УДК: 004.6+ 004.4

# Современные промышленные приложения графов знаний

*Апанович З.В (Институт систем информатики СО РАН)*

Графы знаний проделали большой путь эволюции от простого множества RDF-триплет до систем получения новых знаний. Если в прежние годы основным приложением графов знаний считался семантический поиск, то на современном этапе графы знаний проникают во все области промышленного производства. Данная работа представляет обзор новых вариантов графов знаний, таких виртуальные графы знаний, динамические графы знаний и исполняемые графы знаний, применяемые в современном производстве, а также основную область их применения когнитивные цифровые двойники. Также в работе кратко рассмотрены способы генерации графов знаний при помощи больших языковых моделей и повышение качества работы больших языковых моделей за счет применения графов знаний.

**Ключевые слова:** *виртуальный граф знаний, динамический граф знаний, исполняемый граф знаний, когнитивный цифровой двойник, большие языковые модели*

## 1. Введение

Графы знаний (ГЗ) с момента их громкого появления под лозунгом «вещи, а не строки» (“things not strings”) прошли большой путь эволюции. Если первоначально граф знаний понимался просто как RDF-граф, то есть множество триплет в виде (субъект, предикат, объект), то к настоящему моменту графы знаний превратились в системы получения новых знаний [1]. Если в начальные годы основным приложением графов знаний считался семантический поиск, то сейчас в список приложений входят вопросно-ответные системы, рекомендательные системы, системы принятия решений и многие другие. В последние несколько лет графы знаний все чаще используются при разработке цифровых двойников и даже появился специальный термин «когнитивный цифровой двойник», означающий системы, совместно использующие методы семантического моделирования и глубокого обучения.

Одним из факторов такой эволюции графов знаний является то, что графы знаний унаследовали от *Открытых связанных данных* (Linked Open Data) стандарты, входящие в стек технологий Semantic Web [2].

1. Использование глобальной идентификации ресурсов на основе интернационализированных идентификаторов ресурсов (Internationalised Resource Identifier, IRI), обеспечивает глобальную однозначность представления каждого ресурса.
2. Использование Resource Description Framework (RDF) [3] позволяет стандартизировать представление данных в форме триплетов.
3. Использование онтологий для классификации ресурсов позволяет генерацию новых триплет как при помощи логического вывода, так и при помощи методов глубокого обучения на основе текстовых, визуальных и других данных.
4. Использование специализированных систем хранения, таких как RDF-хранилища и графовые базы данных, позволяет использовать наиболее эффективный способ хранения данных. 1
5. Язык запросов SPARQL позволяет не только осуществлять запросы, но и обновлять данные в хранилищах через конечную точку. Федеративные запросы предлагают возможность извлекать данные из нескольких источников данных и анализировать их одновременно, включая возможность объединения данных из общедоступных и частных источников данных, принадлежащих любому количеству различных сторон.
6. Доступ данным при помощи Виртуальных графов знаний, ранее известный как Доступ к данным на основе онтологий (Ontology-Based Data Access, OBDA) позволяет выполнять онтологические запросы над реляционными и NoSQL базами данных.

В настоящее время семантические модели внедряются в различных отраслях Промышленного Интернета вещей для моделирования и управления знаниями предметной области. Их приложения варьируются от управления следующим поколением производства до объяснимого транспорта и энергосбережения в зданиях, от использования семантической интеграции различных датчиков Интернета Вещей (IoT) до автоматизации аналитики созданных данных. Количество разновидностей графов знаний значительно расширяется за счет новых приложений. Ведущие производители такие как Siemens, Bosh, IBM предлагают и внедряют новые вариации ГЗ для промышленных приложений, такие как Динамические графы знаний, Исполняемые графы знаний, Виртуальные графы знаний и др. В данной

работе будет сделан обзор современных графов знаний и их приложений, обусловленных потребностями четвертой и пятой промышленных революций.

## 2. Виртуальные графы знаний для интеграции множественных источников данных

Тенденция цифровизации в обрабатывающей промышленности приводит к огромному росту объема и сложности данных, генерируемых устройствами, участвующими в производственных процессах. Эти данные становятся важным активом для улучшения эффективности производства. Однако раскрытие потенциала этих данных - серьезная проблема для многих организаций. Часто данные находятся в разрозненных хранилищах, которые не связаны между собой физически, но содержат *семантически связанные* данные, возможно с избыточной и противоречивой информацией. Поэтому эффективное использование данных требует *интеграции данных,* которая включает их очистку, дедупликацию и семантическую гомогенизацию. По оценке компании Bosch [4], усилия по интеграции данных составляют примерно 70-80 процентов по сравнению с 20-30 процентами, необходимыми для анализа данных. В последние годы для решения этой проблемы используются методы семантической интеграции данных на основе *виртуальных графов знаний* (ВГЗ) [5].

В подходе ВГЗ онтология моделирует предметную область и определяет общий словарь виртуального графа знаний, который скрывает от пользователя физическую структуру источников данных, а также обогащает данные из источников некоторыми общими знаниями. Онтология связана с конкретными источниками данных через декларативные спецификации данных в терминах отображений (mappings), которые связывают классы и свойства онтологии с конкретными представлениями данных в разрозненных источниках. Отображения указывают, каким образом превратить значения, хранящиеся в разрозненных реляционных таблицах в триплеты единого графа знаний.

Онтология вместе с отображениями представляет собой виртуальный граф знаний, к которому можно осуществлять запросы SPARQL. Запросы формулируются в терминах онтологии, описывающей предметную область и пользователю не требуется понимание конкретных источников данных, знание о взаимосвязях между этими источниками или способе кодирования данных в отдельных источниках. Ключевой технологией является технология переписывания запросов (query rewriting), которая позволяет избежать физической материализации источников в графе знаний. Запрос SPARQL транслируется в

серию SQL запросов к различным источникам данных, а затем результаты запросов собираются в единый ответ на исходный SPARQL запрос. Схема этого подхода показана на рисунке 1.

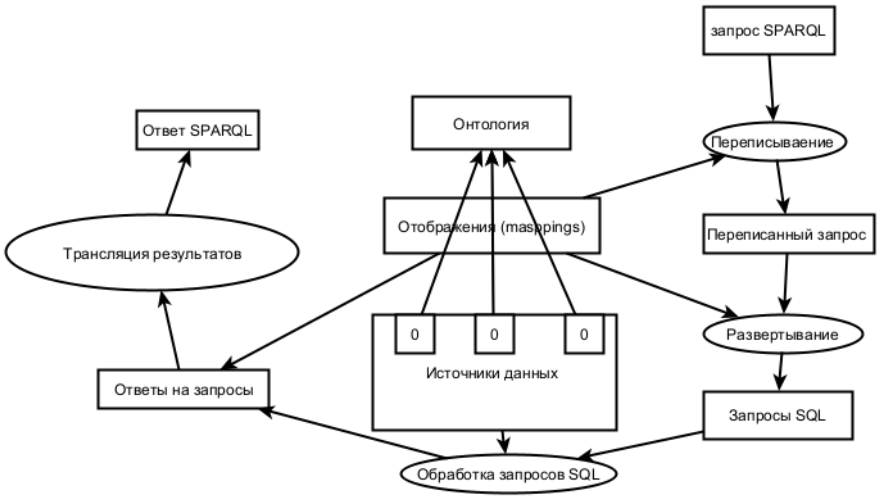



Рис. 1. Схема метода переписывания запросов для работы с виртуальным графом знаний.

Преимущество этого подхода заключается, с одной стороны, в знаниях о предметной области, закодированных в онтологии. Эти знания используются для обогащения ответов на запросы пользователей, описывающих задачи анализа продукта. Другим ключевым преимуществом является использование семантически насыщенных отображений ВГЗ. Они устраняют несоответствие между уровнем данных и уровнем онтологии, используя основанный на шаблонах механизм R2RML [6] для построения уникальных идентификаторов (IRI) графа знаний из значений базы данных. Более того, сопоставления обеспечивают решение задачи интеграции, поскольку семантически однородная информация, поступающая из разных файлов журналов, которые используют синтаксически различные представления, может быть согласована на уровне ГЗ. Это упрощает выполнение запросов ко всем данным интегрированным образом, используя семантику извлеченной информации.

В качестве примера рассмотрим использование виртуальных графов знаний для анализа качества поверхностного монтажа печатных плат на заводах BOSH [4]. Процесс поверхностного монтажа (surface mounting Process, SMT) включает четыре основных этапа (см. Рисунок 2):

(1) Нанесение паяльной пасты: этап состоит из нанесения паяльной пасты на печатные платы (ПП) и выполняется с помощью машины для печати паяльной пастой;

(2) Поверхностный монтаж: этап, где электронные компоненты фактически устанавливаются на печатные платы при поверхностном монтаже устройств (surface mount devices, SMD); по-русски говорят SMD компоненты, SMT технологи, и SMD-монтаж;

(3) Нагрев: для правильной пайки смонтированных компонентов на этом этапе. платы нагреваются в печи оплавления;

(4) Автоматизированная оптическая инспекция (automated optical inspection, AOI.

На заключительном этапе платы проверяются устройством AOI, чтобы выяснить, не произошел ли на предыдущих этапах какой-либо сбой, например, неправильная установка компонентов или неисправность пайки. Когда весь процесс завершен, система генерирует файлы, которые содержат два типа данных:

- журналы размещения, создаваемые SMD-устройством, содержат информацию о том, какой компонент установлен на какой плате;
- журналы неисправностей, генерируемые устройством AOI, показывают, где на плате и с какой компонентой случилась неисправность

Проблема состоит в том, что разные этапы этого процесса реализуются при помощи разных устройств, а эти устройства, как правило, поставляются разными производителями и имеют разные форматы и схемы для управления одними и теми же данными в процессе поверхностного монтажа.

Следовательно, необработанные, не интегрированные данные не дают целостного представления обо всем процессе SMT и затрудняют анализ выпускаемой продукции.

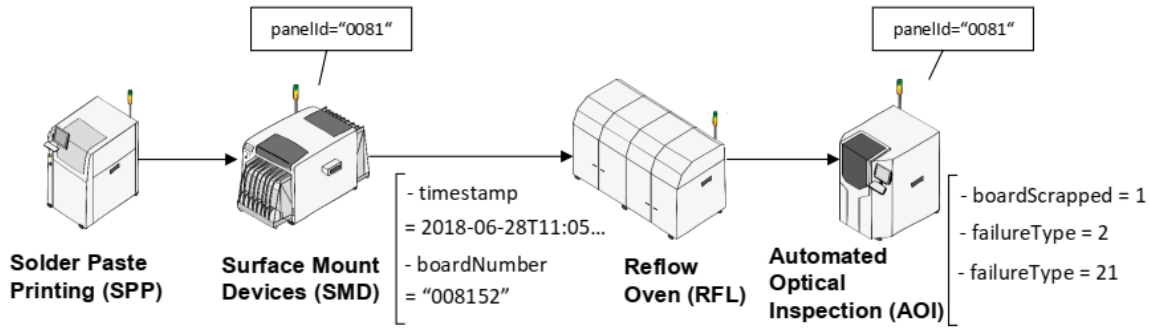



Рис. 2. Устройства, участвующие в процессе SMD-монтажа [4].

Каждое из устройств генерирует свои собственные данные (журналы), регистрирующие детали процесса обработки. Например, устройство оптического контроля, порождает таблицы такие как aoi_event, aoi_location, aoi_panel и aoi_failures. На рисунке 3 показаны фрагменты таблиц устройств. Таблица aoi_failures содержит столбец failureCode,

принимающий различные целочисленные значения в зависимости от типа сбоя. Эти численные значения могут быть различными даже в рамках одного предприятия.

**SMD Tables**

smd_event
smd_location
smd_panel
smd_components

**AOI Tables**

aoi_event
aoi_location
aoi_panel
aoi_failures

smd_panel

| panelId | boardNo | machineName | processedTS | location |
|---|---|---|---|---|
| p01 | b01 | SMD Machine 1 | 24-04-2020 | mes01 |

smd_components

| panelId | boardNo | headId | nozzleId | turnNo | pickSeqNo | placeSeqNo |
|---|---|---|---|---|---|---|
| p01 | b01 | h01 | n01 | 2 | 1 | 3 |

aoi_failures

| panelId | boardNo | refDesignator | windowNo | cPinNo | failureCode |
|---|---|---|---|---|---|
| p01 | b01 | rd01 | w01 | pn01 | 1 |

Рис. 3. Реляционные таблицы, используемые устройствами SMD и AOI [4].

На рисунке 3 показаны фрагменты трех таблиц. Таблицы smd-panel smd_components создаются SMD-устройством, а таблица aoi_failures - AOI устройством. Можно видеть, что SMD-устройство обрабатывает информацию о панели (столбец panelId), монтируемых на этой панели компонентах (столбец boardNo), и о моменте времени, в который данная панель была обработана (столбец processed TS), в то время как устройство AOI генерирует информацию о том, была ли панель списана или нет (столбец failureCode). Значение этой информации закодировано в числах: значение ’1’ соответствует случаю, когда панель списывается, 0 - в противном случае. Кроме того, устройство AOI содержит информацию о различных типах сбоев, например, число 2 соответствует ситуации «ложный вызов» (FalseCall), а число 21 соответствует случаю «Неправильно размещенный компонент» (MisplacedComponent). Смысл этой информации, закодированной в числах, имеется только во внутренних документах предприятия и в головах его инженеров.

Теперь предположим, что мы хотим получить ответ на следующий запрос. «Для панелей, обработанных в заданный период времени, получить количество отказов, связанных со списанными платами и сгруппированных по типам отказов». Чтобы получить эту информацию, необходимо семантически интегрировать данные, которые находятся на устройстве поверхностного монтажа SMD и на устройстве автоматического оптического контроля AOI.

На рисунке 4 показана часть онтологии SMT, которая используется в качестве модели предметной области для интеграции семантических данных и доступа к предметной области. Онтология разделена на три модуля:

- Онтология продуктов SMT (product ontology, префикс psmt:), описывающая продукты SMT;
- Онтология сбоев SMT (failure ontology, префикс fsmt:), моделирующая отказы, которые могут произойти во время процесса SMT;и
- Онтология устройств SMT (machine ontology, префикс msmt:), моделирующая устройства SMT. Общая онтология SMT включает 76 классов, 30 объектных свойств и 57 свойств типов данных.

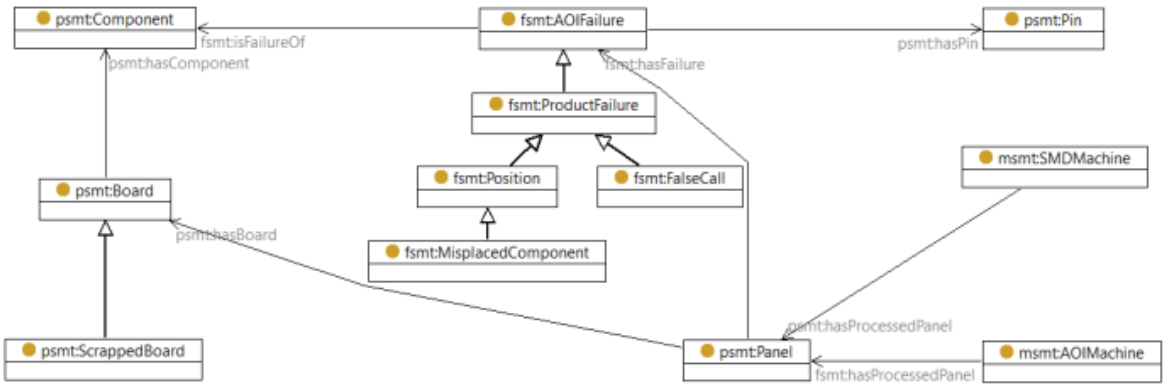



Рис. 4. Онтология SMT включает в качестве подмодулей онтологию продуктов, онтологию устройств и онтологию сбоев [4].

Как правило, способ кодирования отказов различается на разных предприятиях внутри одной и той же организации, что затрудняет идентификацию одних и тех же отказов, описанных в разных подсистемах. Чтобы решить эту проблему, предлагается вместо численных значений, принимающих различные значения в различных реляционных таблицах, использовать соответствующие классы в Онтологии сбоев. Например, для представления такого типа отказа как «неправильное размещение компонента» создается класс *fsmt:MisplacedComponent*. Этот класс является подклассом класса *fsmt:Position*, а также класса *fsmt:ProductFailure*, которые семантически группируют все отказы, попадающие в эти категории. Точно так же создается класс *psmt:ScrappedBoard* для семантического представления всех плат, которые списываются. Этот класс представлен как подкласс класса *psmt:Board.*

На рисунке 5 показаны два примера отображений. Одно из отображений называется “Ложный вызов” ("FalseCall"), а второе – “Неправильно размещенная компонента” (“MisplacedComponent”). Оба отображения определяют, какие триплеты графа знаний должны быть сгенерированы в зависимости от численного значения столбца failureCode в таблице aoi_failures.

```
mappingId FalseCall
target    fsmt:AOIFailure/{panelId}/{boardNo}/{refDesignator}/{windowNo}/{cPinNo}
            rdf:type fsmt:FalseCall ;
            fsmt:failureType "FalseCall" .
source    SELECT panelId, boardNo, refDesignator, windowNo, cPinNo
          FROM aoi_failures WHERE failureCode = 2

mappingId MisplacedComponent
target    fsmt:AOIFailure/{panelId}/{boardNo}/{refDesignator}/{windowNo}/{cPinNo}
            rdf:type fsmt:MisplacedComponent ;
            fsmt:failureType "MisplacedComponent" .
source    SELECT panelId, boardNo, refDesignator, windowNo, cPinNo
          FROM aoi_failures WHERE failureCode = 21
```

Рис. 5. Отображения FalseCall и MisplacedComponent [4].

Если значение failureCode равно 2, то в графе знаний должны быть сгенерированы две триплеты. Обе триплеты будут иметь в качества субъекта объект, имеющий URI fsmt:AOIFailure/{panelId}/{boardNo}/{refDesignator}/{windowNo}/{PinNo}, где на месте местодержателей будут указаны конкретные идентификаторы панели, компоненты, контакта и т.д., например fsmt:AOIFailure/p01/b01/rd01,wx1/p01. Сгенерированные по этому отображению триплеты будут иметь вид:

fsmt:AOIFailure/p01/b01/rd01/wx1/p01 rdf:type fsmt:FalseCall.

fsmt:AOIFailure/p01/b01/rd01/wx1/p01 fsmt:failureType “FalseCall”.

Аналогичным образом, отображение MisplacedComponent указывает, какие триплеты графа знаний должны быть сгенерированы, в случае, когда таблице aoi_failures код сбоя в столбце failureCode равен 21. Этому случаю в сгенерированном графе знаний также будет соответствовать две триплеты.

К результирующему графу знаний можно писать запросы SPARQL в терминах онтологии, описывающей предметную область, и пользователю не требуется понимание конкретных источников данных, знание о взаимосвязях между этими источниками или способе кодирования данных в отдельных источниках.

В работе [8] описано применение подхода ВГЗ в норвежской многонациональной нефтегазовой компании Equinor (ранее Statoil ASA). Одна из обычных задач геологов Equinor - найти новые пригодные для эксплуатации скопления нефти или газа в заданных областях, своевременно анализируя данные об этих областях. Однако сбор необходимых данных не является тривиальной задачей, так как данные хранятся в нескольких сложных и больших источниках данных, таких как EPDS, Recall, CoreDB, GeoChemDB, OpenWorks, Compass и NPD FactPages. Построение правильных запросов ко всем этим источникам невозможно для геологов Equinor, поэтому они должны сообщать свои потребности в информации ИТ-специалистам, которые затем превращают их в запросы SQL. Это резко влияет на

эффективность поиска правильных данных для поддержки принятия решений. Эта проблема была решена путем создания виртуального графа знаний, и создания каталога запросов SPARQL в терминах предметной области.

Еще одно применение подхода виртуальных графов знаний, реализованное в фирме Siemens, описано в работе [9]. Подразделение Siemens Energy управляет несколькими сервисными центрами, которые удаленно контролируют и выполняют диагностику нескольких тысяч устройств, таких как газовые и паровые турбины, генераторы и компрессоры, установленные на электростанциях. Для выполнения диагностики доступ к данным и интеграция как статических данных (например, конфигурация и структура турбин), так и динамических данных (например, данные от датчиков) особенно важны, но очень сложны. Опять же использование виртуальных графов знаний позволяет справиться с этой проблемой.

В работе [10] описано применение ВГЗ в аэрокосмической промышленности для извлечения и валидации информации о результатах тестирования электронных устройств, запускаемых в космос.

## 3. Исполняемые графы знаний для контроля качества точечной контактной сварки в BOSH

Точечная контактная сварка – это пример полностью автоматизированного и эффективного производственного процесса, широко применяемого в самолето-, судо- и автомобилестроительной промышленности, в сельскохозяйственном машиностроении и других отраслях промышленности. Для осуществления сварки два колпачка электродов сварочной горелки сжимают два или три металлических листа между электродами и пропускают ток высокого напряжения. Материал на небольшой площади между электродами плавится и образует точечный сварочный шов, соединяющий рабочие листы, известный как *литое ядро сварной точки* [11]. Качество операции сварки количественно оценивается как диаметр литого ядра сварной точки, как предписано в международных стандартах. Для точного определения диаметра литого ядра сварной точки в машиностроении обычной практикой является разрезание сваренного кузова автомобиля и измерение размера этого ядра, что разрушает сваренные автомобили и является чрезвычайно дорогой операцией. В настоящее время Bosch [12] разрабатывает методы оценки качества сварки, основанные на машинном обучении, чтобы уменьшить потребность в разрушенных кузовах автомобилей.

Сварка — это процесс, требующий больших объёмов данных. Каждая сварочная машина производит одну сварную точку за несколько секунд или минут, а кузов автомобиля может иметь до 6000 сварных точек. Для каждой точки генерируется несколько сотен характеристик, включая состояние сварки, показатели качества и показания датчиков, которые измеряют важные физические характеристики каждую миллисекунду, такие как ток, сопротивление, мощность.

Для оценки качества сварки применяется три важные направления машинного обучения такие как визуальная аналитика, статистическая аналитика и аналитика машинного обучения, основанная, например, на нейронных сетях.

Более того, у Bosch много источников данных для схожих производственных процессов, поэтому возможность повторного использования решений машинного обучения очень желательна. Эти решения можно перенести на аналогичные данные или вопросы машинного обучения.

В работе [12] предлагается закодировать решения машинного обучения при помощи графа знаний таким образом, чтобы графы знаний помогали в описании знаний о машинном обучении и решениях стандартизированным и прозрачным способом с помощью системы на основе графического интерфейса и визуализации графа знаний. Данный подход называется *исполняемые графы знаний*, потому что такие графы знаний могут быть преобразованы в скрипты машинного обучения, которые можно модифицировать и повторно использовать для решения похожих проблем.

Фреймворк для исполняемого графа знаний представляет возможность создания различных решений на базе методов машинного обучения (конвейеров) для решения проблем машинного обучения. Фрэймворк поддерживает трансляцию исполняемых графов знаний в исполняемые скрипты машинного обучения.

Фрэймворк исполняемого графа знаний определяет связи между такими понятиями как *Данные* (Data), *Метод* (Method) и *Задание* (Task). *Данные* – это множество элементов информации, организованные в разные структуры. *Метод* – это функция, реализованная в форме скрипта, написанного на заданном языке. *Метод* получает *Данные*, удовлетворяющие определенным ограничениям, и выдает в качестве результата другие *Данные*. *Задани*е – это способ вызова *Метода*, подавая ему *Данные*, удовлетворяющие определенным ограничениям. Некоторые задания имеют метод для выполнения этого задания, а некоторые задания не могут быть выполнены при помощи одного метода и должны быть развернуты в последовательность заданий, где каждое задание является частью более сложного задания в

этом случае можно говорить о *Конвейере* (Pipeline) заданий. Связи между этими понятиями показаны на рисунке 6.

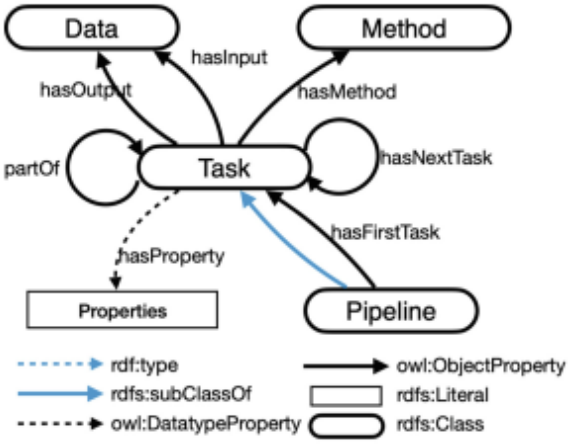



Рис. 6. Связи между основными классами фреймворка исполняемого графа знаний [12].

В зависимости от типа аналитического приложения могут вызываться различные методы, описанные в специализированных онтологиях, и генерироваться различные конвейеры заданий. Общий вид исполняемого графа знаний показан на рисунке 7.

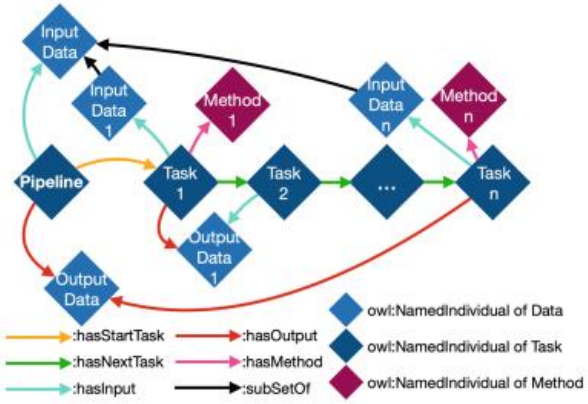



Рис. 7. Общая схема исполняемого графа знаний [12].

Архитектура системы для работы с исполняемыми графами знаний показана на рисунке 8. Система состоит из пяти слоев: слой данных (не граф знаний), уровень приложений, уровень базы данных графа знаний, уровень семантических модулей и уровень семантических артефактов. Модули одного уровня покрашены в один цвет. Например, самый нижний уровень данных содержит пять модулей, покрашенных в серый цвет. На этом уровне находятся входные и выходные данные системы. На вход Модуля Интеграции данных подаются необработанные данные сварки (нижний левый угол). Эти данные преобразуются Модулем Интеграции данных (с помощью онтологий предметной области) в графы знаний

машинного обучения для сварочных аппаратов, которые представляет собой тип графа знаний о данных сварки с некоторой аннотацией машинного обучения. Эти графы знаний используются четырьмя типами аналитических приложений на уровне приложений (показаны бежевым цветом).

