Khoroshilov A., Shchepetko I.** *Modeling and verification of access control security policies in operating systems,* Moscow, Goryachaya-liniya-telekom, 2019, 261 p. (in Russian).
5. **Why3.** Where Programs Meet Provers, available at: http://why3.lri.fr
6. **Shelekhov V.** Deductive verification of a predicate program of binary search of of an arbitrary type object, *Sistemnaya informatika*, 2019, no. 15, pp. 45—64 (in Russian).
7. **Shelekhov V.** Verification of the heapsort predicate program using inverse transformations, *Sistemnaya informatika*, 2020, no. 16, pp. 75—102 (in Russian).
8. **Shelekhov V.** Deductive verification of a string concatenation program using a reverse transformation, *Znaniya—ontologii—teorii (ZONT-19)*, Novosibirsk, Institut matematiki SO RAN, 2019, pp. 369—378 (in Russian).
9. **Kharnaukhov N., Perchine D., Shelekhov V.** *Yazyk predikatnogo programmirovaniya P,* Novosibirsk, ISI SB RAN, 2018, 45 p. (in Russian).
10. **Shelekhov V.** Formal Semantics of the P Predicate Language, *ZONT-15*, Novosibirsk, 2015. 13 p., available at: http://persons.iis.nsk.su/files/persons/pages/semZont1.pdf (in Russian).
11. **Shelekhov V.** Model-based construction, verification and synthesis of predicate programs, *Znaniya—ontologii—teorii (ZONT-17)*, vol. 2. Institut matematiki SO RAN, Novosibirsk, 2018, pp. 156—165 (in Russian).
12. **Shelekhov V.**, **Chushkin M.** Verification of a quicksort program with two pivots, *Nauchnyj servis v seti internet, Trudy XX vserossijskoj nauchnoj konferencii*, 2018, pp. 511—524 (in Russian).
13. **Kablukhov I., Shelekhov V.** Implementation of optimizing transformations in the predicate programming system, *Sistemnaya informatika*, 2017, no. 11, pp. 21—48 (in Russian).
14. **Boockmann J. H., Lüttgen G., Mühlberg J. T**. Generating Inductive Shape Predicates for Runtime Checking and Formal Verification, *Leveraging Applications of Formal Methods, Verification and Validation*, 2018, LNCS 11245, pp. 64—74.
15. **Jung C., Clark N.** DDT: Design and evaluation of a dynamic program analysis for optimizing data structure usage, *42nd Int. Symposium on Microarchitecture (MICRO 42)*, NY, 2009, pp. 56—66.
16. **White D., Rupprecht T., Lüttgen G.** DSI: An Evidence-based Approach to Identify Dynamic Data Structures in C Programs, *Intl. Symposium on Software Testing and Analysis (ISSTA 2016)*, ACM, 2016, pp. 259—269.
17. **Haller I., Slowinska A., Bos H**. Scalable data structure detection and classification for C/C++ binaries, *Empirical Software Engineering*, 2016, vol. 21, no. 3, pp. 778—810.
18. **Dudka K., Holík L., Peringer P., Trtík M., Vojnar T.** From Low-Level Pointers to High-Level Containers, *Verification, Model Checking, and Abstract Interpretation*, 2016, LNCS 9583, pp. 431—452.
19. **White D. H., Lüttgen G.** Identifying Dynamic Data Structures by Learning Evolving Patterns in Memory, *Tools and Algorithms for the Construction and Analysis of Systems,* 2013, LNCS 7795, pp. 354—369.
20. **Holík L., Lengál O., Rogalewicz A., Šimáček J., Vojnar T.** Fully Automated Shape Analysis Based on Forest Automata, *Computer Aided Verification. CAV 2013*, 2013, LNCS 8044, pp. 740—755.
21. **Plaisted D**. Source-to-Source Translation and Software Engineering, *J. of Software Engineering and Applications*, 2013, vol. 6, no. 4A, pp. 30—40.
22. **Trudel M., Furia C. A., Nordio M., Meyer B., Oriol M**. C to O-O Translation: Beyond the Easy Stuff, *19th Working Conference on Reverse Engineering*, 2012, pp. 19—28.
23. **Martin J., Müller H. A**. Strategies for migration from C to Java. CSMR, *IEEE Computer Society*, 2001, pp. 200—210.
24. **Novosoft** C2J: a C to Java translator, available at: http://www.novosoft-us.com/solutions/product_c2j.shtml
25. **The Coq Proof Assistant**, available at: http://coq.inria.fr
26. **Hoare C. A. R**. An axiomatic basis for computer programming, *Communications of the ACM,* 1969, vol. 12, no. 10, pp. 576—585.
27. **Reynolds J. C**. Separation Logic: A Logic for Shared Mutable Data Structures, *LICS '02 Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science,* IEEE, 2002, pp. 55—74.
28. **Leino K. R. M.** *Specification and verification of object-oriented software,* Marktoberdorf International Summer School, 2008, 36 p.
29. **Meyer B**. Towards a Calculus of Object Programs, *Patterns, Programming and Everything, Judith Bishop Festschrift*, Springer-Verlag, 2012, pp. 91—128.
30. **Shelekhov V.** Deductive verification of a predicate program for inverting singly linked lists, *Informacionnye i matematicheskie tekhnologii v nauke i upravlenii*, 2018, no. 3 (11), pp. 136—146 (in Russian).
31. **Owre S., Rushby J. M., Shankar N.** PVS: Specification and Verification System, *CADE-11: Automated Deduction*, *LNCS*, 1992, vol. 607, pp. 748—752.
32. **Jones C. B., Yatapanage N**. Reasoning about Separation Using Abstraction and Reification, *SEFM 2015*, 2015, LNCS 9276, pp. 3—19.
33. **Shelekhov V.** Predicate program for insertion into AVL-tree, *Sistemnaya informatika*, 2017, no. 9, pp. 23—42 (in Russian).
34. **Blanchard A., Kosmatov N., Loulergue F**. Ghosts for Lists: A Critical Module of Contiki Verified in Frama-C, *Tenth NASA Formal Methods*, 2018, LNCS 10811, pp. 37—53.
35. **Volkov G., Mandrykin M., Efremov D.** Lemma functions for Frama-C: C programs as proofs, *Proceedings of the 2018 Ivannikov ISPRAS Open Conference (ISPRAS-2018)*, 2018, pp. 31—38.

ИНФОРМАЦИЯ

УДК 004.912 DOI: 10.17587/prin.12.140-149

С. И. Паринов, д-р техн. наук, гл. науч. сотр., sparinov@gmail.com,
Федеральное государственное бюджетное учреждение науки Центральный экономико-математический институт РАН Российская академия народного хозяйства и государственного управления (РАНХ и ГС), Москва

Тематическое моделирование контекстов цитирований из научных публикаций: структура научного потребления автора

Контексты цитирований из научных публикаций, как правило, содержат информацию о причинах и характере использования учеными результатов друг друга. Извлекая эту информацию из контекстов, можно создавать наборы наукометрических характеристик. В статье систематизируются общие возможности использования данных из контекстов цитирований для развития анализа сетей цитирований. Как одно из приложений представлен подход к построению тематической структуры научного потребления ученого на основе топиков, полученных в результате тематического моделирования массива контекстов цитирований из его публикаций. Обсуждены особенности представления тематической структуры в виде "дерева слов" и потоковой диаграммы. Рассмотрены возможные направления развития данного подхода. Подобные тематические структуры являются новым перспективным источником данных как для наукометрических исследований, так и для создания новых научных сервисов.

Ключевые слова: *проект Сиртек, контексты цитирований, тематическое моделирование, топики, структура научного потребления ученого, наукометрический анализ*

Введение

Контексты цитирований в научной публикации представляют собой текстовые фрагменты, расположенные слева и справа от ссылок на источники списка литературы в полном тексте публикации. В последние десятилетия контексты цитирований стали объектом различных и многочисленных исследований, так как исследователи получили доступ к некоторым источникам подобных данных. Например, издательство Public Library of Science (PLOS) предоставляет публикации в размеченном XML-формате, что позволяет достаточно просто извлекать из них контексты цитирований [1—3]. Аналогичные возможности предоставляют издательства PubMed и Elsevier, а также крупный агрегатор научных публикаций Microsoft Academic.

В России пока нет крупных академических издательств, которые бы предоставляли доступ к большому числу размеченных научных публикаций с выделенными в них содержательными и структурными элементами, включая цитирования. В этих условиях требуются сервисы, которые выполняют распознавание и выделение данных о цитированиях из полных текстов публикаций в формате PDF. Такой сервис был создан в проекте Сиртек[1], финансируемом РАНХиГС[2] [4].

Сиртек предоставляет открытый доступ к данным о цитированиях для большого числа научных публикаций, включая контексты цитирований [5]. Созданная в Сиртек программная технология по извлечению из PDF-версий публикаций данных о цитированиях получает исходные данные из научных информационных система RePEc[3] и Соционет[4], а также из репозиториев некоторых организаций.

Расширение традиционного набора данных о научных цитированиях и сетях цитирований [6] путем добавления к ним данных из контекстов цитирований существенно обогащает возможности наукометрического анализа процессов создания нового научного знания и участия ученых в этом процессе. Контексты цитирований, в частности, уже используются для анализа отношений между цитирующими и цитируемыми публикациями [7], включая изучение тональности и функций цитирований [8, 9]. Достаточно полное представление о примерах исследований, выполненных на основе контекстов цитирований, дает обзор публикаций 2006—2018 гг. на эту и близкие темы [10], а также обзор публикаций о применении для обработки контекстов цитирований методов анализа естественного языка и машинного обучения [11].

Такие исследования, как правило, основываются на гипотезе, что в содержании контекстов цитирований из научных публикаций отражены характеристики процесса трансформации научного знания, включая данные об использованных в этом процессе

[1] http://cirtec.ranepa.ru/
[2] https://www.ranepa.ru/
[3] http://repec.org/
[4] https://socionet.ru/

результатах исследований из цитируемых публикаций, а также особенности участия в этом процессе авторов цитирующих и цитируемых публикаций.

Фактически предполагается, что в содержании публикации и, в частности, в контекстах цитирований, автор определенным образом описал:

а) какую информацию из цитируемых публикаций он использовал;

б) каким образом он эту информацию применил в своей работе;

в) какие цели он при этом преследовал;

г) что у него получилось в результате.

По сложившейся практике научного цитирования непосредственно в контекстах цитирований ученый обычно приводит информацию, упомянутую в п. а). Эта информация, как правило, содержит данные: 1) о цитируемых источниках, включая их авторов, название, год публикации и т. д.; 2) признаки со-цитирования источников; 3) множество слов, содержащие, в том числе, профессиональные термины, лексические клише, слова-маркеры, указывающие на определенные функции или тональность цитирований.

Подобные данные, собранные вместе из публикаций некоторого заданного ученого, могут рассматриваться как характеристики этого ученого. В них содержится информация о том, какие публикации каких авторов заданный ученый использовал в своей работе и с какой частотой он их цитирует. В терминах социально-экономических наук подобные характеристики использования чего-то для создания нового продукта можно называть характеристиками научного потребления ученого. Дополнительные характеристики научного потребления ученого содержатся также в значении слов из контекстов цитирований.

Множество слов из контекстов цитирований, в частности, позволяют получить тематические характеристики научного потребления ученого. Для этого необходимо выделить в контекстах цитирований из публикаций ученого характерные ключевые слова или фразы, представляющие тематику цитирований им работ других ученых. Число повторов таких слов в контекстах цитирований ученого дает представление о значимости отдельных слов (тем) в составе научного потребления, а сочетание этих слов позволяет получить представление о тематической структуре научного потребления.

Характеристики научного потребления ученых на основе данных из контекстов цитирований и, в частности, тематическая структура научного потребления дополняют традиционные наукометрические показатели и характеристики. Например, добавление подобных данных к традиционному индексу цитирований ученого позволяет расширить возможности анализа публикационной активности и других аспектов научной деятельности ученых.

Одним из методов выявления тематической структуры содержания контекстов цитирований является популярный в компьютерной лингвистике метод тематического моделирования[1]. В наукометрических исследованиях этот метод используется для решения различных задач. К их числу относятся: классификация авторов [12, 13]; анализ изменений во времени тематики научных публикаций [14]; анализ содержания научных статей [15] и др. В приложении к анализу цитирований с помощью тематического моделирования исследуются, например, межстрановые потоки научных знаний, проявляющиеся в данных о публикациях и цитированиях [16], поиск тематических связей в цитированиях [17], комплексное тематическое моделирование текста публикаций и цитирований [18] и т. п.

В проекте Сиртек, в ходе выполнения которого получены представленные в настоящей статье результаты, с помощью тематического моделирования проведены эксперименты по созданию тематической структуры научного потребления. В качестве исходных данных для этого проекта были использованы контексты цитирований, созданные в этом же проекте. Контексты цитирований были сгруппированы для множества публикаций некоторого числа заданных авторов. Для каждого отдельного автора на множестве его контекстов цитирований было проведено тематическое моделирование при заданных условиях: создаются 20 топиков по пять слов каждый. В результате для каждого автора получен набор топиков, характеризующих обобщенную тематику содержания массива контекстов заданного автора.

С использованием данных о связи полученных топиков с контекстами цитирований проведены группировки и упорядочивания слов из топиков. При этом учитывались наличие определенных видов общности между словами и значимость этих слов для заданного автора. Значимость слов определяется числом связей топиков, в которые входит соответствующее слово, с контекстами цитирования. Чем больше таких связей, тем более значимо соответствующее слово.

Проведено визуальное представление упорядоченных структур слов и связей между ними в виде диаграмм трех видов: хордовая диаграмма, "дерево слов" и потоковая диаграмма. Таким образом, было создано несколько вариантов представления структуры научного потребления заданного автора, которые могут быть использованы как для наукометрического анализа, так и для создания новых сервисов для развития онлайновой научной инфраструктуры и повышения эффективности научных исследований.

Возможности развития анализа сетей цитирований с использованием контекстов цитирований

Контексты цитирований по определению содержат качественную информацию о связях между цитирующей и цитируемой публикациями и/или об отношении автора цитирующей публикации к цитируемым источникам. Следовательно, содержание этих контекстов может быть использовано для получения качественных характеристик связей как между цитирующими и цитируемыми публикациями, так и между их авторами. С точки зрения сетей цитирования[2], это означает, что традиционные связи цитирований (дуги между

[1] https://ru.wikipedia.org/wiki/Тематическое_моделирование.

[2] https://en.wikipedia.org/wiki/Citation_network

узлами сети цитирований) в данном случае получают дополнительные качественные атрибуты, выводимые из содержания соответствующих контекстов.

Рассмотрим это подробнее на примере сети цитирований ученых: в ее узлах находятся ученые-авторы, представленные множеством своих публикаций, а дуги сети цитирований связывают публикации ученых с цитируемыми и цитирующими публикациями. В данном случае каждый узел (ученый) такой сети цитирований характеризуется множеством исходящих и входящих дуг (связей цитирования).

Исходя из общепринятой практики оформления цитирований в научных публикациях, контексты цитирований в общем случае позволяют извлечь перечисленные далее типы данных, являющиеся атрибутами соответствующих связей цитирования.

1. Источники, которые цитируются в контекстах. Источники, упоминаемые в контекстах, извлеченных из некоторого множества публикаций, могут быть представлены как список уникальных, включая подсчет числа их повторов в данном множестве.

2. Авторы источников. Например, в виде списка уникальных авторов с числом повторов упоминаний (цитирований) публикаций данного автора в контекстах.

3. Со-цитированные источники и авторы источников, т. е. публикации и их авторы, которые имеют общий контекст цитирования и цитируются перечислением. Содержание контекстов с со-цитированиями позволяет получить некоторую информацию о характере общности со-цитируемых источников и их авторов.

4. Семантическое структурирование содержания контекстов с точки зрения тематики цитирования, функций цитирования, тональности и т. п. Например, выделение часто используемых ключевых слов, профессиональных терминов, слов семантических маркеров (слова-маркеры) и т. п.

5. Данные о расположении контекстов цитирований в теле публикации. Цитирования во введении и заключении обычно отличаются по значимости от цитирований в основном тексте публикации, в котором описываются используемые методы, данные и полученные результаты.

6. Временные характеристики цитирующих и цитируемых публикаций, которые позволяют группировать и упорядочивать перечисленные выше данные по времени, а также исследовать временной характер их изменений.

Как уже отмечалось, использование цитирующим автором содержания цитируемых публикаций в терминах социально-экономических наук может быть определено как процесс научного потребления цитирующего автора содержания цитируемых публикаций. Следовательно, перечисленные выше шесть типов данных представляют собой разнокачественные характеристики такого научного потребления.

Контексты цитирований также имеют атрибуты, которые позволяют различным образом группировать их, создавая массивы данных под определенные аналитические задачи. Подобные группировки могут выполняться как на основе контекстов внутри одной публикации, так и из контекстов, принадлежащих разным публикациям.

Рассмотрим контексты цитирований из разных публикаций, связанные сетью цитирований ученых. Пусть в центре такой сети цитирований находится некий заданный ученый со своими публикациями. В данном случае сеть цитирований включает публикации заданного ученого, а также его ближайшее окружение, состоящее из публикаций, цитируемых данным ученым, и публикаций, цитирующих данного ученого.

Контексты цитирований из данного множества публикаций, очевидно, можно группировать следующим образом.

А. Контексты из публикаций, принадлежащих заданному ученому. Содержание данного массива контекстов представляет ученого-автора как "потребителя" существующих результатов исследований, включая свои собственные результаты из предыдущих публикаций (самоцитирование). Этот массив позволяет анализировать научное потребление заданного ученого.

Б. Контексты из цитируемых публикаций, которые заданный ученый цитирует в своих публикациях. Данное множество контекстов характеризует научное потребление группы цитируемых авторов, находящихся в ближайшем окружении заданного автора и удаленных в сети цитирования по отношению к нему на один уровень. Этот массив данных позволяет анализировать научное потребление группы авторов, которые являются "поставщиками" по отношению к заданному ученому. Эти данные вместе с данными о научном потреблении заданного ученого характеризуют цепочку "ученый — поставщики".

В. Контексты из цитирующих публикаций, в которых среди прочего цитируются публикации заданного ученого. Этот массив контекстов характеризует спрос группы авторов цитирующих, в том числе результаты заданного ученого, как "потребителей" по отношению к заданному ученому. Вместе с предыдущими данными это образует набор характеристик научного потребления в цепочке "потребители — ученый — поставщики".

В результате подобных группировок контекстов могут быть получены различные обобщенные характеристики, которые дают представление о научном потреблении не только заданного ученого, но и его ближайшего окружения в сети цитирований.

Кроме группировок типов А—В, некоторые из типов данных 1—6 могут дополнительно группироваться на основе совпадения или семантической близости значений.

Например, могут группироваться контексты, в которых один и тот же источник/автор источника цитируется (тип 1-2) или со-цитируется (тип 3). Таким образом, можно получить сводные данные о характеристиках потребления определенного источника или автора источника при различных сценариях выборки данных.

Аналогично, контексты могут группироваться, если в них встречаются одинаковые содержательные признаки, такие как термины, ключевые слова и т. п. (тип 4).

Перечисленные выше правила группировки контекстов цитирований и созданных на их основе типов данных позволяют генерировать достаточно большое число содержательно осмысленных характеристик научного потребления. Например, на основе контекстов цитирований, которые сгруппированы по типу А,

можно создавать разнообразные характеристики научного потребления заданного ученого:

- определить наиболее часто цитируемые источники и их авторов (тип 1-2):
  - для этих источников/авторов определить их со-цитирования (тип 3);
  - а также какие семантические признаки характерны для их цитирования, например, темы, ключевые слова, профессиональные термины и т. п. (тип 4);
  - а также как их цитирования распределены по телу публикаций (тип 5);
  - а также как список цитируемых источников и их авторов меняется в зависимости от времени, т. е. от даты публикации статей заданного ученого (тип 6).

Аналогичным образом можно создавать характеристики научного потребления для типов группировки Б и В, для других типов данных, а также в виде вложенных структур данных.

Различные варианты данных, которые таким образом могут быть сгенерированы, в общем случае характеризуют структуру и содержание научного потребления по отношению к определенным ученым или группам ученых.

Информация такого рода является ценным дополнением к традиционным индексам цитирований, так как позволяет получать более полную картину сети связей цитирований с расширенным набором характеристик, в центре которой находится заданный автор.

Исходные данные

Полные тексты научных публикаций в формате PDF, а также их метаданные собираются в проекте Сиртек из научных информационных систем RePEc и Соционет, а также из репозиториев других организаций.

Извлечение из полных текстов научных публикаций в PDF расширенного набора данных о цитированиях включает применение следующих методов: а) конвертация бинарных PDF-файлов в текст в формате JSON, позволяющая получить расширенный список атрибутов для извлекаемых данных; б) применение набора эвристических правил для распознавания в JSON-версии текстов публикаций сносок на источники, включая числовые и символьные виды сносок; в) связывание извлеченных данных о цитированиях с исходными публикациями и их метаданными, а также их представление в виде XML-файлов для дальнейшего использования. Подробные данные об этом приведены в работе [5].

Для выборки из множества обработанных таким образом в Сиртек публикаций данных о публикациях заданных ученых была использована информация о связях персонального профиля ученого с его публикациями, которая есть в RePEc и в Соционет [19]. Этот способ является предпочтительным, так как в данном случае, как правило, множество публикаций автора сформировано и проверено самим автором или его представителем.

Подготовка исходных данных, включая создание программного механизма для их регулярного обновления, выполнена участниками проекта Сиртек Т. Крихелем и В. Ляпуновым.

Для проведения необходимых экспериментов были сгруппированы публикации для 12 ученых, в разной степени отличающихся областями исследований. Полные данные для этих ученых доступны на рабочей площадке проекта[17] в разделе "Показатели научной кооперации авторов".

Ниже все результаты приводятся для одного из этих ученых, которому присвоен код "АК". Полный набор данных для АК приведен в дополнительных материалах к этой статье[2] [20].