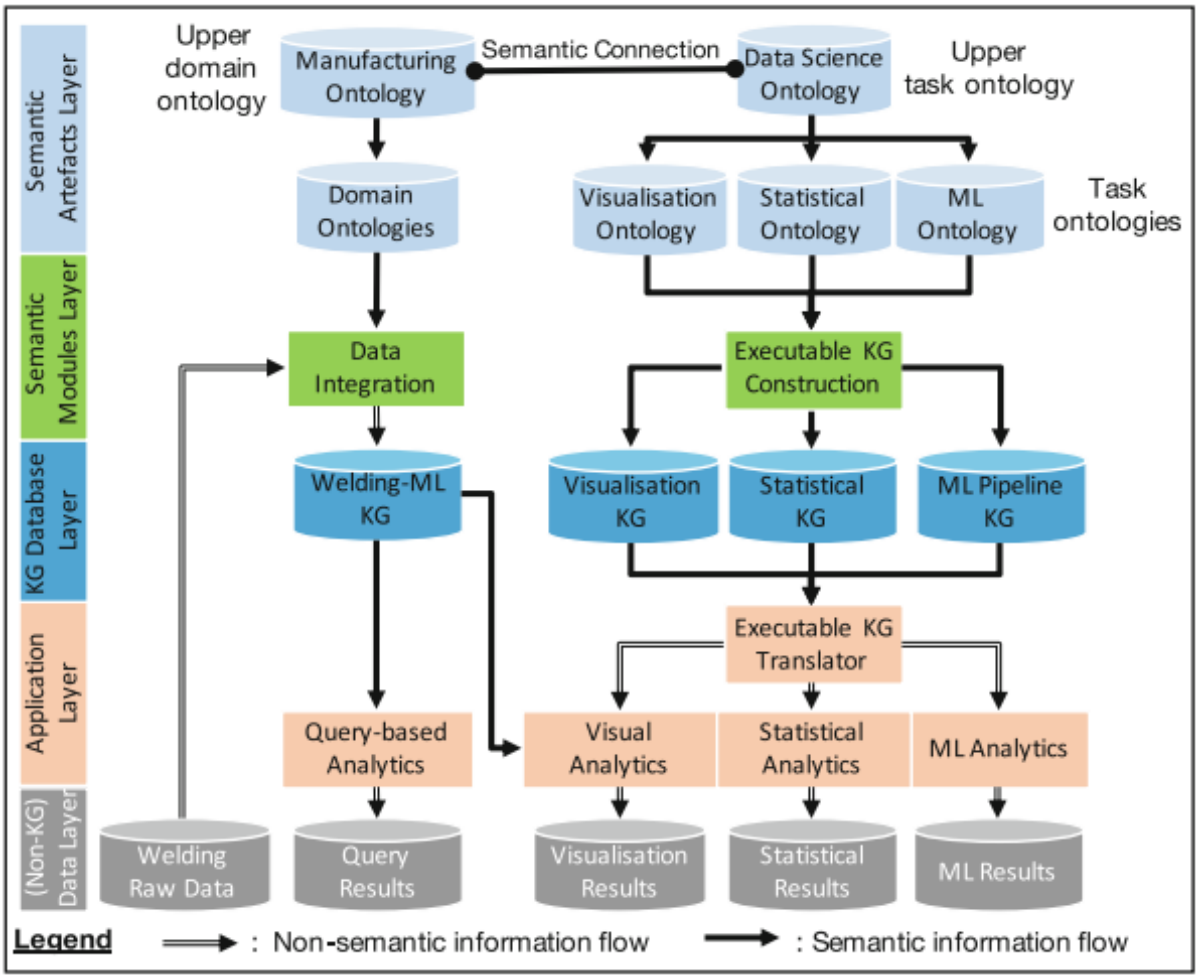



Рис. 8. Архитектура системы исполняемых графов знаний [12].

На самом верхнем уровне Семантических артефактов находятся две онтологии верхнего уровня. Это Онтология Производства (Manufacturing ontology) и Онтология интеллектуального анализа данных ( data science ontology, $O_{ds}$). Специализацией онтологии производства являются онтологии предметных областей, такие как различные онтологии сварки, например, онтология точечной сварки. Эти онтологии создаются на основе онтологии производства верхнего уровня [13].

Онтология производства семантически связана с онтологией верхнего уровня Онтологии интеллектуального анализа данных (data science ontology, Ods), таким образом, что атрибутные свойства (DataTypeProperty) онтологии производства аннотированы некоторыми классами онтологии интеллектуального анализа данных Ods.

На рисунке 9 показаны Онтология интеллектуального анализа данных (Ods, Data Science Ontology) и три ее специализации Онтология статистики (StatisticalOntology), онтология Визуализации (Visualization Ontology) и Онтология машинного обучения (ML-Ontology). Три онтологии заданий StatisticalOntology, Visualization Ontology и ML-Ontology являются специализацией онтологии Ods, поскольку все классы этих онтологий являются подклассами онтологии Ods, а все свойства являются подсвойствами Ods.

Можно видеть, что онтология верхнего уровня содержит такие классы как Data (Данные), Method (Метод) и Task (Задание), в то время как онтологии-специализации содержат методы и задания, специфические для конкретного приложения. Например, методы, необходимые для выполнения статистической аналитики описаны в онтологии статистического анализа (Statistical_Ontology, Ostats), методы, необходимые для выполнения визуальной аналитики, описаны в онтологии визуализации (Visualization_Ontology, Ovisu), а методы, необходимые для выполнения аналитики, Машинного обучения, описаны в онтологии машинного обучения (ML_Ontology, Oml).

Так, например, в онтологии визуализации Visualisation_Ontology описаны специфические для визуализации методы такие как LineplotMethod (метод построения линейчатых диаграмм), ScatterplotMethod (метод построения диаграмм рассеяния BarplotMethod (метод построения гистограмм). Все эти методы являются экземплярами класса VisualMethod, являющегося подклассом класса Method.

Аналогичным образом методы, специфические для статистической аналитики, такие как MeanCalculationMethod и StandardDeviationMethod являются подклассами класса StatisticalMethod онтологии Statistical-Ontology.

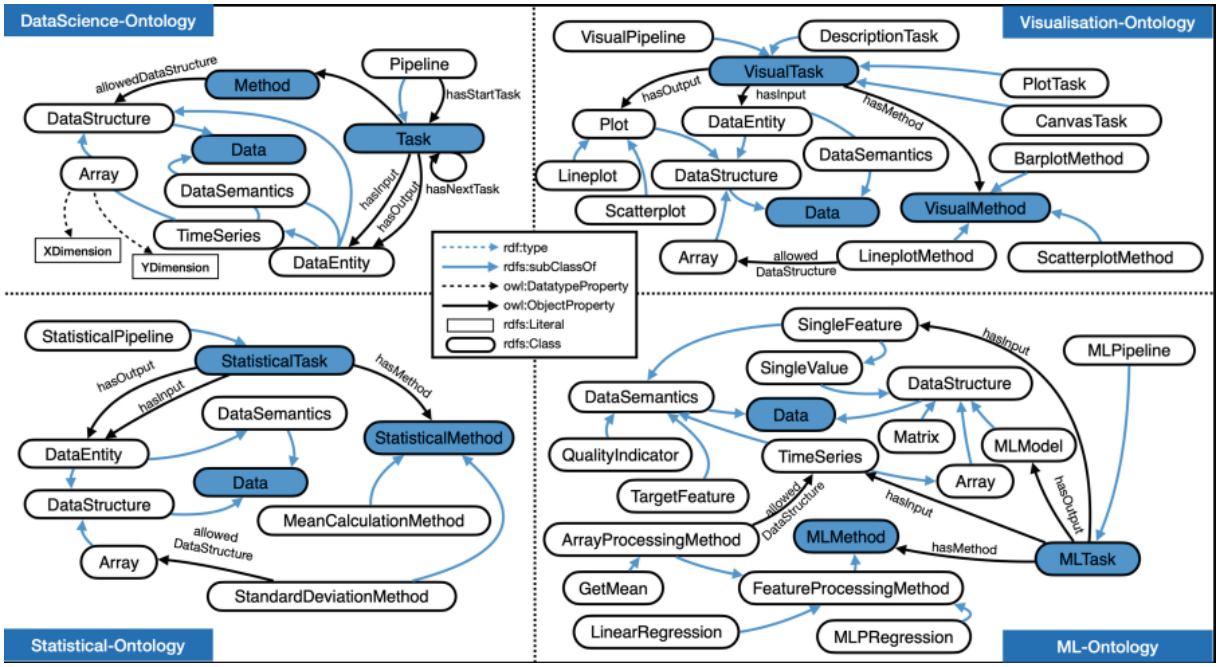



Рис. 9. Онтологии задач интеллектуального анализа данных [12].

Эти специализированные онтологии служат схемами для Модуля Построения исполняемого графа знаний, который кодирует различные конвейеры исполняемых графов знаний. Все исполняемые графы знаний принадлежат классу Pipeline (Конвейер) и соответствуют конкретным решаемым проблемам. На рисунке 10 показан один из возможных вариантов графа знаний для визуализации.

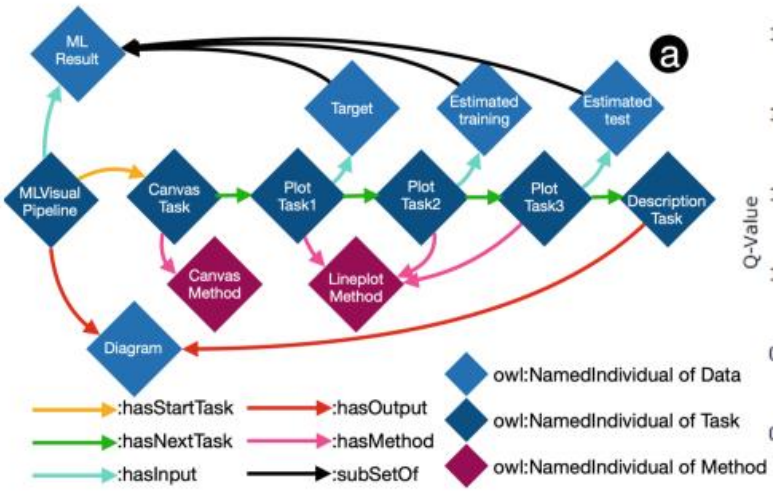



Рис. 10. Один из исполняемых графов знаний для визуализации [12].

Эти исполняемые графы могут быть переведены модулем Executable Knowledge Graph Translator в исполняемые скрипты, создавая тем самым приложения визуальной либо статистической аналитики или же приложения аналитики машинного обучения.

Возможно три способа построения новых исполняемых графов знаний: Создание графа знаний с нуля на основе библиотеки имеющихся модулей, Модификация ГЗ и интеграция

нескольких ГЗ. Например, можно интегрировать конвейер статистического анализа, который идентифицирует выбросы, с конвейером визуализации для того, чтобы визуализировать результаты идентификации выбросов.

## 4. Динамические графы знаний для цифровых двойников

Одной из областей применения семантических моделей являются цифровые двойники. Цифровой двойник (ЦД) — это цифровое представление системы IoT, способное непрерывно обучаться в течение всего жизненного цикла системы и прогнозировать поведение системы IoT. Цифровые двойники предназначены для постоянного использования на протяжении всего жизненного цикла системы - от предоставления рекомендаций при создании системы, до автоматизации ее производства и оптимизации ее работы путем диагностики аномалий или улучшения контроля и прогнозирования. В последние несколько лет наблюдается тенденция к объединению ЦД с передовыми технологиями семантического моделирования и методами глубокого обучения для обеспечения ЦД когнитивными возможностями.

Термин «когнитивные цифровые двойники» по отношению к промышленным приложениям впервые был предложен Адлом [14] в 2016 году, хотя и до этого в литературе рассматривались возможности улучшения когнитивных способностей цифровых двойников с помощью семантических технологий.

В ходе презентации на отраслевом семинаре в 2016 году Ахмед Эль Адл рассказал о когнитивной эволюции технологий Интернета вещей и предложил концепцию когнитивного цифрового двойника, а также её характеристики и категории. Она была определена как «цифровое представление, дополнение и интеллектуальный компаньон своего физического двойника в целом, включая его подсистемы на всех этапах жизненного цикла и эволюции».

Хотя в настоящее время нет широко распространенного консенсуса по определению КЦД, можно выделить некоторые их общие элементы и характеристики [15].

(1) Основаны на ЦД: КЦД является расширенной или дополненной версией ЦД. Он содержит как минимум три основных элемента ЦТ, включая физическую сущность (системы, подсистемы, компоненты и т. д.), цифровое (или виртуальное) представление, связи между виртуальным и физическим пространствами. Основное отличие заключается в том, что КЦД обычно содержит несколько моделей ЦД с определениями топологии и семантики. В частности, для сложной промышленной системы КЦД должен включать цифровые модели подсистем и компонентов, каждая из которых имеет разный статус на протяжении всего жизненного цикла. Для более сложных бизнес-сценариев ожидается подключение большого количества связанных цифровых моделей.

(2) Когнитивные способности: КЦД должен обладать определенными когнитивными способностями, то есть он позволяет выполнять интеллектуальные действия, подобные человеческим, такие как внимание, восприятие, понимание, память, рассуждение, прогнозирование, принятие решений, решение проблем, реакция и т. д.

(3) Управление полным жизненным циклом: КЦД должен состоять из цифровых моделей, охватывающих различные этапы всего жизненного цикла системы, включая начало жизненного цикла (например, проектирование, сборка, тестирование), середину жизненного цикла (например, эксплуатация, использование, техническое обслуживание) и конец жизненного цикла (например, разборка, переработка, восстановление). КЦД также должен быть способен интегрировать и анализировать все доступные данные, информацию и знания из различных этапов жизненного цикла.

(4) Способность к автономности: КЦД должен выполнять автономные действия без помощи человека или минимального уровня человеческого вмешательства. Эта способность частично перекрывается и подкрепляется когнитивными способностями КЦД. Например, основываясь на результатах восприятия и прогнозирования, КЦД может автономно принимать решения и адаптивно реагировать на проектирование, производство или эксплуатацию.

(5) Непрерывное развитие: КЦД должен иметь возможность развиваться вместе с реальной системой на протяжении всего жизненного цикла. Существует три уровня развития. Во-первых, для одной цифровой модели она обновляется в соответствии с изменением соответствующих данных, информации и знаний, полученных из реальной системы; во-вторых, благодаря взаимодействию между различными цифровыми моделями, содержащимися в одной и той же фазе жизненного цикла, каждая модель динамически развивается в соответствии с влиянием других моделей; в-третьих, благодаря обратной связи от других фаз жизненного цикла.

Одной из интересных моделей КЦД является World Avatar [16], использующий в качестве основного инструмента динамический граф знаний, который задуман как универсальный и всеобъемлющий. Универсальность Мирового Аватара основана на значительном количестве различных модульных онтологий предметных областей и экосистеме автономных агентов, способных модифицировать граф знаний.

Схематическое устройство World Avatar показано на рисунке 11. В основании конструкции находятся расширяемое множество модульных онтологий предметных областей. Это множество прямоугольников голубого цвета. Реальные экземпляры графа знаний, изображенные при помощи эллипсов, описываются при помощи этих онтологий

(зеленый слой). В зеленом слое также имеются красные эллипсы, изображающие экземпляры агентов. Агенты имеют разные типы (атомарный, составной и композитный) и являются частью графа знаний, управляемой Онтологией Агентов (OntoAgent) [17]. Агенты действуют автономно и непрерывно на графе знаний, постоянно обновляя его и, таким образом, заставляя его развиваться во времени.

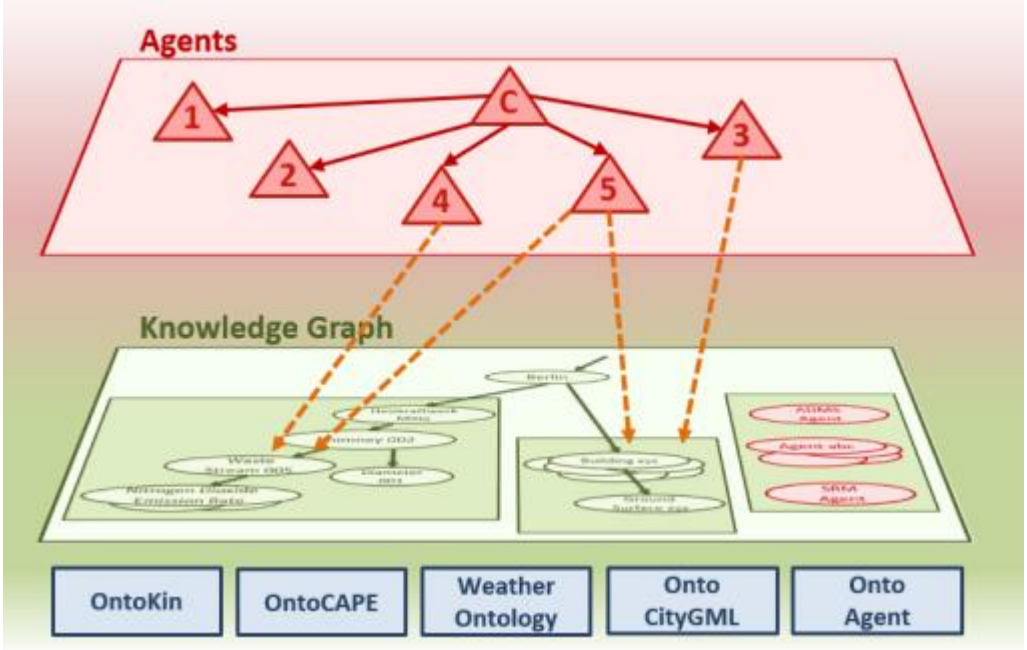



Рис. 11: Иллюстрация основных принципов Мирового Аватара (World Avatar) на основе динамического графа знаний [16].

В верхней части схемы World Avatar показаны активные агенты (красные треугольники), действующие над графом знаний и взаимодействующие друг с другом.

Понятие «Агент» может соответствовать программному обеспечению, методам, приложениям, сервисам и т. д., которые используют семантические веб-технологии и работают с графом знаний для чтения/записи, оценивания, моделирования, оптимизирования и/или запросов и т. д. для достижения конкретных целей.

Кроме того, чтобы облегчить использование агентов и упростить идентификацию агента, подходящего для конкретной задачи в среде с большим количеством агентов (где доступно множество сервисов), был создан рынок агентов на основе технологии блокчейн и смарт-контрактов [18].

World Avatar позволяет подключение к различным подграфам облака открытых связанных данных (LOD, lod-cloud.net), и, таким образом, может использовать богатство всех данных, доступных в Интернете. Благодаря своей универсальной конструкции, основанной на онтологиях и автономных агентах, World Avatar улучшает совместимость между

разнородными форматами данных, а также программным обеспечением и, таким образом, позволяет использовать междоменные приложения в более широком контексте.

Текущее онтологическое покрытие World Avatar включает:

- онтологию механизмов химических кинетических реакций OntoKin [19];
- технологических процессов OntoCAPE[20];
- погоды Weather Ontology [21];
- онтологию 3D моделей городов и ландшафтов OntoCityGML[22];
- систем электроснабжения OntoPowerSys, [23].

В области химии используются такие онтологии как

- онтология для квантовой химии OntoCompChem, [24];
- видов химических веществ OntoSpecies,[25];
- экспериментов по горению OntoChemExp [26];
- Онтологию выделяемых при сгорании веществ в зависимости от типа двигателя корабля OntoShip [27].

Далее будет показано два варианта использования принципиально разных аспектов фреймворка World Avatar – управление и проектирование.

Первый вариант демонстрирует, как цифровые двойники на основе динамического графа знаний могут сократить затраты и потребляемую энергию посредством интеллектуальных стратегий управления.

Второй вариант показывает, как можно использовать структуру параллельного мира для создания «живого» цифрового мира, т.е. сценария, который позволяет исследовать различные технологические альтернативы и эффект политики по переходу к энергосберегающим технологиям.

## 4.1 Кросс-доменное вычисление качества воздуха

В Сингапуре расположен один из самых загруженных портов в мире. Естественно, возникает вопрос о том, как выбросы углеводородов из такого трудно поддающегося сокращению сектора, такого как судоходство, влияют на такие факторы как качество воздуха в разных районах Сингапура.

Рисунок 12 иллюстрирует, каким образом интероперабельность World Avatar позволяет оценивать в режиме реального времени вклад выбросов от судоходства в качество воздуха в Сингапуре. Он демонстрирует, чего можно достичь с точки зрения совместимости, как между моделями и данными из разных предметных областей (погода, расположение зданий в

городе, корабли в гавани и топливо, которое они используют, выбросы вредных веществ и их рассеяние в атмосфере) и между различными масштабами размеров (от масштаба атомной длины в расчетах вычислительной химии до километровых масштабов моделирования атмосферной дисперсии).

Агенты, работающие над World Avatar, обновляют в графе знаний в режиме реального времени информацию о погоде и о кораблях, находящихся в окрестностях Сингапура. Агент по выбросам способен использовать информацию о судах для оценки выбросов несгоревших углеводородов, CO, NO2, NOx, O3, SO2, PM2,5 и PM10 с каждого корабля. Агент Атмосферной дисперсии может использовать информацию о погоде, выбросах каждого корабля и расположении зданий Сингапура для моделирования рассеивания выбросов.

Агенты виртуальных датчиков сообщают результирующие оценки качества воздуха в разных локациях.

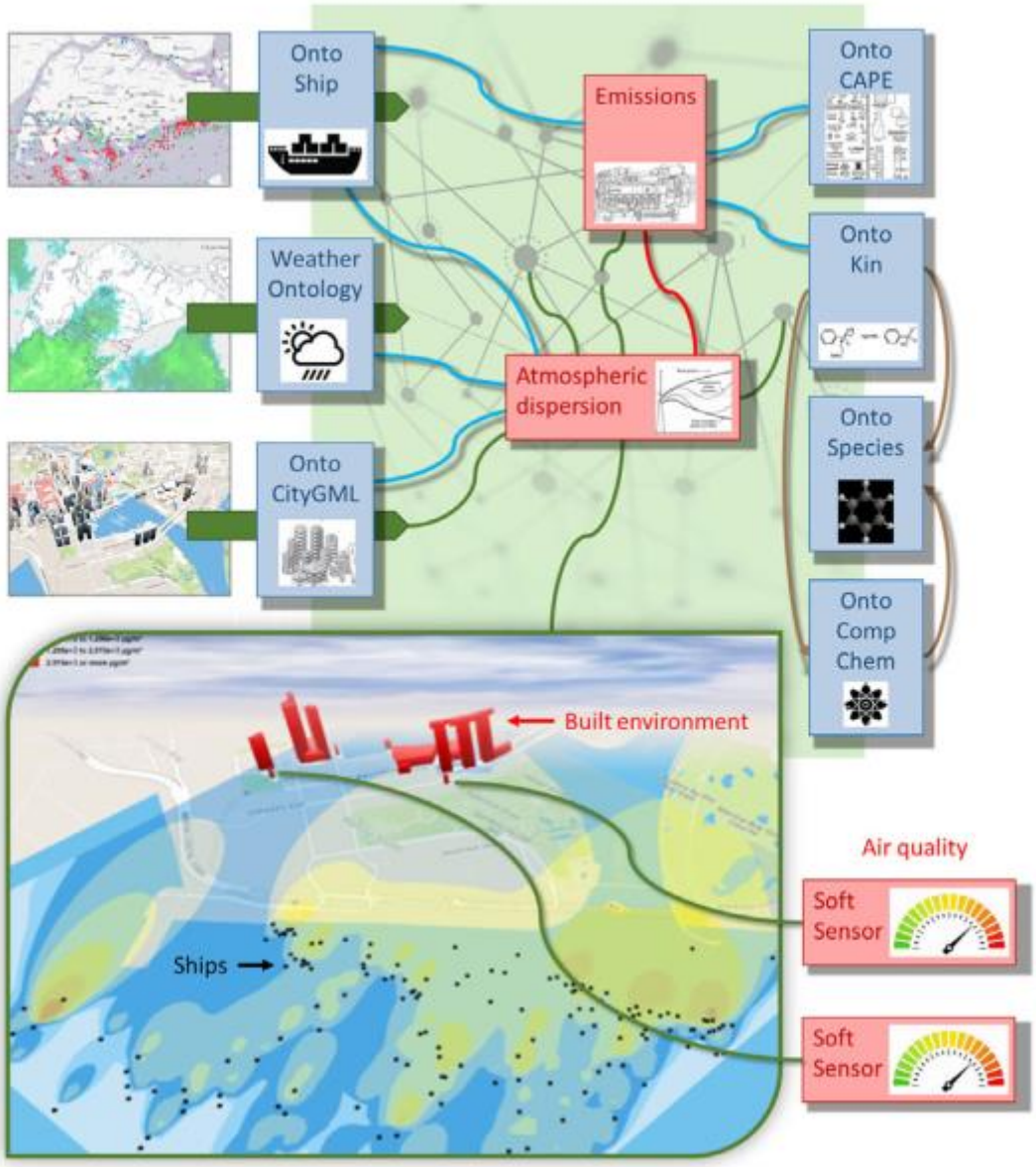



Рис. 12. Междоменная оценка в режиме реального времени вклада выбросов от судоходства в качество воздуха в Сингапуре [25].

## 4.2. Интеллектуальные стратегии проектирования систем электроснабжения

Этот вариант использования иллюстрирует, как цифровое дублирование и анализ сценариев «что, если» с использованием динамического графа знаний может помочь лицам, принимающим решения, понять взаимное влияние различных вариантов использования источников энергии и инструментов политики.

Возможности параллельных миров World Avatar поддерживают принятие решений в сложной среде, позволяя исследовать различные сценарии и связанные с ними результаты.

Параллельные миры используют структуру динамического графа знаний для использования агентов сценариев для группировки новых экземпляров любых сущностей, которые изменяются в сценарии, и сохраняют их в специфичной для сценария части графа знаний. Доступ, запросы и обновления экземпляров, специфичных для сценария, осуществляются через агентов сценария.

Специфическую для сценария информацию в параллельном мире можно рассматривать как наложение на базовый мир, поэтому неизмененные сущности остаются связанными с базовым миром, и любые изменения в базовом мире отражаются в сценарии. И наоборот, любые изменения в параллельном мире остаются изолированными в рамках специфической для конкретного сценария части графа знаний и поэтому не мешают базовому миру. Основанные на идее, очень похожей на системы контроля версий, широко используемые среди разработчиков программного обеспечения, контейнеры параллельного мира хранят различия с базовым миром в именованных графах, где сценарий предоставляет контекст. Технические подробности можно найти в [28].

Рисунок 13 иллюстрирует использование концепции параллельного мира для исследования влияния введения налога на выбросы углерода в результате процессов производства электроэнергии, чтобы мотивировать переход от ископаемого топлива на экологически чистые энергетические технологии. Рассматриваются следующие вопросы:

• Какая сумма налога на выбросы углерода необходима для того, чтобы сделать переход на малые модульные ядерные реакторы (Small Modular Reactor, SMR) выгодным для заданного набора условий (например, срок службы проекта, нормы амортизации, профили электрической нагрузки и характеристики генератора)?