Список из 41 публикации ученого АК, в которых распознаны данные о цитированиях (список литературы и ссылки на источники списка литературы) доступен по ссылке, приведенной в п. 1 [20]. Для каждой публикации в этом списке приводится ссылка на ее страницу в системе Соционет, а также дано число распознанных источников в ее списке литературы и число найденных в ее тексте ссылок на эти источники.

Для данного подмножества публикаций заданного автора были извлечены и сгруппированы 1703 контекстов цитирований (ссылка приведена в п. 2 [20]). В этом списке контексты сгруппированы блоками для каждой публикации заданного автора. В начале каждого блока приводится ссылка на страницу соответствующей публикации в Соционет, а также общее число выделенных из публикации контекстов цитирований. Для каждого отдельного контекста цитирования приводятся следующие данные:

- его координата в соответствующем полном тексте (число в поле `start`, означающее номер первого символа ссылки на цитируемый источник от начала полного текста);
- левая часть контекста (поле `Prefix`);
- вид ссылки на цитируемый источник (поле `Exact`), включая гиперссылку для ее просмотра в полном тексте соответствующей публикации (для перехода на страницу со ссылкой нужно кликнуть внизу на рамке справа метку 1 или 2, ссылка будет выделена желтым на соответствующей странице);
- правая часть контекста (поле `Suffix`).

Список контекстов для публикаций АК включает 3 публикации из 41, которые имеют данные о цитированиях (список литературы), но ссылки на источники списка литературы в них не распознаны.

Небольшое количество контекстов (меньше 1 %) имеют группы спецсимволов, которые не являются буквами или цифрами. Эти спецсимволы являются проявлением известной проблемы с кодировкой для PDF-файлов, в частности для формул.

Кроме этого, небольшое количество контекстов (также меньше 1 %) выделены неправильно, так как алгоритм распознал часть формул как ссылки на источники.

Поскольку количество контекстов с ошибками в сумме около 1 % от их общего числа, считаем, что данный массив может быть использован для дальнейшего анализа.

[1] http://cirtec.ranepa.ru/
[2] https://clck.ru/SjLSh

Сгруппированные контексты цитирований были лемматизированы[1] с удалением стоп-слов[2]. Лемматизация выполнена с помощью процедуры UDPipe[3]. Ссылка на подготовленный таким образом массив контекстов цитирований приведен в п. 3 [20].

Контексты цитирований в этом массиве являются строками, в начало которых добавлены четыре идентификатора:

- признак принадлежности контекста к типу группировки (`linked _ papers` — группировка типа А; `cited _ papers` — группировка типа Б; `citing _ papers` — группировка типа В);
- код публикации, к которой принадлежит данный контекст (например, `repec:gai:wpaper:0096`);
- порядковый номер в списке литературы цитируемого в данном контексте источника;
- координата первого символа ссылки на источник (аналогично числу в поле `start`).

Эти идентификаторы позволяют сформировать уникальный код для каждого контекста цитирований.

Структура научного потребления на основе топиков

Определение структуры научного потребления заданного автора означает тематическое структурирование содержания контекстов цитирований. Для этих целей используется описанный выше лемматизированный массив контекстов цитирований из публикаций заданного автора, что соответствует созданию характеристик типа А-4 (тип группировки А, а тип данных 4), описанных выше.

Задача тематического структурирования (выявление тематических признаков) в данном случае рассматривается как выделение из контекстов цитирований слов и словосочетаний, из которых может быть построена необходимая тематическая структура. Для решения этой задачи в рамках рассматриваемого исследования используется метод тематического моделирования, реализованный в виде алгоритма LDA (Латентное размещение Дирихле)[4].

Метод LDA позволяет выявить в заданном наборе текстов скрытую семантическую структуру, которая выражается в виде набора абстрактных топиков. Применительно к анализу контекстов цитирований такой метод используется: для автоматической классификации тональности цитирований [21]; для автоматического построения онтологии цитирований [22]; для измерения близости между текстами [23] или близости между контекстами цитирований и цитируемыми в них публикациями [24]; для тематического моделирования контекстов цитирований [19].

Для тематического моделирования в проекте Сиртек были использованы модули LDA[5] из библиотеки Gensim. Таким образом, для подготовленного массива контекстов цитирований были построены 20 топиков по пять слов каждый. Разработку соответствующего программного обеспечения[6] выполнил А. Бакаров. Обработка данных и их визуализация выполнена Р. Пузыревым.

В системе Сиртек построенные топики автоматически обновляются при изменениях в данных о цитированиях для заданного автора (при появлении новых публикаций и т. п.). В п. 4 [20] приведены топики, построенные в Сиртек на 10.01.2021. Все описанные ниже результаты получены на основе данного массива топиков.

Массив с данными о топиках состоит из трех разделов, соответствующих типу группировки (`linked _ papers` — группировка типа А; `cited _ papers` — группировка типа Б; `citing _ papers` — группировка типа В). В каждом разделе приводятся:

- в массиве `topics` общие данные о топиках, включая сам топик (пять слов в поле `topic`), число контекстов цитирования для которых вероятность (релевантность) их отнесения к данному топику больше нуля (поле `number`), средняя вероятность (поле `probability _ average`) и стандартное отклонение для средней вероятности (поле `probability _ std`);
- в массиве `contexts` для каждого контекста цитирования приводится его уникальный код, наиболее релевантный топик и коэффициент вероятности для данной пары "топик и контекст цитирования".

Построенные топики представляют собой наборы по пять слов, которые с позиций метода LDA при заданных параметрах наилучшим образом представляют тематику исходного массива контекстов цитирований.

Порядок слов в топиках не имеет значения. Поэтому их отличия друг от друга определяются содержащимися словами и количествами релевантных контекстов цитирований. Топики могут содержать одинаковые слова.

В таблице представлены пять слов из полного набора топиков, построенных для ученого АК, которые наиболее часто повторяются в топиках[7]. Для каждого слова приведены номера топиков (в скобках), в которых это слово встречается, а также подсчитаны суммарные значения значимости слов. Например, слово "страна" входит в состав 12 из 20 топиков, и оно в составе этих топиков релевантно со средней вероятностью не менее 0,5 к 1811 контекстам цитирований (данное число вследствие повторного счета больше общего числа контекстов 1703, так как разные топики с данным словом могут быть релевантны одинаковым контекстам).

Слова из топиков являются элементами тематической структуры научного потребления. Они тематически связаны с другими словами, с которыми они соседствуют в топиках. На рис. 1 (см. третью сторону обложки) в виде хордовой диаграммы (тип диаграммы Chord[8]) представлены связи между словами из построенных 20 топиков на основе их соседства в топиках.

Минимальное число связей соседства одного слова, которое имеет место, если слово присутству-

[1] https://ru.wikipedia.org/wiki/Лемматизация
[2] См. ссылки на словари стоп-слов на https://clck.ru/SjLSh
[3] https://github.com/ufal/udpipe
[4] https://ru.wikipedia.org/wiki/Латентное_размещение_Дирихле
[5] https://radimrehurek.com/gensim/models/ldamodel.html
[6] https://github.com/bakarov/cirtec/blob/master/topic_modeling/scripts/parse_contexts.py
[7] Полный список слов находится в п. 4 [20].
[8] Использована библиотека https://www.amcharts.com/demos/toggleable-chord-diagram/

Пять наиболее часто повторяющихся слов из топиков

Слова из топиков (№ топиков)	Суммарное число контекстов
страна (0, 1, 3, 4, 5, 7, 9, 12, 13, 16, 17, 18)	1811
модель (3, 4, 5, 6, 9, 10, 13, 15, 16, 17)	1384
работа (1, 4, 5, 6, 10, 12, 13, 16, 18, 19)	1341
товар (1, 6, 7, 9, 13, 14, 15)	1029
год (3, 5, 8, 12, 13, 14, 17)	849

ет только в одном топике из пяти слов, равно 4, а максимальное — 80 (при условии, что другие слова в топиках не повторяются). На рис. 1 (см. третью сторону обложки) видно, что слово "страна" имеет связи с 24 словами, но теоретически оно могло бы иметь 48 связей, так как оно входит в 12 топиков. Меньшее число связей объясняется тем обстоятельством, что в этих топиках есть еще и другие слова, которые тоже повторяются.

Хордовое графическое представление позволяет оценить общую "тесноту" связей соседства между словами всех топиков. Общее число слов для 20 топиков по пять слов составляет 100 слов, но за счет повторов число уникальных слов в построенных топиках составляет только 48. Диаграмма на рис. 1 (см. третью сторону обложки) показывает число повторяющихся слов в топиках. Это 14 слов из 48, которые расположены от слова "страна" до слова "качество" по часовой стрелке. Эти слова имеют более четырех связей.

Остальные 34 слова из этих топиков являются уникальными, т. е. встречаются только в одном топике.

Хордовое представление связей на рис. 1 (см. третью сторону обложки) не позволяет отследить принадлежность слов топикам, что необходимо для полноты представлений об отношениях между словами в структуре научного потребления.

Для более наглядного представления слов из топиков в виде структуры научного потребления построена диаграмма типа "деревья слов" (тип диаграммы Wordtree[1]). Данная диаграмма представляет множество слов как набор "деревьев", имеющих определенное число "ветвей".

В нашем случае деревья слов должны быть простроены таким образом, что ветви дерева совпадали с имеющимися топиками. Для этого все слова из всех топиков были представлены в виде списка, упорядоченного по убыванию их значимости, которая понимается как число контекстов цитирований, к которым данное слово релевантно в составе соответствующих топиков. Пример этого списка приведен в таблице. Полный список упорядоченных слов приведен в п. 4 [20].

На пером шаге построения тематической структуры из этого списка слов выбираются корневые слова, т. е. слова, которые не имеют между собой связей соседства. Для этого перебором каждого слова в списке, начиная с первого, определяются слова, которые не имеют общих топиков ни с одним из слов,

[1] Использована библиотека https://developers.google.com/chart/interactive/docs/gallery/wordtree

Рис. 2. Фрагмент тематической структуры потребления в виде "дерева слов"

расположенных выше заданного в списке слов. Данные корневые слова являются началом (корнем) для построения дерева слов. Число выделенных таким образом корневых слов соответствует числу деревьев в создаваемой структуре научного потребления.

На втором шаге для каждого из отобранных на первом шаге корневых слов отбираются слова, которые имеют с корневым словом общие топики. Затем аналогичное действие повторяется для слов, отобранных на втором шаге и т. д. Таким образом, формируются ветви деревьев.

В результате создается множество деревьев слов, где число деревьев равно числу корневых слов. В каждом дереве имеется различное число ветвей, но каждая ветвь соответствует одному из топиков и содержит ровно пять слов.

На рис. 2 показано самое большое дерево из построенных на основе слов (представленных на рис. 1, см. третью сторону обложки) из топиков для ученого АК.

В соответствии с алгоритмом построения значимость слов в ветвях дерева уменьшается слева направо. Размер шрифта слова на диаграмме примерно соответствует значимости слова. Числа в скобках на концах ветвей дерева являются суммарной значимостью соответствующей ветви (топика). Для всего дерева подсчитывается его значимость, как сумма релевантных контекстов цитирований входящих в него ветвей (топиков). Эти значения отражают степень связанности дерева с полным массивом контекстов цитирований.

Представление "дерево слов" позволяет выявлять слова, которые являются общими для разных топиков (места разветвления дерева), а также удобно для анализа различий в значимости слов (близость или удаленность от корня). Подобная диаграмма показывает, какие топики являются тематически связанными (группировка топиков в единое дерево), или, наоборот, являются тематически обособленными.

Применение описанного алгоритма к топикам заданного ученого позволило сформировать шесть "деревьев слов", которые приведены в п. 6 [20]. Из них два дерева состоят из одной ветви, три дерева содержит по две ветви и еще одно, приведенное на рис. 2, содержит 12 ветвей. Значимостью дерева, приведенного на рис. 2, более чем в 2 раза больше, чем суммарно для пяти остальных деревьев.

Наличие среди шести деревьев одного доминирующего дерева с 12 ветвями можно интерпретировать

как достаточно высокую тематическую однородность научной структуры потребления ученого АК.

Вместе с тем в диаграмме "дерево слов" одинаковые слова могут входить в несколько деревьев. Это не позволяет оценить полную значимость слов для структуры потребления, так как она может быть распределена между разными деревьями.

Для устранения этого недостатка была построена потоковая диаграмма (тип диаграммы Sankey[1]), в которой каждое слово из топиков встречается только один раз (занимает только один узел). В данной диаграмме узлами являются уникальные слова, через которые проходят связи (потоки) с другими словами, образуя топики. Такая диаграмма позволяет увидеть общую значимость каждого слова и отследить число связей, проходящих через него. Чем больше связей и релевантных контекстов ассоциированы со словом, тем выше его значимость в рамках тематической структуры научного потребления ученого.

Алгоритм построения потоковой диаграммы использует тот же упорядоченный список слов из топиков, который был создан для диаграммы "дерево слов". Алгоритм работает следующим образом.

Шаг 1. На этом шаге, аналогично, как для "дерева слов", выделяются корневые слова. Найденные слова фиксируются как массив корневых слов нулевого уровня.

Шаг 2. Исключаем корневые слова нулевого уровня из упорядоченного списка слов и повторяем в нем поиск корневых слов. Для каждого выявленного корневого слова проверяем, есть ли хотя бы одно слово из группы корневых нулевого уровня (в общем случае из предыдущей группы), с которым у этого слова есть общий топик. Таким образом проверяем, связано ли каждое слово из текущей группы корневых слов хотя бы с одним словом из предыдущей группы. Связь здесь понимается как наличие хотя бы одного топика, где оба эти слова присутствуют. Если для текущего корневого слова не найдено ни одного слова из предыдущей группы корневых слов с общим топиком, то текущее корневое слово переносится в предыдущую группу корневых слов, и для него повторяется аналогичная проверка (теоретически слово, таким образом, может оказаться в нулевой группе). Фиксируем найденные слова, за исключением перенесенных в предыдущую группу, как корневые слова уровня 1. Фиксируем новый состав нулевой группы, если в нее были перенесены слова из группы 1. В общем случае фиксируем состав всех обновленных групп корневых слов.

Шаг 3. Повторяем процедуру шага 2 еще 2 раза для формирования корневых слов уровней 2 и 3.

В результате получаем четыре группы корневых слов уровней 0-3, а также все остальные слова из упорядоченного списка слов, которые не попали в корневые и образовали пятую группу слов.

Представленный алгоритм гарантирует, что корневые слова, которые не имеют связи через общий топик ни с одним из корневых слов предыдущей группы, постепенно переходят в группу, в которой подобная связь обнаруживается, или в конце концов они оказываются в нулевой группе. Число построенных групп слов соответствует числу слов в топиках. Каждое слово в любой группе имеет одну или более связей со словами из рядом стоящих групп. Связанные таким образом слова соответствуют определенным топикам.

На рис. 3 (см. третью сторону обложки) в виде потоковой диаграммы представлена тематическая структура из слов, связи между которыми созданы по описанному выше алгоритму. Набор слов на рис. 3 аналогичен набору на рис. 1 (см. третью сторону обложки), и эти же слова использованы для построения дерева на рис. 2.

Потоковая диаграмма, пример которой приведен на рис. 3, объединяет достоинства хордовой диаграммы (рис. 1, см. третью сторону обложки) и диаграммы "дерево слов" (см. рис. 2). Как и хордовая, она показывает, какие слова являются наиболее значимыми. На это указывают размер полоски узла соответствующего слова в диаграмме, и число ветвей, проходящих через него.

Вместе с тем потоковая диаграмма как и "дерево слов" показывает, как слова связаны между собой в виде топиков. Топиками здесь являются группы слов, связанные между собой по горизонтали. Электронная версия диаграммы, ссылка на которую приведена в п. 7 [20], позволяет наведением курсора на фрагмент любой связи увидеть все связанные слова, принадлежащие определенному топику. На рис. 3 (см. третью сторону обложки) приведен пример такого выделения связанных слов, начинающихся со слов "страна" — "модель" — "работа" и т. д.

Как и "дерево слов", потоковая диаграмма позволяет увидеть тематическую (не)однородность научной структуры потребления. На это указывает число корневых слов нулевого уровня: чем их больше, тем больше в структуре потребления тематически самостоятельных компонентов и, следовательно, тем выше ее тематическая разнородность. Дополнительно к представлению "дерево слов" потоковая диаграмма позволяет увидеть возможные пересечения тематически разнородных структур потребления через общие слова.

Потоковая диаграмма в сравнении с "деревом слов" дает на том же наборе топиков более компактное представление научной структуры потребления. В этом смысле она является более наглядной и информативной.

Однако у потоковой диаграммы есть особенность построения: каждое слово может встречаться в диаграмме только один раз и корневые слова одного уровня не могут иметь связей между собой. Также для нашего случая число уровней корневых слов должно совпадать с числом слов в топиках, т. е. быть равно 5. Это означает, что если среди слов из топиков есть пять или более слов, которые встречаются в пяти или более топиках, то алгоритм не может разместить корневые слова по пяти уровням так, чтобы не нарушалось одно из приведенных выше правил. Таким образом, в подобных случаях нельзя построить потоковую диаграмму, "потоки" (линии связей) в которых не разветвляются и точно соответствуют топикам.

Для разрешения подобных ситуаций при построении потоковой диаграммы типа Sankey допускается разветвление связей. В п. 7 [20] приведен пример диаграммы с разветвлением связей.

[1] Использована библиотека https://www.amcharts.com/demos/traceable-sankey-diagram/

Обсуждение результатов

Представленный выше подход для построения тематических структур научного потребления ученого имеет отмеченные далее возможности для развития и улучшений уже полученных результатов.

- Более аккуратное формирование содержания контекстов цитирований путем исключения из них текста, не имеющего отношения к цитируемым источникам. В проведенных экспериментах в контексты цитирований включались текстовые фрагменты слева и справа от ссылки на источник. В Сиртек в эти фрагменты помещается не менее 250 символов, расширенных до полного предложения. В литературе описаны способы, как формировать содержание контекстов цитирований более точно [25].
- Разметка в тематических структурах типов слов. Например, выделение в них профессиональных терминов, лексических клише и слов-маркеров. В полученных топиках из 47 слов два слова являются глаголами, пять слов — прилагательными, а остальные — существительными (см. размеченный список слов из топиков в п. 4 в [20]). Представляется, что профессиональные термины в большей части состоят из существительных и прилагательных. В литературе описано, что глаголы часто могут служить словами-маркерами, указывающими, например, на функции и тональность цитирования [26]. Размеченная таким образом тематическая структура потребления ученого может не только характеризовать разнообразие специфических для ученого профессиональных терминов и их тематических комбинаций, но и показывать внутри тематической структуры также структуру функций/тональности цитирований.
- Добавление в тематическую структуру потребления характеристик цитируемых публикаций и их авторов для определения их вкладов и значимости для заданного ученого. Наличие в данных о цитированиях сквозных идентификаторов позволяет привязать к тематически связанным группам слов (топикам) названия публикаций и/или авторов, которые данный ученый процитировал в контекстах, релевантных к соответствующим топикам. В этом случае тематическая структура потребления получает дополнительную размерность, состоящую из множеств названий публикаций и их авторов, привязанных к определенным темам. Такая двухмерная структура данных позволяет анализировать, в каком тематическом контексте ученый цитирует различных авторов, и выполнять другой наукометрический анализ.
- Использование других методов для определения тематики контекстов цитирований. Тематическое моделирование представляет тематику массива контекстов цитирований сравнительно небольшим набором слов. Если контекстов очень много и их тематика достаточно разнообразна, то данный метод может терять важную информацию. В таких случаях целесообразно попробовать другие методы. Например, использование метода *n*-грамм[1] для выделения из лемматизированных контекстов часто повторяющихся групп слов и построения из них тематической структуры потребления.

[1] https://ru.wikipedia.org/wiki/N-грамма

Заключение

Использование данных, извлеченных из контекстов цитирований, для расширения наукометрической базы и, в том числе, для создания характеристик научного потребления ученых является очевидным современным направлением развития наукометрических исследований. Это направление стало возможным в связи с появлением в последние годы больших массивов соответствующих данных. Описанные выше эксперименты по созданию тематических структур научного потребления ученых являются примерами использования этих новых возможностей. Они способствуют лучшему пониманию места и роли подобных новых наукометрических инструментальных средств, а также их потенциальных практических приложений.

Очевидно, что изложенный подход в первую очередь является развитием анализа научных сетей цитирований. Для сетей цитирований, узлами которых являются авторы публикаций, построение тематических структур научного потребления ученых позволяет существенно расширить и обогатить традиционные представления о характеристиках как узлов, так и дуг сетей цитирований.

В результате такого подхода традиционное представление ученого в сети цитирований расширяется за счет представления каждой дуги графа цитирований в виде пучков тематических связей между учеными. Это позволяет более эффективно, чем раньше, решать важные для научного сообщества задачи, а именно:

- позиционирование ученого в сложившейся системе научных тематических направлений, а в более широком плане — в системе научного разделения труда и кооперации, включая более точное определение влияния ученого на процессы развития научного знания;
- определение тематической близости между учеными с точки зрения их потенциальной научной кооперации, включая определение их возможных ролей друг для друга в виде "поставщика" или "потребителя" результатов исследований;
- анализ отношений заданного ученого в цепочке его научной кооперации, включая как "поставщиков" (тех, кого он цитирует), так и "потребителей" его научной продукции (тех, кто цитируют его публикации);
- создание комплексного цифрового образа ученого как участника научной кооперации, который позволит развивать подходы к построению нового механизма глобальной научной кооперации, лишенного ограничений и недостатков традиционных публикаций.

Цифровой образ ученого как участника глобальной научной кооперации представляет интерес как регулярно обновляемый статистический портрет места автора в системе научной кооперации, основанной на публикациях. Это создает основу для компьютерной модернизации механизмов научной кооперации за счет выявления потенциальных участников научной кооперации. Данное направление развития механизмов совместной деятельности ученых обещает существенное повышение эффективности научной системы в целом.