• Какой завод(ы) следует заменить, и где следует расположить новые SMR(ы)?


Рис. 13. Концепция параллельного мира для анализа сценариев «что, если» [28].

В верхней части рисунка показана электрическая сеть из реального мира (слева) и оптимизированная сеть, облагаемая налогом на выбросы углерода (справа). Сеть из реального мира описана в базовом мире. Изменения в сети в параллельном мире описаны в специфичной для сценария части графа знаний. Розовые треугольники обозначают генераторы на природном газе, которые присутствуют как в базовом мире, так и в параллельном мире. Синий квадрат обозначает масляный генератор, который присутствует только в базовом мире. Символ радиации обозначает небольшой модульный ядерный реактор (SMR), который присутствует только в параллельном мире.

Параллельный мир показывает, что нефтяные генераторы будут заменяться на SMR, как только налог на выбросы углерода будет увеличен с пяти до ста семидесяти сингапурских долларов за тонну. Типы имеющихся генераторов и соответствующие оценки выбросов CO2 автоматически обновляются в параллельном мире, чтобы отразить эти изменения, и Агент оптимальной мощности потока (optimal power flow, OPF) вызывается для минимизации общих эксплуатационных затрат сценария в параллельном мире [28, 27].

В этом примере проблема, решаемая параллельным миром, достаточно проста, и ее можно было сформулировать как классическую задачу оптимизации с четко определенной целевой функцией. Однако многие сценарии будут слишком сложными для того, чтобы это

произошло. Как решить эту проблему, обсуждается в контексте цифрового двойника Великобритании на основе графа знаний [30].

The World Avatar можно рассматривать как пример новой парадигмы для информационных систем GeoWeb, выходящих за рамки Web 2.0. Основанный на базе Semantic 3D City Database [31], он переносит существующие стандарты ГИС в новую графовую базу данных и использует преимущества концепции открытого мира (Open World Assumption), которая отсутствует в аналогичных реляционных геопространственных базах данных. Сочетание её с системой интеллектуальных автономных агентов, основанных на когнитивной архитектуре, расширяет и масштабирует существующие инструменты преобразования геопространственных данных. Её агенты, как оказалось, способны автоматически создавать семантическую модель Берлина, состоящую в общей сложности из 419 909 661 утверждений типа «субъект–предикат–объект». Помимо традиционно трудоёмкого и подверженного ошибкам процесса создания модели, агенты также автономно создали представление модели, пригодное для взаимодействия с ней через веб-интерфейсы. Более того, агенты продемонстрировали способность отслеживать взаимодействие пользователя с моделью в сети, создавать новые знания и также автономно отображать их пользователю [31]. Данные работы положили начало большому множеству исследований по цифровым двойникам для моделирования умных городов, интегрирующим информацию о потреблении энергии, траффике, землепользовании, и др. [32, 33].

## 5. Взаимное обогащение ГЗ и больших языковых моделей

Появление Больших Языковых Моделей таких как Gemini от Google и серии GPT от OpenAI привело к повышению качества работы многих приложений, таких как автоматический перевод между разными языками, генерация контента и виртуальные помощники. Большие Языковые Модели используются в чатботах и сервисах клиентской поддержки, они прекрасно справляются с резюмированием текстов и анализом тональности документов для изучения общественного мнения. К сожалению, знание языковых моделей «замораживается» в их параметрах во время обучения, что приводит к определенным ограничениям. Прежде всего они способны генерировать неточную или бессмысленную информацию (галлюцинации), испытывают недостаток специфических знаний, а также нехватку знаний, появившихся после обучения, более того, не всегда их ответы возможно интерпретировать. Поэтому в последние несколько лет активно исследуются возможности повышения качества работы Больших Языковых моделей при помощи Графов Знаний. Недавние обзоры [34, 35] рассматривают возможность включения знаний из ГЗ в Большие

Языковые Модели. Наиболее простой способ включения знаний из ГЗ состоит генерации текстовых промптов на основе ГЗ, как это показано на рисунке 14 (вариант 1). Например, в работе [36] описан метод KAPING (Knowledge-Augmented language model PromptING, подсказки, расширенные знаниями для языковой модели), который извлекал факты из ГЗ и использовал их для создания подсказок LLM, чтобы получить быстрый ответ на вопросы.

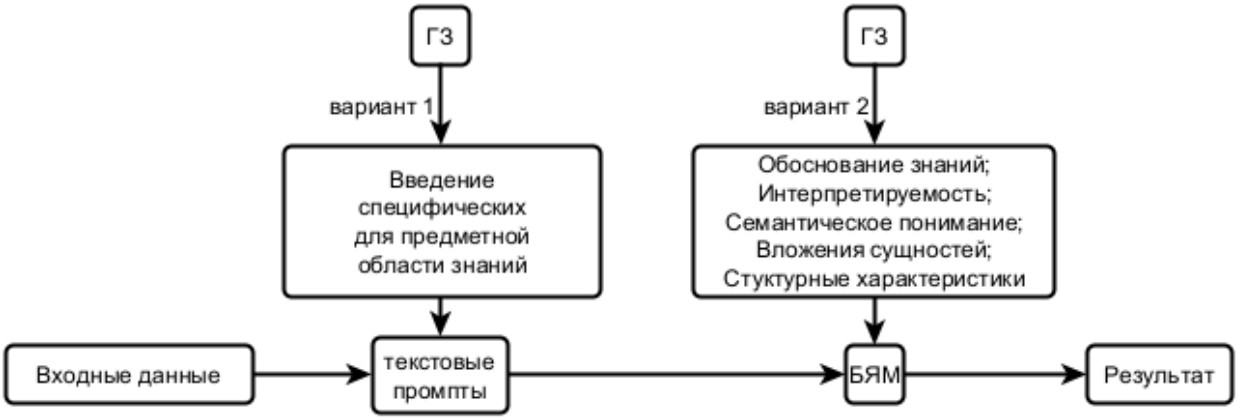



Рис. 14 Варианты введения знаний их графов знаний в большие языковые модели.

Возможны и другие варианты. Например, Knowledge Solver [37] вместо того, чтобы включать факты из ГЗ в промпты для БЯМ, обучает БЯМ многошаговому обходу ГЗ для поиска ответа на вопрос. Таким образом, ГЗ могут предоставлять факты, на основе которых БЯМ могут рассуждать, обосновывая эти факты. С другой стороны, ГЗ могут внести больший вклад в БЯМ, чем просто предоставляя факты для обоснования знаний. Для задачи ответа на вопросы QA-GNN (вопросно-ответная графовая нейронная сеть) [38] выполняла совместный вывод над векторными представлениями (embeddings) контекста вопроса, полученным при помощи БЯМ, и ГЗ для объединения двух представлений. Для лучшей интерпретируемости модели использовалась графовая нейронная сеть (GNN), вычислявшая веса между вершинами графа, предоставляя путь рассуждений, который модель прошла в ГЗ, чтобы получить ответ. Другим примером является LMExplainer [39], который использовал ГЗ и графовую нейронную сеть с механизмом внимания для понимания ключевых сигналов принятия решений в БЯМ, которые были преобразованы в объяснения на естественном языке для лучшей объяснимости. Таким образом, ГЗ также могут обеспечить лучшую интерпретируемость БЯМ и дать представление о процессах вывода БЯМ, что, в свою очередь, повышает доверие людей к БЯМ.

ГЗ также могут применяться для добавления семантического понимания или встраивания сущностей в БЯМ. Например, LUKE (Language Understanding with Knowledge-based Embeddings) [40], как расширение BERT, является механизмом само-внимания, осознающим

сущности, который может помочь модели обрабатывать слова и сущности в заданном тексте как независимые токены и выводить их контекстуализированные представления.

Что касается добавления семантического понимания, то недавняя методология, называемая «Правильное по правильным причинам» (**R**ight for **R**ight **R**easons, R3) [41] для выполнения запросов на естественном языке к контенту графов знаний (KGQA) с использованием БЯМ, представляет проблему поиска ответов на такие запросы здравого смысла как древовидную структуру поиска для полного использования выявленных аксиом здравого смысла – ключевого свойства, делающего процедуру вывода проверяемой, так что семантическое понимание из ГЗ может быть добавлено в БЯМ. Эти более продвинутые возможности обогащения БЯМ при помощи ГЗ показаны на рисунке 14 (вариант 2).

Несмотря на вышеуказанные достоинства графов знаний, одной существенной проблемой остается весьма дорогостоящий, трудозатратный и затратный по времени процесс создания графов знаний. Он требует значительного количества промежуточных шагов, таких как извлечение сущностей и отношений в соответствии с заданной онтологией, разрешение кореференций, слияние знаний. Более того, ГЗ специфичны для разных предметных областей, поэтому для разных приложений создаются разные графы знаний, а уже созданные графы знаний могут устаревать, если знание не обновляется.

Поэтому появление больших языковых моделей породило огромное количество исследований автоматической или полуавтоматической генерации графов знаний при помощи Больших Языковых Моделей. БЯМ обучаются на больших и разнообразных наборах данных и хранят эти знания неявно. В работе [42] описан полуавтоматический конвейер построения ГЗ при помощи ChatGPT-3.5, который побуждал эту модель генерировать высокоуровневые вопросы компетентности (competence questions) о данных. LLM было поручено извлекать сущности и отношения из этих вопросов для формирования онтологии, а затем отображать полученную информацию из документов в онтологию для построения ГЗ. Аналогичным примером является платформа AutoRD [43], недавно представленная для извлечения информации о редких заболеваниях и построения соответствующих графов знаний. Эта система может обрабатывать неструктурированный медицинский текст в качестве входных и выходных результатов извлечения, а также граф знаний, где БЯМ используется для извлечения сущностей и отношений из медицинских онтологий. Совсем недавно неконтролируемый фреймворк, называемый TKGCon (построение тематически-специфичного графа знаний) [44], использовал БЯМ для построения как онтологий, так и тематически-специфичных ГЗ, полагаясь на БЯМ для генерации и определения отношений между сущностями и построения ребер графа.

Эти методы свидетельствуют о том, что БЯМ способны на большее, чем извлечение знаний из неструктурированных данных. Они также могут обрабатывать и анализировать данные для построения и заполнения графов знаний. Кроме того, в работе [45] описаны другие методы, которые используют БЯМ для конкретных задач построения групп знаний, таких как сопоставление текста с онтологиями, извлечение сущностей и выравнивание онтологий. БЯМ также использовались для валидации групп знаний посредством проверки фактов и обнаружения несоответствий. Варианты введения информации из больших языковых моделей в графы знаний показаны на рисунке 15.

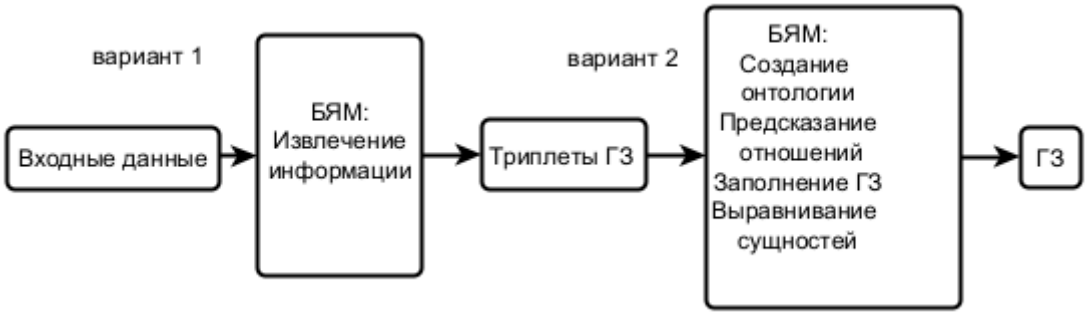



Рис. 15. Варианты введения информации из больших языковых моделей в графы знаний.

## 6. Заключение

В данной работе представлен обзор по современным приложениям различных вариаций графов знаний, таких как Виртуальные графы знаний, Исполняемые графы знаний и Динамические графы знаний. В работе сделан акцент именно на промышленных приложениях графов знаний, таких как цифровые двойники и их более современная версия когнитивные цифровые двойники. Также рассмотрено взаимное влияние графов знаний и больших языковых моделей, позволяющее создавать все более продвинутые приложения на основе ИИ. Приведенные примеры демонстрируют, что роль графов знаний при создании промышленный приложений в ближайшем будущем будет только возрастать.

## Список литературы

1. Апанович З.В. Эволюция понятия и жизненного цикла графов знаний // Системная информатика. — 2020. — № 16. — С. 57-74
2. Berners-Lee T, Hendler J and Lassila O (2001) The semantic web// Scientific American 284(5), 28–37.

3. Klyne G and Carroll JJ (2004) Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 10 February 2004. World Wide Web Consortium (W3C). [Электронный ресурс]. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (дата обращения: 1.10.2025).
4. Kalaycı, E.G., Grangel Gonz´alez, I., L¨osch, F., Xiao, G., Kharlamov, E., Calvanese, D., et al.: Semantic integration of Bosch manufacturing data using virtual knowledge graphs. // International Semantic Web Conference (ISWC). — 2020. — pp. 464–481.
5. Xiao, Guohui, Ding, Linfang, Cogrel, Benjamin, Calvanese, Diego, Virtual Knowledge Graphs: An Overview of Systems and Use Cases Data. //Intelligence 1(2019), 201-223.
6. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. // W3C Recommendation, World Wide Web Consortium (September 2012)
7. Calvanese, D., et al.: Ontop: answering SPARQL queries over relational databases. //Semant. Web J. **8**(3), 471–487 (2017).
8. Kharlamov E., Hovland D., Skjæveland M.G., Bilidas D., Jiménez-Ruiz E., Xiao G., Soylu A., Lanti D., Rezk M., Zheleznyakov D., Giese M., Lie H., Ioannidis Y.E., Kotidis Y., Koubarakis M., Waaler A. Ontology based data access in Statoil. //Journal of Web Semantics. — 442017. — pp. 3–36.
9. Kharlamov E., Mailis T., Mehdi G., Neuenstadt C., Özçep Ö. L., Roshchin M., Solomakhina N., Soylu A., Svingos C., Brandt S., Giese M., Ioannidis Y.E., Lamparter S., Möller R., Kotidis Y., Waaler A. Semantic access to streaming and static data at Siemens// Journal of Web Semantics. — 44. — 2017. — pp. 54–74.
10. De Santis, A. *et al.* (2025). Integrating Large Language Models and Knowledge Graphs for Extraction and Validation of Textual Test Data. In: Demartini, G., *et al.* The Semantic Web – ISWC 2024. ISWC 2024. // Lecture Notes in Computer Science, vol 15233. Springer, Cham. https://doi.org/10.1007/978-3-031-77847-6_17.
11. [Электронный ресурс]. URL:https://commons.wikimedia.org/wiki/Category:Spot_welding. (дата обращения: 1.10.2025).
12. Zheng, Z. et al. Executable Knowledge Graphs for Machine Learning: A Bosch Case of Welding Monitoring // The Semantic Web – ISWC 2022. ISWC 2022. — Lecture Notes in Computer Science. — vol 13489. Springer, Cham. https://doi.org/10.1007/978-3-031-19433-7_45.
13. Svetashova, Y., et al.: Ontology-enhanced machine learning: a bosch use case of welding quality monitoring. // ISWC (2020).

14. Adl, Ahmed El. 2016. The Cognitive Digital Twins: Vision,Architecture Framework and Categories. Technical Report. [Электронный ресурс]. https://www.slideshare.net/slideshow/embed_code/key/JB60Xqcn. (дата обращения: 1.10.2025).
15. Xiaochen Zheng, Jinzhi Lu, Dimitris Kiritsis (2022) The emergence of cognitive digital twin: vision, challenges and opportunities// International Journal of Production Research, 60:24, 7610-7632, DOI: 10.1080/00207543.2021.2014591.
16. Mei Qi Lim, Xiaonan Wang, Oliver Inderwildi Markus Kraft August 3, 2021 The World Avatar – a world model for facilitating interoperability.
17. Zhou X, Eibeck A, Lim MQ, Krdzavac N and Kraft M (2019) An agent composition framework for the J-Park simulator – A knowledge graph for the process industry. // Computers and Chemical Engineering 130, 106577.
18. Zhou X, Lim MQ and Kraft M (2020a) A smart contract-based agent marketplace for the J-Park simulator – A knowledge graph for the process industry// Computers and Chemical Engineering 139, 106896.
19. Farazi F, Akroyd J, Mosbach S, Buerger P, Nurkowski D, Salamanca M and Kraft M (2020a) OntoKin: An ontology for chemical kinetic reaction mechanisms. //Journal of Chemical Information and Modeling 60(1), 108–120.
20. Marquardt W, Morbach J, Wiesner A and Yang A (2010) OntoCAPE – A re-Usable Ontology for // Chemical Process Engineering, 1st Edn. Berlin: Springer-Verlag.
21. Weather Ontology, Institute of Computer Engineering at Technical University of Vienna, [Электронный ресурс]. https://www.auto.tuwien.ac.at/downloads/thinkhome/ontology/WeatherOntology.owl (дата обращения: 8.08.2025).
22. Eibeck A, Lim MQ and Kraft M (2019) J-Park simulator: An ontology-based platform for cross-domain scenarios in process industry. // Computers and Chemical Engineering 131, 106586.
23. Devanand A, Kraft M and Karimi IA (2019) Optimal site selection for modular nuclear power plants. //Computers and Chemical Engineering 125, 339–350.
24. Krdzavac N, Mosbach S, Nurkowski D, Buerger P, Akroyd J, Martin J, Menon A and Kraft M (2019) An ontology and semantic web service for quantum chemistry calculations. // Journal of Chemical Information and Modeling 59(7), 3154–3165.
25. Farazi F, Krdzavac N, Akroyd J, Mosbach S, Menon A, Nurkowski D and Kraft M (2020b) Linking reaction mechanisms and quantum chemistry: An ontological approach. // Computers and Chemical Engineering 137, 106813.
26. Bai J, Geeson RM, Farazi F, Mosbach S, Akroyd J, Bringley EJ and Kraft M (2021) Automated calibration of a poly(oxymethylene) dimethyl ether oxidation mechanism using knowledge-graph technology. //Journal of Chemical Information and Modeling 61(4), 1701–1717.

27. Farazi F, Salamanca M, Mosbach S, Akroyd J, Eibeck A, Aditya LK, Chadzynski A, Pan K, Zhou X, Zhang S, Lim MQ and Kraft M Knowledge graph approach to combustion chemistry and interoperability. // ACS Omega 5(29), 18342–18348.
28. Eibeck A, Chadzynski A, Lim MQ, Aditya LK, Ong L, Devanand A, Karmakar G, MosbachS, Lau R, Karimi IA, Foo EYS and Kraft M (2020) A parallel world framework for scenario analysis in knowledge graphs. // Data-Centric Engineering 1, e6.
29. Devanand A, Kraft M and Karimi IA (2019) Optimal site selection for modular nuclear power plants. //Computers and Chemical Engineering 125, 339–350.
30. Akroyd J, Mosbach S, Bhave A, Kraft M. Universal Digital Twin - A Dynamic Knowledge Graph. //Data-Centric Engineering. 2021;2:e14. doi:10.1017/dce.2021.10
31. Chadzynski A, Li S, Grišiūtė A, et al. Semantic 3D city interfaces—Intelligent interactions on dynamic geospatial knowledge graphs. //Data-Centric Engineering. 2023;4:e20. doi:10.1017/dce.2023.14
32. Quek HY, Sielker F, Akroyd J, et al. The conundrum in smart city governance: Interoperability and compatibility in an ever-growing ecosystem of digital twins. //Data & Policy. 2023;5:e6. doi:10.1017/dap.2023.1.
33. Hofmeister M, Bai J, Brownbridge G, et al. Semantic agent framework for automated flood assessment using dynamic knowledge graphs. //Data-Centric Engineering. 2024;5:e14. doi:10.1017/dce.2024.11.
34. Linyao Yang, Hongyang Chen, Zhao Li, Xiao Ding, and Xindong Wu. Give us the facts: Enhancing large language models with knowledge graphs for fact-aware language modeling. //*IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2024. doi:10.1109/TKDE.2024.3360454.
35. Amanda Kau, Xuzeng He, Aishwarya Nambissan, Aland Astudillo, Hui Yin, Amir Aryani Combining knowledge graphs and large language models// arXiv:2407.06564.
36. Jinheon Baek, Alham Fikri Aji, Amir Saffari, Knowledge-Augmented Language Model Prompting for Zero-Shot Knowledge Graph Question Answering // arXiv:2306.04136.
37. Chao Feng, Xinyu Zhang, and Zichu Fei. Knowledge solver: Teaching llms to search for domain knowledge from knowledge graphs, 2023. //arXiv:2309.03118.
38. Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. Qagnn: Reasoning with language models and knowledge graphs for question answering, 2022. arXiv:2104.06378.
39. Zichen Chen, Ambuj K Singh, and Misha Sra. LMExplainer: a knowledge-enhanced explainer for language models, 2023. //arXiv:2303.16537.
40. Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. Luke: Deep contextualized entity representations with entity-aware self-attention, 2020. //arXiv:2010.01057.
41. Armin Toroghi, Willis Guo, Mohammad Mahdi Abdollah Pour, and Scott Sanner. Right for right reasons: Large language models for verifiable commonsense knowledge graph question answering, 2024. //arXiv:2403.01390.

42. Vamsi Krishna Kommineni, Birgitta König-Ries, and Sheeba Samuel. From human expert to machines: An llm supported approach to ontology and knowledge graph construction, 2024. //arXiv:2403.08345.
43. Lang Cao, Jimeng Sun, and Adam Cross. Autord: An automatic and end-to-end system for rare disease knowledge graph construction based on ontologies-enhanced large language models, 2024. //arXiv:2403.00953
44. Linyi Ding, Sizhe Zhou, Jinfeng Xiao, and Jiawei Han. Automated construction of theme specific knowledge graphs. //*arXiv preprint arXiv:2404.19146*, 2024.
45. Hanieh Khorashadizadeh, Fatima Zahra Amara, Morteza Ezzabady, Frédéric Ieng, Sanju Tiwari, Nandana Mihindukulasooriya, Jinghua Groppe, Soror Sahri, Farah Benamara, and Sven Groppe. Research trends for the interplay between large language models and knowledge graphs, 2024. arXiv:2406.08223.

УДК 519.681.2, 519.681.3

# Тестовые эквивалентности с обратимостью для временных сетей Петри

*Боженкова Е.Н. (Институт систем информатики СО РАН)*

В статье определяется и исследуется семейство тестовых эквивалентностей в контексте непрерывно-временных безопасных сетей Петри (ВСП) с возможностью отмены (обратимости) выполненных действий. Тестовые эквивалентности рассматриваются в интерливинговой и шаговой семантиках, семантике частичного порядка и комбинации этих семантик. Для представления вычислений ВСП используется частично-упорядоченная семантика временных причинных сетей-процессов. Обратимость действий рассматривается как возможность отмены в вычислении одиночных или параллельных действий, максимальных в данном вычислении относительно отношения причинной зависимости. В статье устанавливается иерархия взаимосвязей между рассматриваемыми эквивалентностями.

*Ключевые слова*: временные сети Петри, тестовые эквивалентности, отмена действий, обратимые вычисления

## 1. Введение

При моделировании и изучении поведения вычислительных процессов было введено значительное количество поведенческих эквивалентностей. При этом также активно исследовались и взаимосвязи эквивалентностей в разных семантиках от интерливинговой до семантик частичного порядка (см., например, классический обзор Р. ван Глаббека [34]). При интерливинговом подходе невозможно различить процессы с паралелльным и последовательным поведением. Для увеличения мощности эквивалентностей были введены шаговая семантика, в которой сравнение поведения происходит относительно выполнения множества независимых действий, и семантика частичного порядка, при которой в качестве подпроцессов выполнения берутся уже частично-упорядоченные множества.

В множестве известных подходов к определению понятия эквивалентности, от трассовой до бисимуляционной, большой подкласс занимают тестовые эквивалентности. Понятие тестовой эквивалентности параллельных процессов было предложено М. Хеннесси и Р. де Николой в статье [15]. Тест — это специальный процесс, который выполняется

параллельно с тестируемым процессом. Такое выполнение считается успешным, если тест достигает выделенного успешного состояния, и процесс проходит тест, если каждое его совместное выполнение с процессом является успешным. Два процесса считаются тестово эквивалентными, если они проходят одни и те же наборы тестов. Для облегчения применения тестовой эквивалентности обычно используются определения на основе найденных альтернативных характеризаций, в одном из наиболее распространенных определений тест состоит из процесса-эксперимента и допустимого множества возможных его продолжений вместо единственного действия ([4, 11]). Так, альтернативные характеризации временных тестовых эквивалентностей с использованием понятия допустимых множеств в работах [12] и [23] были даны для систем переходов с дискретным временем, в [5] — для непрерывно-временных структур событий. Другой подход к альтернативной характеризации был предложен в статьях [19] и [13], в них авторы нашли характеризации тестовых предпорядков для алгебр процессов с временными ограничениями через трассы отказов.

Для модели сетей Петри интерливинговые тестовые отношения были исследованы в работе [9], в которой кроме альтернативной характеризации получены результаты по дискретизации временных характеристик модели, сопоставленных фишкам и дугам.

Для представления семантики частичного порядка в модели сетей Петри обычно используются причинные сети-процессы ([20, 28, 35]), в которых выполнениям переходов соответствуют события, разметке — условия, частичный порядок моделируется отношениями причинной зависимости и параллелизма. Сравнение разновидностей тестовой эквивалентности в частично-упорядоченной семантике сетей Петри было проведено в статье [32].

Для сетей Петри с временными характеристиками частично-упорядоченная семантика была предложена для дискретно-временных сетей Петри в работах [2, 33], где время в модель введено как длительность срабатывания перехода; для непрерывно-временных безопасных сетей Петри (ВСП) — в работе [6], где переходам сопоставлены интервалы временных задержек их срабатывания. В работе [8] сравнение моделей временных автоматов и временных сетей Петри проведены с использованием интерливинговой трассовой и бисимулиционной эквивалентностями.

Иерархия трассовых и бисимуляционных эквивалентностей в частично-упорядоченной семантике ВСП изучалась в работе [37]. Для этой же модели исследования тестовых

эквивалентностей в семантиках причинных сетей-процессов и причинных деревьев проведены в работах [1, 10], при этом установлено совпадение данных семантик в контексте тестовых эквивалентностей.