Кроме этого, цифровой образ может быть использован для модернизации системы оценки научной результативности ученых, так как в сравнении

с традиционной наукометрией он дает представление о деятельности ученого в цепочках научной кооперации, т. е. показывает место ученого в глобальной системе научного разделения труда.

Все вместе это позволяет решать задачу повышения эффективности системы научной кооперации, так как информация о процессах кооперации может быть объектом для алгоритмов поиска, оптимизации, рекомендательных систем, связывания ученых и т. п.

Часть данного исследования, связанная с развитием методов анализа данных о цитированиях для суперкомпьютерного моделирования научного сообщества в виде взаимодействий между агентами и окружающей средой, финансируется за счет гранта РНФ (проект № 19 -18-00240).

Список литературы

1. **Bertin M., Atanassova I.** A study of lexical distribution in citation contexts through the IMRaD standard // In Proceedings of the First Workshop on Bibliometric-enhanced Information Retrieval co-located with 36th European Conference on Information Retrieval (ECIR 2014). — 2014. — Vol. 1143. — P. 5—12.

2. **Bertin M., Atanassova I.** Factorial Correspondence Analysis Applied to Citation Contexts // BIR@ECIR 2015. — 2015. — P. 22—29.

3. **Bertin M., Atanassova I.** InTeReC: In-text Reference Corpus for Applying Natural Language Processing to Bibliometrics // In Proc. of the Seventh Workshop on Bibliometric-enhanced Information Retrieval (BIR). Grenoble, France, CEURWS.org 2018. — 2018. — P. 54—62.

4. **Parinov S.** Extraction and visualisation of citation relationships and its attributes for papers in PDF // International Journal of Metadata, Semantics and Ontologies (IJMSO). — 2017. — Vol. 12, No. 4. — P. 195—203. DOI: 10.1504/IJMSO.2017.093626.

5. **Kogalovsky M., Krichel T., Lyapunov V.** et al. Open Citation Content Data //Metadata and Semantic Research. MTSR 2018. Communications in Computer and Information Science / Eds by E. Garoufallou, F. Sartori, R. Siatri, M. Zervas. Springer, Cham. — 2019. — Vol. 846. — P. 355—364.

6. **Бредихин С. В., Ляпунов В. М., Щербакова Н. Г.** Структура сети цитирования научных журналов // Проблемы информатики. — 2017. — № 2. — С. 38—52.

7. **Boyack K. W., van Eck N. J., Colavizza G., Waltman L.** Characterizing in-text citations in scientific articles: A large-scale analysis // Journal of Informetrics — 2018. — Vol. 12, No. 1. — P. 59—73.

8. **Hernandez-Alvarez M., Soriano J. M. G., Martínez-Barco P.** Citation function, polarity and influence classification // Natural Language Engineering. — 2017. — Vol. 23, No. 4. — P. 561—588.

9. **Kilicoglu H., Peng Z., Tafreshi S., Tran T., Rosemblat G., Schneider J.** Confirm or Refute?: A Comparative Study on Citation Sentiment Classification in Clinical Research Publications // Journal of Biomedical Informatics. — 2019. — Vol. 91. — Article 103123.

10. **Tahamtan I., Bornmann L.** What Do Citation Counts Measure? An Updated Review of Studies on Citations in Scientific Documents Published between 2006 and 2018 // Scientometrics. — 2019. — Vol. 121, No. 3. — P. 1635—684. DOI: 10.1007/s11192-019-03243-4.

11. **Iqbal S., Hassan S. U., Aljohani N. R.** et al. A Decade of In-text Citation Analysis based on Natural Language Processing and Machine Learning Techniques: An overview of empirical studies. arXiv preprint 2020, arXiv:2008.13020, https://arxiv.org/pdf/2008.13020.pdf

12. **Kongthon A., Haruechaiyasak C., Thaiprayoon S.** Enhancing the Literature Review Using Author-Topic Profiling // In International Conference on Asian Digital Libraries. Berlin, Heidelberg, Springer. — 2008. — P. 335—338.

13. **Lu K., Wolfram D.** Measuring author research relatedness: A comparison of wordbased, topic-based, and author cocitation approaches // Journal of the American Society for Information Science and Technology. — 2012. — Vol. 63, No. 10. — P. 1973—1986.

14. **Jensen S., Liu X., Yu Y., Milojevic S.** Generation of topic evolution trees from heterogeneous bibliographic networks// Journal of Informetrics. — 2016. — Vol. 10, No. 2. — P. 606—621.

15. **Leydesdorff L., Nerghes A.** Co-word maps and topic modeling: A comparison using small and medium-sized corpora (N < 1,000) // Journal of the Association for Information Science and Technology. — 2017. — Vol. 68, No. 4. — P. 1024—1035.

16. **Hassan S. U., Haddawy P.** Analyzing knowledge flows of scientific literature through semantic links: a case study in the field of energy // Scientometrics. — 2015. — Vol. 103, No. 1. — P. 33—46.

17. **Kim H. J., An J., Jeong Y. K., Song M.** Exploring the leading authors and journals in major topics by citation sentences and topic modeling // In Proceedings of the Joint Workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL) 2016. — 2016. — P. 42—50.

18. **Nallapati R. M., Ahmed A., Xing E. P., Cohen W. W.** Joint latent topic models for text and citations // In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM. — 2008. — P. 542—550.

19. **Parinov S., Antonova V.** Citation content/context data as a source for research cooperation analysis // International Journal of Metadata, Semantics and Ontologies. — 2020. — Vol. 14, No. 2. — P. 149—157.

20. **Дополнительные** материалы к статье. URL: https://clck.ru/SjLSh

21. **Yousif A., Niu Z., Nyamawe A. S., Hu Y.** Improving citation sentiment and purpose classification using hybrid deep neural network model// In International Conference on Advanced Intelligent Systems and Informatics. Springer, Cham. — 2018. — P. 327—336.

22. **Bakhti K., Niu Z., Yousif A., Nyamawe A. S.** Citation Function Classification Based on Ontologies and Convolutional Neural Networks // In International Workshop on Learning Technology for Education in Cloud. Springer, Cham. — 2018. — P. 105—115.

23. **Knoth P., Herrmannova D.** Towards semantometrics: A new semantic similarity based measure for assessing a research publication's contribution // D-Lib Magazine. — 2014. — Vol. 20, No. 11/12. — Article 8.

24. **Parinov S., Bakarov A., Vodolazcky D.** Layout logical labelling and finding the semantic relationships between citing and cited paper content // International Journal of Metadata, Semantics and Ontologies. — 2020. — Vol. 14, No. 1. — P. 54—62.

25. **Nielsen B. L., Skau S. L., Meier F., Larsen B.** Optimal Citation Context Window Sizes for Biomedical Retrieval // In BIR@ ECIR 2019. — 2019. — P. 51—63.

26. **Bertin M., Atanassova I., Sugimoto C. R.** et al. The linguistic patterns and rhetorical structure of citation context: an approach using n-grams // Scientometrics. — 2016. — Vol. 109, No. 3. — P. 1417—1434. DOI: 10.1007/s11192-016-2134-8.

Topic Modeling of the Research Papers' Citation Contexts: a Structure of an Author's Research Consumption

S. I. Parinov, sparinov@gmail.com, Central Economics and Mathematics Institute of RAS (CEMI RAS), Russian Presidential Academy of National Economy and Public Administration (RANEPA), Moscow, 117418, Russian Federation

Corresponding author:

Parinov Sergey I., D. Sc., Chief Researcher, Central Economics and Mathematics Institute of RAS (CEMI RAS), Russian Presidential Academy of National Economy and Public Administration (RANEPA), Moscow, 117418, Russian Federation
E-mail: sparinov@gmail.com

Received on January 11, 2021
Accepted on February 01, 2021

Citation contexts from research papers, as a rule, contain information about the reasons and the character of using the cited research outputs. By extracting this information from the citation contexts, one can create different data sets for scientometric studies. The paper systematizes general possibilities of using data from the citation contexts for the development of the author-citation network analysis. As one of applications, the paper presents an approach to constructing the thematic structure of a research consumption based on topic modelling of the citation contexts from researcher's papers. The thematic structure features built in the forms of a "word tree" and a flowchart are discussed. Possible directions of development of this approach are considered. The proposed thematic structure of the research consumption is a promising new data source for both scientometric studies and creation of new research services.

Keywords: *Cirtec project, citation contexts, topic modeling, structure of research consumption*

For citation:

Parinov S. I. Topic Modeling of the Research Papers' Citation Contexts: a Structure of an Author's Research Consumption, *Programmnaya Ingeneria,* 2021, vol. 12, no. 3, pp. 140—149

DOI: 10.17587/prin.12.140-149

References

1. **Bertin M., Atanassova I.** A study of lexical distribution in citation contexts through the IMRaD standard, *In Proceedings of the First Workshop on Bibliometric-enhanced Information Retrieval co-located with 36th European Conference on Information Retrieval (ECIR 2014),* 2014, vol. 1143, pp. 5—12.

2. **Bertin M., Atanassova I.** Factorial Correspondence Analysis Applied to Citation Contexts, *In BIR@ ECIR 2015*, 2015, pp. 22—29.

3. **Bertin M., Atanassova I.** InTeReC: In-text Reference Corpus for Applying Natural Language Processing to Bibliometrics, *In Proc. of the Seventh Workshop on Bibliometric-enhanced Information Retrieval (BIR),* Grenoble, France, CEURWS.org 2018, 2018, pp. 54—62.

4. **Parinov S**. Extraction and visualisation of citation relationships and its attributes for papers in PDF, *International Journal of Metadata, Semantics and Ontologies (IJMSO),* 2017, vol. 12, no. 4, pp. 195—203. DOI: 10.1504/IJMSO.2017.093626.

5. **Kogalovsky M., Krichel T., Lyapunov V., Medvedeva O., Parinov S., Sergeeva V.** Open Citation Content Data, *Metadata and Semantic Research. MTSR 2018. Communications in Computer and Information Science* / Eds by E. Garoufallou, F. Sartori, R. Siatri, M. Zervas. Springer, Cham., 2019, vol 846, pp. 355—364.

6. **Bredikhin S., Lyapunov V., Shcherbakova N.** The structure of the citation network of scientific journals, *Problemy informatiki,* 2017, no. 2, pp. 38—52 (in Russian).

7. **Boyack K. W., van Eck N. J., Colavizza G., Waltman L.** Characterizing in-text citations in scientific articles: A large-scale analysis, *Journal of Informetrics,* 2018, vol. 12, no.1, pp. 59—73.

8. **Hernandez-Alvarez M., Soriano J. M. G., Martínez-Barco P.** Citation function, polarity and influence classification, *Natural Language Engineering,* 2017, vol. 23, no. 4, pp. 561—588.

9. **Kilicoglu H., Peng Z., Tafreshi S., Tran T., Rosemblat G., Schneider J.** Confirm or Refute?: A Comparative Study on Citation Sentiment Classification in Clinical Research Publications, *Journal of Biomedical Informatics,* 2019, vol. 91, article 103123.

10. **Tahamtan I., Bornmann L.** What Do Citation Counts Measure? An Updated Review of Studies on Citations in Scientific Documents Published between 2006 and 2018, *Scientometrics,* 2019, vol. 121, no. 3, pp. 1635—1684. DOI: 10.1007/s11192-019-03243-4.

11. **Iqbal S., Hassan S. U., Aljohani N. R., Alelyani S., Nawaz R., Bornmann L.** A Decade of In-text Citation Analysis based on Natural Language Processing and Machine Learning Techniques: An overview of empirical studies. arXiv preprint 2020, arXiv:2008.13020, https://arxiv.org/pdf/2008.13020.pdf

12. **Kongthon A., Haruechaiyasak C., Thaiprayoon S.** Enhancing the Literature Review Using Author-Topic Profiling, *International Conference on Asian Digital Libraries*, Springer, Berlin, Heidelberg, 2008, pp. 335—338.

13. **Lu K., Wolfram D.** Measuring author research relatedness: A comparison of wordbased, topic-based, and author cocitation approaches, *Journal of the American Society for Information Science and Technology*, 2012, vol. 63, no. 10, pp. 1973—1986.

14. **Jensen S., Liu X., Yu Y., Milojevic S.** Generation of topic evolution trees from heterogeneous bibliographic networks, *Journal of Informetrics,* 2016, vol. 10, no. 2, pp. 606—621.

15. **Leydesdorff L., Nerghes A.** Co-word maps and topic modeling: A comparison using small and medium-sized corpora (N < 1,000), *Journal of the Association for Information Science and Technology*, 2017, vol. 68, no. 4, pp. 1024—1035.

16. **Hassan S. U., Haddawy P.** Analyzing knowledge flows of scientific literature through semantic links: a case study in the field of energy, *Scientometrics,* 2015, vol. 103, no. 1, pp. 33—46.

17. **Kim H. J., An J., Jeong Y. K., Song M.** Exploring the leading authors and journals in major topics by citation sentences and topic modeling, *Proceedings of the Joint Workshop on Bibliometric-enhanced Information Retrieval and Natural Language Processing for Digital Libraries (BIRNDL) 2016*, 2016, pp. 42—50.

18. **Nallapati R. M., Ahmed A., Xing E. P., Cohen W. W.** Joint latent topic models for text and citations, *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2008, pp. 542—550.

19. **Parinov S., Antonova V.** Citation content/context data as a source for research cooperation analysis, *International Journal of Metadata, Semantics and Ontologies,* 2020, vol. 14, no. 2, pp. 149—157.

20. **Dopolnitel'nye** materialy k stat'e, available at: https://clck.ru/SjLSh

21. **Yousif A., Niu Z., Nyamawe A. S., Hu Y.** Improving citation sentiment and purpose classification using hybrid deep neural network model, *International Conference on Advanced Intelligent Systems and Informatics*, Springer, Cham., 2018, pp. 327—336.

22. **Bakhti K., Niu Z., Yousif A., Nyamawe A. S.** Citation Function Classification Based on Ontologies and Convolutional Neural Networks, *International Workshop on Learning Technology for Education in Cloud*, Springer, Cham., 2018, pp. 105—115.

23. **Knoth P., Herrmannova D.** Towards semantometrics: A new semantic similarity based measure for assessing a research publication's contribution, *D-Lib Magazine,* 2014, vol. 20, no. 11/12, article 8.

24. **Parinov S., Bakarov A., Vodolazcky D.** Layout logical labelling and finding the semantic relationships between citing and cited paper content. *International Journal of Metadata, Semantics and Ontologies,* 2020, vol. 14, no. 1, pp. 54—62.

25. **Nielsen B. L., Skau S. L., Meier F., Larsen B.** Optimal Citation Context Window Sizes for Biomedical Retrieval, *BIR@ ECIR 2019*, 2019, pp. 51—63.

26. **Bertin M., Atanassova I., Sugimoto C. R.** et al. The linguistic patterns and rhetorical structure of citation context: an approach using n-grams, *Scientometrics,* 2016, vol. 109, no. 3, pp. 1417—1434. DOI: 10.1007/s11192-016-2134-8.

УДК 519.7; 004.056 DOI: 10.17587/prin.12.150-156

А. В. Галатенко, канд. физ.-мат. наук, вед. науч. сотр., Высшая школа экономики, Москва, agalatenko@hse.ru, **В. А. Кузовихина**, учитель, pletnyova_va@mail.ru, Новая школа, Москва

Об одной модели безопасного функционирования компьютерных систем*

Введена формальная модель безопасности компьютерных систем, позволяющая отражать оперативность реакции на нарушения безопасности. Система задается автоматом, состояния которого разбиты на два класса — безопасные и небезопасные, и натуральным параметром k. Входная последовательность считается безопасной, только если длина непрерывной подпоследовательности небезопасных состояний не превосходит k. Исследованы свойства модели при вариации параметра k, а также сложность задачи восстановления параметров системы с помощью кратных условных экспериментов.

Ключевые слова: *формальные модели безопасности, конечные автоматы, регулярные языки, кратные условные эксперименты*

Введение

В настоящее время известно значительное число формальных моделей безопасности компьютерных систем, как классических (Харрисона—Руззо—Ульмана [1], Белла—Лападула [2], невлияния [3, 4]), так и возникших в последние годы [5, 6]. Основным средством моделирования, как правило, служат либо автоматы (в явном или неявном виде), либо нагруженные графы. В работе [7] показано, что автоматная модель из работы [4] может быть естественным образом вложена в графовую модель [6]. Заметим, что диаграмма Мура автомата является нагруженным графом, поэтому автоматные модели можно считать одновременно графовыми за счет переформулировки условий на поведение в терминах путей по диаграмме Мура.

В большинстве современных систем принципиально возможны переходы из безопасных состояний в небезопасные. Причинами таких переходов могут служить, например, ошибки в программном обеспечении или конфигурировании, злоумышленные действия суперпользователей, обход аутентификации или закладки в программном обеспечении и аппаратуре. В качестве одного из методов защиты предлагаются системы активного аудита, также называемые системами обнаружения вторжений (*Intrusion Detection Systems*, IDS) или системами предупреждения вторжений (*Intustion Prevention Systems*, IPS). Своевременное выявление нарушений безопасности позволяет минимизировать ущерб.

В. Б. Кудрявцев предложил автоматную модель, в рамках которой удается естественным образом отражать факт наличия небезопасных состояний и оперативность реакции на нарушения. В этой модели состояния автомата разбиваются в объединение двух непересекающихся множеств — безопасные и небезопасные. В работе [8] исследованы свойства безопасных входных последовательностей (языков), не выводящих автомат за пределы безопасного подмножества, а также k-безопасных языков, для которых доля небезопасных состояний не превосходит заданной константы ε. Заметим, что при таком определении при условии $\varepsilon > 0$ сколь угодно длинная последовательность небезопасных состояний, поданная после достаточно длинной последовательности безопасных состояний, будет признана безопасной, что противоречит интуитивному представлению о реальном процессе. В работе [9] анализировалась задача определения параметров системы (подмножества безопасных состояний, значения ε) с помощью кратных условных экспериментов. Оказалось, что восстановление разбиения множества состояний может быть проведено с помощью эксперимента квадратичного (от мощности множества состояний) объема, тогда как определение значения ε с помощью конечного эксперимента даже при известном разбиении, как правило, невозможно.

В настоящей работе рассмотрено новое ограничение в модели [8]. Последовательность команд считается безопасной, если максимальная длина непрерывной последовательности небезопасных состояний, порожденных командами, не превосходит заданную константу $k \in \mathbb{N} \cup \{0\}$. Содержательно это ограничение является требованием оперативности реакции на нарушения безопасности. Оказывается, что множество всех входных последовательностей, безопасных для заданного автомата и некоторого k, можно представить как множество всех входных последовательностей, безопасных для $k' < k$ и какого-то другого автомата. Обратный переход (от k к $k'' > k$) в общем случае невозможен.

В работе также рассмотрена задача восстановления параметров системы с помощью кратного условного эксперимента. Содержательно такая задача

* Работа поддержана грантом РФФИ 18-07-01055.

может возникнуть при анализе системы. В частности, высокая сложность восстановления означает, что злоумышленнику для анализа потребуется большое число запросов. В случае, если мощность входного алфавита ограничена сверху некоторой константой, порядок сложности не меняется по сравнению со случаем $k = 0$, рассмотренным в работе [9]. Однако если мощность входного алфавита не ограничена, сложность существенно растет.

1. Основные определения

Приведенные в этом разделе определения следуют идеологии монографии [10].

Определение 1. Конечным инициальным детерминированным автоматом называется шестерка $(A, Q, B, \varphi, \psi, q_0)$, где A, Q, B — конечные непустые множества (входной алфавит, алфавит состояний и выходной алфавит соответственно), φ: $A\times Q\to Q$ функция переходов; ψ: $A\times Q\to B$ — функция выходов; $q_0 \in Q$ — начальное состояние.

В дальнейшем для краткости слова "конечный инициальный детерминированный" будут опускаться. Содержательно автомат представляет собой устройство, функционирующее в дискретном времени и имеющее конечную память (там хранится один символ из множества Q), каждый такт принимающее на вход команду из множества A, перезаписывающее память в соответствии с функцией φ и выдающее выход в соответствии с функцией ψ.

Перейдем от локального описания функционирования к глобальному. Для этого сначала перейдем от однобуквенных входов к входам произвольной длины. Пусть A^* — множество всех конечных слов в алфавите A; Λ — пустое слово. Продолжим функции φ и ψ на множество $(A^*\backslash\{\Lambda\})\times Q$ — по мультипликативности:

$$\varphi(\alpha a, q)=\varphi(a, \varphi(\alpha, q)),$$
$$\psi(\alpha a, q)=\psi(a, \varphi(\alpha, q)).$$

Дополнительно положим $\varphi(\Lambda, q) = q$.

Введем еще одну модификацию функций φ и ψ. Исходные функции на входной последовательности выдают только итоговое состояние и последний выход; новые функции $\overline{\varphi} : A^*\times Q \to Q^*$ и $\overline{\psi} : A\times Q \to B^*$ выдают всю последовательность состояний и выходов соответственно:

$$\overline{\varphi}(\Lambda, q) = q,$$
$$\overline{\psi}(\Lambda, q) = \Lambda,$$
$$\overline{\varphi}(a_1...a_n, q) = \varphi(a_1, q)\varphi(a_1a_2, q)...\varphi(a_1...a_n, q).$$
$$\overline{\psi}(a_1...a_n, q) = \psi(a_1, q)\psi(a_1a_2, q)...\psi(a_1...a_n, q).$$

Произвольное подмножество A^* будем называть языком в алфавите A или просто языком. Важным классом языков, связанных с автоматами, являются регулярные языки. Понятие регулярного языка вводится индуктивно. Рассмотрим операцию конкатенации двух языков $L_1, L_2 \subseteq A^*$, определяемую соотношением $L_1L_2 = \{\gamma \in A^* \mid \exists\alpha \in L_1, \beta \in L_2 : \gamma = \alpha\beta\}$. Легко увидеть, что эта операция ассоциативна, поэтому корректно определена операция возведения в степень относительно конкатенации. Используем возведение в степень для определения операции итерации языка $L \subseteq A^*$:

$$L^* = \bigcup_{i=0}^{\infty} L^i,$$

где $L^0 = \{\Lambda\}$.