Для увеличения выразительной мощности в вычислительные модели стали вводить возможность отменить выполнение некоторых действий. Такие модели с обратимостью нашли применение в моделировании биохимических реакций, отладке микропроцессоров и программных систем. В ходе исследований выделилось три основные способа введения отмены действий: обратный ход; причинная обратимость и произвольная обратимость. Метод обратного хода подразумевает, что отмена действий производится в том же порядке, в котором действия происходили (см., например, [14]). При отмене действий в методе причинной обратимости выбор происходит среди максимальных действий в текущем вычислении, т.е действия, для которых они являются необходимыми предшественниками, уже отменены или еще не были соверешны (см. [24]). Такой метод используется для моделирования систем, протоколов, при отладке программ ([21, 22, 36]). При методе произвольной обратимости, правила отмены действий задаются в самой модели (см., например, [3, 25, 31]). Расширение таких вычислительных моделей как сети Петри и структуры событий возможностью отмены действий и их взимосвязи исследовались для разных подходах к обратимости. Так в [24] найдено соответствие между подклассом обратимых первичных структур событий с причинной обратимостью и обратимыми О-сетями. В [25] предложен подкласс сетей Петри, в котором реализована произвольная обратимость, и для него установлено соответствие с произвольно-обратимыми структурами событий. Среди других исследований проводились поиски способов построения сетей с обратимостью из обычных (см., например, [7, 24]).

Для моделей с обратимыми вычислениями также изучаются и поведенческие эквивалентности. Иерархия эквивалентностей с причинной обратимостью в интерливинговой, шаговой семантиках, семантиках частичного порядка и сохраняющих историю исследована И.Филипсом и И.Улидовски для стабильных структур конфигураций с причинной обратимостью в работе [29]. Авторами установлено, что бисимуляции с обратимостью сильнее обычных бисимулиций, так интерливинговая бисимуляция с интерливинговой обратимостью сильнее сохраняющей историю бисимуляции без обратимости. Эти же авторы ввели логику EIL в работе [30], расширив логику HML ([18]) модальностями обратимости, что позволило дать логическую характеризацию подклассу сохраняющих

историю бисимуляций.

Отметим, что исследования эквивалентнтостей с учетом прошлого поведения делались и без введения в модель отмены действий. Например, в [16] подход к исследованию прошлого вычислений совпадает с методом обратного хода, интерливинговые (back-forth) бисимуляции определены на последовательностях вычислений. При таком определении бисимулиции введение обратимости не привнесло дополнительных возможностей, т.е. они оказазались слабее бисимуляций с причинной обратимостью из [29]. В работе [27] исследовалась $\delta$-бисимуляция на путях с использованием максимальных независимых элементов, что дало некоторое усиление, но такой подход отличается от причинной обратимости и слабее нее.

Цель данной работы — изучить тестовые эквивалентности с возможностью отмены выполненных действий в контексте безопасных ВСП, в которых переходы помечены временными интервалами и каждый переход, имеющий достаточное количество фишек во входных местах, должен срабатывать в момент времени, когда значение его счетчика принадлежит его временному интервалу.

Для представления вычислений ВСП используется частично-упорядоченная семантика причинных сетей-процессов. Тестовые эквивалентности для обычных прямых вычислений рассматриваются в семантиках от интерливинга до семантики частичного порядка, а обратимость действий рассматривается как прошлое вычислений с точки зрения причинной обратимости в интерливинговой и шаговых семантиках. Материал статьи разбит на части следующим образом. В первых двух главах будут рассмотрены основные определения ВСП и причинных сетей-процессов. Далее, в гл. 4, будут введены понятия обратимости действий. Гл. 5 посвящена определению тестовых эквивалентностей в разных семантиках с обратимостью. В гл. 6 будут исследованы взаимосвязи тестовых эквивалентностей. Заключительные замечания будут приведены в гл.7.

## 2. Временные сети Петри: синтаксис и шаговая семантика

В этой главе рассмотрим базовую терминологию непрерывно-временных сетей Петри и их шаговую семантику. Сначала напомним определения структуры и поведения сетей Петри. Пусть $Act$ — множество действий.

**Определение 1.** *(Помеченная над Act) сеть Петри* (СП) — это набор $\mathcal{N} = (P, T, F, M_0, L)$, где $P$ — конечное множество мест, $T$ — конечное множество переходов ($P \cap T = \emptyset$

и $P \cup T \neq \emptyset$), $F \subseteq (P \times T) \cup (T \times P)$ — отношение инцидентости, $\emptyset \neq M_0 \subseteq P$ — начальная разметка, $L : T \to Act$ — помечающая функция. Для элемента $x \in P \cup T$ определим множество ${}^\bullet x = \{y \mid (y, x) \in F\}$ входных и множество $x^\bullet = \{y \mid (x, y) \in F\}$ выходных элементов, которые для подмножества $X \subseteq P \cup T$ элементов обобщаются соответственно до множеств ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ и $X^\bullet = \bigcup_{x \in X} x^\bullet$.

*Разметка* $M$ СП $\mathcal{N}$ — это произвольное подмножество $P$. Переход $t \in T$ *готов сработать* при разметке $M$, если ${}^\bullet t \subseteq M$. Обозначим через $En(M)$ множество всех переходов, готовых сработать при разметке $M$.

Непустое множество переходов $\emptyset \neq U \subseteq T$ называется *шагом, готовым сработать при разметке* $M$, если $\forall t \in U \circ t \in En(M)$ и $(\forall t \neq t' \in U : {}^\bullet t \cap {}^\bullet t = t^\bullet \cap t'^\bullet = \emptyset$[1]. Если шаг $U$ готов сработать при разметке $M$, то его срабатывание приводит к новой разметке $M' = (M \setminus {}^\bullet U) \cup U^\bullet$ (обозначается $M \xrightarrow{U} M'$).

Под непрерывно-временной сети Петри (ВСП) [6] понимается СП, в которой с каждым переходом связан временной интервал, указывающий возможные временные моменты срабатывания перехода, готового по наличию фишек в его входных местах; готовый переход может сработать, только когда достигнута нижняя граница и не превышена верхняя граница его интервала, и, если он еще не сработал, то обязан сработать, когда достигнута верхняя граница его интервала.

Область $\mathbb{T}$ временных значений — множество неотрицательных рациональных чисел. Считаем, что $[\tau_1, \tau_2]$ — замкнутый интервал между двумя временными значениями $\tau_1, \tau_2 \in \mathbb{T}$. Также, бесконечность может появляться как правая граница в открытых справа интервалах. Пусть $Interv$ — множество всех таких интервалов.

**Определение 2.** *(Помеченная над Act) временная сеть Петри* (ВСП) — это пара $\mathcal{TN} = (\mathcal{N}, D)$, где $\mathcal{N}$ — (помеченная над $Act$) базовая сеть Петри и $D : T \to Interv$ — статическая временная функция, сопоставляющая каждому переходу временной интервал. Границы временного интервала $D(t) \in Interv$ называются ранним ($Eft$) и поздним ($Lft$) временами срабатывания перехода $t \in T$.

*Состояние ВСП* $\mathcal{TN}$ — это пара $S = (M, I)$, где $M$ — разметка СП $\mathcal{N}$ и $I : En(M) \longrightarrow \mathbb{T}$ — динамическая временная функция. Начальное состояние ВСП $\mathcal{TN}$ — это пара $S_0 = (M_0, I_0)$, где $M_0$ — начальная разметка СП $\mathcal{N}$ и $I_0(t) = 0$ для всех $t \in En(M_0)$.

[1]Для удобства последующего определения временных сетей Петри требование $M \cap U^\bullet = \emptyset$, необходимое для обеспечения безопасности сети, будет введено в определении свойства свободы от контактов.

Шаг $U$, готовый сработать при разметке $M$, *готов сработать в состоянии* $S = (M, I)$ *в относительный момент времени* $\theta \in \mathbb{T}$, если $(Eft(t) \leq I(t) + \theta)$ для всех $t \in U$ и $(I(t') + \theta \leq Lft(t'))$ для всех $t' \in En(M)$. *Срабатывание* шага $U$ приводит к новому состоянию $S' = (M', I')$ (обозначается $S \xrightarrow{(U,\theta)} S'$), при этом $M \xrightarrow{U} M'$ и $\forall t' \in T$ ∘

$$I'(t') = \begin{cases} I(t') + \theta, & \text{если } t' \in En(M \setminus {}^\bullet U), \\ 0, & \text{если } t' \in En(M') \setminus En(M \setminus {}^\bullet U), \\ \text{не определено}, & \text{в остальных случаях.} \end{cases}$$

Будем писать $S \xrightarrow{(A,\theta)} S'$ для $A = L(U) = \Sigma_{t \in U} L(t) \in Act^{\mathbb{N}}$, т.е. $A$ является мультимножеством над $\{a \in Act \mid a = L(t) \text{ и } t \in U\}$. Также будем использовать запись $S_0 \xrightarrow{\sigma} S'$, если $\sigma = (U_1, \theta_1) \dots (U_k, \theta_k)$ и $S_0 \xrightarrow{(U_1,\theta_1)} S^1 \dots S^{k-1} \xrightarrow{(U_k,\theta_k)} S^k = S'$ $(k \geq 0)$. Тогда $\sigma$ называется *последовательностью срабатываний в* $\mathcal{TN}$ *из* $S_0$ *(в* $S'$*)*, а $S'$ — *достижимым состоянием* $\mathcal{TN}$ *из* $S_0$. Множество всех последовательностей срабатываний в $\mathcal{TN}$ из $S_0$ обозначим $\mathcal{FS}_s(\mathcal{TN})$ и множество достижимых состояний $\mathcal{TN}$ из $S_0$ — $RS(\mathcal{TN})$. Для $\sigma = (U_1, \theta_1) \dots (U_k, \theta_k)$ $L(\sigma) = (A_1, \theta_1) \dots (A_k, \theta_k)$, если $A_i = L(U_i)$ для всех $1 \leq i \leq k$.

Если $\mid U_i \mid = 1$ для всех $1 \leq i \leq k$, т.е. $\sigma = (t_1, \theta_1) \dots (t_k, \theta_k)$, то $\sigma$ будем называть интерливинговой последовательностью срабатываний. Подмножество таких последовательностей в $\mathcal{FS}_s(\mathcal{TN})$ будем обозначать $\mathcal{FS}_i(\mathcal{TN})$. Для интерливинговых последовательностей срабатываний $L(\sigma) = (a_1, \theta_1) \dots (a_k, \theta_k)$, если $a_i = L(t_i)$ для всех $1 \leq i \leq k$.

ВСП $\mathcal{TN}$ называется $T$*-ограниченной*, если ${}^\bullet t \neq \emptyset \neq t^\bullet$ для всех переходов $t \in T$; *свободной от контактов*, если для любого состояния $S \in RS(\mathcal{TN})$ и любого шага $U$, готового сработать в состоянии $S$ в относительный момент времени $\theta$ верно: $(M \setminus {}^\bullet U) \cap U^\bullet = \emptyset$; *прогрессирующей по времени*, если для любой последовательности переходов $\{t_1, t_2, \dots, t_n\} \subseteq T$ такой, что $t_i^\bullet \cap^\bullet t_{i+1} \neq \emptyset$ $(1 \leq i < n)$ и $t_n^\bullet \cap^\bullet t_1 \neq \emptyset$, выполняется неравенство $\sum_{1 \leq i \leq n} Eft(t_i) > 0$. В дальнейшем будем рассматривать только $T$-ограниченные, свободные от контактов и прогрессирующие по времени ВСП.

**Пример 1.** Пример помеченной над $Act = \{a, b, c, d\}$ ВСП $\mathcal{TN}$ показан на рис. 1, где места представлены окружностями, переходы — барьерами; рядом с элементами ВСП размещены их имена; между элементами, включенными в отношение инцидентности, изображены стрелки; каждое место, входящее в начальную разметку, отмечено наличием в нем фишки (жирной точки); значения помечающей и статической временной функций указаны рядом с переходами. Нетрудно проверить, что шаг $U = \{t_1, t_3\}$ готов сработать при начальной разметке $M_0 = \{p_1, p_2\}$, а также готов сработать в начальном состоянии

$S_0 = (M_0, I_0)$, где $I_0(t) = \begin{cases} 0, \text{ если } t \in \{t_1, t_3\}, \\ \text{не определено иначе,} \end{cases}$ в относительный момент времени $\theta \in [2, 3]$. При этом, $\sigma = (U, 3)\ (t_2, 2)\ (U, 2)\ (\{t_5, t_4\}, 2)$ — последовательность срабатываний из $S_0$ в ВСП $\mathcal{TN}$, $L(\sigma) = ([b : 2], 3)(a, 2)([b : 2], 3)([d : 1, c : 1], 2)$. Кроме того, $\mathcal{TN}$ является $T$-ограниченной, свободной от контактов и прогрессирующей по времени. □

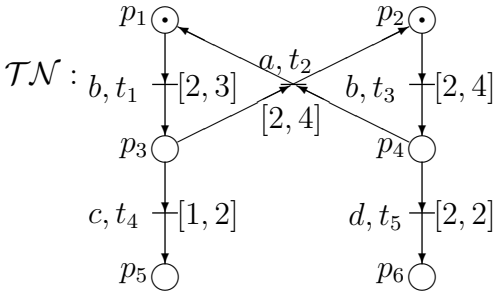



*Рис. 1.* Пример временной сети Петри.

## 3. Причинно-зависимые семантики временных сетей Петри

### 3.1. Временные сети

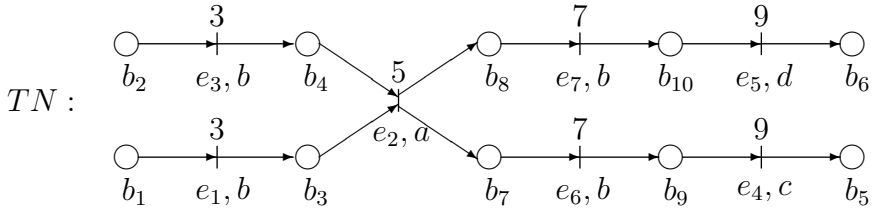



*Рис. 2.* Пример временной причинной сети.

Для анализа поведения ВСП используется семантика временных причинных сетей-процессов. Прежде чем перейти к рассмотрению данной семантики напомним базовые определения и обозначения, связанные с временными сетями.

**Определение 3.** *(Помеченной над Act) временной сетью* называется конечная, ациклическая сеть $TN = (B, E, G, l, \tau)$, где $B$ — множество условий, $E$ — множество событий, $G \subseteq (B \times E) \cup (E \times B)$ — отношение инцидентости (причинной зависимости $TN$) такое, что $\{e \mid \exists b \in B\ (e, b) \in G\} = \{e \mid \exists b \in B\ (b, e) \in G\} = E$, $l : E \to Act$ — помечающая функция и $\tau : E \to \mathbb{T}$ — временная функция такая, что $(e, e') \in G^+ \Rightarrow \tau(e) \leq \tau(e')$, т.е. событие не может произойти раньше своих предшественников относительно причинной зависимости $TN$.

В дальнейшем будут полезны следующие обозначения для временной сети $TN = (B, E, G, l,\tau)$. Пусть $\prec = G^+$, $\preceq = G^*$ и $\tau(TN) = \max\{\tau(e) \mid e \in E\}$. Определим множества: ${}^\bullet x = \{y \mid (y,x) \in G\}$ и $x^\bullet = \{y \mid (x,y) \in G\}$ для $x \in B \cup E$; ${}^\bullet X = \bigcup_{x\in X} {}^\bullet x$ и $X^\bullet = \bigcup_{x\in X} x^\bullet$ для $X \subseteq B \cup E$; ${}^\bullet TN = \{b \in B \mid {}^\bullet b = \emptyset\}$ и $TN^\bullet = \{b \in B \mid b^\bullet = \emptyset\}$.

$TN = (B, E, G, l, \tau)$ называется *(помеченной над Act) временной причинной сетью*, если $|{}^\bullet b| \le 1$ и $|b^\bullet| \le 1$ для всех условий $b \in B$.

Введем дополнительные определения и обозначения для временной причинной сети $TN = (B, E, G, l, \tau)$:

- $\downarrow e = \{x \mid x \preceq e\}$ — множество *предшественников события* $e \in E$, $Earlier(e) = \{e' \in E \mid \tau(e') < \tau(e)\}$ — множество *временных предшественников события* $e \in E$;
- $E' \subseteq E$ — *левозамкнутое* подмножество $E$, если $\downarrow e' \cap E \subseteq E'$ для каждого $e' \in E'$. Для такого подмножества будем использовать обозначение $Cut(E') = (E'^\bullet \cup {}^\bullet TN) \setminus {}^\bullet E'$. $E' \subseteq E$ — *непротиворечивое по времени* подмножество $E$, если $\tau(e') \le \tau(e)$ для всех $e' \in E'$ и $e \in E \setminus E'$;
- $e \smile e' \iff \neg((e \prec e') \vee (e' \prec e))$. Множество $\emptyset \ne V \subseteq E$ называется *шагом*, если $e \smile e'$ и $\tau(e) = \tau(e')$ для всех пар $e \ne e' \in V$. Обозначим $\downarrow V = \{\downarrow e \subseteq E \mid e \in V\}$ множество предшественников $V$ и $\tau(V) = \tau(e)$ для произвольного $e \in V$ — время $V$. Тогда шаг $V$ является предшественником шага $V'(V \prec V')$, если выполняется $(\downarrow V' \cap V \ne \emptyset) \wedge (V' \cap \downarrow V = \emptyset)$; $V \smile V' \iff (\downarrow V \cap V' = \emptyset) \wedge (V \cap \downarrow V' = \emptyset)$.
- последовательность шагов $\rho = V_1 \dots V_k$ $(k \ge 0)$ — *s-линеаризация* $TN$, если $\bigcup_{1\le i\le k} V_i = E$, $\sum_{1\le i\le k} \mid V_i \mid = \mid E \mid$ и для всех $V_i, V_j$ $(1 \le i, j \le k,\ i \ne j)$ выполняются условия: $((V_i \smile V_j) \vee (V_i \prec V_j) \vee (V_i \prec V_j))$ и $((V_i \prec V_j \vee \tau(V_i) < \tau(V_j)) \Rightarrow i < j)$. Очевидно, $\tau(V_k) = \tau(TN)$.
- $s$-линеаризация $TN$ $\rho = V_1 \dots V_k$ $(k \ge 0)$ — будет *i-линеаризацией* $TN$, если для всех $1 \le i \le k \mid V_i \mid = 1$, т.е. $\rho = e_1 \dots e_k$.

Заметим, что $\eta(TN) = (E_{TN}, \preceq_{TN} \cap (E_{TN} \times E_{TN}), l_{TN}, \tau_{TN})$ является *(помеченным над Act) временным частично-упорядоченным множеством (ВЧУМ)*[2].

Обозначим через $\mathcal{TP}(Act)$ множество всех ВЧУМов, помеченных над $Act$, $\mathcal{TP}_e(Act) = \{\eta \in \mathcal{TP}(Act) \mid \mid X_\eta \mid = 1\}$ — подмножество ВЧУМов, состоящих из одного элемента.

[2](Помеченый над $Act$) ВЧУМ — это набор $\eta = (X, \preceq, \lambda, \tau)$, состоящий из конечного множества элементов $X$; рефлексивного, антисимметричного и транзитивного отношения $\preceq$; помечающей функции $\lambda : X \to Act$ и временной функции $\tau : X \to \mathbb{T}$ такой, что $e \preceq e' \Rightarrow \tau(e) \le \tau(e')$. Пусть $\tau(\eta) = \max\{\tau(x) \mid x \in X\}$.

Для двух ВЧУМов $\eta = (E, \preceq, l, \tau)$ и $\eta' = (E', \preceq', l', \tau') \in \mathcal{TP}(Act)$ и $\star, * \in \{i, s, p\}$ введем дополнительные обозначения: $\eta'$ является $*$-*расширением* $\eta$ ($\eta \sqsubset_* \eta'$), если $E$ — левозамкнутое и непротиворечивое по времени подмножество $E'$, $\preceq = \preceq' \cap (E \times E)$, $l = l' \mid_E$; если $* = i$, то должно выполняться дополнительное требование $\mid E' \setminus E \mid = 1$, и если $* = s$, то для любой пары $e \neq e' \in E' \setminus E$ должно выполняться $e \smile e'$. Тогда для отношения $\sqsubset_{\star - *} = (\sqsubset_* \cup \sqsubset_\star^{-1})\ \eta'$ является $\star - *$-*расширением* $\eta$. Обозначим через $\mathcal{TP}^{\sqsubset}_{\star - *}(Act) = \{P_1 P_2 \dots P_k, 0 \leq k \mid P_j \in \mathcal{TP}(Act)(1 \leq j \leq k), P_j \sqsubset_{\star - *} P_{j+1}, 1 \leq j < k)\}$ множество последовательностей ВЧУМов, в которых соседние ВЧУМы являются $\star - *$-расширениями.

Временные причинные сети $TN = (B, E, G, l, \tau)$ и $TN' = (B', E', G', l', \tau')$ *изоморфны* (обозначается $TN \simeq TN'$), если существует биективное отображение $\beta : B \cup E \to B' \cup E'$ такое, что: (а) $\beta(B) = B'$ и $\beta(E) = E'$; (б) $x\ G\ y \iff \beta(x)\ G'\ \beta(y)$ для всех $x, y \in B \cup E$; (в) $l(e) = l'(\beta(e))$ и $\tau(e) = \tau'(\beta(e))$ для всех $e \in E$. Кроме того, будем говорить,что для $* \in \{i, s, p\}$ $TN'$ является $*$-расширением $TN$ (обозначается $TN \longrightarrow_* TN'$), если $\eta(TN')$ является $*$-расширением $\eta(TN)$, $B \subseteq B'$ и $G = G' \cap (B \times E \cup E \times B)$.

**Пример 2.** На рис. 2 показана временная причинная сеть $TN = (B,\ E,\ G,\ l,\ \tau)$, где условия представлены окружностями, а события — барьерами; рядом с элементами сети размещены их имена; между элементами, включенными в отношение инцидентности, изображены стрелки; значения функций $l$ и $\tau$ указаны рядом с событиями. Определим временные причинные сети $TN' = (B',\ E',\ G',\ l',\ \tau')$, где $B' = \{b_1, b_2\}$, $E' = \emptyset$, $G' = \emptyset$, $l' = \emptyset$, $\tau' = \emptyset$ и $TN'' = (B'',\ E'',\ G'',\ l'',\ \tau'')$, где $B'' = \{b_1, b_2, b_3, b_4\}$, $E'' = \{e_1, e_3\}$, $G'' = G \cap (B'' \times E'' \cup E'' \times B'')\}$, $l'' = l \mid_{E''}$, $\tau'' = \tau \mid_{E''}$. Легко проверить, что $TN''$ является $s$-расширением $TN'$. □

## 3.2. Временные причинные сети-процессы временных сетей Петри

В этом разделе рассмотрим понятие временных причинных сетей-процессов ВСП, предложенное в статье [6], и использованное при исследовании временных тестовых эквивалентностей в работах [1, 10].

**Определение 4.** Пусть $\mathcal{TN} = ((P,\ T,\ F,\ M_0,\ L),\ D)$ — ВСП и $TN = (B, E, G, l, \tau)$ — временная причинная сеть. Отображение $\varphi : B \cup E \to P \cup T$ называется *гомоморфизмом из* $TN$ *в* $\mathcal{TN}$, если выполняются следующие условия:

- $\varphi(B) \subseteq P$, $\varphi(E) \subseteq T$;
- ограничение $\varphi$ на ${}^\bullet e$ является биекцией между ${}^\bullet e$ и ${}^\bullet\varphi(e)$ и ограничение $\varphi$ на $e^\bullet$ является биекцией между $e^\bullet$ и $\varphi(e)^\bullet$ для всех $e \in E$;
- ограничение $\varphi$ на ${}^\bullet TN$ является биекцией между ${}^\bullet TN$ и $M_0$;
- $l(e) = L(\varphi(e))$ для всех $e \in E$.

Пара $\pi = (TN, \varphi)$ называется *временным причинным сетью-процессом ВСП* $\mathcal{TN}$, если $TN$ — временная причинная сеть и $\varphi$ — гомоморфизм из $TN$ в $\mathcal{TN}$.

Пусть $\pi = (TN, \varphi)$ — временной причинный сеть-процесс ВСП $\mathcal{TN}$, $B' \subseteq B_{TN}$ и $t \in En(\varphi(B'))$. Тогда глобальный момент времени, когда фишки появляются во всех входных местах перехода $t$, определяется следующим образом: $\mathbf{TOE}_\pi(B', t) = \max\Big(\{\tau_{TN}({}^\bullet b) \mid b \in B'_{[t]} \setminus {}^\bullet TN\} \cup \{0\}\Big)$, где $B'_{[t]} = \{b \in B' \mid \varphi_{TN}(b) \in {}^\bullet t\}$.

Для того, чтобы значения временных функций временных причинных сетей-процессов ВСП соответствовали временным интервалам срабатывания сетевых переходов, вводится понятие корректных временных причинных сетей-процессов ВСП.

**Определение 5.** Временной причинный сеть-процесс $\pi = (TN, \varphi)$ ВСП $\mathcal{TN}$ называется *корректным*, если для каждого $e \in E$ выполняются следующие условия:

- $\tau(e) \geq \mathbf{TOE}_\pi({}^\bullet e, \varphi(e)) + Eft(\varphi(e))$,
- $\forall t \in En(\varphi(C_e)) \circ \tau(e) \leq \mathbf{TOE}_\pi(C_e, t) + Lft(t)$, где $C_e = Cut(Earlier(e))$.

Пусть $\mathcal{CP}(\mathcal{TN})$ — множество корректных временных причинных сетей-процессов ВСП $\mathcal{TN}$, а $\pi_0 = (TN_0 = (B_0, \emptyset, \emptyset, \emptyset, \emptyset), \varphi_0) \in \mathcal{CP}(\mathcal{TN})$ с $\varphi_0(B_0) = M_0$ — начальный временной причинный сеть-процесс ВСП $\mathcal{TN}$.

Через $\mathcal{TP}os(\mathcal{TN}) = \{TP \mid \exists \pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})\colon\ TP \simeq^3 \eta(TN)\}$ обозначим множество ВЧУМов, изоморфных ВЧУМам, полученным из корректных временных причинных сетей-процессов ВСП $\mathcal{TN}$.

**Пример 3.** Определим отображение $\varphi$ из временной причинной сети $TN$ (см. рис. 2) в ВСП $\mathcal{TN}$ (см. рис. 1) следующим образом: $\varphi(b_i) = p_i$ $(1 \leq i \leq 6)$, $\varphi(b_i) = p_{i-6}$ $(7 \leq i \leq 10)$ и $\varphi(e_i) = t_i$ $(1 \leq i \leq 5)$, $\varphi(e_6) = t_1$, $\varphi(e_7) = t_3$. Легко видеть, что $\pi = (TN, \varphi)$ является корректным временным причинным сетью-процессом ВСП $\mathcal{TN}$. □

Будем говорить, что $\pi = (TN, \varphi)$ и $\pi' = (TN', \varphi')$ из $\mathcal{CP}(\mathcal{TN})$ *изоморфны* (обозначается $\pi \simeq \pi'$), если существует изоморфизм $f : TN \simeq TN'$ такой, что $\varphi(x) = \varphi'(f(x))$ для всех

[3]Два ВЧУМ $\eta = (X, \preceq, \lambda, \tau)$ и $\eta' = (X', \preceq', \lambda', \tau')$ изоморфны (обозначается $\eta \simeq \eta'$), если существует биекция $\beta : X \to X'$ такая, что (а) $x \preceq y \iff \beta(x) \preceq' \beta(y)$ для всех $x, y \in X$; (б) $\lambda(x) = \lambda'(\beta(x))$ и $\tau(x) = \tau'(\beta(x))$ для всех $x \in X$.

$x \in B \cup E$; $\pi'$ является $*$-расширением $\pi$ в $\mathcal{TN}$, (обозначается $\pi \longrightarrow_* \pi'$) ($* \in \{i, s, p\}$), если $TN \longrightarrow_* TN'$ и $\varphi = \varphi'|_{B \cup E}$.

В дальнейшем понадобятся дополнительные обозначения для расширений временных причинных сетей-процессов. Для $\pi$, $\pi' \in \mathcal{CP}(\mathcal{TN})$ будем писать:

- $\pi \xrightarrow{(a,\theta)}_i \pi'$, если $\pi'$ является $i$-расширением $\pi$ и $\{e\} = E' \setminus E$, $l_{\pi'}(e) = a$, $\tau_{\pi'}(e) = \theta$;
- $\pi \xrightarrow{(A,\theta)}_s \pi'$, если $\pi'$ является $s$-расширением и $E' \setminus E$ является шагом в $\pi'$, $A = L(\varphi(E' \setminus E))$ и $\theta = \tau(E' \setminus E)$;
- $\pi \xrightarrow{P}_p \pi'$, если $\pi'$ является $p$-расширением $\pi$ и ВЧУМ $P \sim \eta(\pi') \setminus \eta(\pi)$.

Для $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ определим функцию $FS_\pi$, которая отображает $s$-линеаризацию $\rho = V_1 \dots V_k$ $TN$ в последовательность вида: $FS_\pi(\rho) = (\varphi(V_1), \tau(V_1) - 0) \dots (\varphi(V_k), \tau(V_k) - \tau(V_{k-1}))$.

Следующие утверждение и лемма являются обобщениями результатов из [1] (Утверждение 1 и Лемма 2), которые устанавливют взаимосвязи между последовательностями срабатываний и корректными временными причинными сетями-процессами ВСП,

**Утверждение 1.** *Пусть $\mathcal{TN}$ — ВСП. Тогда*

*(а) если $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ и $\rho$ — $s(i)$-линеаризация $TN$, то существует единственная последовательность срабатываний $FS_\pi(\rho) \in \mathcal{FS}_{s(i)}(\mathcal{TN})$;*

*(б) если $\sigma \in \mathcal{FS}_{s(i)}(\mathcal{TN})$, то существует единственный (с точностью до изоморфизма) временной причинный сеть-процесс $\pi_\sigma = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ и единственная $s(i)$-линеаризация $\rho_\sigma$ $TN$ такие, что $FS_{\pi_\sigma}(\rho_\sigma) = \sigma$.*

**Лемма 1.** *Пусть $\sigma \in \mathcal{FS}_{s(i)}(\mathcal{TN})$ и $\pi \in \mathcal{CP}(\mathcal{TN})$ такие, что $\sigma = FS_\pi(\rho)$, где $\rho$ — $s(i)$-линеаризация $TN_\pi$. Тогда*

*(а) если $\sigma(U, \theta) \in \mathcal{FS}_{s(i)}(\mathcal{TN})$, то существует $\widetilde{\pi} \in \mathcal{CP}(\mathcal{TN})$ такой, что $\pi \to_{s(i)} \widetilde{\pi}$ в $\mathcal{TN}$ и $\sigma(U, \theta) = FS_{\widetilde{\pi}}(\rho V)$, где $\rho V$ — $s(i)$-линеаризация $TN_{\widetilde{\pi}}$;*

*(б) если $\pi \to_{s(i)} \widetilde{\pi}$ в $\mathcal{TN}$, то существует $\sigma(U, \theta) \in \mathcal{FS}_{s(i)}(\mathcal{TN})$ такая, что $\sigma(U, \theta) = FS_{\widetilde{\pi}}(\rho V)$, где $\rho V$ — $s(i)$-линеаризация $TN_{\widetilde{\pi}}$.*

**Пример 4.** Для временного причинного сети-процесса $\pi = (TN, \varphi)$ ВСП $\mathcal{TN}$ (см. пример 3) и $s$-линеаризации $\rho = \{e_1, e_3\}\ \{e_2\}\ \{e_7, e_6\}\ \{e_5, e_4\}$ временной причинной сети $TN$ получаем, что $FS_\pi(\rho) = (\{t_1, t_3\}, 3)\ (t_2, 2)\ (\{t_1, t_3\}, 2)\ (\{t_5, t_4\}, 2)$ является последовательностью срабатываний ВСП $\mathcal{TN}$ (см. пример 1). □

## 4. Обратимые вычисления

При исследовании поведения вычислительных систем появилась необходимость отменять часть уже выполненных действий. При введении понятий тестовых эквивалентностей с учетом обратимых действий в дальнейшем будет использоваться подход причинной обратимости. Будем определять обратимые вычисления как вычисления в которых возможна отмена выполненного действия, если действие не является причиной для других действий процесса.

Введем обозначения для множества отменяемых действий $\overleftarrow{Act} = \{\overleftarrow{a} \mid a \in Act\}$, $\overleftrightarrow{Act} = \overleftarrow{Act} \cup Act$, при этом выполняется условие что $\overleftarrow{Act} \cap Act = \emptyset$, и биекцию $[\![\overleftarrow{\cdot}]\!] : Act \to \overleftarrow{Act}$, такую что $[\![\overleftarrow{\cdot}]\!](a) = \overleftarrow{a}$.

Аналогичным образом вводятся обозначения для отменяемых ВЧУМов. $\overleftarrow{\mathcal{TP}}(Act) = \{\overleftarrow{P} \mid P \in \mathcal{TP}(Act)\}$; $\overleftarrow{\mathcal{TP}}_e(Act) = \{\overleftarrow{P} \mid P \in \mathcal{TP}_e(Act)\}$, $\overleftrightarrow{\mathcal{TP}}(Act) = \overleftarrow{\mathcal{TP}}(Act) \cup \mathcal{TP}(Act)$, и биекция $[\![\overleftarrow{\cdot}]\!]$ расширяется на $\mathcal{TP}(Act)$.

Через $\overleftrightarrow{a}$ и $\overleftrightarrow{P}$ будем обозначать элементы множеств $\overleftrightarrow{Act}$ и $\overleftrightarrow{\mathcal{TP}}(Act)$ соответственно.

Отметим, что отменяемые действия используются только в качестве обозначения действий для отмены, а не являются самостоятельными действиями. Отменяемое действие означает что для текущего процесса можно найти процесс, который может быть расширен до текущего, и расширение будет содержать событие, помеченное выбранным для отмены действием. Для простоты восприятия далее под $\overleftarrow{a}$ и $\overleftarrow{P}$ подразумеваются $[\![\overleftarrow{\cdot}]\!](a)$ и $[\![\overleftarrow{\cdot}]\!](P)$ для $a \in Act$, $P \in \mathcal{TP}(Act)$ соответственно.

Пусть $\pi = (TN, \varphi)$, $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$, $a \in Act$, $A \in Act^{\mathbb{N}}$, $\theta \in \mathbb{T}$, $P \in \mathcal{TP}(Act)$. Введем обозначения для выполнения отмены действий и обратимых вычислений.

- Если $\pi' \xrightarrow{P}_p \pi$ ($\pi' \xrightarrow{(a,\theta)}_i \pi$, $\pi' \xrightarrow{(A,\theta)}_s \pi$ ), будем писать $\pi \xrightarrow{\overleftarrow{P}}_p \pi'$ ($\pi \xrightarrow{(\overleftarrow{a},\theta)}_i \pi'$, $\pi \xrightarrow{(\overleftarrow{A},\theta)}_s \pi'$ ),
- Для $\star, * \in \{i, s, p\}$ будем писать $\pi \xrightarrow{\overleftrightarrow{P}}_{\star - *} \pi'$, если $\pi \xrightarrow{P}_* \pi'$ или $\pi' \xrightarrow{\overleftarrow{P}}_\star \pi$. Аналогичным образом определяются обозначения $\pi \xrightarrow{(\overleftrightarrow{a},\theta)}_{i-i} \pi'$ и $\pi \xrightarrow{(\overleftrightarrow{A},\theta)}_{s-s} \pi'$.

Если нет необходимости в уточнении или смысл понятен из контекста, символы $\overleftrightarrow{P}$ и/или $\star, *$ будут опускаться.

Если $\pi \xrightarrow{\overleftrightarrow{P}} \pi'$, обозначим $E' \between E = \begin{cases} E' \setminus E, \text{ если } \pi \xrightarrow{P} \pi', \\ E \setminus E', \text{ если } \pi \xrightarrow{\overleftarrow{P}} \pi'. \end{cases}$

Кроме того, введем обозначения для вычислений во временных сетях-процессах с сохранением истории. Пусть $TP = P_1 P_2 \dots P_k \in \mathcal{TP}^{\sqsubset}_{\star - *}(Act)$ при $\star, * \in \{i, s, p\}$, тогда $\pi \overset{TP}{\hookrightarrow}_{\star - *} \pi'$, если существуют $\pi_j \in \mathcal{CP}(\mathcal{TN})$ $(1 \le j \le k)$ такие что $\pi_0 \xrightarrow{P_j} \pi_j$, $\pi' = \pi_k$ и $\pi_{j-1} \longrightarrow_{\star - *} \pi_j$.

В дальнейшем, при определении тестовых эквивалентностей с обратимостью понадобятся обозначения отменяемых действий в последовательностях срабатываний в ВСП $\mathcal{TN}$. Для обратимых вычислений в корректных временных сетях-процессах введем связанные с ними обозначения для последовательностях срабатываний.

Пусть $\star, * \in \{i, s\}$. Если существуют $\pi_j \in \mathcal{CP}(\mathcal{TN})(1 \le j \le k)$ такие, что $\pi_0 \xrightarrow{(A_1,\theta_1)} \pi_1 \xrightarrow{(\overleftrightarrow{A}_2,\theta_1)} \pi_2 \ldots \pi_{k-1} \xrightarrow{(\overleftrightarrow{A}_k,\theta_k)} \pi_k$ обозначим $U_j = \varphi(E_j \langle\!\rangle E_{j-1})$, где $E_l = E_{TN_{\pi_l}}$ $(0 \le l \le k)$. Тогда последовательность $\overleftrightarrow{\sigma} = U_1 \overleftrightarrow{U_2} \ldots \overleftrightarrow{U_k}$ с $L(\overleftrightarrow{\sigma}) = A_1(\theta_1)\ \overleftrightarrow{A_2}(\theta_2) \ldots\ \overleftrightarrow{A_k}(\theta_k)$ будем называть обратимой $\star - *$-последовательностью срабатываний $\mathcal{TN}$.

Обозначим через $\mathcal{FS}_{\star-*}(\mathcal{TN})$ множество обратимых $\star - *$-последовательностей срабатываний в ВСП $\mathcal{TN}$ из $S_0$. Определим *обратимые* языки ВСП $\mathcal{TN}$, $\star, * \in \{i, s\}$ следующим образом:

$\mathcal{L}_{\star-*}(\mathcal{TN}) = \{L(\overleftrightarrow{\sigma}) \mid \overleftrightarrow{\sigma} \in \mathcal{FS}_{\star-*}(\mathcal{TN})\}$;

$\mathcal{TP}om_{i-p}(\mathcal{TN}) = \{\overleftrightarrow{P}_1 \overleftrightarrow{P}_2 \ldots \overleftrightarrow{P}_k 0 \le k \mid P_j \in \mathcal{TP}(Act) \cup \overleftarrow{\mathcal{TP}}_e(Act)(1 \le j < k),\ \exists \pi_j (1 \le j \le k) \in \mathcal{CP}(\mathcal{TN})\ \pi_l \xrightarrow{\overleftrightarrow{P}_l}_{i-p} \pi_{l+1},\ 0 \le l < k)\}$;

$\mathcal{TP}os_{i-p}(\mathcal{TN}) = \{TP = P_1 P_2 \ldots P_k \in \mathcal{TP}^{\sqsubset}_{i-p}(Act)(0 \le k) \mid P_j \in \mathcal{TP}os(\mathcal{TN}),\ \exists \pi \in \mathcal{CP}(\mathcal{TN})\ \pi_0 \overset{TP}{\hookrightarrow}_{i-p} \pi (1 \le j \le k)\}$.

## 5. Тестовые эквивалентности

При интерливинговом и шаговом подходах к определению тестовой эквивалентности в качестве тестов рассматриваются последовательности $w$ выполняемых действий или мультмножеств независимых действий (вычисления процесса) и множества $W$ возможных дальнейших действий или шагов. Процесс проходит тест, если после выполнения каждой последовательности $w$ дальше может выполниться хотя бы один из элементов из $W$. Два процесса тестово эквивалентны, если они проходят одно и то же множество тестов. Во временном варианте добавляется информация о временах выполнения действий. Для сравнения поведения процессов с обратимыми вычислениями в тестовые последовательности включаются отменяемые действия.

**Определение 6.** Пусть $\mathcal{TN}$ и $\mathcal{TN}'$ — ВСП, $\star, * \in \{i, s\}$. Для последовательности $w \in (Act^{\mathbb{N}} \times \mathbb{T})^*$ $(w \in (\overleftrightarrow{Act}^{\mathbb{N}} \times \mathbb{T})^*)$ и множества $W \subseteq Act^{\mathbb{N}} \times \mathbb{T}$, $(W \subseteq \overleftrightarrow{Act}^{\mathbb{N}} \times \mathbb{T})$, $\mathcal{TN}$ **after** $w$ $\mathbf{MUST}_{*(\star-*)}$ $W$, если для любой $\sigma \in \mathcal{FS}_*(\mathcal{TN})$ $(\overleftrightarrow{\sigma} \in \mathcal{FS}_{\star-*}(\mathcal{TN}))$ такой, что $L(\sigma) = w$ $(L(\overleftrightarrow{\sigma}) = w)$, существуют $(A, \theta) \in W$ и $\sigma(U, \theta) \in \mathcal{FS}_*(\mathcal{TN})$ $((\overleftrightarrow{A}, \theta) \in W$ и $\overleftrightarrow{\sigma}(\overleftrightarrow{U}, \theta) \in \mathcal{FS}_{\star-*}(\mathcal{TN}))$ такие, что $L(\sigma(U, \theta)) = w\ (A, \theta)$ $(L(\overleftrightarrow{\sigma}(\overleftrightarrow{U}, \theta)) = w\ (\overleftrightarrow{A}, \theta))$.

$\mathcal{TN}$ и $\mathcal{TN}'$ называются $*-(\star-*)$-*тестово эквивалентными* (обозначается $\mathcal{TN} \sim_* TN'$ ($\mathcal{TN} \sim_{\star-*} TN'$)), если для любой последовательности $w \in (Act^{\mathbb{N}} \times \mathbb{T})^*$ ($w \in (\overleftrightarrow{Act}^{\mathbb{N}} \times \mathbb{T})^*$) и любого множества $W \subseteq Act^{\mathbb{N}} \times \mathbb{T}$ ($W \subseteq \overleftrightarrow{Act}^{\mathbb{N}} \times \mathbb{T}$), $\mathcal{TN}$ **after** $w$ $\mathbf{MUST}_{*(\star-*)}$ $W \iff \mathcal{TN}'$ **after** $w$ $\mathbf{MUST}_{*(\star-*)}$ $W$.

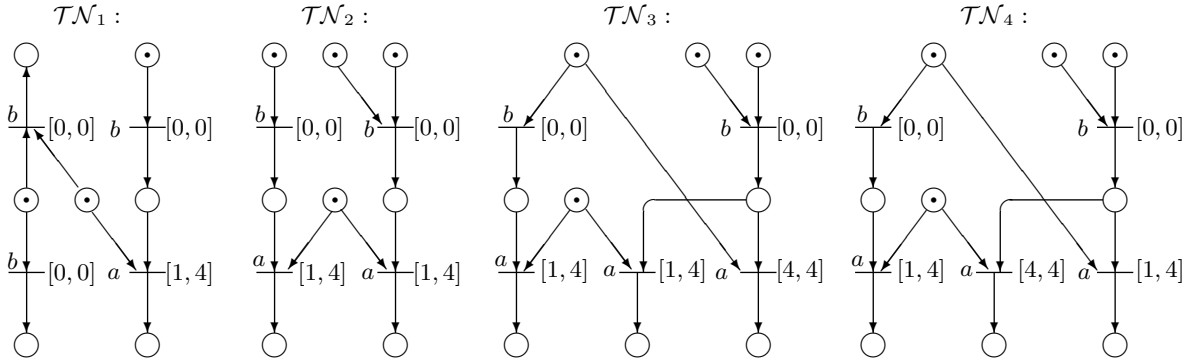



*Рис. 3.*

**Пример 5.** ВСП $\mathcal{TN}_2$, $\mathcal{TN}_3$ и $\mathcal{TN}_4$, изображенные на рис. 3, $i$-тестово эквивалентны, тогда как $\mathcal{TN}_1$ и $\mathcal{TN}_2$ не являются таковыми. Легко проверить, что $\mathcal{TN}_2$ **after** $w = (b,0)(b,0)$ $\mathbf{MUST}_i$ $W = \{(a, 3.9)\}$. Однако в $\mathcal{FS}_i(\mathcal{TN}_1)$ существует последовательность срабатываний, которая помечена $w$ и после которой невозможно срабатывание перехода, помеченного $a$, в относительный момент времени 3.9. Таким образом, не выполняется $\mathcal{TN}_1$ **after** $w$ $\mathbf{MUST}_i$ $W$. □

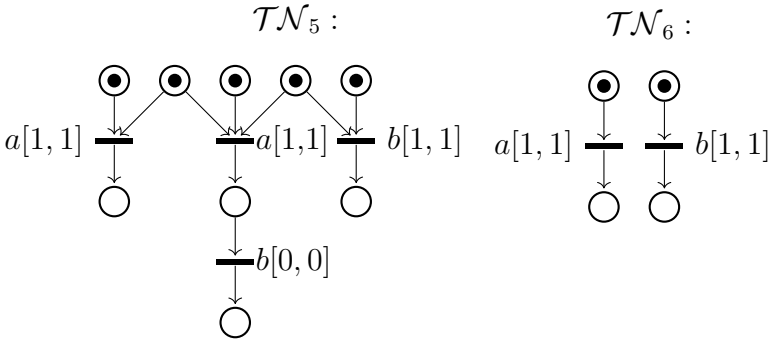



*Рис. 4.* Пример $s$-, но не $i-i$-тестово эквивалентных ВСП.

**Пример 6.** Рассмотрим ВСП $\mathcal{TN}_5$ и $\mathcal{TN}_6$, изображенные на рис. 4. ВСП $i$- и $s$-тестово эквивалентны, но не $i-i$-тестово эквивалентны.

Нетрудно проверить, что $\mathcal{TN}_6$ **after** $w = (a,1)(b,0)$ $\mathbf{MUST}_{i-i}$ $W = \{(\overleftarrow{a}, 1)\}$. Но в $\mathcal{FS}_{i-i}(\mathcal{TN}_5)$ существует обратимая последовательность срабатываний, которая помечена $w$ и в которой невозможна отмена действия $(a,1)$. Таким образом, не выполняется $\mathcal{TN}_5$ **after** $w$ $\mathbf{MUST}_{i-i}$ $W$. □

Исследования тестовых эквивалентностей в семантике частичного порядка были впервые проведены Асето и др. в статье [4] в контексте моделей структур событий. При таком подходе в качестве вычислений процесса рассматриваются частично-упорядоченные мультимножества выполняемых действий (ЧУММы), и вместо множеств дальнейших действий могут использоваться непосредственные расширения выполняемых ЧУММов ([17]). Также, в качестве вычислений использовались и ЧУМы выполняемых действий, как в работе [17] при определении одной из версий причинной тестовой эквивалентности.

Далее определяются временные тестовые эквивалентности в семантике частичного порядка для ВСП с использованием ее корректных временных причинных сетей-процессов. Сначала рассмотрим тестовые эквивалентности со слабо-сохраняющей историю семантикой.

**Определение 7.** Пусть $\mathcal{TN}$ и $\mathcal{TN}'$ — ВСП. Для последовательности $w \in (Act \times \mathbb{T})^*$ ($w \in (\overleftrightarrow{Act} \times \mathbb{T})^*$) и множества **TP** ВЧУМов $\mathcal{TN}$ **after** $w$ $\mathbf{MUST}_{wh(i-wh)}$ **TP**, если для любого $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ такого что $\pi_0 \xrightarrow{w} \pi$ существуют $TP' \in \mathbf{TP}$ и $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$ такие, что $\pi \longrightarrow_{p(i-p)} \pi'$ и $\eta(TN') \sim TP'$.

$\mathcal{TN}$ и $\mathcal{TN}'$ называются *$wh$–$(i-wh)$-тестово эквивалентными* (обозначается $\mathcal{TN} \sim_{i-wh} \mathcal{TN}'$), если для любой последовательности $w \in (Act \times \mathbb{T})^*$ ($w \in (\overleftrightarrow{Act} \times \mathbb{T})^*$) и любого множества **TP** ВЧУМов выполняется условие: $\mathcal{TN}$ **after** $w$ $\mathbf{MUST}_{wh(i-wh)}$ $\mathbf{TP}'$ $\iff$ $\mathcal{TN}'$ **after** $w$ $\mathbf{MUST}_{wh(i-wh)}$ $\mathbf{TP}'$.

Далее для ВЧУМ $TP \in \mathcal{TP}(Act)$ будем обозначать $\mathbf{TP}^p_{TP} \subseteq \mathcal{TP}(Act)$ множество $i$-расширений $TP$, $\mathbf{TP}^{i-p}_{TP} \subseteq \mathcal{TP}(Act)$ множество $i-p$-расширений $TP$.

**Определение 8.** Пусть $\mathcal{TN}$ и $\mathcal{TN}'$ — ВСП. Для последовательности ВЧУМов $TP = P_1 \dots P_k \in \mathcal{TP}(Act)^*$ ($TP = \overleftrightarrow{P}_1 \dots \overleftrightarrow{P}_k \in (\mathcal{TP}(Act) \cup \overleftarrow{\mathcal{TP}}_e(Act))^*$) и множества $\mathbf{TP} \subseteq \mathcal{TP}(Act)$ ($\mathbf{TP} \subseteq \mathcal{TP}(Act) \cup \overleftarrow{\mathcal{TP}}_e(Act)$) $\mathcal{TN}$ **after** $TP$ $\mathbf{MUST}_{pom(i-pom)}$ **TP**, если для любого $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ такого что $\pi \xrightarrow{TP} \pi'$ ($\pi_0 \xrightarrow{TP}_{i-p} \pi'$) существуют $TP' \in \mathbf{TP}$ и $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$ такие, что $\pi \xrightarrow{TP'} \pi'$ ($\pi \xrightarrow{TP'}_{i-p} \pi'$).

$\mathcal{TN}$ и $\mathcal{TN}'$ называются *$pom-(i-pom)$-тестово эквивалентными* (обозначается $\mathcal{TN} \sim_{pom(i-pom)} TN'$), если для любой последовательности ВЧУМов $TP = P_1 \dots P_k \in \mathcal{TP}(Act)^*$ ($TP = \overleftrightarrow{P}_1 \dots \overleftrightarrow{P}_k \in (\mathcal{TP}(Act) \cup \overleftarrow{TP}_e(Act))^*$) и множества $\mathbf{TP} \subseteq \mathcal{TP}(Act)$ ($\mathbf{TP} \subseteq \mathcal{TP}(Act) \cup \overleftarrow{TP}_e(Act)$) выполняется условие: $\mathcal{TN}$ **after** $TP$ $\mathbf{MUST}_{pom(i-pom)}$ $\mathbf{TP}'$ $\iff$ $\mathcal{TN}'$ **after** $TP$ $\mathbf{MUST}_{pom(i-pom)}$ $\mathbf{TP}'$.

**Пример 7.** ВСП $\mathcal{TN}_7$ и $\mathcal{TN}_8$, изображенные на рис. 5, *pom*-тестово эквивалентны. Проверим, что они не $i-pom$-тестово эквивалентны.

Определим ВЧУМы $TP_1 = (\{x_1, x_2\}, \preceq_1, \lambda_1, \tau_1)$, где $\preceq_1 = \{(x_i, x_i) \mid 1 \le i \le 2\} \cup \{(x_1, x_2)\}$, $\lambda_1(x_1) = a$, $\lambda(x_2) = b$, $\tau_1(x_1) = \tau'(x_2) = 1$; $TP_2 = (\{x_1\}, \preceq_2, \lambda_2, \tau_2)$, где $\preceq_2 = \{(x_1, x_1)\}$, $\lambda_2(x_1) = b$, $\tau_2(x_1) = 1$; и ВЧУМ $TP' = (\{x_3\}, \preceq_3, \lambda_3, \tau_3)$, где $\preceq_3 = \{(x_3, x_3)\}$, $\lambda_3(x_3) = c$, $\tau_3(x_3) = 1$. Для единственного временного причинного сети-процесса $\pi_8^1 = (TN_8^1, \varphi_8^1) \in \mathcal{CP}(\mathcal{TN}_8)$, в котором $E_{TN_8^1}$ состоит из двух событий с пометками $a$ и $b$, находящихся в причинной зависимости, существует временной причинный сеть-процесс $\pi_8^2 = (TN_8^2, \varphi_8^2) \in \mathcal{CP}(\mathcal{TN}_8)$, в котором $E_{TN_8^2}$ состоит из события с пометкой $a$, такой что $\pi_8^2 \xrightarrow{TP_2}_i \pi_8^1$, т.е. $\pi_8^0 \xrightarrow{TP_1}_{i-p} \pi_8^1 \xrightarrow{\overleftarrow{TP_2}}_{i-p} \pi_8^2$. И для $\pi_8^2$ существует временной причинный сеть-процесс $\pi_8^3 = (TN_8^3, \varphi_8^3) \in \mathcal{CP}(\mathcal{TN}_8)$, в котором $E_{TN_8^3}$ состоит из двух событий с пометками $a$ и $c$, такой, что $\pi_8^2 \xrightarrow{TP'}_i \pi_8^3$. Однако, в случае ВСП $\mathcal{TN}_7$ не верно,что $\mathcal{TN}_7$ **after** $TP_1\overleftarrow{TP_2}$ $\mathbf{MUST}_{i-pom}$ $\{TP'\}$, а именно для временного сети-процесса, элементы которого отображаются в элементы ВСП $\mathcal{TN}_7$, выделенные на рисунке красным цветом.

Также легко убедиться, что ВСП $\mathcal{TN}_2$ и $\mathcal{TN}_4$ (рис. 3) являются $i-wh$-, но не *pom*-тестово эквивалентными.