Определение 2. Пусть A — конечное непустое множество.

1. $\varnothing$, $\{\Lambda\}$, $\{a\}$, где $a \in A$ — регулярные языки.
2. Пусть $L_1, L_2 \subseteq A^*$ — регулярные языки. Тогда языки $L_1 \cup L_2$, L_1L_2, L_1^* также регулярны.
3. Регулярность произвольного языка устанавливается в соответствии с п. 1 и 2 за конечное число шагов.

По теореме Клини язык является регулярным тогда и только тогда, когда он распознается автоматом. Автоматное распознавание может проводиться либо с помощью выделенного подмножества состояний $Q_0 \subseteq Q$ (слово $\alpha \in A^*$ принадлежит языку тогда и только тогда, когда $\varphi(\alpha, q_0) \in Q_0$), либо с помощью подмножества выходов $B_0 \subseteq B$ (слово $\alpha \in A^*$ принадлежит языку тогда и только тогда, когда $\psi(\alpha, q_0) \in B_0$; в этом случае распознавание возможно с точностью до пустого слова).

Рассмотрим конечный автомат $V = (A, Q, \{0, 1\}, \varphi, \psi, q_0)$. Пусть $Q = S \sqcup I$ (состояния множества S будем называть безопасными, а состояния множества I — небезопасными). Функция выходов будет уточнена позднее, при рассмотрении восстановления параметров автомата с помощью экспериментов. Будем считать, что $q_0 \in S$. Рассмотрим $k \in \mathbb{N} \cup \{0\}$. Автомату V можно поставить в соответствие язык $L(V, k)$ по следующему правилу:

1) $\Lambda \in L(V, k)$;

2) непустое слово $\alpha \in A^*$ принадлежит $L(V, k)$ тогда и только тогда, когда в слове $\overline{\varphi}(\alpha, q_0)$ число идущих подряд небезопасных состояний не превосходит k.

Содержательно п. 1 означает, что в первый момент времени система находилась в безопасном состоянии, а п. 2 — что реакция на выход из множества безопасных состояний, т. е. на нарушения безопасности, каждый раз оказывается достаточно оперативной.

Определение 3. Пусть $k \in \mathbb{N} \cup \{0\}$. Язык $L \subseteq A^*$ называется k-безопасным, если существует автомат V, для которого выполнено равенство

$$L = L(V, k).$$

Легко увидеть, что при $k = 0$ получается в точности класс безопасных языков, введенных в работе [8]. Известно характеристическое свойство таких языков.

Утверждение 1 [8]. Язык L является безопасным языком тогда и только тогда, когда выполнены следующие условия:

1) L непуст;

2) L регулярен;

3) $[L] \subseteq L$, где $[L]$ — множество всех начал слов, принадлежащих L.

В работе [9] рассматривалась задача восстановления параметров автомата с помощью кратного условного эксперимента. В качестве выходной функции рассматривался оракул, возвращавший значение 1 тогда и только тогда, когда входное слово принадлежало соответствующему безопасному языку. Функция переходов φ автомата V считалась известной, и требовалось восстановить разбиение $Q = S \sqcup I$. Заметим, что даже в случае $k = 0$, рассмотренном в работе [9], восстановление возможно с точностью до класса эквивалентности, порожденного равенством задаваемых автоматами безопасных языков. С ростом k мощность таких классов растет, поэтому в настоящей работе сразу перейдем к классам эквивалентности и будем рассматривать задачу восстановления k-безопасного языка.

Определение 4. Простым условным экспериментом назовем конечное ориентированное от корня дерево T с разметкой:

1) каждой вершине v приписано слово $E_v \in A^*$;

2) каждому ребру, исходящему из вершины v, поставлено в соответствие слово из $\{0, 1\}^*$, имеющее длину $|E_v|$, причем разным ребрам, исходящим из одной вершины, приписаны различные слова.

Содержательно определение означает, что имеется "черный ящик", на вход которому можно подавать команды (слова, приписанные вершинам), причем выбор новой команды определяется результатами выполнения старых (переходами по соответствующим ребрам). Эксперимент заканчивается, когда из текущей вершины не исходит ребра, помеченного выходом, полученным на слове, приписанном вершине. Формально результат применения простого условного эксперимента T к автомату V определяется следующим образом. Рассматривается последовательность $v_1, e_1, v_2, ..., v_{n-1}, e_{n-1}, v_n$, где v_1 — корень дерева, ребро e_i идет из v_i в v_{i+1} и имеет пометку $\bar{\psi}\left(\varphi\left(q_0, E_{v_1}...E_{v_{i-1}}\right), E_{v_i}\right)$ (для первого ребра в φ подставляется пустое слово), и из v_n не исходит ребра с соответствующей пометкой. Тогда результат есть $\left(E_{v_1}...E_{v_n}, \bar{\psi}\left(q_0, E_{v_1}...E_{v_n}\right)\right)$.

Целью эксперимента является восстановление тех или иных свойств "черного ящика". Известно [10], что для широкого класса свойств возможностей, предоставляемых простыми экспериментами, недостаточно. Возможный выход из положения — рассмотрение кратных экспериментов. Содержательно это означает, что у "черного ящика" имеется кнопка сброса в начальное состояние, которую разрешается нажимать произвольное число раз. Формально кратным условным экспериментом называется конечное ориентированное от корня дерево, вершинам которого приписаны простые условные эксперименты, а исходящим из вершины ребрам — результаты приписанного вершине эксперимента, причем пометки исходящих из одной вершины ребер попарно различны. Заметим, что переход по ребру подразумевает сброс автомата в начальное состояние.

Будем считать, что эксперимент корректно восстанавливает k-безопасный язык, задаваемый автоматом из класса K, если из условия $V_1, V_2 \in K$, $L(V_1, k) \neq L(V_2, k)$ следует, что результаты эксперимента для V_1 и V_2 отличаются. При этом желательно, чтобы эксперимент имел минимально возможные сложностные характеристики. В контексте настоящей работы рассматриваем кратность M, т. е. число поданных на вход слов, и объем Vol — общее число поданных на вход букв. Строгая постановка задачи и определение функций Шеннона для изучаемых классов приведены в разд. 3.

2. Взаимосвязь k-безопасных языков

В этом разделе изучается взаимосвязь k-безопасных языков при вариации параметра k.

Сначала сформулируем и докажем важное техническое утверждение.

Лемма 1. Пусть $k \in \mathbb{N}$, $L \subseteq A^*$ является k-безопасным языком, заданным автоматом V. Тогда язык L регулярен.

Доказательство. По автомату $V = (A, Q, B, \varphi, \psi, q_0)$ построим автомат $V_k = (A, Q \times C, \{0, 1\}, \varphi_k, \psi_k, q_0')$ следующим образом: $C = \{0, 1, ..., k+1\}$; $q_0' = (q_0, 0)$;

$$\varphi_k\left(a,(q,c)\right) = \begin{cases} (\varphi(a, q), c+1) \text{ при } c \leqslant k \text{ и } \varphi(a, q) \in I; \\ (\varphi(a, q), 0) \text{ при } c \leqslant k \text{ и } \varphi(a, q) \in S; \\ (\varphi(a, q), k+1) \text{ иначе,} \end{cases}$$

$$\psi_k\left(a, (q, c)\right) = \begin{cases} 1 \text{ при } c < k \text{ или } c = k \text{ и } \varphi(a, q) \in S; \\ 0 \text{ в противном случае.} \end{cases}$$

Содержательно компонента C является счетчиком, в котором хранится текущая длина подпоследовательности состояний из множества I. Если эта длина превышает k, автомат переходит в поглощающее состояние и начинает выдавать тождественный 0. Если же все выходы из S имеют длину не больше k, выход равен тождественной единице.

Рассмотрим язык L', задаваемый автоматом V_k с помощью выходного символа 1. Покажем, что $L = L' \cup \{\Lambda\}$. Действительно, $\Lambda \in L$ по определению. Пусть $\alpha \in A^*$, $\alpha \neq \Lambda$. Если $\alpha \in L'$, то $\psi_k\left(\alpha, (q_0, 0)\right) = 1$, т. е. автомат не перешел в поглощающее состояние, максимальная длина непрерывной подпоследовательности состояний из I не превысила k и $\alpha \in L$.

Обратно, пусть $\alpha \notin L'$. Это значит, что $\psi(\alpha, (q_0, 0)) = 0$, т. е. автомат V_k перешел в поглощающее состояние. Значит, нашлась непрерывная подпоследовательность состояний из I длины не меньше $k+1$ и $\alpha \notin L$.

Язык $\{\Lambda\}$ регулярен по определению, язык L' регулярен по теореме Клини, объединение является регулярной операцией. Значит, язык L также регулярен.

Из утверждения 1 и леммы 1 непосредственно вытекает следующий факт.

Следствие 1. Пусть $k \in \mathbb{N}$, $L \subseteq A^*$ является k-безопасным языком. Тогда L является 0-безопасным языком.

Замечание 1. Легко увидеть, что если в V_k в качестве множества безопасных состояний выбрать все состояния вида $(q, 0)$, будет выполнено равенство $L = L(V_k, k)$. Если же положить множество S, равным $\{(q, c) \mid c \leqslant k\}$, то $L = L\left(V_k, 0\right)$. Таким образом,

использование в качестве выходной функции ψ характеристической функции k-безопасного языка не нарушает корректности определения автомата.

Покажем, что на самом деле k-безопасный язык также является k-безопасным не только для $k' = 0$, но и для любого натурального k', меньшего k.

Теорема 1. Пусть $k, k' \in \mathbb{N}$, $k' < k$, $L \subseteq A^*$ является k-безопасным языком, заданным автоматом V. Тогда существует автомат V', такой что $L = L(V', k')$.

Доказательство. Аналогично доказательству леммы 1 по автомату V построим автомат V_k. Заметим, что из условия теоремы вытекает неравенство $k > 1$. Значит, множество C содержит по крайней мере четыре элемента. Рассмотрим множество $S' = \{(q, c) \mid c < 2\}$. Выберем S' в качестве множества безопасных состояний автомата V_k. Покажем, что в этом случае $L = L(V_k, k-1)$. Заметим, что оба языка по определению содержат пустое слово, поэтому достаточно рассмотреть только непустые слова. Кроме того, в силу замечания 1 без ограничения общности можно считать, что автомат V, задающий язык L, есть автомат V_k с множеством безопасных состояний $\{(q, 0)\}$.

Пусть $\alpha \in A^*$, $\alpha \neq \Lambda$. Предположим, что $\alpha \in L$. Значит, длина непрерывной подпоследовательности небезопасных состояний V_k относительно множества S в $\overline{\varphi}(\alpha, (q_0, 0))$ не превосходит k. Заметим, что в каждой такой подпоследовательности первый элемент соответствует состоянию вида $(q, 1)$, т. е. безопасен относительно множества S'. Значит, длина непрерывной последовательности небезопасных состояний относительно S' на единицу меньше исходной, и $\alpha \in L(V_k, k-1)$.

Обратно, пусть $\alpha \notin L$. Значит, в $\overline{\varphi}(\alpha, (q_0, 0))$ имеется непрерывная подпоследовательность небезопасных состояний длины не меньше $k+1$. По построению эта подпоследовательность начинается с элементов $(q_{i_1}, 1), (q_{i_2}, 2), \ldots, (q_{i_k}, k), (q_{i_{k+1}}, k+1)$. Но подпоследовательность $(q_{i_2}, 2), \ldots, (q_{i_k}, k), (q_{i_{k+1}}, k+1)$ состоит из небезопасных относительно S' состояний и имеет длину k. Значит, $\alpha \notin L(V_k, k-1)$.

Следствие 2. Пусть L — k-безопасный язык для некоторого $k \in \mathbb{N}$. Тогда он является k'-безопасным для любого целого неотрицательного $k' < k$.

Замечание 2. Из доказательства теоремы следует, что для перехода от исходного значения k к меньшему значению достаточно "раздуть" множество состояний в $(k+2)$ раз.

Замечание 3. Заметим, что если L является k-безопасным для некоторого $k \in \mathbb{N}$, то по определению L содержит все слова длины k, $(A^k \subseteq L)$. Легко построить автомат, для которого $L = A^k$. Таким образом, существуют k-безопасные языки, не являющиеся k''-безопасными ни для какого $k'' \in \mathbb{N}$, $k'' > k$.

3. Восстановление параметров системы с помощью кратных условных экспериментов

В этом разделе рассматриваются две подзадачи. В первом случае считаются известными функция переходов автомата V и разбиение $Q = S \sqcup I$, и требуется восстановить значение k. Эта задача оказывается простой — для ее решения достаточно простого условного эксперимента линейной по n и k длины. Во втором случае известна функция переходов автомата V и значение k, и требуется восстановить разбиение $Q = S \sqcup I$. В обоих случаях легко построить примеры, когда решение неоднозначно, поэтому ограничимся восстановлением соответствующего k-безопасного языка.

3.1. Восстановление параметра *k*

Пусть задан $V = (A, Q, \{0,1\}, \varphi, \psi, q_0)$. Известны начальное состояние q_0, функция переходов φ и разбиение $Q = S \sqcup I$; значение k неизвестно. Функция выходов ψ является характеристической функцией соответствующего k-безопасного языка. Требуется восстановить значение k с помощью условного эксперимента. Оказывается, что для решения задачи достаточно рассматривать простые условные эксперименты, т. е. кратность M тождественно равна единице, а объем Vol вырождается просто в длину входного слова.

Определим функцию Шеннона объема эксперимента. $Vol(V, k)$ есть минимальная длина эксперимента, достаточного для восстановления параметра k автомата V. Пусть $n \in \mathbb{N}$. Через $K(n)$ обозначим класс всех автоматов, имеющих не более n состояний. Положим

$$Vol(n, k) = \sup_{V \in K(n)} Vol(V, k).$$

Теорема 2. Задача восстановления параметра k решается с помощью простого условного эксперимента. При этом в случае, если все непрерывные пути по множеству небезопасных состояний в диаграмме Мура автомата V ограничены некоторой константой, то $Vol(V, k) \leqslant |Q| - 1$, в противном случае $Vol(V, k) \leqslant |Q| + k - 1$, и обе оценки достигаются.

Доказательство. Рассмотрим два возможных случая.

1. Все непрерывные пути в множестве небезопасных состояний I ограничены сверху некоторой константой.

В этом случае для восстановления значения k достаточно подать одно слово, длина которого не превышает $|Q| - 1$. Действительно, рассмотрим слово, которое сперва переводит автомат в первое состояние самого длинного непрерывного небезопасного пути, а потом проводит автомат по этому пути. Так как начальное состояние q_0 лежит в S, число букв в поданном слове не превосходит $|S| - 1$ (максимальное число состояний до перехода в небезопасное состояний) плюс $|I|$ (оценка сверху на длину небезопасного пути), т. е. $|Q| - 1$. Если в каждый момент времени выходная функция равнялась единице, то автомат задает язык A^*; в противном случае значение k определяется моментом, когда автомат в первый раз выдал 0 (т. е. на единицу меньше длины пути по небезопасным состояниям, на котором в первый раз был выдан 0).

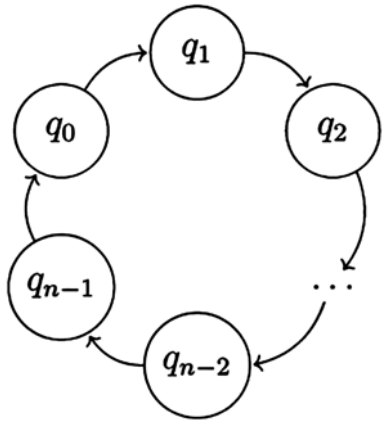


Рис. 1. Диаграмма автомата, на котором достигается оценка в случае 1

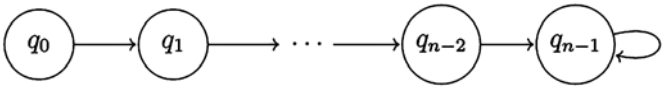


Рис. 2. Диаграмма автомата, на котором достигается оценка в случае 2

Покажем, что оценка является неулучшаемой. Рассмотрим следующий пример, в котором $S = \{q_0, ..., q_{n-2}\}$, $I = \{q_{n-1}\}$ (рис. 1).

Для того чтобы найти число k, в худшем случае нужно побывать во всех состояниях, т. е. подать слово длины $|Q| - 1$. В противном случае, например, не удастся отличить случай $k = 0$ от случая $k = 1$.

2. Есть сколь угодно длинный путь в множестве небезопасных состояний I.

В этом случае для восстановления параметра k также достаточно подать на вход одно слово, по которому автомат сперва перейдет в начало пути по небезопасным состояниям неограниченной длины, затем пройдет по этому пути еще k шагов. Длина первой части пути не превосходит $|Q| - 1$, длина второй части равна k. Таким образом, общая длина не превосходит $|Q| + k - 1$.

Покажем, что оценка неулучшаема.

Рассмотрим пример, в котором $S = \{q_0, ..., q_{n-2}\}$, $I = \{q_{n-1}\}$ (рис. 2).

Следствие 3. $Vol(n, k) = n + k - 1$.

Таким образом, задача восстановления параметра k является простой.

3.2. Восстановление подмножеств *S* и *I*

Пусть $V = (A, Q, \{0, 1\}, \varphi, \psi, q_0)$ — автомат, причем функция φ известна, $k \in \mathbb{N}$ известная константа (случай $k = 0$ разобран в работе [9]), разбиение $Q = S \sqcup I$ неизвестно. В этом случае уже для случая $k = 0$ простого эксперимента недостаточно [9], поэтому приходится переходить к кратным экспериментам. Помимо класса $K(n)$ рассмотрим подмножество этого класса $K(m, n)$, состоящее из автоматов с k-элементным входным алфавитом. Аналогично приведенному ранее, введем функции $M(V, k)$ и $Vol(V, k)$ и рассмотрим соответствующие функции Шеннона

$$M(n, k) = \sup_{V \in K(n)} M(V, k),$$

$$Vol(n, k) = \sup_{V \in K(n)} Vol(V, k),$$

$$M(m, n, k) = \sup_{V \in K(m,n)} M(V, k),$$

$$Vol(m, n, k) = \sup_{V \in K(m,n)} Vol(V, k).$$

Для класса $K(n)$ порядок функций Шеннона растет с ростом k.

Теорема 3. $M(n, k) = \Omega\left(n^{k+1}\right)$, $n \to \infty$.

Напомним, что обозначение $f(x) = \Omega(g(x))$ показывает, что порядок функции f асимптотически ограничен снизу порядком функции g.

Доказательство. Рассмотрим автомат V, диаграмма Мура которого устроена следующим образом. Пусть $n \geqslant 2k + 3$. Помимо начального состояния (пусть оно находится на нулевом слое) есть еще $(k + 1)$ слой из $d = \left[\frac{n-1}{k+1}\right]$ (квадратные скобки означают взятие целой части) состояний каждый; оставшиеся состояния недостижимы из начального. Входной алфавит состоит из d символов. Из каждого состояния на слое номер t, $0 \leqslant t \leqslant k$, можно попасть в любое состояние слоя $t + 1$. На последнем слое, имеющем номер $k + 1$, все переходы являются петлями (рис. 3).

Покажем, что для восстановления распознаваемого k-безопасного языка необходимо подать на вход все d^{k+1} входных слов длины $k + 1$. Будем выбирать множество S таким образом, чтобы на всех поданных входах, кроме, может быть, одного, выдавалось значение 1. Предположим, что по крайней мере одно слово α длины $k + 1$ не вошло в состав эксперимента. Без ограничения общности $\overline{\varphi}(\alpha, q) = q_1 q_{d+1} \cdots q_{kd+1}$ (если это не так, переименуем элементы множества Q). Положим $I_1 = \{q_1, q_{d+1}, ..., q_{(k-1)d+1}\}$, $I_2 = \{q_1, q_{d+1}, ..., q_{kd+1}\}$. Заметим, что в первом случае задаваемый автоматом k-безопасный язык равен

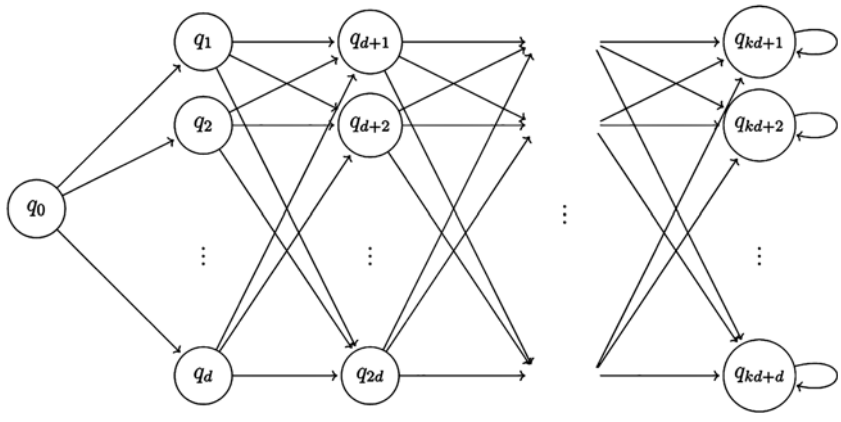


Рис. 3. Диаграмма автомата, на котором достигается нижняя оценка теоремы 3

A^*, так как максимальная длина пути по состояниям из I_1 равна k, а во втором случае этот язык равен $A^*\backslash\{\alpha\}A^*$, так как в любом пути, начало которого порождено словом α, встречается $(k+1)$ состояние из множества I_2. При этом по предположению в составе эксперимента нет входных слов, начинающихся с α, поэтому результаты эксперимента для различных языков, порожденных множествами I_1 и I_2, совпадут, что противоречит определению корректности эксперимента.

Таким образом,

$$M(V, k) \geqslant d^{k+1} = \left\lceil \frac{n-1}{k+1} \right\rceil^{k+1} \geqslant 1\left(\frac{n-1}{k+1} - 1 \right)^{k+1}.$$

Заметим, что начиная с некоторого N выполнено неравенство $\frac{n-1}{k+1} - 1 \geqslant \frac{n}{2(k+1)}$. Поэтому

$$M(V, k) \gtrsim \left(\frac{n}{2(k+1)} \right)^{k+1} \text{ при } n \to \infty.$$

Для завершения доказательства осталось заметить, что знаменатель является константой (так как k фиксировано), а $M(n, k) \geqslant M(V, k)$.

Следствие 4. $Vol(n, k) = \Omega\left(n^{k+1}\right)$, $n \to \infty$.

В случае фиксированного размера входного алфавита в худшем случае достаточно линейного по n числа слов и квадратичного числа поданных букв.

Теорема 4. Пусть $k, m \in \mathbb{N}$, $m \geqslant 2$. Тогда $M(m, n, k) = \Theta(n)$, $Vol(m, n, k) = \Theta\left(n^2\right)$, $n \to \infty$.

Напомним, что обозначение $f(n) = \Theta(g(n))$ показывает равенство порядков функций f и g.

Доказательство. Для доказательства верхней оценки рассмотрим процедуру послойного восстановления множества S. Упорядочим состояния по мере удаления от начального состояния в диаграмме Мура. Начальное состояние безопасно по определению. Восстановим принадлежности всех состояний, достижимых из начального за один переход. Для этого очевидным образом достаточно рассмотреть все пути в диаграмме Мура длины k, начинающиеся в исследуемом состоянии. Таких путей не больше, чем m^k, т. е. $O(1)$, $n \to \infty$. Состояние q принадлежит множеству I, только если найдется входное слово, по которому автомат сперва перейдет в q, а затем пройдет k шагов по состояниям из I. Восстановление принадлежности состояний следующих слоев проводится аналогично. Состояния, не достижимые из начального, не влияют на задаваемый автоматом язык и могут быть классифицированы произвольно. Таким образом, процедура позволяет однозначно восстановить задаваемый автоматом язык. Для каждого состояния используется константное число входных слов, т. е. общее число слов линейно по n, а общее число поданных букв квадратично.

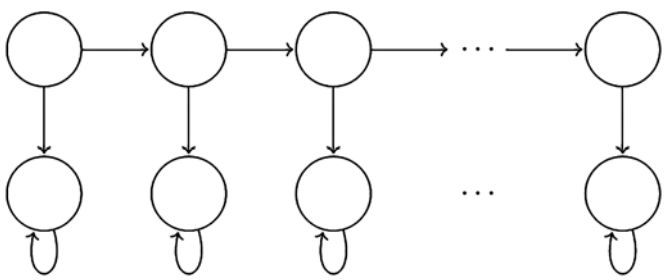


Рис. 4. Диаграмма автомата, на котором достигается нижняя оценка теоремы 4

Нижние оценки следуют из рассмотрения класса автоматов, диаграмма Мура которых имеет вид, представленный на рис. 4.

Все состояния верхнего ряда принадлежат множеству S. Для полного восстановления множества S необходимо, чтобы автомат побывал в каждом состоянии нижнего ряда. Для этого потребуется линейное по n число слов и квадратичное число букв.

Заключение

Проведено обобщение модели, предложенной в работе [8], позволяющее ограничивать длины непрерывных небезопасных подпоследовательностей константой $k \in \mathbb{N} \cup \{0\}$. Показано, что произвольный k-безопасный язык является $(k-1)$-безопасным; обратное, вообще говоря, неверно. Оценены сложностные характеристики кратных условных экспериментов, позволяющих восстановить параметры системы. В дальнейшем планируется одновременное наложение пары ограничений (доля небезопасных состояний не превосходит ε, максимальная длина непрерывного небезопасного участка не превосходит k). Этот случай является существенно более сложным, так как потребует выхода за пределы класса регулярных языков. Более того, из результатов работы [8] следует, что рассмотрение иррациональных ε приводит к континууму попарно различных языков, т. е. выводит за пределы вычислимости.

Де-факто характеристическим свойством безопасных языков, введенных в [8], и, как следствие, k-безопасных языков для любого $k \in \mathbb{N}$, является одновременная регулярность и замкнутость относительно взятия префикса. В частности, это означает, что безопасные языки являются максимальным подклассом регулярных языков, для которого проходят индуктивные доказательства. В работе [11] для моделирования безопасности предлагается задавать состояния объектов системы в виде регулярных языков. Интересным вопросом является исследование задачи при переходе от произвольных регулярных языков к k-безопасным. Ожидается, что возможность проведения индуктивных рассуждений позволит находить локальные свойства объектов, гарантирующие глобальную безопасность.

Авторы благодарят В. Б. Кудрявцева за постановки задач, В. А. Васенина и С. А. Афонина за внимание к работе и ценные замечания.

Список литературы

1. **Harrison M. A., Ruzzo W. L., Ullman J. D.** Protection in Operating Systems // Communications of the ACM. —1976. — Vol. 19, No. 8. — P. 461—471.

2. **Bel D., LaPadula L. J.** Secure Computer Systems: Mathematical Foundations, MITRE Technical Report 2647, vol. I. National Technical Information Service, 1973.

3. **Goguen J. A., Meseguer J.** Security policy and security models // Proc. of the 1982 Symposium on Security and Privacy, IEEE Press, 1982. — P. 11—20.

4. **Moskowitz I. S., Costich O. L.** A classical automata approach to noninterference type problems // Proc. Computer Security Foundations Workshop V, 1992. — P. 2—8.
5. **Васенин В. А., Иткес А. А., Шапченко К. А., Бухонов В. Ю.** Реляционная модель логического разграничения доступа на основе цепочек отношений // Программная инженерия. — 2015. — № 9. — С. 11—19.
6. **Afonin S., Bonushkina A.** Validation of Safety-Like Properties for Entity-Based Access Control Policies // Advances in Soft and Hard Computing. — 2019. — P. 259—271.
7. **Галатенко А. В., Плетнева В. А.** Выразимость моделей безопасности take-grant и невлияния в рамках модели CBAC // Программная инженерия. — 2020. — Т. 11, № 1. — С. 40—46.
8. **Галатенко А. В.** Автоматные модели защищенных компьютерных систем // Интеллектуальные системы. — 2007. — Т. 11, № 1—4. — С. 403—418.
9. **Галатенко А. В.** О восстановлении разбиения безопасности // Интеллектуальные системы. — 2010. — Т. 14, № 1—4. — С. 123—136.
10. **Кудрявцев В. Б., Алешин С. В., Подколзин А. С.** Введение в теорию автоматов. — М.: Наука, 1985. — 320 с.
11. **Кузнецова А. Л., Афонин С. А.** Автоматная модель проверки корректности атрибутной политики информационной безопасности в системах с конечным числом объектов // Вестник Московского университета. Серия 1: Математика. Механика. Принято к печати.

A Model of Secure Functioning of Computer Systems

A. V. Galatenko, agalatenko@hse.ru, HSE University, Moscow, 101000, Russian Federation,
V. A. Kuzovikhina, pletnyova_va@mail.ru, New school, Moscow, 119192, Russian Federation

Corresponding author:

Kuzovikhina Vesta A., Teacher, New school, Moscow, 119192, Russian Federation
E-mail: pletnyova_va@mail.ru

Received on December 29, 2020
Accepted on January 15, 2021

We propose an automata model of computer system security. A system is represented by a finite automaton with states partitioned into two subsets: ''secure'' and ''insecure''. System functioning is secure if the number of consecutive insecure states is not greater than some nonnegative integer k. This definition allows one to formally reflect responsiveness to security breaches. The number of all input sequences that preserve security for the given value of k is referred to as a k-secure language. We prove that if a language is k-secure for some natural and automaton V, then it is also k′-secure for any $0 \leqslant k' < k$ and some automaton $V = V'(k')$. Reduction of the value of k is performed at the cost of amplification of the number of states. On the other hand, for any non-negative integer k there exists a k-secure language that is not k″-secure for any natural $k'' > k$. The problem of reconstruction of a k-secure language using a conditional experiment is split into two subcases. If the cardinality of an input alphabet is bound by some constant, then the order of Shannon function of experiment complexity is the same for al k; otherwise there emerges a lower bound of the order n^k.

Keywords: *formal security model, finite automata, regular languages, multiple conditional experiments*

Acknowledgements:

This work was supported by the Russian Foundation for Basic Research, project no. 18-07-01055.

For citation:

Galatenko A. V., Kuzovikhina V. A. A Model of Secure Functioning of Computer Systems, *Programmnaya Ingeneria*, 2021, vol. 12, no. 3, pp. 150—156.

DOI: 10.17587/prin.12.150-156

References

1. **Harrison M. A., Ruzzo W. L., Ullman J. D.,** Protection in Operating Systems, *Communications of the ACM*, 1976, vol. 19, no. 8, pp. 461—471.
2. **Bel D., LaPadula L. J.** Secure Computer Systems: Mathematical Foundations, MITRE Technical Report 2647, vol. I. National Technical Information Service 1973.
3. **Goguen J. A., Meseguer J.** Security policy and security models, *Proc. of the 1982 Symposium on Security and Privacy*, *IEEE Press*, 1982, pp. 11—20.
4. **Moskowitz I. S., Costich O. L.** A classical automata approach to noninterference type problems, *Proc. Computer Security Foundations Workshop V,* 1992, pp. 2—8.
5. **Vasenin V. A., Itkes A. A., Shapchenko K. A., Bukhonov V. Yu.** Relational Access Control Model Based on Chains of Relations, *Programmnaya Ingeneria*, 2015, vol. 9, no. 11, pp. 19—19 (in Russian).
6. **Afonin S., Bonushkina A.** Validation of Safety-Like Properties for Entity-Based Access Control Policies, *Advances in Soft and Hard Computing*, 2019, pp. 259—271.
7. **Galatenko A. V., Pletneva V. A.** Embeddability of take-grant and noninterference security models in CBAC model, *Programmnaya Ingeneria*, 2020, vol. 11, no. 1, pp. 40—46 (in Russian).
8. **Galatenko A. V.** Automata-based models of secure computer systems, *Intellectualnye systemy*, 2007, vol. 11, no. 1—4, pp. 403—418 (in Russian).
9. **Galatenko A. V.** Reconstruction of security partitioning, *Intellectualnye systemy*, 2010, vol. 14, no. 1—4, pp. 123—136 (in Russian).
10. **Kudryavtsev V. B., Aleshin S. V., Podkolzin A. S.** *Introduction to automata theory*, Moscow, Nauka, 1985, 320 p. (in Russian).
11. **Kuznetsova A. L., Afonin S. A**. An automata model for deciding correctness of attribute-based security policy in systems with a finite number of objects, *Vestnik Moscowskogo universiteta. Seria 1: Matematika. Mehanika* (in Russian). Received.

УДК 004.82 DOI: 10.17587/prin.12.157-168

К. И. Костенко, канд. физ.-мат. наук, доц. кафедры, kostenko@kubsu.ru,
Кубанский государственный университет, Краснодар

Инварианты ядра фундаментальной модели интеллектуальной системы

Приведено целостное описание универсальной математической модели для понятия интеллектуальной системы. Оно основывается на формализованных инвариантах, связанных с процессами построения и применения таких систем. Ядро модели составляют согласованные описания разделов формализмов знаний, многомерной архитектуры компонентов и процессов, а также иерархия агентов управления субъектным существованием интеллектуальных систем. В элементах данных трех разделов совместно реализуются фундаментальные аспекты представления и обработки знаний, разрабатываемые и применяемые в областях знаний, исследующих общие свойства структурной организации памяти и процессов мышления. Инструментами трансформации предложенной абстрактной модели в модели конкретных интеллектуальных систем являются морфизмы гомоморфного расширения. Эти морфизмы конкретизируют содержание основных структурных и функциональных элементов начальной модели. При этом происходит сужение многообразий сущностей, реализуемых элементами модели до семейств объектов, составляющих прикладные интеллектуальные системы, наследующие свойства общих элементов. Промежуточные модели процессов преобразования исходной модели в прикладные допускают математическое исследование. Его результаты составляет основание для последующей разработки специальной технологии построения и применения интеллектуальных систем.

Ключевые слова: *интеллектуальная система, формализм знаний, измерение знаний, поток знаний, онтология, синтез знаний, морфизм знаний, когнитивная цель, интеллектуальный агент*

Введение

Высокоуровневое математическое моделирование интеллектуальных систем (далее ИС) связано с применением специальных множеств объектов и операций над ними. Они ассоциируются с сущностями из разных областей математики, имеющих разный уровень общности. Математические объекты и операции являются формальными аналогами элементов содержания смежных областей знаний, в которых изучаются структуры памяти и процессы мышления. К таким областям относятся философия, кибернетика, лингвистика, когнитивная психология, системная инженерия. Моделирование структур представления и способов обработки знаний в указанных областях базируется на системах понятий (инвариантах), в значительной мере являющихся слабо формализованными. Рассматриваемые понятия связаны с разными аспектами сущностей, самостоятельно развиваемыми в границах соответствующих областей знаний.

Для формализации указанных сущностей естественно применение элементов, относящихся к областям абстрактной математики. Такие элементы являются исследованными и при удачной адаптации позволяют разрабатывать разные по глубине и полноте фрагменты специальных математических моделей ИС. Получаемые при этом модели оказываются междисциплинарными. Они являются применениями фундаментальных моделей. Основу процессов адаптации составляют операции гомоморфного расширения. Расширение моделей заключается в конкретизации и детализации абстрактных сущностей, наделении их новыми свойствами. Таким образом достигается интеграция и согласование неформальных и слабо формализованных представлений о знаниях, развиваемых в разных областях. Трансформации получаемых при этом математических моделей в прототипы прикладных ИС основаны на последовательном сужении общности применяемых в них сущностей. Для этого используются подходящие схемы уточнения форматов представления знаний и описаний классов операций над ними. Обозначенные трансформации реализуются посредством внесения изменений в соответствующие формализованные описания. В частности, преобразования описаний структур применяемых знаний выполняются операциями декомпозиции или проекции. Аналогичные преобразования определений классов операций состоят в уточнении областей определения и значений операций, а также в сужении областей значений переменных, связываемых кванторами в применяемых формулах.

Наделение абстрактных знаний и операций над ними дополнительными свойствами позволяет конструировать модели конкретных ИС. Модели используют системы знаний, представляющие содержание соответствующих областей деятельности. В них обеспечивается возможность реализации аналогов процессов мышления специалистов, поддерживающих приемлемый уровень решения профессиональных задач, основанного на обработке содержания таких областей.

Уточнения систем свойств знаний и классов операций над знаниями в моделях ИС опираются на содержание моделируемых областей знаний. Формализации фундаментальных свойств и принципов работы со знаниями в смежных (нематематических) областях определяют дополнительные инварианты моделей. Процесс развития математических моделей ИС включает расширение многообразия применяемых фундаментальных инвариантов, формализующих сущности разных областей знаний. Таким образом достигается приемлемая полнота заключительных моделей относительно разнообразных экспертных представлений. Для начальных моделей, разрабатываемых на основе обозначенного подхода, сохраняется возможность проведения фундаментального исследования формальных основ ИС. В областях знаний, связанных с процессами мышления и организации памяти, формируются системы инвариантов. Из них составляются специальные слабо формализованные модели. Систематизация и согласование семейств таких инвариантов делает возможным их совместное применение. Формализация инвариантов реализуется с использованием сущностей разных разделов математики. Источниками инвариантов рассматриваемого далее ядра математических моделей ИС являются работы [1—3].

1. Формализмы представления знаний

Формализмы представления знаний составляют специальный класс математических систем. Они основаны на уточнении понятий формы представления и сравнения содержания знаний. Для этого используются фундаментальные инварианты структуры и вложения знаний. Эти инварианты образуют фундамент последующей разработки связанных с ними абстрактных сущностей формируемой модели ИС.

Определение. Формализмом представления знаний называется четверка $\Im = (D_\Im, M_\Im, \circ, \leqslant)$, где $D_\Im$ — перечислимое множество; $M_\Im \subseteq D_\Im$ — разрешимое подмножество $D_\Im$; $\circ : D_\Im \times D_\Im \to D_\Im$ — вычислимая операция; $\leqslant$ — разрешимое отношение на $D_\Im$.

Элементы множества $D_\Im (M_\Im)$ называются фрагментами знаний (знаниями), операция $\circ$ — композицией, а $\leqslant$ — вложением фрагментов знаний. Множество $M_\Im$ содержит пустое знание, обозначаемое как Λ. Такое знание вложено в каждый элемент $D_\Im$.

Всякое выражение, представляющее $z \in D_\Im$ как композицию фрагментов знаний, определяет алгебраическую структуру z. Эта структура представляется нагруженным бинарным деревом. Внутренние вершины указанного дерева размечены операцией композиции, а висячие — фрагментами знаний, из которых конструируется представление z.

Если композиция двух фрагментов знаний равна Λ, то такие фрагменты интерпретируются как несовместимые, а сама комбинация рассматривается как имеющая пустое содержания. В общем случае операции композиции один фрагмент знаний может иметь несколько алгебраических структур. Это усложняет моделирование процессов обработки структурных представлений знаний.

Отношение $\leqslant \subseteq D_\Im \times D_\Im$ называется вложением (вложением содержания) фрагментов знаний. Диаграмма данного отношения определяет семантическую структуру множества $D_\Im$.

Алгебраические структуры отдельных знаний и семантическая структура $D_\Im$ составляют основу исследования многообразий свойств абстрактных знаний и морфизмов знаний. Знание называется элементарным, если оно либо пустое, либо не представляется композицией других фрагментов знаний. Среди неэлементарных знаний особое семейство составляют простые знания, соответствующие тройкам *RDF*. Всякое простое знание составляется из трех элементов. В таком представлении с помощью первой композиции конструируется пара элементарных знаний. Вторая композиция связывает такую пару с семантическим отношением, которое выполняется между элементами пары. В применяемом формализме это отношение также относится к элементарным знаниям. В структурах знаний такое элементарное знание наделяется специальной ролью. Содержание элементарного знания, соответствующего отношению, раскрывается с помощью множества простых знаний, использующих это отношение.

Среди формализмов знаний, для которых любое неэлементарное знание представляется единственной композицией элементарных знаний, особое положение занимают пространства конфигураций (семантических иерархий). Множество формализмов семантических иерархий обозначается как *SH*. Такие пространства определяются как четверки $(M, R, \varepsilon, \psi)$. Отдельные знания в них называются конфигурациями. Множество конфигураций M является перечислимым. Множество разрешимых отношений на M составляет перечислимый класс R. Для этого класса разрешимо свойство вложения отношений. Множество R содержит специальные отношения $\perp = \varnothing$ и $T = M \times M$. Пустое отношение $\perp$ связывает любые две конфигурации в конструкцию пары несвязанных конфигураций.

Алгебраическая структура всякой неэлементарной конфигураций $z \in M$ связана с процессом декомпозиции конфигураций. Последний основан на разложении z на пару частей $\varepsilon(z) = (z_1, z_2)$ с помощью вычислимого отображения $\varepsilon : M \to M \times M$. Конфигурация z называется элементарной, если $\varepsilon(z) = (\Lambda, \Lambda)$. Отношение, составляющее из z_1 и z_2 конфигурацию z, вычисляется с помощью вычислимого отображения связывания $\psi : M \to R$. Поэтому, если $\varepsilon(z) = (z_1, z_2)$ и $\psi(z) = r$, то $(z_1, z_2) \in r$.

Отображения ε и ψ определяют полные структурные представления (ПСП) произвольных кон-

фигураций. Они имеют вид нагруженных бинарных деревьев. Висячие вершины ПСП размечены элементарными конфигурациями, а остальные — отношениями из R. Корень дерева ПСП неэлементарной конфигурации z размечен отношением $\psi(z)$, а левое и правое поддеревья корня — это ПСП конфигураций z_1 и z_2. Вершины деревьев ПСП конфигураций задаются двоичными наборами. Множество всех таких наборов обозначается как I. Всякий набор $\alpha \in I$ определяет путь из корня ПСП конфигураций в вершину α. Пустой набор, являющийся корнем бинарного дерева, обозначается как λ. Отношение вложения двоичных наборов основано на свойстве префиксности ($\alpha \subseteq \beta$ означает, что α является префиксом (началом) β).

Множество конфигураций, ПСП которых имеют глубину $k \in N$, обозначается как M_k. Из них M_0 составляют элементарные знания, а M_1 — класс простых знаний рассматриваемого формализма.

Определение пространств семантических иерархий как формализмов представления знаний получается расширением M до множества всех фрагментов конфигураций. Последние представляются парами (z, ρ), где $z \in M$, а $\rho \in R$. Операция композиции в пространствах конфигураций моделирует процесс конструирования ПСП конфигураций снизу-вверх и слева-направо. Разные уточнения этой операции определяют разные формализмы пространств семантических иерархий. Рассмотрим случай, когда алгебраическая структура произвольной неэлементарной конфигурации $z \in M$, для которой $\varepsilon(z) = (z_1, z_2)$ и $\psi(z) = r$, конструируется из алгебраических структур z_1 и z_2, задаваемых выражениями σ_1 и σ_2 в виде комбинации $(\sigma_1 \circ r) \circ \sigma_2$. В остальных случаях композиции фрагментов принимают значение, равное Λ. Определение операции композиции фрагментов, для которой всякая непустая конфигурация имеет ровно одну алгебраическую структуру, представляется следующим образом:

$$\sigma_1 \circ \sigma_2 = \begin{cases} (z, r), \text{ если } \sigma_1 \text{ — алгебраическая структура } z \in M \ \& \ r \in R; \\ z, \text{ если } \sigma_1 \text{ — алгебраическая структура фрагмента } (z_1, r) \ \& \\ \quad \& \ \sigma_2 \text{ — алгебраическая структура } z_2 \in M \ \& \ \varepsilon(z) = (z_1, z_2) \ \& \ \psi(z) = r); \\ \Lambda \text{ — в остальных случаях.} \end{cases}$$

Если z — фрагмент конфигурации в формализме семантических иерархий, то выражение $\Sigma(z)$ обозначает алгебраическую структуру этого фрагмента. Множество всех структур фрагментов конфигураций обозначается как Σ.