□

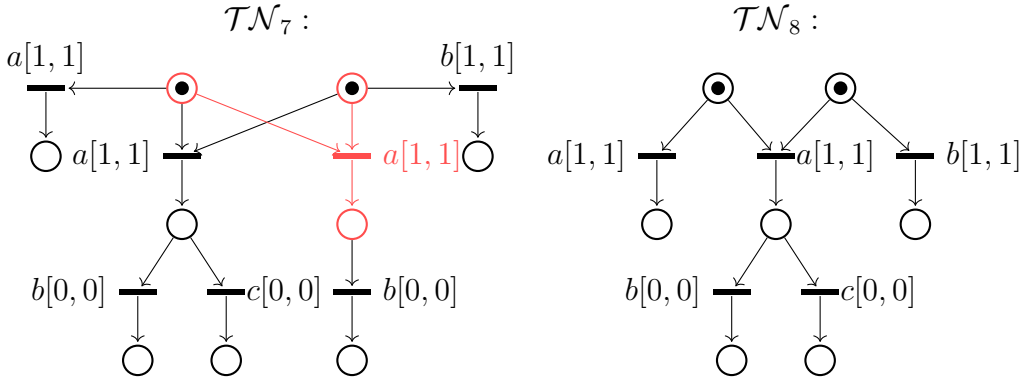



*Рис. 5.* Пример $i-i$-, *pom*-, не $wh$-, не $i-pom$-тестово эквивалентных ВСП.

**Определение 9.** Пусть $\mathcal{TN}$ и $\mathcal{TN}'$ — ВСП. Для ВЧУМ $TP$ и множества **TP** ВЧУМов такого, что $\mathbf{TP} \subseteq \mathbf{TP}^p_{TP}$, $\mathcal{TN}$ **after** $TP$ $\mathbf{MUST}_{pos}$ **TP**, если для любого $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$ и для любого изоморфизма $f : \eta(TN) \longrightarrow TP$ существуют $TP' \in \mathbf{TP}$, $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$ и изоморфизм $f' : \eta(TN') \longrightarrow TP'$ такие, что $\pi \to_p \pi'$ и $f \subseteq f'$.

$\mathcal{TN}$ и $\mathcal{TN}'$ называются *pos-тестово эквивалентными* (обозначается $\mathcal{TN} \sim_{pos} TN'$), если для любого ВЧУМ $TP$ и любого множества **TP** ВЧУМов такого, что $\mathbf{TP} \subseteq \mathbf{TP}^p_{TP}$, выполняется условие: $\mathcal{TN}$ **after** $TP$ $\mathbf{MUST}_{pos}$ $\mathbf{TP}'$ $\iff$ $\mathcal{TN}'$ **after** $TP$ $\mathbf{MUST}_{pos}$ $\mathbf{TP}'$.

Введем обозначение для двух изоморфизмов: $f \gtrless f'$, если $f \subset f'$ или $f' \subset f$.

**Определение 10.** Пусть $\mathcal{TN}$ и $\mathcal{TN}'$ — ВСП. Для последовательности ВЧУМов $TP = P_1 \dots P_k \in \mathcal{TP}^{\sqsubset}_{i-p}(Act)$ и множества ВЧУМов **TP** такого, что $\mathbf{TP} \subseteq \mathbf{TP}^{i-p}_{P_k}$, $\mathcal{TN}$ **after** $TP$ $\mathbf{MUST}_{i-pos}$ **TP**, если для любого $\pi = (TN, \varphi) \in \mathcal{CP}(\mathcal{TN})$, такого что $\pi_0 \overset{TP}{\hookrightarrow}_{i-p} \pi'$, и для любого изоморфизма $f : \eta(TN) \longrightarrow P_k$ существуют $TP' \in \mathbf{TP}$, $\pi' = (TN', \varphi') \in \mathcal{CP}(\mathcal{TN})$ и изоморфизм $f' : \eta(TN') \longrightarrow TP'$ такие, что $\pi \longrightarrow_{i-p} \pi'$ и $f \gtrless f'$.

$\mathcal{TN}$ и $\mathcal{TN}'$ называются *$i-pos$-тестово эквивалентными* (обозначается $\mathcal{TN} \sim_{i-pos} TN'$), если для любой последовательности ВЧУМов $TP = P_1 \dots P_k \in \mathcal{TP}^{\sqsubset}_{i-p}(Act)$ и любого множества **TP** ВЧУМов такого, что $\mathbf{TP} \subseteq \mathbf{TP}^{i-p}_{P_k}$, выполняется условие: $\mathcal{TN}$ **after** $TP$ $\mathbf{MUST}_{i-pos}$ **TP** $\iff$ $\mathcal{TN}'$ **after** $TP$ $\mathbf{MUST}_{i-pos}$ **TP**.

**Пример 8.** Рассмотрим ВСП $\mathcal{TN}_2$, $\mathcal{TN}_3$ и $\mathcal{TN}_4$, изображенные на рис. 3. Легко проверить, что $\mathcal{TN}_2$ и $\mathcal{TN}_3$ *pos*-тестово эквивалентны, тогда как $\mathcal{TN}_3$ и $\mathcal{TN}_4$ не являются таковыми. Убедимся в последнем. Определим ВЧУМ $TP = (\{x_1, x_2\}, \preceq, \lambda, \tau)$, где $\preceq = \{(x_i, x_i) \mid 1 \le i \le 2\}$, $\lambda(x_1) = \lambda(x_2) = b$, $\tau(x_1) = \tau'(x_2) = 0$; и ВЧУМ $TP' = (\{x_1, x_2, x_3\}, \preceq', \lambda', \tau')$, где $\preceq' = \{(x_i, x_i) \mid 1 \le i \le 3\} \cup \{(x_2, x_3)\}$, $\lambda'(x_1) = \lambda'(x_2) = b$, $\lambda'(x_3) = a$, $\tau'(x_1) = \tau'(x_2) = 0$ и $\tau'(x_3) = 3.9$. Для любого временного причинного сети-процесса $\pi_3 = (TN_3, \varphi_3) \in \mathcal{CP}(\mathcal{TN}_3)$, в котором $E_{TN_3}$ состоит из двух параллельных событий с пометками $b$ и временными значениями, равными 0, и для любого изоморфизма $f_3 \colon \eta(TN_3) \longrightarrow TP$ можно найти временной причинный сеть-процесс $\pi'_3 = (TN'_3, \varphi'_3) \in \mathcal{CP}(\mathcal{TN}_3)$, в котором $E_{TN'_3}$ состоит из двух параллельных событий с пометками $b$ и временными значениями 0 и третьего события с пометкой $a$ и временным значением 3.9, находящегося в отношении причинной зависимости с одним из $b$, и изоморфизм $f'_3 \colon \eta(TN'_3) \longrightarrow TP'$ такие, что $\pi_3 \to_p \pi'_3$ и $f_3 \subset f'_3$. Однако, это не так в случае ВСП $\mathcal{TN}_4$. Таким образом, $\mathcal{TN}_3$ **after** $TP$ $\mathbf{MUST}_{pos}$ $\{TP'\}$, но не верно, что $\mathcal{TN}_4$ **after** $TP$ $\mathbf{MUST}_{pos}$ $\{TP'\}$.

□

## 6. Взаимосвязи тестовых эквивалентностей

Следующая лемма устанавливает взаимосвязь между совпадением обратимых языков ВЧУМов для ВСП с наличием между ними временных тестовых эквивалентностей с интерливинговой и шаговой обратимостью.

**Лемма 2.** *Пусть $\mathcal{TN}_1$ и $\mathcal{TN}_2$ — ВСП, $\star, * \in \{i, s\}$. Тогда*

$$\mathcal{TN}_1 \sim_{\star-*} \mathcal{TN}_2 \Rightarrow \mathcal{L}_{\star-*}(\mathcal{TN}_1) = \mathcal{L}_{\star-*}(\mathcal{TN}_2),$$

$$\mathcal{TN}_1 \sim_{pom} \mathcal{TN}_2 \Rightarrow \mathcal{TP}os(\mathcal{TN}_1) = \mathcal{TP}os(\mathcal{TN}_2),$$
$$\mathcal{TN}_1 \sim_{i-pom} \mathcal{TN}_2 \Rightarrow \mathcal{TP}om_{i-p}(\mathcal{TN}_1) = \mathcal{TP}om_{i-p}(\mathcal{TN}_2),$$
$$\mathcal{TN}_1 \sim_{i-pos} \mathcal{TN}_2 \Rightarrow \mathcal{TP}os_{i-p}(\mathcal{TN}_1) = \mathcal{TP}os_{i-p}(\mathcal{TN}_2).$$

Далее представлена иерархия введенных временных тестовых эквивалентностей в различных семантиках без обратимости и с обратимостью.

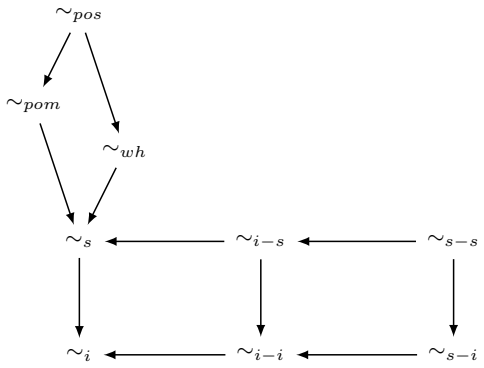



*Рис. 6.*

**Теорема 1.** *Пусть* $\star, * \in \{i, s, pos, pom, wh\}$ *и* $\dagger, \ddagger \in \{i, s, \circ\}$, *где* $\circ$ *означает "пусто". Если* $\sim_{\dagger-*}$ *и* $\sim_{\dagger-*}$ *представлены на рис. 6, то для ВСП* $\mathcal{TN}_1$ *and* $\mathcal{TN}_2$ *выполняется:* $\mathcal{TN}_1 \sim_{\dagger-*} \mathcal{TN}_2 \implies \mathcal{TN}_1 \sim_{\ddagger-\star} \mathcal{TN}_2 \iff$ *существует путь из* $\sim_{\dagger-*}$ *в* $\sim_{\ddagger-\star}$ *на рис. 6.*

*Доказательство.* ($\Longleftarrow$) Очевидным образом следует из определений временных тестовых эквивалентностей.

($\Longrightarrow$) Для доказательства отсутствия на рис. 6 других прямых стрелок и стрелок в обратном направлении рассмотрим следующие контрпримеры.

1) ВСП $\mathcal{TN}_5$ и $\mathcal{TN}_6$, рассмотренные ранее в примере 6 $s$-тестово эквивалентны, и легко проверить, что данные ВСП ни $i-i$-, ни $pom$-, ни $pos$-тестово эквивалентны.

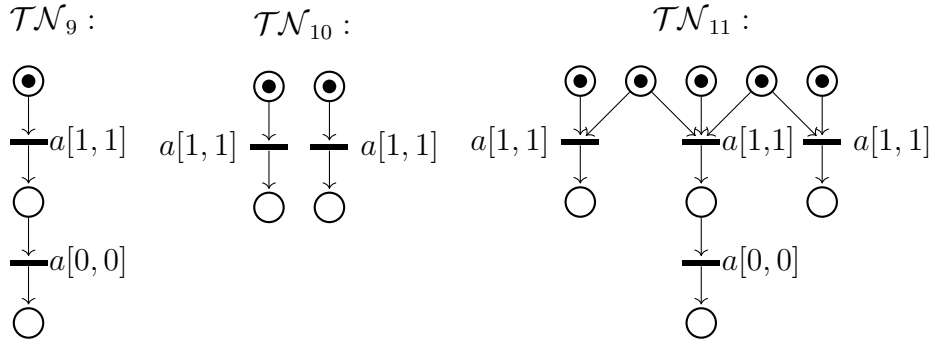



*Рис. 7.* Примеры $i-i$-тестово эквивалентных ВСП.

2) ВСП $\mathcal{TN}_9$ и $\mathcal{TN}_{10}$, изображенные на рис. 7, $i-i$-тестово эквивалентны, но ни $s$-, ни $s-i$-тестово эквивалентны. Нетрудно проверить, что $\mathcal{TN}_{10}$ **after** $w = (a,1)(a,0)$ $\mathbf{MUST}_{s-i}$

$W = \{([\overleftarrow{a:2}], 1)\}$. Однако в $\mathcal{TN}_9$ после последовательности срабатываний, помеченной $w$, невозможно срабатывание обратного шага, помеченного $([\overleftarrow{a:2}], 1)$. Таким образом, не выполняется $\mathcal{TN}_9$ **after** $w$ $\mathbf{MUST}_{s-i}$ $W$.

3) ВСП $\mathcal{TN}_{10}$ и $\mathcal{TN}_{11}$, изображенные на рис. 7, $i-s$-тестово эквивалентны. Но данные ВСП не $s-i$-тестово эквивалентны, так как ВСП $\mathcal{TN}_{11}$ не проходит тест, описанный выше в п. 2.

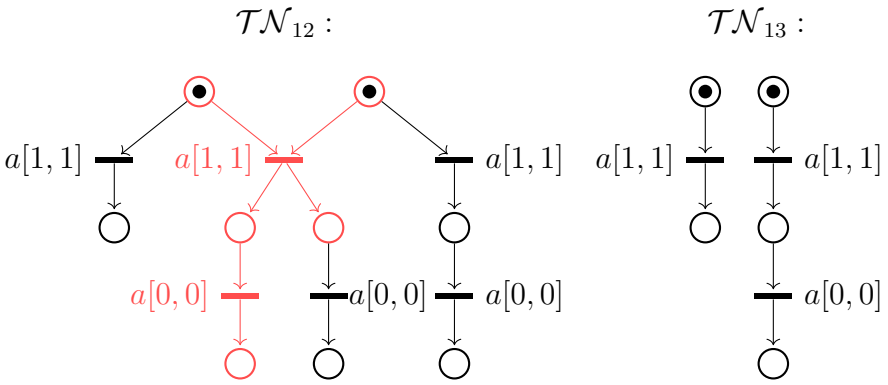



*Рис. 8.* Пример $s-s$-, не $wh$, не $pom$-тестово эквивалентных ВСП.

4) ВСП $\mathcal{TN}_{12}$ и $\mathcal{TN}_{13}$, изображенные на рис. 8, $s-s$-тестово эквивалентны, но ни $pom$-, ни $wh$-тестово эквивалентны. Проверим последнее.

Определим последовательность $w = (a,1)(a,0) \in (Act \times \mathbb{T})^*$ и ВЧУМ $TP' = (\{x_1, x_2, x_3\}, \preceq', \lambda', \tau')$, где $\preceq' = \{(x_i, x_i) \mid 1 \le i \le 3\} \cup \{(x_1, x_2)\}$, $\lambda'(x_1) = \lambda'(x_2) = \lambda'(x_3) = a$, $\tau'(x_1) = \tau'(x_2) = \tau'(x_3) = 1$. Для любого временного причинного сети-процесса $\pi_{13} = (TN_{13}, \varphi_{13}) \in \mathcal{CP}(\mathcal{TN}_{13})$, в котором $E_{TN_{13}}$ состоит из двух событий с пометками $a$ и временными значениями, равными 1, т.е. $\pi^0_{13} \xrightarrow{w} \pi_{13}$, существует его $i$-расширение, единственный временной причинный сеть-процесс $\pi'_{13} = (TN'_{13}, \varphi'_{13}) \in \mathcal{CP}(\mathcal{TN}_{13})$, в котором $E_{TN'_{13}}$ состоит из трех событий: двух параллельных событий с пометками $a$ и третьего события с пометкой $a$, находящегося в отношении причинной зависимости с одним из $a$, и временными значениями 1, т.е. $\pi_{13} \to_i \pi'_{13}$, $\eta(TN'_{13}) \simeq TP'$ и $\mathcal{TN}_{13}$ **after** $w$ $\mathbf{MUST}_{wh}$ $\{TP'\}$. В случае ВСП $\mathcal{TN}_{12}$, например, для временного сети-процесса, элементы которого отображаются в элементы ВСП $\mathcal{TN}_{12}$, выделенные на рисунке красным цветом, не существует необходимого $i$-расширения.

5) ВСП $\mathcal{TN}_7$ и $\mathcal{TN}_8$, изображенные на рис. 5 и рассмотренные ранее в примере 7, $i-i$- и $pom$-тестово эквивалентны, но ни $pos$-, ни $wh$-тестово эквивалентны. Убедимся в последнем.

Определим последовательность $w = (a,1) \in (Act \times \mathbb{T})^*$ и ВЧУМы $TP_1 = (\{x_1, x_2\}, \preceq_1,$

$\lambda_1$, $\tau_1$), где $\preceq_1 = \{(x_1, x_1), (x_2, x_2), (x_1, x_2)\}$, $\lambda_1(x_1) = a$, $\lambda(x_2) = c$, $\tau_1(x_1) = \tau_1(x_2) = 1$; $TP_2 = (\{x'_1, x'_2\}, \preceq_2, \lambda_2, \tau_2)$, где $\preceq_2 = \{(x'_1, x'_1), (x'_2, x'_2)\}$, $\lambda_2(x'_1) = a$, $\lambda_2(x'_2) = b$, $\tau_2(x'_1) = \tau_2(x'_2) = 1$.

Легко видеть, что $\mathcal{TN}_8$ **after** $w$ $\mathbf{MUST}_{wh}$ $\{TP_1, TP_2\}$. Однако, это не так для ВСП $\mathcal{TN}_7$, так как, например, для временного сети-процесса, элементы которого отображаются в элементы ВСП $\mathcal{TN}_7$, выделенные на рисунке красным цветом, не существует $i$-расширения, которому бы соответствовало ВЧУМ, изоморфное $TP_1$ или $TP_2$.

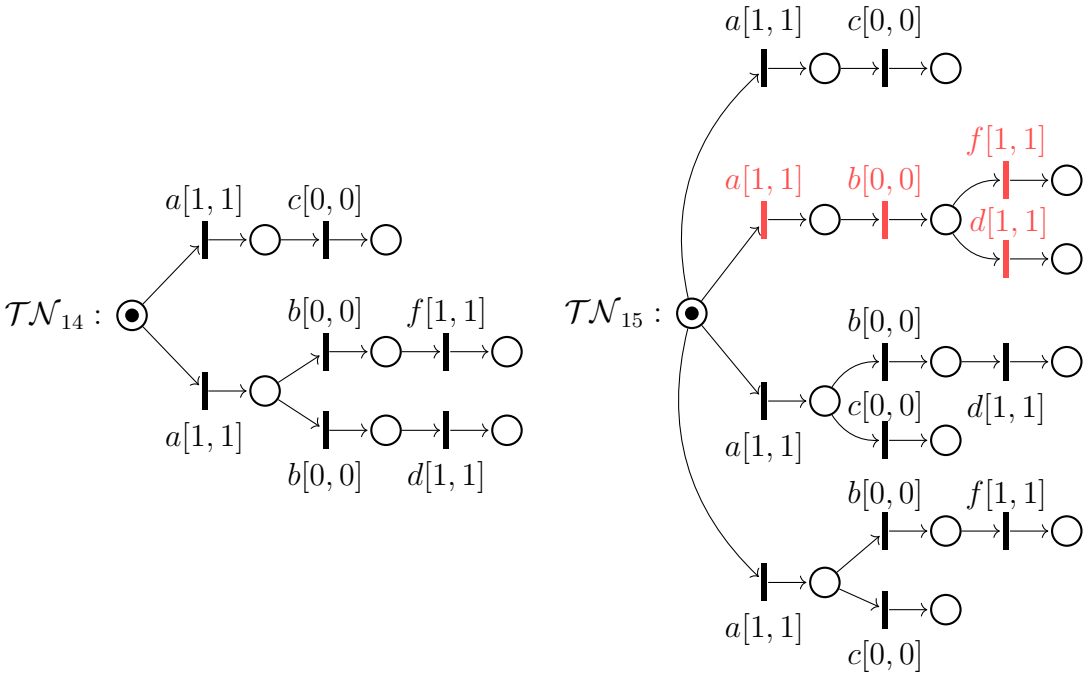



*Рис. 9.* Пример *pos*-, не $i-i$-тестово эквивалентных ВСП.

6) ВСП $\mathcal{TN}_{14}$ и $\mathcal{TN}_{15}$, изображенные на рис. 9, *pos*-тестово эквивалентны, но не $i-i$-тестово эквивалентны. $\mathcal{TN}_{14}$ **after** $w = (a,1)(b,0)(d,1)(\overleftarrow{d},1)(f,1)$ $\mathbf{MUST}_{i-i}$ $W = \{(e,1)\}$, так как в ВСП $\mathcal{TN}_{14}$ не существует обратимой $i-i$-последовательности срабатываний, которая помечена $w$. Однако в ВСП $\mathcal{TN}_{15}$ существует такая обратимая $i-i$-последовательность срабатываний, входящие в нее переходы выделены на рисунке красным цветом, и для неё невозможно расширение посредством срабатывания шага, помеченного $(e,1)$. Таким образом, не выполняется $\mathcal{TN}_{15}$ **after** $w$ $\mathbf{MUST}_{i-i}$ $W$.

7) ВСП $\mathcal{TN}_{16}$ и $\mathcal{TN}_{17}$, изображенные на рис. 10, $s-i$-тестово эквивалентны, но не $s$-тестово эквивалентны.

Нетрудно проверить, что $\mathcal{TN}_{16}$ **after** $w = (a,1)$ $\mathbf{MUST}_s$ $W = \{([d_1,\ d_2],\ 1);\ (d_3,1)\}$. Но в $\mathcal{TN}_{17}$ после последовательности срабатываний, помеченной $w$ и соответствующей средней части ВСП на рисунке (переход выделен красным цветом), невозможны срабатывания ни

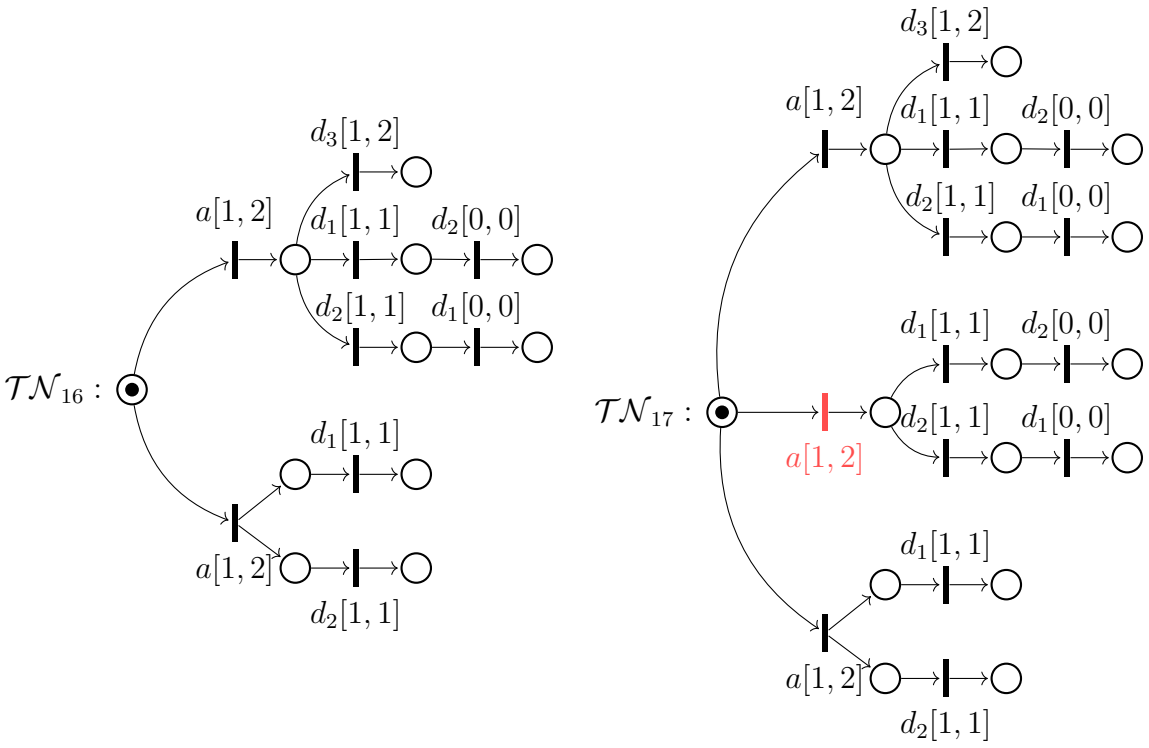



*Рис. 10.* Пример $s-i$-, не $s$-тестово эквивалентных ВСП

шага, помеченного $([d_1,\ d_2], 1)$, ни шага $(d_3, 1)$. Таким образом, не выполняется $\mathcal{TN}_{17}$ **after** $w$ $\mathbf{MUST}_s$ $W$. □

Вопросы о взаимосвязях тестовых эквивалентностей с обратимостью в частично-упорядоченной семантике в основном пока остаются открытыми. Как следствие Теоремы 1 и примера 7 на рис. 11 представлена иерархия рассмотренных в статье временных тестовых эквивалентностей с обратимостью. Стрелки с белыми наконечниками обозначают оставшиеся открытыми вопросы о взаимосвязях.

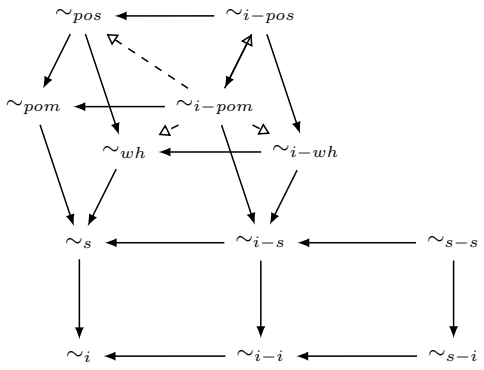



*Рис. 11.*

## 7. Заключение

В данной статье была сделана попытка расширить тестовые эквивалентности введением тестов с возможностью отмены выполненных действий. В контексте непрерывно-временных сетей Петри были введены понятия тестовых эквивалентностей в интерливинговой, шаговой и частично-упорядоченной семантиках с обратимостью в интерливинговой и шаговой семантиках. Для введения отмены действий применялся метод причинной обратимости, который хорошо поддерживается использованной для представления вычислений ВСП частично-упорядоченной семантикой причинных сетей-процессов. Были исследованы взаимосвязи между изученными ранее ([1, 10]) и введенными тестовыми эквивалентностями. Установлено, что тестовая эквивалентность в частично-упорядоченной семантике слабее интерливинговой тестовой эквивалентности с обратимостью в интерливинговой семантике. Также в полученной иерархии показано, что, в отличие от бисимуляций с обратимостью ([29]), интерливинговая тестовая эквивалентность с шаговой обратимостью не сильнее шаговой тестовой эквивалентности с интерливинговой обратимостью. Заметим, что и для вариантов эквивалентностей с "расширенным будущим", тестовые эквивалентности для случаев с "единичным будущим" и "расширенным будущим" различаются, в отличие от бисимуляционной эквивалентности, для которых установлено совпадение ([10]). Также остался открытым вопрос о взаимоотношении некоторых тестовых эквивалентностей в частично-упорядоченной семантике. Поэтому в дальнейшем планируется продолжить изучение тестовых эквивалентностей с шаговой обратимостью, а также исследовать тестовые эквивалентности с обратимостью в частично-упорядоченной семантике.