Отношение вложения фрагментов знаний формализма семантических иерархий связано с расширением понятия трассирования конфигураций на множество $D_\Im$ [4]. Трассирование $z_1 \in M$ в $z_2 \in M$ выполняется, если существует изотонное отображение $\xi : I \to I$, для которого внутренние (висячие) вершины ПСП z_1 отображаются во внутренние (висячие) вершины ПСП z_2. Для ξ выполняются дополнительные условия:

1) если $\xi(\alpha) \subset \xi(\alpha\sigma)$, где $\alpha \in I$ и $\sigma \in \{0, 1\}$, то $\xi(\alpha\sigma) = \xi(\alpha)\beta\sigma\gamma$, где $\beta, \gamma \in I$;

2) для разметок внутренних (висячих) вершин z_1 и z_2, связываемых отображением ξ, имеет место вложение отношений из R (вложения содержания элементарных знаний), приписанных таким вершинам.

Важными частными случаями отображений трассирования являются сжатия и растяжения. Отображение трассирования ξ называется сжатием (растяжением), если $\beta, \gamma = \lambda$ (ξ — инъективно и $\beta = \lambda$). Общий случай, когда допускаются как сжатия, так и растяжения, представляют o-трассирования ($\beta = \lambda$). Класс o-трассирований составляет основу общего случая моделирования вложения фрагментов знаний в формализмах семантических иерархий [4].

Понятие трассирования конфигураций может быть перенесено с бинарных деревьев ПСП конфигураций на бинарные деревья алгебраических структур знаний в произвольных формализмах с сохранением их основных свойств. Поэтому для ПСП конфигураций обеспечивается универсальность использования как формата моделирования структурных представлений знаний в произвольных формализмах знаний [4, 5].

1.1. Базы морфизмов формализмов знаний

Структурные представления отдельных знаний образуют содержание фрагментов памяти ИС. В общем случае такие представления реализуются в формате алгебраических структур знаний применяемых формализмов. Для формализмов семантических иерархий используются также форматы ПСП отдельных конфигураций и их фрагментов. Многообразие структур представлений знаний в формализмах распадается на классы фрагментов знаний с общими структурными свойствами. Классы называют базами. Их применяют в качестве областей определения и значения для операций над знаниями (морфизмов) в моделях ИС. В абстрактной модели ИС базы составляют перечислимое семейство перечислимых классов объектов $\Psi = \{B_i \mid i \in N \ \& \ B_i \subseteq \Sigma\}$. Для этого семейства отношения принадлежности элементов отдельным базам и вложения баз являются разрешимыми. Также $\Sigma \in \Psi$.

Примерами баз операций в формализме семантических иерархий являются серии и окрестности элементарных конфигураций, обозначаемые как S и O. Полные структурные представления знаний, являющихся элементами указанных баз, приведены на рис. 1.

Элементами баз первого типа являются серии знаний из некоторой базы (например, базы элементарных знаний в применяемом формализме). Серию образуют знания $z_1, ..., z_k$ (см. рис. 1, *а*). Эти знания составлены в структуру серии с помощью специального отношения $\vdots$. Окрестность знания z, изображенную на рис. 1, *б*, составляют знания $z_1, ..., z_k$, находящиеся с z в отношении ρ.

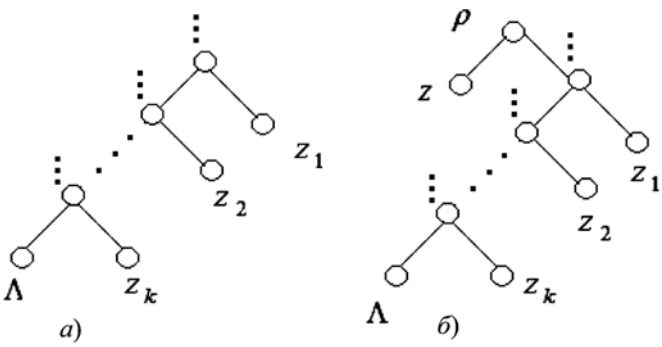


Рис. 1. Конфигурации серии знаний и окрестности знания

Определение. Композицией (прямой суммой) произвольных классов конфигураций $B_i, B_j \in \Psi$ (обозначается как $B_i \circ B_j$ или $B_i \oplus B_j$) называется класс $B = \{z_1 \circ z_2 \mid z_1 \in B_i \,\&\, z_2 \in B_j\}$ $\left(B = \{z_1 \oplus z_2 \mid z_1 \in B_i \,\&\, z_2 \in B_j\}\right)$.

В данном определении отображение $\oplus : M \times M \to M$, задаваемое соотношением $\forall z_1, z_2 \in M \left(z_1 \oplus z_2 = z \to \varepsilon(z) = (z_1, z_2) \,\&\, \psi(z) = \perp\right)$, называется прямой суммой конфигураций.

Морфизм прямой суммы $\oplus$ дополняется обратными к нему морфизмами левой Pr_0 и правой Pr_1 проекций конфигураций. Если $z = z_1 \oplus z_2$, где z_1 и z_2 принадлежат произвольным базам морфизмов B_1 и B_2, то выражения $(z)_0$ и $(z)_1$ обозначают z_1 и z_2 соответственно как значения морфизмов левой и правой проекции для морфизма прямой суммы.

Обозначения $D(z)$ и $O(z)$, где $z \in M$, применяются для множества вершин и множества висячих вершин ПСП конфигурации z соответственно. Для значений разметки вершины $\alpha \in D(z)$ и конфигурации с корнем в такой вершине применяются обозначения $[z]_\alpha$ и $(z)_\alpha$ соответственно.

Определение. Произведением баз B_i и B_j в формализме $\Im \in SH$ называется множество

$$B_i \otimes B_j =$$

$$= \{z \mid \exists z_1 \in B_i \, \exists D \subseteq O(z_1)(\forall \alpha \in D(z_1) \setminus O(z_1)([z]_\alpha = [z_1]_\alpha) \,\&$$

$$\&\forall \alpha \in D \, \exists z_2 \in B_j((z)_\alpha = z_2) \,\&\, \forall \alpha \in O(z_1) \setminus D((z)_\alpha = [z_1]_\alpha))\}.$$

Произведение баз $B_i, B_j \in \Psi$ также обозначается как B_iB_j. Алгебраические структуры знаний из произведений баз формируются на основе ПСП знаний из первой базы, в которых висячие вершины структурных представлений заменяются на ПСП знаний из второй базы. Допускается неполная замена висячих вершин ПСП конфигураций первой базы на ПСП конфигураций базы второго множителя произведения баз.

Дополнительные свойства семейства Ψ являются аналогами понятия замкнутости операций из разных областей математики. Ими обеспечивается возможность абстрактного конструирования и исследования свойств баз в произвольных формализмах знаний. Примерами таких условий являются:

1) для Ψ выполняются аксиомы топологии, предполагающие, что Σ и $\varnothing$ являются элементами Ψ и Ψ замкнуто относительно перечислимых объединений и конечных пересечений элементов Ψ;

2) Ψ замкнуто относительно операции прямой суммы (композиции) баз;

3) Ψ замкнуто относительно операции произведения баз.

Формулы, составляемые как композиции сумм и произведений баз, определяют новые базы. Например, композиция $S \otimes M_1$ задает семейство неупорядоченных серий простых знаний. Аналогично, прямое произведение $\Sigma \otimes (S \otimes M_1)$ соответствует конфигурациям, в ПСП которых висячие вершины заменены на неупорядоченные серии простых знаний. В общем случае пространств семантических иерархий операция прямой суммы неассоциативная и некоммутативная. Операция произведения баз ассоциативная, но в общем случае она некоммутативная. Формулы, составленные из элементов Ψ и операций над ними, применяются для моделирования структурной организации памяти ИС. Такие структуры адаптированы к процессам когнитивного синтеза и операциям над знаниями, реализуемым в таких системах. Возможность единой классификации структур содержания памяти для разных формализмов знаний (формализмов семантических иерархий) обеспечивает сходство форматов алгебраических структур знаний в таких формализмах.

Формализмы семантических иерархий составляют унифицированный и универсальный подход к моделированию понятий формы и содержания знаний. С ними связано развиваемое семейство баз конфигураций, применяемых для моделирования процессов обработки знаний в ИС [4]. Последние включают неструктурированные знания (M), неструктурированные знания глубины $k \in N$ (M_k), ПСП конфигураций (Σ), ПСП конфигураций глубины $k \in N$ (Σ_k), неупорядоченные и упорядоченные серии элементарных конфигураций (S и $\bar{S}$), а также окрестности конфигураций разной глубины.

Если $B \in \Psi$, то обозначение B^* применяется для базы, составленной знаниями, принадлежащими прямым суммам $\left(\ldots\left(\{\Lambda\} \oplus_: B\right) \oplus \ldots\right) \oplus B$. Такими суммами представляются параллельные серии знаний из B. Обратные операции к произвольным суммам вида $\left(\ldots\left(\{\Lambda\} \oplus_: B_{i_1}\right) \oplus \ldots\right) \oplus B_{i_r}$ моделируются как проекции на фрагменты серий.

1.2. Классы морфизмов абстрактных знаний

Морфизмы в формализмах знаний являются аналогами операций над знаниями в ИС. Унификация атрибутов формализмов позволяет адаптировать функциональные сущности из разных разделов математики к специфике знаний. При этом операции остаются достаточно общими, поскольку они применяются к представлениям знаний в абстрактных моделях. Поэтому многообразие морфизмов формируется как система абстрактных классов морфизмов.

Многообразие морфизмов над знаниями удобно представлять в виде перечислимого множества классов вычислимых морфизмов $\Phi = \{F_i \mid i \in N\}$. Здесь $F_i = \{f_{i,j} \mid i, j \in N \,\&\, f_{i,j} : B_i^1 \to B_i^2\}$, где B_i^1 и B_i^2 —

базы из Ψ. Они составляют области определения и значения морфизмов $f_{i,j}$. Для Φ разрешимо отношение вложения классов из этого семейства. Всякий класс представляется формализованным математическим описанием. Формулами определений классов морфизмов реализуются предикаты, характеризующие элементы отдельных классов. Классы могут быть не перечислимыми. Перечисление классов, применяемое в конкретных моделях ИС, моделирует индуктивный процесс появления в таких моделях новых классов или новых морфизмов.

Пример системы классов вычислимых морфизмов для формализмов семантических иерархий приведен на рис. 2.

Элементы этих классов формализуют унифицированный фрагмент многообразия морфизмов над абстрактными знаниями. Они являются аналогами фундаментальных операций из разных областей математики. Комбинации морфизмов приведенных классов являются основой высокоуровневого моделирования морфизмов, представляющих когнитивные операции и процессы.

Формальные определения классов морфизмов из приведенной иерархии даны в работе [5]. Развитие иерархии связано с добавлением независимых классов и гомоморфных расширений представленных классов. В последнем случае применяются операции гомоморфного расширения, детализирующие или конкретизирующие структурные и функциональные свойства их элементов. В прикладных моделях ИС процессы расширения заканчиваются одноэлементными классами морфизмов.

Операция произведения классов морфизмов применяется к классам морфизмов с общей областью определения для морфизмов из таких классов. Если $F_i,\ F_j \in \Phi$ — семейства, для которых $B_i^1 = B_j^1$, то произведением классов F_i и F_j (обозначается как $F_i \times F_j$) называется класс

$$F_k = \{f \mid f : B_i^1 \to B_i^2 \oplus B_j^2 \ \&\ \exists f' \in F_i,$$
$$f'' \in F_j\ \forall z \in B_i^1 \left(f(z) = (f'(z) \oplus f''(z))\right)\}.$$

1.3. Эволюции знаний

Эволюции знаний — это формальные объекты, применяемые для моделирования процессов обработки знаний в ИС. В формализме семантических иерархий они определяются как вычислимые последовательности ПСП конфигураций (в общем случае формализмов — алгебраических структур знаний), конструируемые в дискретном времени [6]. Эволюции конфигураций задают форматы моделирования процессов обработки знаний, представляемых последовательными сериями ПСП конфигураций. Исходными данными эволюций конфигураций также являются последовательные серии ПСП конфигураций. Новые конфигурации добавляются в такие серии в возрастающие моменты времени. Семейства эволюций конфигураций с общими алгоритмическими инструментами (базисами) их формирования для произвольных начальных данных называются пространствами эволюций конфигураций [6]. Эволюции конфигураций — это абстрактные сущности формализмов знаний. Они применяются для моделирования жизненных циклов ИС. Основу функциональных описаний таких циклов составляют базисы эволюций.

Определение. Базисом пространства эволюций конфигураций называется семейство пар операторов перехода и остановки, представляемое как $\underset{\alpha \in I}{\oplus} (T_\alpha, S_\alpha)$, где $T_\alpha : \Sigma \circ \Sigma \to R \cup M_1,\ \ S_\alpha : \Sigma \circ \Sigma \to \{0, 1\}$.

Отображения T_α, S_α в последнем определении называются операторами перехода и остановки соответственно. Операторы перехода вычисляют значения разметок вершин ПСП конфигураций, последовательно добавляемых в реализации эволюций конфигураций в возрастающие моменты времени. Области определения таких операторов составляют множества начал эволюций и фрагментов начальных данных эволюций. Операторы остановки распознают фрагменты вычисляемых конфигураций, которые включаются в значения результатов отдельных эволюций, связанных с $\alpha \in I$.

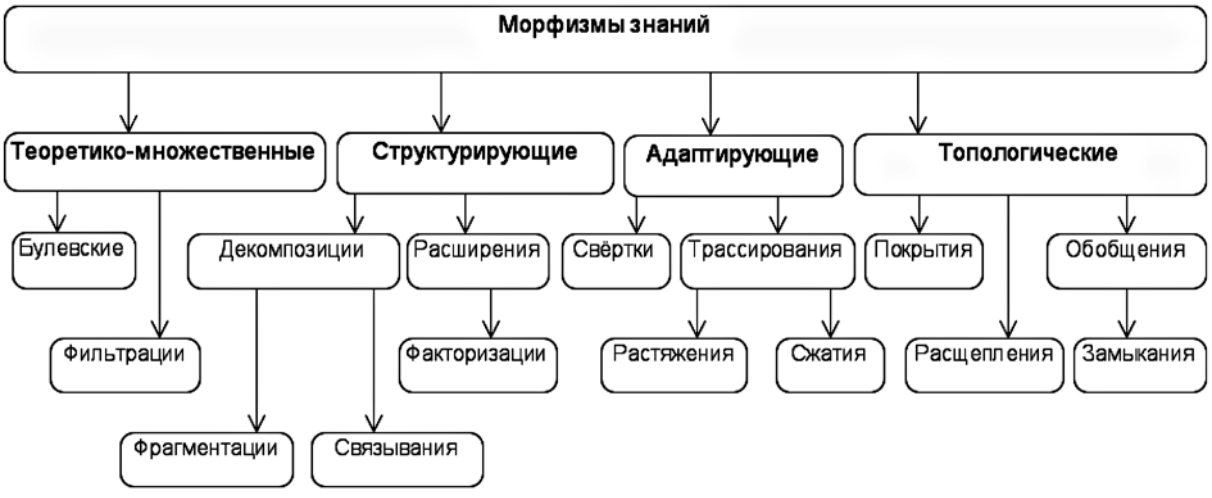


Рис. 2. Классы морфизмов для формализмов семантических иерархий

Области определения и значения морфизма, пошагово моделирующего процессы, порождаемые фиксированным базисом пространства эволюций конфигураций, представляются как прямая сумма трех вычислимых бесконечных серий конфигураций: начального данного эволюции, серий конфигураций, составленных значениями операторов перехода и операторов остановки. Они составляются из элементов базы $S \otimes \Sigma \in \Psi$. В возрастающие моменты дискретного времени эти серии пополняются новыми конфигурациями. Если в некоторый момент времени нет конфигурации, расширяющей начальное данное, то в соответствующую серию добавляется пустая конфигурация. Для обозначения таких развиваемых во времени неограниченных вычислимых серий применяется выражение $(S \otimes \Sigma)^{\infty}$.

Унифицированный формат совместного представления серий, используемых для моделирования эволюций конфигураций, соответствует прямой сумме таких серий. Если z', z'', z''' — фрагменты трех рассматриваемых серий в некоторый момент времени, им соответствует сумма $z' \oplus (z'' \oplus z''')$. Она интегрирует информацию об эволюции конфигураций на данный момент времени. Этой суммы достаточно для моделирования шага эволюции с помощью операторов базиса пространства эволюций конфигураций. Для представления конфигураций, составляющих серию z''', применяются нагруженные бинарные деревья. Вершины таких деревьев размечаются значениями операторов остановки S_{α} для $\alpha \in D(z)$, где z — конфигурация, формируемая в данный момент времени с помощью операторов перехода. Область значений морфизма эволюции конфигураций составляют серии, формируемые в неограниченном времени. Многообразие таких серий определяется выражением $(S \otimes (\Sigma \oplus \Sigma))^{\infty}$.

2. Архитектура интеллектуальной системы

Архитектура ИС основана на инвариантах измерения знаний, потока и процесса трансформации знаний. Измерения знаний определяют многомерную структуру многообразия компонентов ИС. Потоки знаний моделируют схемы переноса знаний между компонентами, а процессы трансформации знаний — обработку знаний, выполняемую в отдельных компонентах [1].

2.1. Измерения знаний

Инвариант измерения знаний связан с моделированием структур компонентов ИС. Реализациями этого инварианта являются разные аспекты знаний [7]. Аспекты позволяет группировать знания, применяемые в ИС, в классы знаний с заданными значениями аспектов. Измерения позволяют создавать и исследовать структуры компонентов архитектуры ИС, применяющих знания с близкими общими свойствами [8].

Определение. Пространством измерений называется семейство линейно упорядоченных множеств $\mathfrak{X} = \{X_i \mid i = 1, ..., n\}$.

Элементы $\mathfrak{X}$ называются измерениями знаний. Множество значений всякого измерения $X_i = \{\sigma_{i,j} \mid j = 1, ..., m_i\}$ упорядочено в порядке возрастания значений индекса j.

С измерениями связаны вычислимые отображения $\chi_i : A \to 2^{X_i}$. Здесь A — это семейство всех фрагментов знаний, применяемых в модели ИС. Если $z \in A$ и $\sigma \in \chi_i(z)$, то $z \in A$ соответствует значению σ в измерении X_i.

Пространством компонентов ИС в пространстве измерений $\mathfrak{X} = \{X_i \mid i = 1, ..., n\}$ называется $\Xi = X_1 \times ... \times X_n$. Элементы последнего множества — это наборы с фиксированными значениями применяемых измерений. Каждый набор из Ξ определяет компонент структуры ИС. Атрибутами компонентов являются формализмы знаний, применяемые для знаний, конструируемых и размещаемых в таких компонентах. К компонентам также относятся процессы обработки знаний, моделирующие жизненные циклы ИС.

К компоненту структуры ИС, определяемому набором $(\sigma_1, ..., \sigma_n) \in \Xi$, относятся все фрагменты знаний в ИС, которые соответствуют значениям измерений $\sigma_1 \in X_1$, ..., $\sigma_n \in X_n$.

Цели разработки и исследования отдельных измерений являются многопредметными. Они связаны с независимыми характеристиками знаний, позволяющими совместно использовать атрибуты знаний, существенные в разных предметных областях. Полнота и независимость пространств измерений знаний в общем случае являются содержательными, поскольку относятся к слабо формализуемым представлениям о знаниях.

Примерами измерений являются: абстрактность, структурированность, уровень и время [8, 9]. Для них можно предложить следующие упорядоченные наборы значений: (*поверхностный*, *алгоритмический*, *аналитический*), (*целостный образ*, *слабо структурированный*, *атомарный*), (*междисциплинарный*, *профессиональный*, *исследовательский*), (*прошлое*, *настоящее*, *будущее*).

Модель всякой ИС конструируется с использованием фрагмента пространства измерений. Неиспользуемые элементы пространства компонентов считаются несущественными для разрабатываемой модели системы. Количество применяемых измерений определяет размерность модели.

Определение. Система значений измерения X_i называется независимой, если для каждого $\sigma \in X_i$ выполняется условие $\bigcap_{\sigma \in \chi_i(z)} (\chi_i(z)) = \{\sigma\}$.

Наборы $(\alpha_1, ..., \alpha_n)$, $(\beta_1, ..., \beta_n) \in \Xi$ называются соседними, если они различаются ровно в одном элементе. Значения этого элемента в наборах являются соседними в упорядоченном представлении значений соответствующего измерения. Отношение соседства наборов обозначается символом $\ll$. Диаграмма этого отношения на Ξ определяет специальную структуру пространства компонентов ИС. Она применяется для моделирования жизненных циклов ИС в форме по-

следовательностей этапов, реализуемых в отдельных компонентах и продолжаемых этапами в соседних с ними компонентах структуры ИС.

Пустой набор значений измерений (□, ..., □) соответствует внешнему окружению ИС. В нем моделируются знания и процессы их обработки, реализуемые вне такой системы. В этом компоненте формируется содержание, представляемое интеллектуальными ресурсами и процессами в форматах, принятых среди специалистов. Информационная система называется автономной системой, если для нее не выполняются переносы знаний из внешнего окружения.

Абстрактной архитектурой (архитектурой) ИС называется всякое семейство $\mathrm{K} \subseteq \Xi$, элементы которого применяются в формализованных описаниях потоков и процессов обработки знаний.

Определение. Архитектура $\mathrm{K} \subseteq \Xi$ называется связной, если

$$\forall c', c'' \in \mathrm{K}\ \exists c_1, ..., c_n \in$$
$$\in \mathrm{K}\left(c_1 = c' \,\&\, c_n = c'' \,\&\, \forall i = 1, ..., n-1\left(c_i \ll c_{i+1}\right)\right).$$

Последовательность $c_1, ..., c_n$ из последнего определения образует путь в K. Он соответствует системе локальных переходов между соседними компонентами в конкретных измерениях. Перемещения составляют основу моделирования потоков знаний между компонентами архитектуры ИС [8, 9]. Потоки позволяют связывать этапы процессов обработки знаний. Этапы реализуются в отдельных компонентах ИС. Результаты выполнения этапов формируются и хранятся в области памяти этих компонентов. Формат представления результата основан на алгебраических структурах синтезируемых знаний. Для пространств семантических иерархий таким форматом являются ПСП конфигураций. Переход к следующему этапу потока инициируется переносом знания, синтезированного в текущем компоненте. Морфизмы переноса знаний между компонентами архитектуры реализуют перевод представлений знаний из формата одного формализма знаний в формат другого формализма знаний. В общем случае морфизм не является тождественным, но сохраняет необходимые свойства переносимых знаний.

Трансформация знания при его переносе между компонентами определяется отличиями свойств знаний из разных компонентов архитектуры. Это связано с несовпадением схем структурного представления фрагментов содержания областей знаний, применяемых в разных компонентах ИС.

Пусть $c', c'' \in \mathrm{K}$ — это соседние компоненты ИС, которым соответствуют формализмы $\mathfrak{I}_1 = \left(D_{\mathfrak{I}_1}, M_{\mathfrak{I}_1}, \circ, \leqslant\right)$ и $\mathfrak{I}_2 = \left(D_{\mathfrak{I}_2}, M_{\mathfrak{I}_2}, \circ, \leqslant\right)$. Перенос знаний из c' в c'' реализуется как преобразование знаний для $\mathfrak{I}_1$ в знания для $\mathfrak{I}_2$. Если знание из одного формализма не имеет аналогов в другом формализме, то значением морфизма переноса знаний является Λ. Для общего случая формализмов знаний морфизмы переноса уточняются следующим определением.

Определение. Морфизм $h : D_{\mathfrak{I}_1} \to D_{\mathfrak{I}_2}$ называется морфизмом переноса знаний из $\mathfrak{I}_1$ в $\mathfrak{I}_2$, если

$$\forall z \in D_{\mathfrak{I}_1}\left(h(z) \neq \Lambda \to (\exists \sigma \in \{0, 1\}(h((z)_\sigma) =\right.$$
$$= \Lambda \,\&\, h(z) = h((z)_{\bar{\sigma}})) \vee (h((z)_0) \neq \Lambda \,\&\, h((z)_1) \neq \Lambda) \,\&$$
$$\left.\&\, h(z) = h((z)_0) \oplus h((z)_1))\right).$$

Пусть $DA(z)$ — алгебраическая структура фрагмента знания в некотором формализме. Тогда сечением $DA(z)$ называется множество $L \subseteq DA(z)$ вершин этой структуры, для которых выполнены условия префиксности и полноты (всякий путь из корня в лист содержит вершину из L). Если $h : D_{\mathfrak{I}_1} \to D_{\mathfrak{I}_2}$ — морфизм переноса знаний, то выполняется соотношение:

$$\forall z \in D_{\mathfrak{I}_1}\left(h(z) \neq \Lambda \to \exists L =\right.$$
$$= \{\alpha_1, ..., \alpha_k\}(z = \Sigma((z)_{\alpha_1}, ..., (z)_{\alpha_k}) \,\&\, h(z) =$$
$$\left.= \Sigma((h(z))_{\alpha_1}, ..., (h(z))_{\alpha_k}))\right).$$

Здесь $L = \{\alpha_1, ..., \alpha_k\}$ — сечение алгебраической структуры фрагмента z, упорядоченное лексикографически, а $\Sigma\left((z)_{\alpha_1}, ..., (z)_{\alpha_k}\right)$ — алгебраическая структура z, конструируемая из фрагментов $(z)_{\alpha_1}, ..., (z)_{\alpha_k}$. То есть преобразование, реализуемое произвольным морфизмом переноса знаний в формализмах, заключается в такой замене непустых фрагментов знаний одного формализма в непустые фрагменты знаний другого формализма, для которого сохраняются алгебраические структуры знаний, составляемых из этих фрагментов.

2.2. Диаграммы процессов в интеллектуальных системах

Процессами в ИС моделируются схемы синтеза структурных представлений знаний. Процессы реализуется в компонентах таких систем и являются аналогами эволюций знаний в формализмах знаний. Специальный формат структур формализованных описаний процессов определяют диаграммы процессов. Последние подобны диаграммам морфизмов в теории категорий [10].

Диаграммы представляются нагруженными ориентированными графами. Вершины диаграмм размечаются классами и базами морфизмов. Разметками ребер являются элементы управления процессами. Последовательное конструирование диаграмм связано с операциями гомоморфного расширения общих диаграмм. Применение таких операций позволяет преобразовывать общие абстрактные диаграммы в диаграммы конкретных процессов, реализуемых в заданных формализмах представления знаний и форматах структур памяти компонентов ИС.

Начальные диаграммы морфизмов задаются последовательностями морфизмов. Вершины таких диаграмм размечены классами морфизмов. Для пары классов морфизмов, приписанных вершинам, соединяемым ориентированным ребром, должно выполняться условие вложения области (базы) значений морфизмов класса вершины начала ребра в базу определения морфизмов класса вершины — конца ребра. В диаграммах также обозначаются начальная и конечная вершины.

Расширение диаграмм морфизмов в диаграммы этапов процессов получается однократным разбиением ребер. При этом вершины разбиения ребра размечаются объединениями областей значения и определения классов морфизмов, связанных ребром. Расширение полученных диаграмм до диаграмм управления процессами получается дополнительной разметкой ребер, связывающих соседние вершины. Элементами разметок являются параметры статуса ребер и предикаты условий прохождения ребра процессом. Примеры операций гомоморфного расширения диаграмм приведены в работе [8].

Предикат условия ребра представляется специальной формулой. В ней используются алгебраические структуры элементов баз начальной или конечной вершины ребра, а также онтологии компонента.

Пример последовательного развития диаграммы процессов, связанного с рассмотренными типами форм диаграмм, приведен на рис. 3.

К операциям расширения диаграмм относится также замена класса морфизмов диаграммы на диаграмму процесса, реализующего морфизмы соответствующего класса применением комбинаций разных классов морфизмов.

Приведем формализацию понятия диаграммы процесса. Определим два бесконечных перечислимых дизъюнктных множества: $VB = \{a_1, ..., a_i, ...\}$ и $VF = \{b_1, ..., b_i, ...\}$. Для VB и VF определим вычислимые отображения разметки вершин $\varphi : VB \to \Psi$ и $\psi : VF \to \Phi$, которые связывают элементы множества VB с базами морфизмов, а элементы VF — с классами морфизмов.

Обозначим как $\mathfrak{P} = \{P_1, ..., P_i, ...\}$ бесконечное перечислимое множество формул для предикатов условий ребер диаграмм.

Определение. Диаграмма процесса (диаграмма) — это четверка $\mathsf{D} = (V, U, I, S)$, где (V, U) — ориентированный граф и $V \subseteq VB \cup VF$ — множество вершин, а U — множество размеченных ребер ($U \subseteq VB \times VF \times \mathfrak{P} \cup VF \times VB \times \mathfrak{P}$), $I, S \in (VB \cap V)$ — начальная и заключительная вершины диаграммы соответственно.

Ребра $(a, b, p) \in VB \times VF \times \mathfrak{P}$ $((b, a, p) \in VF \times VB \times \mathfrak{P})$ соединяют вершины, связанные областями определения морфизмов, с вершинами, связанными с классами морфизмов (вершинами, связанными с классами морфизмов с вершинами, связанными с областями значения морфизмов). Предикат ребра p определяет условие прохождения этого ребра процессом. При этом, если $(a, b, p) \in VB \times VF \times \mathfrak{P}$ $((b, a, p) \in VF \times VB \times \mathfrak{P})$, то $\psi(a)$ вложено в область определения морфизмов из $\varphi(b)$ (область значений морфизмов из $\varphi(b)$ вложена в $\psi(a)$).

Пусть DS — это множество всех диаграмм процессов для заданных множеств VB, VF, C, а также отображений Ψ и Φ. Для этого множества определим однозначную вычислимую нумерацию $\mu : \mathrm{N} \to DS$. В конкретных моделях ИС такая нумерация состоит в назначении номеров новым диаграммам в процессе их добавления в модель. Нумерация μ применяется для конструирования гомоморфных расширений диаграмм, в которых вершины диаграмм $\nu \in VF$ заменяются на диаграммы из DS. Такие расширения можно задавать указанием μ номеров диаграмм, заменяющих вершины.

Всякая диаграмма определяет класс морфизмов из Φ. Области определения и значения этих морфизмов соответствуют базам, поставленным в соответствие вершинам I, $S \in DS$. При подстановке диаграммы вместо вершины класса морфизмов в диаграмме вершины I и S отождествляются с входной и выходной вершинами заменяемой вершины соответственно. Корректность выполняемой замены связана с выполнением отношения вложения классов заменяющих морфизмов и баз в замещаемые ими классы.

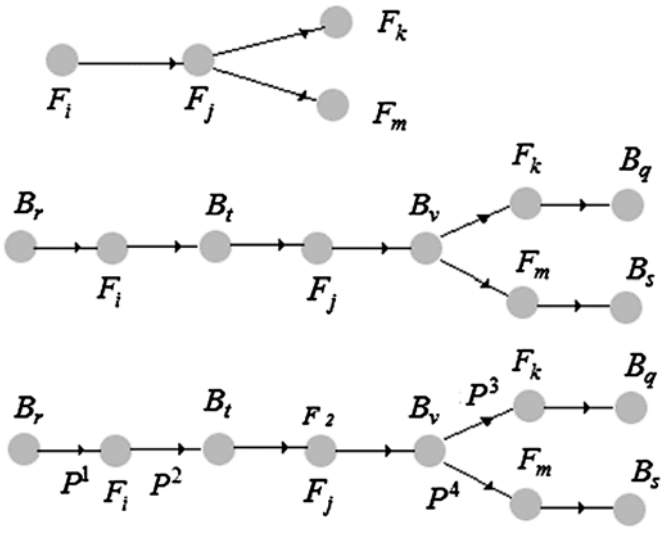


Рис. 3. Формы описаний диаграмм процессов:

F_i, F_j, F_k, F_m — классы морфизмов; B_r, B_t, B_v, B_q, B_s — базы; P^1, P^2, P^3, P^4 — условия ребер

2.3. Потоки знаний в интеллектуальных системах

Понятие потока знаний в ИС связано с моделированием жизненного цикла времени существования такой системы. Потоки разбиваются на этапы, относящиеся к отдельным компонентам архитектуры. Реализация этапа начинается с переноса знания из компонента предыдущего этапа, преобразуемого в формат знаний нового компонента с помощью подходящего морфизма. Перенесенное знание обрабатывается в текущем компоненте потока в соответствии с выбранной диаграммой процесса. Переносы знаний выполнятся между соседними компонентами архитектуры ИС.

Вершины диаграммы потоков представляются парами (α, D), где α — компонент ИС, а D — диаграмма в α. Разметку ребра диаграммы потока $((\alpha, \mathsf{D}_\alpha), (\beta, \mathsf{D}_\beta))$, где $\mathsf{D}_\alpha = (V_\alpha, U_\alpha, I_\alpha, S_\alpha)$ и $\mathsf{D}_\beta = (V_\beta, U_\beta, I_\beta, S_\beta)$, образует четверка (P, p', f, p''). Здесь P — условие переноса знания из α в β, f — морфизм переноса знаний для вершин знаний в диаграммах соседних компонентов, а $p' \in S_\alpha$, $p'' \in I_\beta$ — вершины данных диаграмм, между которыми выпол-

няется указанный перенос. Условие P определяется как предикат, представляемый логической формулой. Основными параметрами формулы являются элементы памяти компонента и фрагменты баз морфизмов определения классов морфизмов диаграмм D_α и D_β.

В общих диаграммах потоков (шаблонах потоков) допускаются неопределенные значения вершин и ребер, обозначаемые символом □. Эти элементы уточняются с помощью гомоморфных расширений шаблонов, моделирующих трансформации шаблонов.

Определение. Элементарным потоком в ИС называется последовательность чередующихся вершин и разметок ребер, соединяющих соседние элементы последовательности

$$W = (\alpha_1, \mathsf{D}_1)(P_1, x_1, f_1, y_1), ..., (\alpha_{k-1}, \mathsf{D}_{k-1}), (P_{k-1}, x_{k-1}, f_{k-1}, y_{k-1}), (\alpha_k, \mathsf{D}_k),$$

в которой любые два компонента α_i и α_{i+1}, $i = 1, ..., k-1$, являются соседними.

Определение. Потоком знаний в ИС называется конечное семейство простых потоков $KF = \{W_1, ..., W_k\}$.

Каждый поток задает систему перемещений знаний между компонентами ИС. Для управления потоками удобно использовать диаграммы потоков. Они позволяют моделировать системы элементарных потоков совместно. Пример такой диаграммы приведен на рис. 4.

Здесь изображен поток, составленный элементарными потоками:

$$W_1 = (\alpha_1, \mathsf{D}_1)(P_5, x_5, f_5, y_5), (\alpha_2, \mathsf{D}_2), (P_6, x_6, f_6, x_6), (\alpha_3, \mathsf{D}_3), (P_7, x_7, f_7, x_7), (\alpha_6, \mathsf{D}_6),$$

$$W_2 = (\alpha_4, \mathsf{D}_4)(P_1, x_1, f_1, y_1), (\alpha_5, \mathsf{D}_5), (P_2, x_2, f_2, y_2), (\alpha_2, \mathsf{D}_2), (P_3, x_3, f_3, y_3), (\alpha_7, \mathsf{D}_7), (P_4, x_4, f_4, y_4), (\alpha_8, \mathsf{D}_8).$$

Потоки W_1 и W_2 использует общую диаграмму D_2 в компоненте α_2. В вершины y_2 и y_5 этой диаграммы переносятся знания, размещенные в позициях x_2 и x_5 диаграмм D_5 и D_1. Применение этих знаний в диаграмме D_2 связано с инициацией процессов,

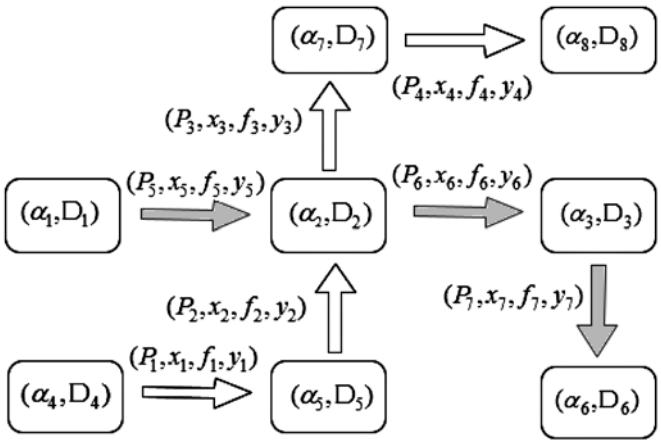


Рис. 4. Фрагмент диаграммы потока ИС

основанных на данных, переносимых из других компонентов.

Описание всякого потока, дополненное условиями активации и завершения, представляется в формате семантической иерархии и сохраняется в области памяти подходящего компонента ИС [8].

2.4. Структурная организация памяти компонентов архитектуры

Память компонента архитектуры ИС имеет формат алгебраической структуры знания. Содержание памяти изменяется в дискретном времени и представляется соответствующими структурами знаний. Унифицированная абстрактная структура памяти компонента включает области онтологии и области реализации процессов компонента. В онтологии аккумулируются знания, актуальные для моделирования жизненных циклов ИС в компоненте. Форматом представления онтологии как конфигурации пространства семантических иерархий является неупорядоченная серия простых знаний. Область памяти процессов, реализуемых в компоненте, составляют последовательные серии. Ими моделируются аналоги эволюций конфигураций [6]. Семантическая структура содержания памяти компонента в фиксированный момент времени называется состоянием компонента в соответствующий момент.

Абстрактный процесс в формате выбранного формализма знаний составляют перечислимые последовательности фрагментов знаний $W_0 = \{z_0^0, ..., z_0^i, ...\}$ и $W_1 = \{z_1^0, ..., z_1^i, ...\}$, где W_0 называется начальным данным процесса, а W_1 — процессом синтеза или эволюцией знаний. При этом $z_1^0 = \Lambda$. Последовательности W_0 и W_1 формируются в дискретном времени, представляемом верхними индексами элементов последовательностей. Начальное данное процесса порождается процедурой пересчета ее элементов, появляющихся в возрастающие моменты времени. Если в очередной момент времени фрагмент знания не появляется в пересчете начального данного, то в W_0 добавляется конфигурация Λ. Эволюция знаний генерируется с использованием оператора перехода $T : M \times M^* \to M$, который вычисляет каждый элемент z_1^{i+1}, $i = 0, 1, ...$, как $T\left(z_1^i \times \{z_0^0, ..., z_0^i\}\right)$. Распознавание фрагментов элементов процесса, составляющих элементы результата этого процесса, выполняется операторами остановки $S_\alpha : M \times M^* \to \{0, 1\}$, $\alpha \in I$. При этом, если $S_\alpha\left(z_1^i \times \{z_0^0, ..., z_0^i\}\right) = 1$, то $\left(z_1^i\right)_\alpha$ входит в результат процесса в момент i. Результат процесса в компоненте $\alpha \in I$ составляют элементы последовательности $\left(z_1^0\right)_\alpha, ..., \left(z_1^i\right)_\alpha, ...$, выбираемые для тех моментов времени, для которых $S_\alpha\left(z_1^i \times \{z_0^0, ..., z_0^i\}\right) = 1$.

Пример структуры памяти для моделирования процесса в формате ПСП конфигураций приведен на рис. 5. Ее составляют развиваемые во времени серии начального данного, эволюции конфигураций и значений операторов остановки.

Вершины 000*1 являются корнями ПСП конфигураций из начального данного процесса $\{z_0^i \mid i = 0, 1, ...\}$. Вершины 0110*1 приведенной структуры — это корни ПСП конфигураций, составляющих эволюцию

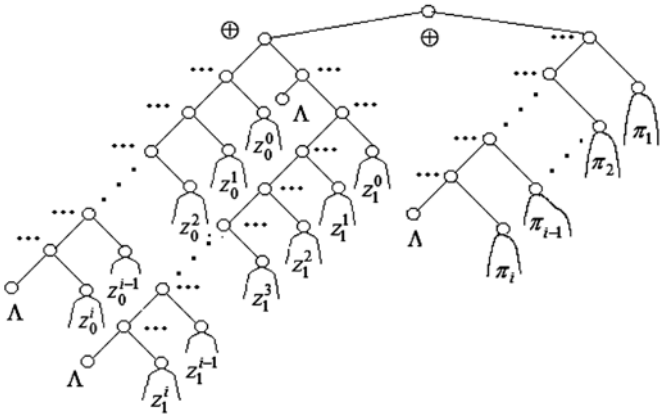


Рис. 5. Структура области памяти реализации процесса

конфигураций. Начальное данное и сама эволюция представляются расширяющимися во времени последовательными сериями с корнями 00 и 011. Эта структура суммируется с последовательной серией ПСП конфигураций, в которой сохраняются значения операторов остановки S_α, $\alpha \in I$. Корни ПСП элементов последней серии образуют вершины вида 10*1. Элемент π_i представляет конфигурацию z_1^i. При этом выполняются условия $D(\pi_i) = D\left(z_1^i\right)$ и $\forall \alpha \in D(\pi_i)\left([\pi_i]^\alpha = S_\alpha\left(z_1^i\right)\right)$. Здесь $[\pi_i]^\alpha \in \{0, 1\}$ — разметка вершины в полном арифметическом представлении фрагмента конфигурации z_1^i [6]. Вершины таких представлений конфигураций размечаются числами — номерами отношений и элементарных знаний.

Для моделирования реализаций диаграмм процессов удобно использовать специальный формат представления эволюций конфигураций. В них конфигурации, составляющие эволюции, содержат значения в вершинах данных диаграмм, полученные к соответствующему моменту времени. Результаты применения морфизмов ставятся в соответствие вершинам баз значений морфизмов на диаграмме. Начальное данное эволюции составляет знание, помещаемое в начальной вершине диаграммы процесса. Оно извлекается из онтологии компонента или переносится потоком из другого компонента.

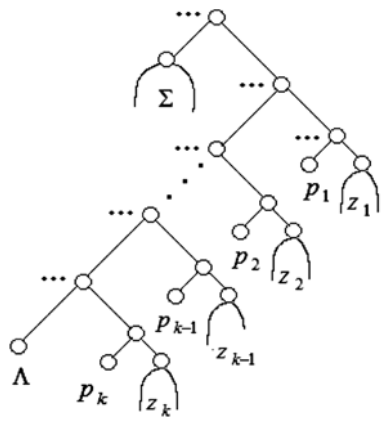


Рис. 6. Структура элемента процесса в эволюции знаний

Реализация диаграммы как эволюции знаний моделируется последовательной серией серий значений, размещаемых в вершинах диаграмм, размеченных базами морфизмов, при пошаговом применении морфизмов диаграммы. Пример соответствующей структуры приведен на рис. 6. Здесь p_1, ..., p_k — имена вершин баз морфизмов диаграммы, а z_1, ..., z_k — ПСП конфигураций, представляющих значения, размещенные в указанных вершинах в заданный момент времени. Фрагмент Σ содержит представление диаграммы моделируемого процесса в виде семантической иерархии, а также ПСП конфигураций, переносимых потоками знаний из других компонентов в качестве начальных данных процесса.