## Список литературы

1. Боженкова Е.Н., Вирбицкайте И.Б. Тестовые эквивалентности временных Сетей Петри // Программирование. 2020. №4. С. 3-13.
2. Вирбицкайте И.Б., Боровлёв В.А., Попова-Цейгманн Л. Истинно-параллельная и недетерминированная семантика временных сетей Петри// Программирование. 2016. №4. С. 4–16.
3. Грибовская Н.С., Вирбицкайте И.Б. Семантика систем переходов структур событий с отменяемыми событиями при сохранении причинно-следственной зависимости// Системная информатика. 2024. №24. С.59–90.
4. Aceto L., De Nicola R., Fantechi A. Testing equivalences for event structures// Lect. Notes in Computer Sci. 1987. Vol. 280. P. 1–20.

5. Andreeva M., Bozhenkova E., Virbitskaite I. Analysis of timed concurrent models based on testing equivalence// Fundamenta Informaticae. 2000. Vol. 43. P. 1–20.
6. Aura T., Lilius J. A causal semantics for time Petri nets// Theor. Computer Sci. 2000. Vol. 243, №1–2. P. 409–447.
7. Barylska, K., Koutny, M., Mikulski, L., Piatkowski, M. Reversible computation vs. reversibility in Petri nets// Sci. Comput. Program. 2018. Vol. 151. P.48-60.
8. Berard B., Cassez F., Haddad S., Lime D., Roux O.H. Comparison of the expressiveness of timed automata and time Petri nets// Lect. Notes in Computer Sci. 2005. Vol. 3829. P. 211–225.
9. Bihler E., Vogler W. Timed Petri Nets: Efficiency of Asynchronous Systems// Lect. Notes in Computer Sci. 2004. Vol. 3185. P. 25–58.
10. Bozhenkova E., Virbitskaite I. Extended Future in Testing Semantics for Time Petri Nets// Concurrency, Specification and Programming — Revised Selected Papers from the 29th International Workshop on Concurrency, Specification and Programming (CS&P'21), Berlin, Germany, in series: Studies in Computational Intelligence. Springer. 2023. Vol. 1091. P. 65-89.
11. Cleaveland R., Hennessy M. Testing equivalence as a bisimulation equivalence// Lect. Notes in Computer Sci. 1989. Vol. 407. P. 11–23.
12. Cleaveland R., Zwarico A.E. A theory of testing for real-time// Proc. 6th IEEE Symp. on Logic in Comput. Sci. (*LICS'91*). Amsterdam, The Netherlands, 1991. P. 110–119.
13. Corradini F., Vogler W., Jenner L. Comparing the Worst-Case Efficiency of Asynchronous Systems with PAFAS// Acta Informatica. 2002. Vol. 38. №11–12. P. 735–792.
14. Danos, V, Krivine, J.: Reversible communicating systems// In: Proceedings of CONCUR'04. Lect. Notes in Computer Sci. 2004. Vol. 3170. P. 292-307.
15. De Nicola R., Hennessy M. Testing equivalence for processes// Theor. Comput. Sci. 1984. Vol. 34. P. 83–133.
16. R. De Nicola, U. Montanari, and F. Vaandrager: Back and forth bisimulations// Proc. CONCUR '90, Theories of Concurrency: Unification and Extension. Lect. Notes in Comp. Sci. 1990. Vol. 458. P. 152–165.
17. Goltz U., Wehrheim H. Causal testing// Lect. Notes in Computer Sci. 1996. Vol. 1113. P. 394–406.
18. M. Hennessy, R.Milner: Algebraic laws for nondeterminism and concurrency// JACM. 1985. Vol. 32. №1. P. 137–161.
19. Hennessy M., Regan T. A process algebra for timed systems// Inform. and Comput. 1995. Vol. 117. P. 221–239.
20. Hoogers P.W., Kleijn H.C.M., Thiagarajan P.S. An event structure semantics for general Petri nets// Theor. Computer Sci. 1996. Vol. 153. №1–2. P. 129–170.
21. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani, J.-B. Concurrent flexible reversibility// In: Felleisen, M., Gardner, P. (eds.) ESOP 2013. Lect. Notes in Computer Sci. Heidelberg:Springer, 2013. Vol. 7792. P. 370–390. URL: https://doi.org/10.1007/978-3-642-37036-6 21.
22. Lanese, I., Palacios, A., Vidal, G. Causal-consistent replay debugging for message passing programs// In: Perez, J.A., Yoshida, N. (eds.) FORTE 2019. Lect. Notes in Computer Sci. Springer,

Cham, 2019. Vol. 11535, P. 167–184. URL: https://doi.org/10.1007/978-3-030-21759-4 10.

23. Llana L., de Frutos D. Denotational semantics for timed testing// Lect. Notes in Computer Sci. 1997. Vol. 1233. P. 368–382.
24. Melgratti, H.C., Mezzina, C.A., Phillips, I., Pinna, G.M., Ulidowski, I. Reversible occurrence nets and causal reversible prime event structures// In I. Lanese and M. Rawski, eds., Reversible Computation - 12th International Conference, RC 2020. Lect. Notes in Computer Sci. Springer, 2020. Vol. 12227. P. 35-53.
25. Melgratti, H.C., Mezzina, C.A., Pinna, G.M. A Reversible Perspective on Petri Nets and Event Structures// Proc. ACM Transactions on Computational Logic, August 2024. Vol. 25, №4. P. 1–38. URL: https://doi.org/10.1145/3686154.
26. Murphy D. Time and duration in noninterleaving concurrency// Fundamenta Informaticae. 1993. Vol. 19. P. 403–416.
27. M. Nielsen, C. Clausen. Bisimulation for Models in Concurrency// In: Proc. of 5th Int. Conf. on Concurrency Theory, CONCUR'94. Lect. Notes in Computer Sci. Springer-Verlag. 1994. Vol. 836. P. 385–400.
28. Nielsen, M., Rozenberg, G., Thiagarajan, P.S. Behavioural notions for elementary net systems // Distributed Computing. 1990. Vol. 4. №1. P. 45–57.
29. Phillips, I.C.C., Ulidowski, I. A hierarchy of reverse bisimulations on stable configuration structures// Math. Struct. Comput. Sci. 2012. Vol.22. P. 333-372.
30. Phillips, I.C.C., Ulidowski, I. Event identifier logic// Math. Struct. Comput. Sci. 24, 2014, P. 1-51.
31. Phillips, I.C.C., Ulidowski, I. Reversibility and asymmetric conflict in event structures// Logic and Algebraic Methods in Programming. 2015. Vol. 84, №6. P. 781-805.
32. Pomello, L., Rozenberg, G., Simone C. A Survey of Equivalence Notions for Net Based Systems// Lect. Notes in Computer Sci. 1992. Vol. 609. P. 410–472.
33. Valero, Vol., de Frutos, D., Cuartero, F. Timed processes of timed Petri nets// Lect. Notes in Computer Sci. 1995. Vol. 935. P. 490–509.
34. van Glabbeek R.J. The linear time – branching time spectrum I: the semantics of concrete, sequential processes// In: Bergstra J.A., Ponse A., and Smolka S.A. (eds.), Handbook of Process Algebra Elsevier. 2001. P. 3–99.
35. van Glabbeek R.J., Goltz U., Schicke J.-W. On causal semantics of Petri nets // Lect. Notes in Computer Sci. 2011. Vol. 6901. P. 43-59.
36. Vassor, M., Stefani, J.-B. Checkpoint/Rollback vs causally-consistent reversibility// In: Kari, J., Ulidowski, I. (eds.) RC 2018. Lect. Notes in Computer Sci. Springer, Cham (2018). Vol. 11106. P. 286-303. URL:https://doi.org/10.1007/978-3-319-99498-7 20.
37. Virbitskaite I., Bushin D., Best E. True concurrent equivalences in time Petri nets// Fundamenta Informaticae. 2016. Vol. 149. №4. P. 401–418.

УДК 519.681.2, 519.681.3

# Теоретико-категорная характеризация семантик систем переходов первичных структур событий с отменяемыми событиями при сохранении причинной зависимости

*Грибовская Н.С. (Институт систем информатики СО РАН)*

*Вирбицкайте И.Б. (Институт систем информатики СО РАН)*

Реверсивные (обратимые) вычисления, широко изучаемые в последние годы, представляют собой нетрадиционную форму вычислений, которые могут быть выполнены как в прямом, так и в обратном направлении. Любая последовательность действий, выполняемых системой, впоследствии может быть отменена по какой-либо причине (например, в случае ошибки), что позволяет восстановить предыдущие состояния системы, как если бы отмененные действия вообще не выполнялись. Структуры событий – это основополагающая модель теории параллелизма, позволяющая понять параллельные процессы путем описания происходящих событий и взаимосвязей между ними. В литературе выделяются два структурно отличающихся подхода к построению семантики систем переходов для моделей структур событий. Один подход основан на конфигурациях, т.е. наборах уже выполненных событий, а другой — на остаточных структурах, т.е. невыполненных фрагментах модели. Системы переходов, основанные на конфигурациях, в основном используются для представления семантики и эквивалентностей моделей параллелизма. Системы переходов, построенные на остаточных структурах, активно применяются для демонстрации согласованности операционной и денотационной семантик алгебраических исчислений параллельных процессов и для визуализации поведения моделей. В настоящей статье дается теоретико-категорная характеризация двух типов семантик систем переходов для обратимых первичных структур событий, учитывающих при отмене событий их причинно-следственные зависимости, и устанавливается взаимосвязь таких семантик, что может помочь при построении алгебраических описаний композиций обратимых параллельных процессов.

*Ключевые слова*: структуры событий, отменяемые события, системы переходов, бисимуляция, функторы, теория категорий

## 1. Введение

В последние годы концепция ‘обратимости’ вычислений широко изучалась в поисках механизмов, позволяющих отменять некоторые выполняемые в вычислительном процессе

действия, которые по какой-либо причине необходимо аннулировать (например, в случае ошибки). 'Обратимые' вычисления могут выполняться не только в традиционном прямом направлении, но и в обратном, восстанавливая прошлые состояния и вычисляя входные данные из выходных. 'Обратимость' вычислений находит свое применение в различных областях, включая абстракции в программировании для разработки надежных и безопасных систем [31, 36], анализ и отладку программ [33], моделирование биохимических процессов [29], разработку аппаратного обеспечения и квантовые вычисления [13] и т.д.

В теории параллельных систем и процессов был изучен ряд аспектов 'обратимых' вычислений, связанных с различными моделями параллелизма: клеточными автоматами [26], алгебраическими исчислениями процессов [11, 32], сетями Петри [5, 14, 38, 41], структурами событий [37, 45, 47], мембранными системами [46] и т.д. Эти исследования привели к выявлению трех основных методов обращения параллельных процессов: обратное отслеживание [12, 42], причинная обратимость [11, 38, 42] и внепричинная обратимость [31, 41, 42], которые отличаются порядком выполнения действий в обратном направлении. Под обратным отслеживанием обычно понимается возможность отменять действия в порядке, обратном тому, в котором они были выполнены. Причинная обратимость предполагает, что действие может быть отменено при условии, что все действия, причинно зависящие от данного действия, (если таковые имеются) уже были отменены. Внепричинная обратимость — это форма обращения вычислений, не сохраняющая причинную зависимость и наиболее характерная для моделирования биохимических систем.

Структуры событий, предложенные Винскелем в его диссертации [48], являются одной из центральных моделей параллельных недетерминированных процессов. Структуры событий использовались для установления связей между различными моделями параллелизма [15, 17, 39, 48], для определения денотационной и операционной семантик алгебраических исчислений и языков спецификации параллельных процессов [9, 18, 27, 28, 30, 48], для определения поведенческих эквивалентностей между процессами [16], для моделирования квантовых стратегий и игр [49].

Известно, что установление связей между моделями систем переходов и структур событий способствует изучению и решению различных проблем анализа и верификации параллельных систем. Различают два метода построения семантики систем переходов для структур событий. В первом методе (см. [2, 15, 16, 25, 27, 48] среди прочих) состояния системы переходов — это конфигурации (наборы уже произошедших событий), а переходы

между состояниями создаются, начиная с начальной (как правило, пустой) конфигурации, посредством расширения конфигураций за счет добавления событий, которые происходят в предыдущем состоянии. Во втором, более ‘структурно-композиционном’, методе (см. [4, 8, 10, 27, 28, 30, 34, 40] среди прочих) начальное состояние системы переходов — это заданная структура событий, состояния — это остаточные структуры, получаемые посредством удаления событий, уже произошедших и конфликтующих с ними в ходе выполнения структуры, а переходы между состояниями создаются по мере построения остаточных структур. В литературе системы переходов, основанные на конфигурациях, преимущественно применяются для определения семантики и эквивалентностей параллельных моделей, а системы переходов, основанные на остаточных структурах, — для построения операционной семантики алгебраических исчислений параллельных процессов и демонстрации согласованности операционной и денотационной семантик. Взаимосвязи между двумя типами систем переходов впервые изучались в статье [35] для наиболее простой модели — первичных структур событий, а затем в статьях [6] и [7] — для широкого спектра моделей структур событий с асимметричным и симметричным конфликтом.

Обратимые структуры событий расширяют структуры событий с целью представления параллельных недетерминированных процессов, способных отменять выполненные действия, позволяя конфигурациям изменяться посредством удаления событий, а не только добавления их. В работах [45, 47] Филлипс и др. определили причинные и внепричинные обратимые формы первичных [45], асимметричных [45, 47] и обобщенных [47] структур событий и показали соответствие между их конфигурациями и конфигурациями традиционных (без отменяемых событий) моделей. В статье [20] Граверсен и др. представили категории различных классов обратимых структур событий, включая упомянутые выше, и построили функторы между этими категориями. В работе [3] Обер и Кристеску разработали в терминах структур конфигураций ‘истинно’ параллельную семантику обратимого расширения CCS, RCCS (без автопараллелизма, автоконфликта и рекурсии). В статье [21] Граверсен и др. построили категорию обратимых структур событий с расслоением и симметричным конфликтом и использовали подкатегорию, учитывающую причинно-следственную зависимость между событиями, для моделирования семантики другого обратимого расширения CCS, CCSK. В работе [22] те же авторы представили $\pi$-исчисления со статической обратимостью, при которой выполнение действия не изменяет структуру процесса, и с динамической обратимостью, при которой выполнение действия перемещает

его в отдельную историю. Для статического исчисления денотационная семантика определена в терминах структур событий с расслоением [27], которая индуктивно генерируется на основе структуры процесса. Для динамического исчисления операционная семантика построена на первичных структурах событий, которая генерируется из асинхронных систем переходов. Показано соответствие между результирующими структурами событий. В статье [44] логика идентификатора событий (EIL) была введена для расширения логики Хеннесси-Милнера с помощью обратных модальностей. EIL-эквивалентность соответствует эквивалентности наследуемой бисимуляции с сохранением истории (hereditary history-preserving bisimulation) в контексте стабильных структур конфигураций. В работах [19, 43] изучаются взаимосвязи различных поведенческих эквивалентностей в обратимых моделях.

Цель данной статьи — дать теоретико-категорную характеризацию двух типов семантик систем переходов, основанных на конфигурациях и на остаточных структурах, для обратимых первичных структур событий, учитывающих при отмене событий их причинно-следственные зависимости, и понять взаимосвязи этих семантик, что может помочь в построении алгебраических исчислений для описания композиций обратимых параллельных процессов.

Эта статья построена следующим образом. В разделе 2 рассматриваются синтаксис обратимых первичных структур событий и их шаговая семантика в терминах конфигураций/трасс. В разделе 3 определяется оператор удаления событий, который используется для построения остаточных структур, и демонстрируется корректность оператора. В разделе 4 разрабатывается два типа семантик систем переходов для обратимых первичных структур событий. В разделе 5 показываются различия между этими семантиками с теоретико-категорной точки зрения. В разделе 6 приводятся заключительные замечания. Приложение А содержит краткую справку по базовым определениям теории категорий, а в приложении Б представлены доказательства лемм и утверждений.

## 2. Обратимость в первичных структурах событий

В этом разделе сначала определяется модель первичных структур событий (ПСС) [48], а затем формулируется понятие обратимых первичных структур событий (ОПСС) [45], а также рассматривается их (шаговая) семантика и свойства.

Для формального описания поведения параллельных систем используются модели струк-

тур событий, в которых элементы поведения представлены событиями. Существуют различные способы определения отношений между событиями. В ПСС причинно-следственная зависимость между событиями задается частичным порядком, а несовместимость событий определяется отношением конфликта. Два события, которые не находятся ни в причинно-следственной зависимости, ни в конфликте, считаются независимыми (параллельными).

**Определение 1.** *(Помеченная)* первичная структура событий (ПСС) *(на множестве* $L = \{a, b, c, \ldots\}$ *действий) — это кортеж* $\mathcal{E} = (E, <, \sharp, l, C_0)$*, где*

- $E$ *— счетное множество* событий*;*
- $< \subseteq E \times E$ *— иррефлексивный частичный порядок (*причинно-следственная зависимость *(ПСЗ)), удовлетворяющий* принципу конечности причин*: для каждого* $e \in E$ *верно, что* $\lfloor e \rfloor_< = \{e' \in E \mid e' < e\}$ *— конечное множество. Для подмножества* $X \subseteq E$ *событий будем писать* $\lfloor X \rfloor_<$*, чтобы обозначать множество* $\{e' \in E \mid e' < e$ *для некоторого* $e \in X\}$ *предшественников событий в* $X$*;*
- $\sharp \subseteq E \times E$ *— иррефлексивное симметричное отношение* конфликта*, удовлетворяющее* принципу наследования конфликта*: для всех* $e, e', e'' \in E$ *верно, что если* $e < e'$ *и* $e \sharp e''$*, то* $e' \sharp e''$*;*
- $l : E \to L$ — помечающая функция.

Итак, ПСС — это модель, основанная на событиях параллельных и недетерминированных процессов, в которой события, помеченные действиями, рассматриваются как атомарные, неделимые и мгновенные действия, некоторые из которых могут происходить только после других (т.е. существует ПСЗ, представленная иррефлексивным частичным порядком $<$ между событиями) и некоторые из которых не могут происходить вместе (т.е. между событиями существует конфликт $\sharp$). Кроме того, необходимы принцип конечности причин и принцип наследования конфликта.

ПСС выполняется по мере того, как происходят события, начиная с начального состояния и переходя из одного состояния в другое. Состояние в ПСС называется конфигурацией и представляет собой множество уже произошедших событий. Подмножество $X \subseteq E$ событий является *лево-замкнутым относительно* $<$, если для всех $e \in X$ верно, что $\lfloor e \rfloor_< \subseteq X$; является *бесконфликтным*, если для всех $e, e' \in X$ верно, что $\neg(e \sharp e')$, запись $CF(X)$ обозначает, что множество $X$ бесконфликтно. Подмножество $C \subseteq E$ является *конфигурацией* в ПСС $\mathcal{E}$, если $C$ конечно, лево-замкнуто относительно $<$ и бесконфликтно.

Обратимые первичные структуры событий (ОПСС) [45, 47] основаны на более слабой форме ПСС, поскольку принцип наследования конфликта может не сохраняться при добавлении обратимости. Кроме того, в ОПСС некоторые события рассматриваются как отменяемые, а также добавляются два отношения между событиями: обратная ПСЗ и отношение предотвращения. Первое отношение — это ПСЗ в обратном направлении, т.е. чтобы отменить событие в текущей конфигурации, в ней должны присутствовать события, от которых это событие обратимо зависит. Второе отношение, напротив, идентифицирует те события, присутствие которых в текущей конфигурации предотвращает отмену события.

**Определение 2.** *(Помеченная)* обратимая первичная структура событий (ОПСС) *(на множестве $L$) — это кортеж $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$, где*

- $E$ — *счетное множество* событий*;*
- $\sharp \subseteq E \times E$ — *иррефлексивное и симметричное отношение* конфликта*;*
- $<\subseteq E \times E$ — *иррефлексивный частичный порядок (*причинно-следственная зависимость*), удовлетворяющая условию: для каждого $e \in E$ верно, что $\lfloor e \rfloor_<$ — конечное и бесконфликтное множество, а также для каждых $e, e' \in E$ верно, что если $e < e'$, то $\neg(e \sharp e')$;*
- $l : E \to L$ — помечающая функция*;*
- $F \subseteq E$ — отменяемые события*, обозначаемые через $\underline{F} = \{\underline{u} \mid u \in F\}$;*
- $\prec\subseteq E \times \underline{F}$ — обратная причинно-следственная зависимость *такая, что для каждого $u \in F$ верно, что $u \prec \underline{u}$ и $\llcorner u \lrcorner_\prec = \{e \mid e \prec \underline{u}\}$ — конечное и бесконфликтное множество;*
- $\rhd \subseteq E \times \underline{F}$ — отношение предотвращения *такое, что для каждого $u \in F$ верно: если $e \prec \underline{u}$, то $\neg(e \rhd \underline{u})$;*
- $\ll$ — *транзитивная* устойчивая причинно-следственная зависимость *такая, что $e \ll e'$, если и только если $e < e'$, а также $e' \rhd \underline{e}$, если $e \in F$. Отношение конфликта $\sharp$* наследуется по устойчивой ПСЗ $\ll$*: если $e \sharp e' \ll e''$, то $e \sharp e''$;*
- $C_0 \subseteq E$ — начальная конфигурация*, которая является конечным, лево-замкнутым относительно $<$ и бесконфликтным множеством.*

*ОПСС с $\emptyset$-компонентами обозначается $\mathcal{O}$.*

Несложно проверить, что любая ПСС также является ОПСС, имеющая $F = \emptyset$ и $C_0 = \emptyset$. Тогда любое понятие, определенное для ОПСС, применимо и к ПСС.

При графическом представлении ОПСС используются следующие обозначения. Действия, связанные с событиями, изображаются рядом с самими событиями. Если не возникнет двусмысленности, будем использовать действия, а не имена событий для обозначения событий. Неотменяемые события рисуются в квадратиках, а отменяемые — в кружочках. Отношение ПСЗ изображается сплошными стрелками (за исключением тех, которые выводятся по транзитивности), отношение обратной ПСЗ — пунктирными стрелками, а также показываются отношения конфликта и предотвращения. События, относящиеся к начальной конфигурации, окрашены в темно-серый цвет. Очевидно, что если начальная конфигурация представляет собой пустое множество, то никакие события не будут окрашены в темно-серый цвет.

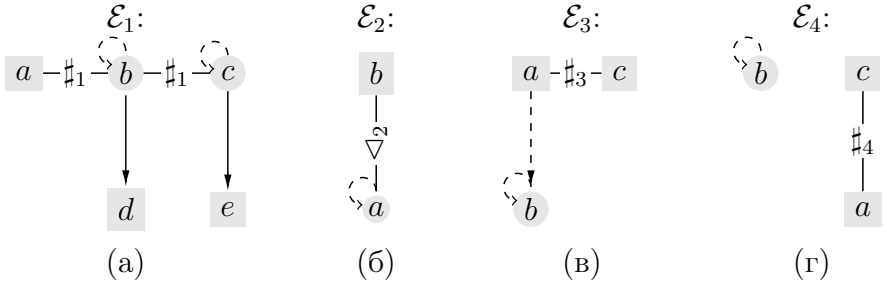



*Рис. 1.* Примеры ОПСС

**Пример 1.** *Рассмотрим показанную на рис. 1(а) структуру* $\mathcal{E}_1$ *с компонентами:* $E_1 = \{a, b, c, d, e\}$; $<_1 = \{(b, d), (c, e)\}$; $\sharp_1 = \{(a, b), (b, a), (b, c), (c, b)\}$; $l_1$ — *идентичная функция;* $F_1 = \{b, c\}$; $\prec_1 = \{(b, \underline{b}), (c, \underline{c})\}$; $\rhd_1 = \emptyset$; $C_0^1 = \emptyset$. *Легко убедиться в том, что компоненты структуры* $\mathcal{E}_1$ *удовлетворяют соответствующим пунктам определения 2. В частности, видим, что для каждого события* $f \in E_1$ *(для каждого события* $u \in F_1$*) множество* $\lfloor f \rfloor_{<_1}$ $(\lfloor u \rfloor_{\prec_1})$ *конечно и бесконфликтно;* $<_1 = \{(b, d), (c, e)\}$ *и* $(b, d), (c, e) \notin \sharp_1$, *а также* $\prec_1 = \{(b, \underline{b}), (c, \underline{c})\}$ *и* $(b, \underline{b}), (c, \underline{c}) \notin \rhd_1$. *Заметим, что* $\sharp_1$ *не наследуется по* $<_1$, *поскольку* $a \sharp_1 b <_1 d$ *и* $\neg(a \sharp_1 d)$. *Кроме того, пары* $(b, d)$ *и* $(c, e)$ *находятся в причинно-следственной зависисмости* $<_1$, *а отношение предотвращения* $\rhd_1$ *пусто. Тогда отношение устойчивой ПСЗ тоже пусто. Поэтому конфликт* $\sharp_1$ *наследуется по* $\ll_1$. *Таким образом, структура* $\mathcal{E}_1$ *является ОПСС.* ◇

ОПСС выполняется по мере того, как происходят и/или отменяются события, начиная с начальной конфигурации и переходя от одной конфигурации к другой. Множества событий, которые происходят/отменяются при таком переходе, называются шагами в ОПСС.

Достижимые конфигурации — это подмножества событий, которые могут быть получены из начальной конфигурации путем выполнения шагов. Последовательность шагов — трасса в ОПСС.

**Определение 3.** *Пусть* $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ *— ОПСС и* $C \subseteq E$ *— конечное, левозамкнутое и бесконфликтное множество. Тогда*

- *для множеств* $A \subseteq E$ *и* $B \subseteq F$ *будем говорить, что шаг* $A \cup \underline{B}$ возможен из $C$, *если выполнены следующие условия:*

  *а)* $A \cap C = \emptyset$, $B \subseteq C$, $(C \cup A)$ *— конечное и бесконфликтное множество;*

  *б)* $\forall e \in A, \forall e' \in E : \ e' < e \Rightarrow e' \in (C \setminus B)$;

  *в)* $\forall e \in B, \forall e' \in E : \ e' \prec \underline{e} \Rightarrow e' \in (C \setminus (B \setminus \{e\}))$;

  *г)* $\forall e \in B, \forall e' \in E : \ e' \rhd \underline{e} \Rightarrow e' \notin (C \cup A)$.