3. Инварианты управления интеллектуальными системами

Абстрактная модель управления ИС основывается на кибернетическом подходе к иерархии управления и автоматического регулирования [11]. В модели ИС принципы управления системой реализуются с использованием формализованных инвариантов моделей формализма представления знаний и архитектуры таких систем.

Основу модели управления ИС составляют агенты управления (далее агенты). Они являются средством реализации активных форм субъектного существования ИС в окружающем мире. Агенты моделируют изменение содержания отдельных систем в дискретном времени. Образом такого содержания в каждый момент времени является состояние системы. Его составляют состояния памяти компонентов архитектуры ИС. При этом агенты реализуют пошаговые преобразования состояний ИС, основанные на формализованных описаниях жизненных циклов таких систем.

Уточнение понятия агента ИС может быть выполнено на основе общей теории систем как аналога понятия автомата [12]. Общее определение агента $\mathfrak{a}$ реализуется как вычислимое отображение $f_\mathfrak{a} : \mathfrak{M} \times \mathfrak{Q} \to \mathfrak{M} \times \mathfrak{Q} \times \mathfrak{O}$. Здесь $\mathfrak{M}$, $\mathfrak{Q}$ и $\mathfrak{O}$ — перечислимые множества структурированных представлений знаний в областях памяти компонентов, состояний агента и дополнительных операций, выполняемых агентом. Отображение $f_\mathfrak{a}$ задает функционирование агента за один момент времени. Область определения отображения $(\mathfrak{M} \times \mathfrak{Q})$ составляют значение содержания области памяти ИС и состояние агента. Область значений $f_\mathfrak{a}$ (множество $\mathfrak{M} \times \mathfrak{Q} \times \mathfrak{O}$) составляют значение содержания памяти ИС и состояние агента в следующий момент времени, а также дополнительные действия, необходимые для реализации специальных аспектов функционирования агентов. Примерами таких действий являются активации и деактивации агентов, передача содержания фрагмента области памяти во внешнюю среду. Многообразие общих видов применяемых агентов составляет иерархию классов. Фрагмент такой иерархии приведен на рис. 7.

Основу иерархии образуют классы всех агентов $(\mathfrak{A})$, агентов, управляющих агентами $(\mathfrak{AA})$, и аген-

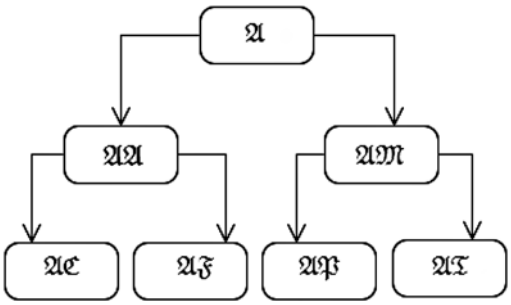


Рис. 7. Иерархия типов агентов управления ИС

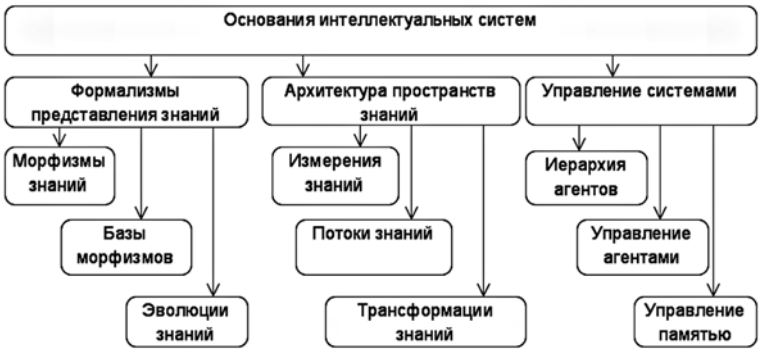


Рис. 8. Базовые инварианты ИС

тов управления памятью системы ($\mathfrak{AM}$). Класс $\mathfrak{AA}$ распадается на классы агентов, управляющих агентами управления агентами ($\mathfrak{AE}$), и агентов, управляющих агентами управления памятью (($\mathfrak{AF}$).). Класс $\mathfrak{AM}$ составляют классы агентов управления переносами знаний ($\mathfrak{AT}$) и реализацией процессов ($\mathfrak{AP}$).

Заключение

Формализация понятия ИС состоит в уточнении базовых элементов и порождающих принципов таких систем. Трудности уточнения связаны с существованием разнообразных представлений об ИС. Для них требуется сбалансированная формализация понятий из разных областей знаний. Формализмы знаний составляют один из аспектов ИС. Они позволяют исследовать структурные и функциональные свойства абстрактных знаний. Общность модели формализма и получаемых в ней результатов вызывает трудности применения формализмов знаний для прикладных ИС. В эффективности таких систем существенное значение имеет моделирование форматов знаний и процессов решения профессиональных задач, применяемых специалистами. Поэтому продуктивные приложения формализмов знаний следует искать в задачах построения ИС сложной многоуровневой структуры, моделирующих развитые семейства процессов мышления и использующих разные форматы представления знаний. Такие системы обеспечивают полное моделирование интеллектуальной деятельности специалистов.

Приведенную в статье математическую модель ИС составляют формализмы знаний, многомерная архитектура и агенты управления. Применяемая система инвариантов и порождающих принципов модели согласована с основными положениями кибернетики [11].

Рассмотренные в работе подходы к абстрактному моделированию ИС основаны на аналогах понятий и конструкций из разных областей математики. В них находят отражение результаты исследования и эмпирический опыт в областях, связанных с применением интеллектуальных объектов и процессов, опыт моделирования представлений знаний и процессов мышления. Многообразие базовых инвариантов модели ИС приведено на рис. 8.

Пусть [$\mathfrak{I}$] — модель формализма представления знаний. Ее основными частями являются базы морфизмов ($\mathfrak{Mem}$), классы морфизмов ($\mathfrak{Mor}$) и эволюции знаний ($\mathfrak{Evol}$); [Ξ] — структурная модель интеллектуальной системы, составленная разделами измерений знаний ($\mathfrak{Space}$) многообразий потоков знаний ($\mathfrak{Flows}$) и диаграмм трансформации знаний в компонентах систем ($\mathfrak{Transforms}$); [$\mathfrak{A}$] — модель управления существованием системы, частями которой являются иерархия типов агентов ($\mathfrak{Agents}$), модель управления агентами ($\mathfrak{Control}$) и модель трансформации состояний интеллектуальной системы ($\mathfrak{Memory}$).

Определение. Тройка $\mathfrak{B}$ = ([$\mathfrak{I}$], [Ξ], [$\mathfrak{A}$]) называется абстрактной интеллектуальной системой.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 20-01-00289.

Список литературы

1. **Burgin M.** Theory of Knowledge: Structures and Processes. World Scientific, 2017. — 948 p.

2. **Bloom B. S., Engelhart M. D., Furst E. J., Hill W. H., Krathwohl D. R.** Taxonomy of educational objectives: The classification Taxonomy of educational goals. Handbook 1: Cognitive domain. New York: David McKay, 1956. — 207 p.

3. **Stanovich K. E.** Rationality and the reactive mind. New York: Oxford Univ. Press, 2011. — 328 p.

4. **Костенко К. И.** Операции когнитивного синтеза формализованных знаний // Программная инженерия. — 2018. — Т. 9, № 4. — С. 174—184.

5. **Костенко К. И.** Моделирование замыканий онтологий в формализмах семантических иерархий // Программная инженерия. — 2020. — Т. 11, № 6. − С. 349—360.

6. **Костенко К. И.** Об алгоритмических свойствах пространств эволюций знаний // Экологический вестник научных центров Черноморского экономического сотрудничества. 2007. — Т. 4, № 4. — С. 14—20.

7. **Zarri G. P.** Functional and Semantic Roles in a High-Level Knowledge Representation Language// Artificial Intelligence Review 51. Springer Verlag. — 2017. — P. 537—575.

8. **Kostenko K.** Knowledge flows processes at multidimensional intelligent systems // Russian Advances in Artificial Intelligence (RCAI 2020), Moscow, Russia, October 10—16, 2020, CEUR Workshops Proceedings. — 2020. — Vol. 2648. — P. 74—84.

9. **Kostenko K.** Conceptual Modeling of Intelligent Systems Components and Functionality// Data Analytics and Management in Data Intensive Domains 2020 (DAMDID\RCDL2020), Voronezh, Russia, October 13—16, 2020. CEUR Workshop Proceedings. — 2020 — Vol. 2790. — P. 4—18.

10. **Ковалев С. П.** Теоретико-категорный подход к проектированию программных систем // Фундаментальная и прикладная математика. — 2014. — Т. 19, № 3. — С. 111—170.

11. **Эшби У. Р.** Введение в кибернетику. — М.: Иностранная литература, 1959. — 432 с.

12. **Месарович М., Такахара Я.** Общая теория систем: математические основы. — М.: Мир, 1978. — 312 с.

Core Invariants of the Mathematical Model of an Intelligent System

K. I. Kostenko, kostenko@kubsu.ru, Kuban State University, Krasnodar, 350040, Russian Federation

Corresponding author:

Kostenko Konstantin I., Associate Professor, kostenko@kubsu.ru, Kuban State University, Krasnodar, 350040, Russian Federation
E-mail: kostenko@kubsu.ru

Received on December 11, 2020
Accepted on January 15, 2021

A holistic description of a universal mathematical model for the concept of an intelligent system is given. It is based on formalized invariants associated with the processes of creating and applying such systems. The core of the model is formed by consistent descriptions for sections of knowledge formalisms, components of multidimensional architecture and knowledge flows processes within it, as well as cybernetic hierarchical multi agents systems that control the intelligent systems' subjective existence. The fundamental invariants of the knowledge presentation and processing are directly implemented by these main sections' basic elements. Invariants form unified set of intelligent systems' general attributes. This set allows carrying out comprehensive formal modelling of the intelligence. These invariants are associated with knowledge aspects. They are developed and used at knowledge areas that deal with exploring the memory structural organization and thinking processes models. The tools for transforming the proposed abstract model into the models of specific intelligent systems are morphisms of homomorphic expansion. These morphisms concretize the content of the main structural and functional elements of the intelligent system fundamental model. At the same time, the varieties of entities implemented by model elements are narrowed to the families of objects that make up applied intelligent systems. These systems inherit the properties of fundamental model common elements. Intermediate models of the processes of converting the original model into applied ones allow studying these models' properties by mathematical tools. Intermediate models form the basis for the subsequent development of the technology of creating and applying multilevel intelligent systems.

Keywords: *intelligent system, knowledge representation formalism, knowledge dimension, knowledge flow process, ontology, knowledge synthesis, knowledge morphism, cognitive goal, intelligent agent*

Acknowledgements:

The reported study was funded by RFBR. Grant project number № 20-01-00289.

For citation:

Kostenko K. I. Core Invariants of the Mathematical Model of an Intelligent System, *Programmnaya Ingeneria,* 2021, vol. 12, no. 3, pp. 157—168

DOI: 10.17587/prin.12.157-168

References

1. **Burgin M.** *Theory of Knowledge: Structures and Processes*, World Scientific, 2017, 948 p.
2. **Bloom B. S., Engelhart M. D., Furst E. J., Hill W. H., Krathwohl D. R.** *Taxonomy of educational objectives: The classification Taxonomy of educational goals.* Handbook 1: Cognitive domain, New York, David McKay, 1956, 207 p.
3. **Stanovich K. E.** *Rationality and the reactive mind*, New York, Oxford Univ. Press, 2010, 328 p.
4. **Kostenko K. I.** Operations of Formalized Knowledge Cognitive Synthesis, *Programmnaya Ingeneria,* 2018, vol. 9, no. 4, pp. 174—184 (in Russian).
5. **Kostenko K. I.** Ontologies' Closures Modeling by Formalisms of Semantic Hierarchies, *Programmnaya Ingeneria,* 2020, vol. 11, no. 6, pp. 349—360 (in Russian).
6. **Kostenko K. I.** Algorithmic Properties of Knowledge Evolutions Spaces, *Ecologocheskij vestnik nauchnyx centrov Chernomorskogo economicheckogo sotrudnichestva*, 2007, vol. 4, no. 4, pp. 14—20 (in Russian).
7. **Zarri G. P.** Functional and Semantic Roles in a High-Level Knowledge Representation Language, *Artificial Intelligence Review 51*, Springer Verlag, 2017, pp. 537—575.
8. **Kostenko K.** Knowledge flows processes at multidimensional intelligent systems, *Russian Advances in Artificial Intelligence (RCAI 2020),* Moscow, Russia, October 10-16, 2020, CEUR Workshops Proceedings, 2020, vol. 2648, pp. 74—84.
9. **Kostenko K.** Conceptual Modeling of Intelligent Systems Components and Functionality, *Data Analytics and Management in Data Intensive Domains 2020 (DAMDID\RCDL2020),* Voronezh, Russia, October 13—16, 2020, CEUR Workshop Proceedings, 2020, vol. 2790, pp. 4—18.
10. **Kovalyov S. P.** Category-theoretic approach to software systems design, *Fundamentalnaya i prikladnaya matematika*, 2014, vol. 19, no. 3, pp. 111—170 (in Russian).
11. **Ashby W. R.** *An Introduction to Cybernetics,* London, Chapman & Hall, 1956, 295 p.
12. **Mesarovich M. D., Takahara Y.** *General Systems Theory: mathematical foundations,* Academic Press, 1975, 322 p.

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева*. Корректор *Н. В. Яшина*

Сдано в набор 10.03.2021 г. Подписано в печать 28.04.2021 г. Формат 60×88 1/8. Заказ PI321
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**

Рисунки к статье **С. И. Паринова**

«ТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ КОНТЕКСТОВ ЦИТИРОВАНИЙ ИЗ НАУЧНЫХ ПУБЛИКАЦИЙ: СТРУКТУРА НАУЧНОГО ПОТРЕБЛЕНИЯ АВТОРА»

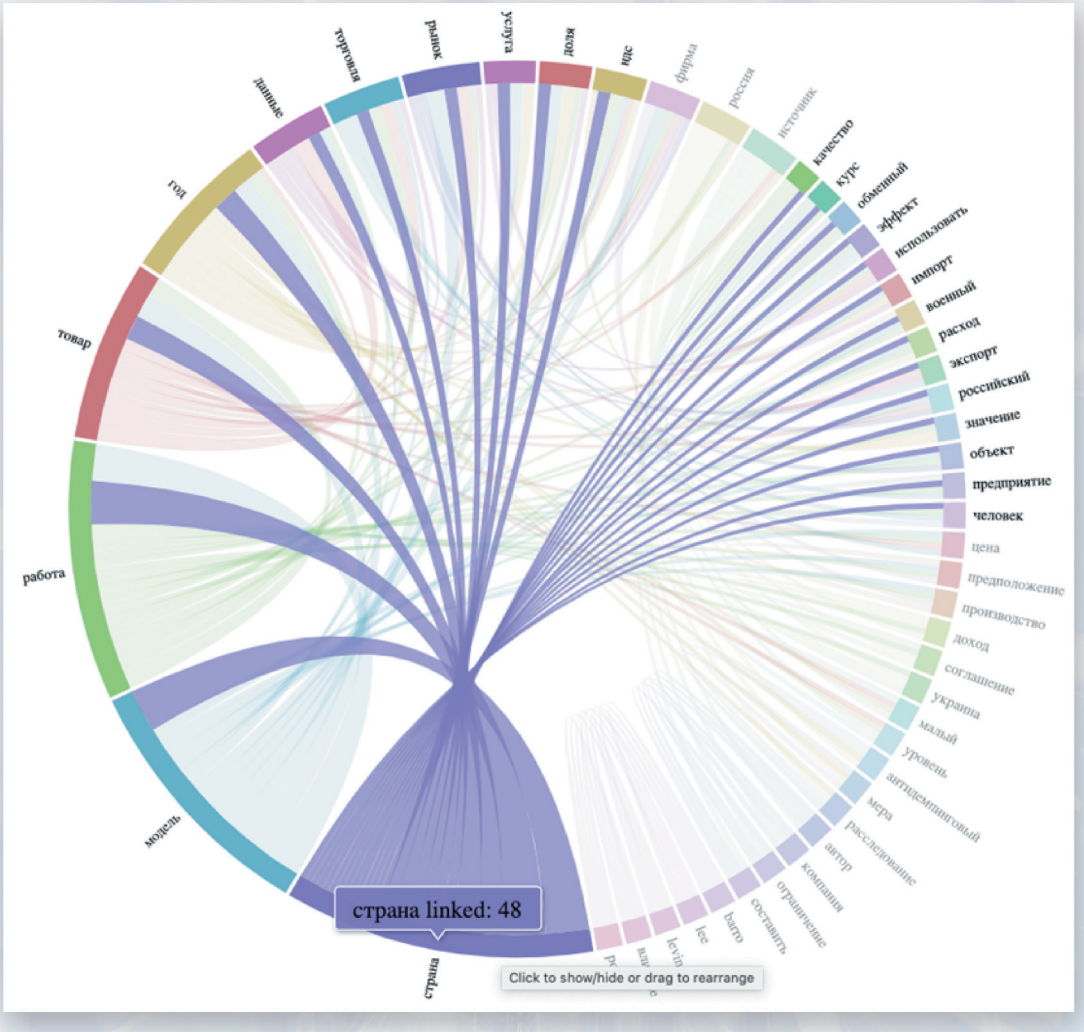


Рис. 1. Пример тематической структуры в виде диаграммы Chord

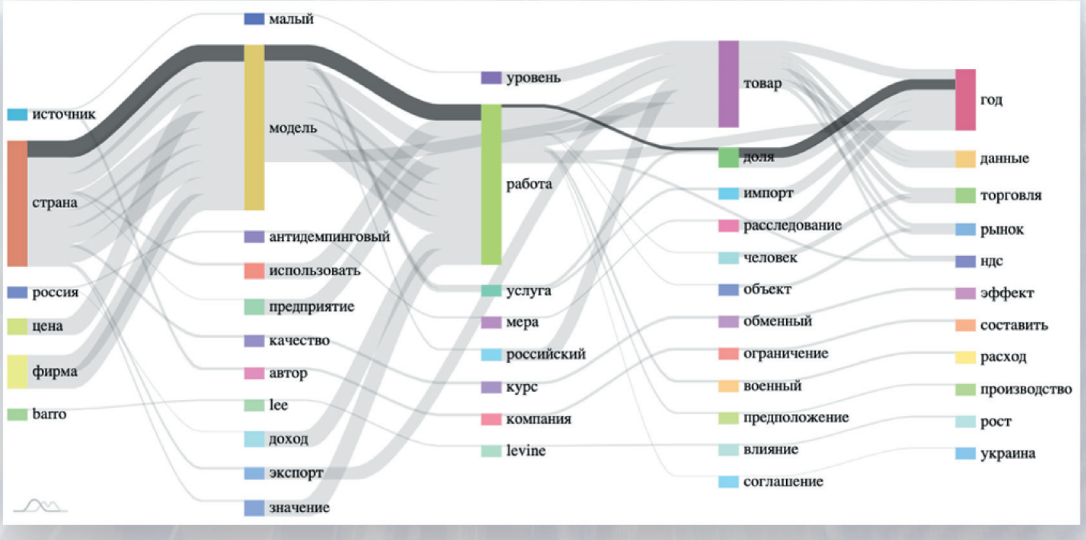


Рис. 3. Пример тематической структуры в виде диаграммы Sankey

Издательство «НОВЫЕ ТЕХНОЛОГИИ»

выпускает *научно-технические журналы*

Теоретический и прикладной научно-технический журнал

ПРОГРАММНАЯ ИНЖЕНЕРИЯ

В журнале освещаются состояние и тенденции развития основных направлений индустрии программного обеспечения, связанных с проектированием, конструированием, архитектурой, обеспечением качества и сопровождением жизненного цикла программного обеспечения, а также рассматриваются достижения в области создания и эксплуатации прикладных программно-информационных систем во всех областях челове ческой деятельности.

Подписной индекс по Объединенному каталогу «Пресса России» – 22765

Ежемесячный теоретический и прикладной научно-технический журнал

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

В журнале освещаются современное состояние, тенденции и перспективы развития основных направлений в области разработки, производства и применения информационных технологий.

Подписной индекс по Объединенному каталогу «Пресса России» – 72656

Междисциплинарный теоретический и прикладной научно-технический журнал

НАНО- и МИКРОСИСТЕМНАЯ ТЕХНИКА

В журнале освещаются современное состояние, тенденции и перспективы развития нано- и микросистемной техники, рассматриваются вопрос разработки и внедрения нанои микросистем в различные области науки, технологии и производства.

Подписной индекс по Объединенному каталогу «Пресса России» – 79493

Ежемесячный теоретический и прикладной научно-технический журнал

МЕХАТРОНИКА, АВТОМАТИЗАЦИЯ, УПРАВЛЕНИЕ

В журнале освещаются достижения в области мехатроники, интегрирующей механику, электронику, автоматику и информатику в целях совершенствования технологий производства и создания техники новых поколений. Рассматриваются актуальные проблемы теории и практики автомати ческого и автоматизированного управления техническими объектами и технологическими процессами в промышленности, энергетике и на транспорте.

Подписной индекс по Объединенному каталогу «Пресса России» – 79492

Научно-практический и учебно-методический журнал

БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

В журнале освещаются достижения и перспективы в области исследований, обеспечения и совершенствования защиты человека от всех видов опасностей производственной и природной среды, их контроля, мониторинга, предотвращения, ликвидации последствий аварий и катастроф, образования в сфере безопасности жизнедеятельности.

Подписной индекс по Объединенному каталогу «Пресса России» – 79963

Адрес редакции журналов для авторов и подписчиков:
107076, Москва, Стромынский пер., 4. Издательство "НОВЫЕ ТЕХНОЛОГИИ".
Тел.: (499) 269-55-10, 269-53-97. Факс: (499) 269-55-10. E-mail: antonov@novtex.ru

ISSN 2220-3397. Программная инженерия. 2021. Том 12. № 3. 113 – 168. Подписной индекс по Объединенному каталогу "Пресса России" – 22765