  *Если шаг* $A \cup \underline{B}$ *возможен из* $C$, *то* $C \stackrel{A \cup \underline{B}}{\longrightarrow} C' = (C \setminus B) \cup A$. *Будем писать* $l(A \cup \underline{B}) = M$, *если* $M$ *— мультимножество на множестве* $L$ *действий, определяемое следующим образом:* $M = l(A \cup \underline{B}) = l(A) \cup \underline{l(B)}$, *где мультимножество* $l(X) = \sum_{a \in L} | \{e \in X \mid l(e) = a\} |$.
- $C$ *—* (достижимая (из $C_0$)) конфигурация в $\mathcal{E}$, *если существуют множества* $A_i \subseteq E$ *и* $B_i \subseteq F$ *для всех* $i = 1, \ldots, n$ $(n \geq 0)$ *такие, что* $C_{i-1} \stackrel{A_i \cup \underline{B_i}}{\longrightarrow} C_i$ *и* $C_n = C$. *В этом случае* $t = (A_1 \cup \underline{B}_1) \ldots (A_n \cup \underline{B}_n)$ $(n \geq 0)$ *—* трасса в $\mathcal{E}$ *и* $last(t) = C_n$. *Множество (достижимых) конфигураций в* $\mathcal{E}$ *обозначается как* $Conf(\mathcal{E})$, *а множество трасс в* $\mathcal{E}$ *— как* $Trace(\mathcal{E})$. *Понятно, что* $C \in Conf(\mathcal{E})$ *— бесконфликтное множество.*
- *Две трассы* $t = (A_1 \cup \underline{B}_1) \ldots (A_n \cup \underline{B}_n)$ $(n \geq 0)$ *и* $t' = (A'_1 \cup \underline{B'}_1) \ldots (A'_m \cup \underline{B'}_m)$ $(m \geq 0)$ *в* $\mathcal{E}$ *называются* эквивалентными *(обозначается* $t \sim t'$*), если* $last(t) = last(t')$. *Эквивалентный класс для трассы* $t$ *обозначается* $[t]$.

**Пример 2.** *Сначала вспомним пример 1 с ОПСС* $\mathcal{E}_1$ *с компонентами:* $E_1 = \{a, b, c, d, e\}$; $<_1 = \{(b, d), (c, e)\}$; $\sharp_1 = \{(a, b), (b, a), (b, c), (c, b)\}$; $l_1$ *— идентичная функция;* $F_1 = \{b, c\}$; $\prec_1 = \{(b, \underline{b}), (c, \underline{c})\}$; $\rhd_1 = \emptyset$; $C_0^1 = \emptyset$. *Рассмотрим возможные шаги из начальной конфигурации в* $\mathcal{E}_1$.

*Так как события* $a$, $b$ *и* $c$ *не имеют предшественников по ПСЗ, то возможны следующие вперед-переходы:* $\emptyset \stackrel{(\{a\} \cup \emptyset)}{\rightarrow} \{a\}$, $\emptyset \stackrel{(\{b\} \cup \emptyset)}{\rightarrow} \{b\}$ *и* $\emptyset \stackrel{(\{c\} \cup \emptyset)}{\rightarrow} \{c\}$. *Поскольку пара* $(b, d)$ $((c, e))$ *принадлежит отношению* $<_1$, *то событие* $d$ $(e)$ *не может произойти перед тем, как событие* $b$ $(c)$ *произойдет. Тогда получаем следующий вперед-переход:* $\{b\} \stackrel{(\{d\} \cup \emptyset)}{\rightarrow} \{b, d\}$

*($\{c\} \overset{(\{e\}\cup\emptyset)}{\to} \{c,e\}$). Из конфигураций $\{b\}$ и $\{b,d\}$ ($\{c\}$ и $\{c,e\}$) возможен назад-шаг $(\emptyset\cup\{\underline{b}\})$ ($(\emptyset \cup \{\underline{c}\})$), поскольку $(b,\underline{b}) \in \prec_1$ ($(c,\underline{c}) \in \prec_1$), т.е. единственный предшественник по обратной ПСЗ для события $b$ ($c$) — само это событие, а также $\rhd_1 = \emptyset$, т.е. нет событий, которые могли бы предотвратить отмену события $b$ ($c$). Далее может иметь место вперед-переход $\{d\} \overset{(\{c\}\cup\emptyset)}{\to} \{c,d\}$ ($\{e\} \overset{(\{b\}\cup\emptyset)}{\to} \{b,e\}$), так как событие $c$ ($b$) не имеет предшественников по ПСЗ и события $c$ и $d$ ($b$ и $e$) независимы (параллельны). Поскольку пара $(c,e)$ ($(b,d)$) принадлежит отношению $<_1$ и события $d$ и $e$ параллельны, получаем вперед-переход $\{c,d\} \overset{(\{e\}\cup\emptyset)}{\to} \{c,d,e\}$ ($\{b,e\} \overset{(\{d\}\cup\emptyset)}{\to} \{b,e,d\}$). Затем возможен назад-переход $\{c,d,e\} \overset{(\emptyset\cup\{\underline{c}\})}{\to} \{d,e\}$ ($\{b,d,e\} \overset{(\emptyset\cup\{\underline{b}\})}{\to} \{d,e\}$), так как $(b,\underline{b}) \in \prec_1$ ($(c,\underline{c}) \in \prec_1$) и $\rhd_1 = \emptyset$. Таким образом, видим, что $\emptyset$, $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$, $\{e\}$, $\{b,d\}$, $\{c,e\}$, $\{c,d\}$, $\{b,e\}$, $\{d,e\}$, $\{b,d,e\}$, $\{c,d,e\}$ — конфигурации в $\mathcal{E}_1$. Исходя из того, что событие $a$ параллельно с событиями $c$, $d$, $e$, имеем такие конфигурации: $\{a,c\}$, $\{a,d\}$, $\{a,e\}$, $\{a,c,e\}$, $\{a,c,d\}$, $\{a,d,e\}$, $\{a,c,e,d\}$. Так как пара $(a,b)$ ($(b,c)$) принадлежит отношению $\sharp_1$, события $a$ и $b$ ($b$ и $c$) не могут быть вместе в какой-либо конфигурации. Следовательно, все достижимые конфигурации в $\mathcal{E}_1$ перечислены выше. Заметим, что следующие конфигурации достижимы только посредством комбинации вперед- и назад-шагов: $\{d\}$, $\{e\}$, $\{b,e\}$, $\{c,d\}$, $\{d,e\}$, $\{b,d,e\}$, $\{c,d,e\}$, $\{a,d\}$, $\{a,e\}$, $\{a,c,d\}$, $\{a,d,e\}$, $\{a,c,e,d\}$.*

*Рассмотрим на рис. 1(б) ОПСС $\mathcal{E}_2$ с компонентами: $E_2 = \{a,b\}$; $<_2 = \emptyset$; $\sharp_2 = \emptyset$; $l_2$ — идентичная функция; $F_2 = \{a\}$; $\prec_2 = \{(a,\underline{a})\}$; $\rhd_2 = \{(b,\underline{a})\}$; $C_0^2 = \emptyset$. Поскольку отношения $<_2$ и $\sharp_2$ пусты, события $a$ и $b$ параллельны и поэтому они могут происходить в любом порядке или одновременно. Тогда возможны вперед-шаги: $\emptyset \overset{(\{a\}\cup\emptyset)}{\to} \{a\} \overset{(\{b\}\cup\emptyset)}{\to} \{a,b\}$, $\emptyset \overset{(\{b\}\cup\emptyset)}{\to} \{b\} \overset{(\{a\}\cup\emptyset)}{\to} \{a,b\}$ и $\emptyset \overset{(\{a,b\}\cup\emptyset)}{\to} \{a,b\}$. Так как верно, что $b \rhd_2 \underline{a}$, событие $b$ предотвращает отмену события $a$, т.е. $a$ не может быть отменено, если $b$ присутствует в конфигурации. Тогда можем идти назад из $\{a\}$ в $\emptyset$ посредством шага $(\emptyset \cup \{\underline{a}\})$ и не можем идти назад из $\{a,b\}$. Конфигурации в $\mathcal{E}_2$ — это множества $\emptyset$, $\{a\}$, $\{b\}$, $\{a,b\}$. Трассы в $\mathcal{E}_2$ — это префиксы последовательностей:*

$(((\{a\} \cup \emptyset)(\emptyset \cup \{\underline{a}\}))^*(\{a\} \cup \emptyset)(\{b\} \cup \emptyset)$,

$(((\{a\} \cup \emptyset)(\emptyset \cup \{\underline{a}\}))^*(\{a,b\} \cup \emptyset)$,

$(((\{a\} \cup \emptyset)(\emptyset \cup \{\underline{a}\}))^*(\{b\} \cup \emptyset)(\{a\} \cup \emptyset)$.

*Посмотрим, как выполняется показанная на рис. 1(в) ОПСС $\mathcal{E}_3$ с компонентами: $E_3 = \{a,b,c\}$; $<_3 = \emptyset$; $\sharp_3 = \{(a,c),(c,a)\}$; $l_3$ — идентичная функция; $F_3 = \{b\}$; $\prec_3 = \{(a,\underline{b}),\ (b,\underline{b})\}$;*

$\rhd_3 = \emptyset$; $C_0^3 = \emptyset$. *Так как отношение* $<_3$ *пусто, то все события в ОПСС* $\mathcal{E}_3$ *не имеют предшественников по ПСЗ и поэтому из начальной конфигуации* $C_0^3$ *возможны такие вперед-шаги:* $\emptyset \overset{(\{a\}\cup\emptyset)}{\to} \{a\}$, $\emptyset \overset{(\{b\}\cup\emptyset)}{\to} \{b\}$ *и* $\emptyset \overset{(\{c\}\cup\emptyset)}{\to} \{c\}$. *Поскольку события* $a$ *и* $b$ *(*$b$ *и* $c$*) параллельны, то имеем вперед-шаги:* $\emptyset \overset{(\{a,b\}\cup\emptyset)}{\to} \{a,b\}$, $\{a\} \overset{(\{b\}\cup\emptyset)}{\to} \{a,b\}$ *и* $\{b\} \overset{(\{a\}\cup\emptyset)}{\to} \{a,b\}$ *(*$\emptyset \overset{(\{b,c\}\cup\emptyset)}{\to} \{b\}$, $\{b\} \overset{(\{c\}\cup\emptyset)}{\to} \{b,c\}$ *и* $\{c\} \overset{(\{b\}\cup\emptyset)}{\to} \{b,c\}$*). Так как события* $a$ *и* $c$ *конфликтуют, они не могут вместе присутствовать ни в какой конфигурации. Тогда конфигурации* $\{a,b\}$ *и* $\{b,c\}$ *не могут быть расширены соответственно событием* $c$ *и событием* $a$. *Отношение* $\prec_3 = \{(a,\underline{b}),\ (b,\underline{b})\}$ *говорит о том, что событие* $b$ *может быть отменено только, если оба события* $a$ *и* $b$ *уже произошли. Так как отношение* $\rhd_3$ *пусто, то получаем* $\{a,b\} \overset{(\emptyset\cup\{\underline{b}\})}{\to} \{a\}$. *Таким образом,* $\emptyset$, $\{a\}$, $\{b\}$, $\{c\}$, $\{a,b\}$, $\{b,c\}$ *— конфигурации в* $\mathcal{E}_3$. *Трассы в* $\mathcal{E}_3$ *— это префиксы последовательностей:*

$(\{b\}\cup\emptyset)(\{c\}\cup\emptyset)$, $(\{c\}\cup\emptyset)(\{b\}\cup\emptyset)$, $(\{b,c\}\cup\emptyset)$,
$(\{a\}\cup\emptyset)((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*(\{b\}\cup\emptyset)((\emptyset\cup\{\underline{b}\})(\{b\}\cup\emptyset))^*$,
$(\{b\}\cup\emptyset)(\{a\}\cup\emptyset)((\emptyset\cup\{\underline{b}\})(\{b\}\cup\emptyset))^*(\emptyset\cup\{\underline{b}\})((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*(\{b\}\cup\emptyset)$,
$(\{a,b\}\cup\emptyset)((\emptyset\cup\{\underline{b}\})(\{b\}\cup\emptyset))^*(\emptyset\cup\{\underline{b}\})((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*(\{b\}\cup\emptyset)$.

*В конце построим конфигурации и трассы изображенной на рис. 1(г) ОПСС* $\mathcal{E}_4$ *с компонентами:* $E_4 = \{a,b,c\}$; $<_4 = \emptyset$; $\sharp_4 = \{(a,c),(c,a)\}$; $l_4$ *— идентичная функция;* $F_4 = \{b\}$; $\prec_4 = \{(b,\underline{b})\}$; $\rhd_4 = \emptyset$; $C_0^4 = \emptyset$. *Поскольку отношение* $<_4$ *пусто и отношение* $\sharp_4$ *содержит только пары* $(a,c)$ *и* $(c,a)$, *события* $a$ *и* $b$ *(*$b$ *и* $c$*) параллельны и поэтому могут происходить в любом порядке или одновременно. Следовательно, имеем следующие вперед-шаги:* $\emptyset \overset{(\{a\}\cup\emptyset)}{\to} \{a\} \overset{(\{b\}\cup\emptyset)}{\to} \{a,b\}$, $\emptyset \overset{(\{b\}\cup\emptyset)}{\to} \{b\} \overset{(\{a\}\cup\emptyset)}{\to} \{a,b\}$ *и* $\emptyset \overset{(\{a,b\}\cup\emptyset)}{\to} \{a,b\}$ *(*$\emptyset \overset{(\{b\}\cup\emptyset)}{\to} \{b\} \overset{(\{c\}\cup\emptyset)}{\to} \{b,c\}$, $\emptyset \overset{(\{c\}\cup\emptyset)}{\to} \{c\} \overset{(\{b\}\cup\emptyset)}{\to} \{b,c\}$ *и* $\emptyset \overset{(\{b,c\}\cup\emptyset)}{\to} \{b,c\}$*). Единственное событие, необходимое для отмены события* $b \in F_4$, *— само это событие, поскольку* $\prec_4 = \{(b,\underline{b})\}$. *Так как* $\rhd_4 = \emptyset$, *событие* $b$ *может быть отменено в любой конфигурации, где оно присутствует. Конфигурации в* $\mathcal{E}_4$ *— это множества* $\emptyset$, $\{a\}$, $\{b\}$, $\{c\}$, $\{a,b\}$, $\{b,c\}$. *Трассы в* $\mathcal{E}_4$ *— префиксы последовательностей:*

$((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*(\{b\}\cup\emptyset)(\{x\}\cup\emptyset)(\emptyset\cup\{\underline{b}\})((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*(\{b\}\cup\emptyset)$,
$((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*(\{x\}\cup\emptyset)((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*(\{b\}\cup\emptyset)$,
$((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*(\{x,b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\})((\{b\}\cup\emptyset)(\emptyset\cup\{\underline{b}\}))^*(\{b\}\cup\emptyset)$,
*где* $x \in \{a,c\}$. ◇

ОПСС способны моделировать такую особенность обратимых вычислений, как согласованность отношения обратимости с отношением ПСЗ: событие может быть отменено

при условии, что все его последователи по ПСЗ были отменены. Это понятие обратимости естественно для надежных параллельных систем, поскольку при возникновении ошибки система пытается корректно вернуться к предыдущему состоянию.

**Определение 4.** *ОПСС* $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ *называется*

- учитывающей причино-следственную зависимость *(со свойством УПСЗ), если для событий* $e, e' \in E$ *верно, что* $e < e' \;\Rightarrow\; e \ll e'$;
- сохраняющей причино-следственную зависимость *(со свойством СПСЗ), если для событий* $e \in E$ *и* $u \in F$ *верно, что* $e \prec \underline{u} \;\Leftrightarrow\; e = u$, *а также* $e \rhd \underline{u} \;\Leftrightarrow\; u < e$.

Неформально говоря, в ОПСС со свойствами УПСЗ и СПСЗ предшественники по ПСЗ могут быть отменены в текущей конфигурации только, если их последователи по ПСЗ не присутствуют в этой конфигурации. Понятно, что ОПСС со свойством СПСЗ также является ОПСС со свойством УПСЗ.

**Пример 3.** *Сначала вспомним ОПСС* $\mathcal{E}_1$ *из примеров 1 и 2 с компонентами:* $E_1 = \{a, b, c, d, e\}$; $<_1 = \{(b, d), (c, e)\}$; $\sharp_1 = \{(a, b), (b, a), (b, c), (c, b)\}$; $l_1$ — *идентичная функция;* $F_1 = \{b, c\}$; $\prec_1 = \{(b, \underline{b}), (c, \underline{c})\}$; $\rhd_1 = \emptyset$; $C_0^1 = \emptyset$. *Известно из примера 1, что отношение устойчивой ПСЗ* $\ll_1$ *пусто, потому что отношение ПСЗ* $<_1$ *содержит пары* $(b, d)$ *и* $(c, e)$, *а отношение предотвращения* $\rhd_1$ *пусто. Поскольку имеем* $\ll_1 \neq <_1$, $\mathcal{E}_1$ *не обладает свойствами УПСЗ и СПСЗ.*

*Проверим свойства ОПСС* $\mathcal{E}_2$ *из примера 2 с компонентами:* $E_2 = \{a, b\}$; $<_2 = \emptyset$; $\sharp_2 = \emptyset$; $l_2$ — *идентичная функция;* $F_2 = \{a\}$; $\prec_2 = \{(a, \underline{a})\}$; $\rhd_2 = \{(b, \underline{a})\}$; $C_0^2 = \emptyset$. *ОПСС* $\mathcal{E}_2$ *имеет свойство УПСЗ, поскольку отношение* $<_2$ *пусто и, следовательно, отношение* $\ll_2$ *пусто, т.е.* $<_2 = \ll_2$. *С другой стороны, ОПСС* $\mathcal{E}_2$ *не обладает свойством СПСЗ, потому что есть события* $a$ *и* $b$ *такие, что* $b \rhd_2 \underline{a}$ *и* $a \not<_2 b$.

*Расссмотрим ОПСС* $\mathcal{E}_3$ *из примера 2 с компонентами:* $E_3 = \{a, b, c\}$; $<_3 = \emptyset$; $\sharp_3 = \{(a, c), (c, a)\}$; $l_3$ — *идентичная функция;* $F_3 = \{b\}$; $\prec_3 = \{(a, \underline{b}),\ (b, \underline{b})\}$; $\rhd_3 = \emptyset$ *и* $C_0^3 = \emptyset$. *Так как множество последователей по ПСЗ для единственно отменяемого события* $b$ *в* $\mathcal{E}_3$ *пусто, то* $\mathcal{E}_3$ *имеет свойство УПСЗ, но она не обладает свойством СПСЗ, поскольку имеем* $(a, \underline{b}) \in \prec_3$ *и* $a \neq b$.

*В конце проанализируем свойства ОПСС* $\mathcal{E}_4$ *из примера 2 с компонентами:* $E_4 = \{a, b, c\}$; $<_4 = \emptyset$; $\sharp_4 = \{(a, c), (c, a)\}$; $l_4$ — *идентичная функция;* $F_4 = \{b\}$; $\prec_4 = \{(b, \underline{b})\}$; $\rhd_4 = \emptyset$; $C_0^4 = \emptyset$. *ОПСС* $\mathcal{E}_4$ *обладает свойством СПСЗ, а значит, и свойством УПСЗ.*

*Это потому, что верно следующее:* $<_4 = \emptyset$ *и* $\rhd_4 = \emptyset$*, а также предшественник по обратной ПСЗ для отмены единственного отменяемого события* $b$ *— само это событие, поскольку имеем* $F_4 = \{b\}$ *и* $\prec_4 = \{(b, \underline{b})\}$. ◇

Следующая лемма говорит об особенности конфигураций в ОПСС со свойсвом УПСЗ, которые остаются лево-замкнутыми относительно ПСЗ при выполнении ОПСС. Благодаря определениям 2 и 3, истинность леммы следует из леммы 13(i) [24].

**Лемма 1.** *Пусть* $\mathcal{E}$ *— ОПСС со свойством УПСЗ и* $C \in Conf(\mathcal{E})$*. Тогда* $C$ *лево-замкнуто относительно* $<$.

Приведенный ниже пример объясняет приведенную выше лемму.

**Пример 4.** *Вспомним ОПСС* $\mathcal{E}_1$ *(*$<_1 = \{(b, d), (c, e)\}$*) из примеров 1–3, не обладающую свойством УПСЗ. Знаем, что* $\{d\}$, $\{e\}$, $\{b, e\}$, $\{c, d\}$, $\{d, e\}$, $\{b, d, e\}$, $\{c, d, e\}$ *и т.д. — конфигурации в* $\mathcal{E}_1$*. Видим, что эти конфигурации не являются лево-замкнутыми относительно* $<_1$.

*Легко проверить, что в ОПСС* $\mathcal{E}_2$ *и* $\mathcal{E}_3$ *из примеров 2–3, обладающих свойством УПСЗ, все конфигурации лево-замкнуты относительно ПСЗ.* ◇

## 3. Остаточные ОПСС

Оператор удаления для ОПСС, основанный на удалении из структуры событий уже произошедших событий и конфликтующих с ними, необходим для построения остаточных ОПСС.

В отличии от работы [1], где оператор удаления задан для более узкого подкласса ОПСС со свойством СПСЗ на основе понятия конфигурации, введем определение оператора удаления, используя понятие трассы. Это определение упрощает определение оператора удаления из статьи [23] в части построения множеств предшественников неотменяемых событий в шагах трассы.

**Определение 5.** *Пусть* $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ *— ОПСС со свойством УПСЗ и* $t = (A_1 \cup \underline{B}_1) \ldots (A_n \cup \underline{B}_n) \in Traces(\mathcal{E})$ *(*$C_0 \overset{A_1 \cup B_1}{\rightarrow} C_1 \ldots C_{n-1} \overset{A_n \cup B_n}{\rightarrow} C_n$*)* $(n \geq 0)$. Остаточная структура $\mathcal{E} \backslash t$ *для* $\mathcal{E}$ *после* $t$ *посредством оператора* $\backslash$ *удаления определяется по индукции* $0 \leq i \leq n$ *следующим образом:*

$i = 0$. $\mathcal{E} \setminus (t_0 = \epsilon) = (E^0 = E,\ <^0 = <,\ \sharp^0 = \sharp,\ l^0 = l,\ F^0 = F,\ \prec^0 = \prec,\ \rhd^0 = \rhd,\ C_0^0 = C_0) = \mathcal{E}$.

$i > 0$. $\mathcal{E} \setminus t_i = (E^i,\ <^i = <^{i-1} \cap (E^i \times E^i),\ \sharp^i = \sharp^{i-1} \cap (E^i \times E^i),\ l^i = l^{i-1}\,|_{E^i},\ F^i,\ \prec^i = \prec^{i-1}$
$\cap\ (E^i \times \underline{F}^i),\ \rhd^i = \rhd^{i-1} \cap\ (E^i \times \underline{F}^i),\ C_0^i)$, *где*

– $E^i = E^{i-1} \setminus (\widetilde{A}_i \cup \sharp^{i-1}(\widetilde{A}_i))$, *где*
$\widetilde{A}_i = (A_i \setminus F^{i-1}) \ \cup \ \lfloor (A_i \setminus F^{i-1}) \rfloor_{<^{i-1}} = \{\widetilde{a} \in E^{i-1} \mid \exists a \in A_i \setminus F^{i-1} \colon \widetilde{a} <^{i-1} a\})$,
$\sharp^{i-1}(\widetilde{A}_i) = \{a \in E^{i-1} \mid \exists \widetilde{a} \in \widetilde{A}_i \ : \ a \sharp^{i-1} \widetilde{a}\}$;
– $F^i = (F^{i-1} \cap E^i) \setminus \left(\hat{A}_i \cup \hat{\hat{A}}_i\right)$, *где*
$\hat{A}_i = \{e \in F^{i-1} \mid \exists a \in \widetilde{A}_i \colon a \rhd^{i-1} \underline{e}\}$,
$\hat{\hat{A}}_i = \{e \in F^{i-1} \mid \exists a \in \sharp^{i-1}(\widetilde{A}_i) \colon a \prec^{i-1} \underline{e}\}$;
– $C_0^i = ((C_0^{i-1} \setminus B_i) \cup A_i) \cap E^i$.

$\mathcal{E} \setminus t = \mathcal{E} \setminus t_n$.

Интуитивная интерпретация приведенного выше определения заключается в следующем.

$i = 0$. Остаточная структура $\mathcal{E} \setminus \epsilon$ для ОПСС $\mathcal{E}$ после пустой трассы $\epsilon$ — это сама ОПСС.

$i > 0$. На каждом этапе $1 \leq i \leq n$ построения остаточной структуры $\mathcal{E} \setminus t_i$ для ОПСС $\mathcal{E}$ после трассы $t_i = (A_1 \cup \underline{B}_1) \ldots (A_i \cup \underline{B}_i)$ выполняется следующее:

– Множество $E^i$ событий на $i$-ом этапе формируется из множества $E^{i-1}$ событий $(i-1)$-го этапа посредством удаления событий из:
  * множества $\widetilde{A}_i = (A_i \setminus F^{i-1}) \ \cup \ \lfloor (A_i \setminus F^{i-1}) \rfloor_{<^{i-1}}$, где $(A_i \setminus F^{i-1})$ — множество неотменяемых событий, происходящих на $i$-ом шаге, и $\lfloor (A_i \setminus F^{i-1}) \rfloor_{<^{i-1}}$ — множество событий, которые являются предшественниками по ПСЗ для событий из $(A_i \setminus F^{i-1})$ и которые происходят перед $i$-ым шагом;
  * множества $\sharp^{i-1}(\widetilde{A}_i) = \{a \in E^{i-1} \mid \exists \widetilde{a} \in \widetilde{A}_i \ : \ a \sharp^{i-1} \widetilde{a}\}$ событий, конфликтующих с событиями из $\widetilde{A}_i$.

  В силу правила выполнения шага и пункта б) в определении 3, события из множества $\widetilde{A}_i$ присутствуют в конфигурации $C_i$, тогда как по пункту а) определения 3, события из множества $\sharp^{i-1}(\widetilde{A}_i)$ отсутствуют в $C_i$. Понятно, неотменяемые события из множества $\widetilde{A}_i$ не могут быть отменены ни на одном последующем шаге. Тогда в ОПСС со свойством УПСЗ [i] отменяемые события из множества $\widetilde{A}_i$ не могут быть отменены впоследствии, согласно пункту г) определения 3. Поскольку, по пункту а) определения 3, события из $\widetilde{A}_i$ в ОПСС со свойством УПСЗ и события из $\sharp^{i-1}(\widetilde{A}_i)$ в любой ОПСС не могут произойти в дальнейшем, то оператор удаляет события из $(\widetilde{A}_i \cup \sharp^{i-1}(\widetilde{A}_i))$ на $i$-ом этапе.

– Множество $F^i$ отменяемых на $i$-ом этапе событий — это пересечение множеств

[i] Для событий $e, e' \in E$ верно, что $e < e' \ \Rightarrow \ e' \rhd e$, если $e \in F$.

$F^{i-1}$ отменяемых событий на $(i-1)$-ом этапе и $E^{i-1}$ оставшихся на $i$-ом этапе событий, из которого удаляются события из:

- множества $\hat{A}_i$ отменяемых событий на $(i-1)$-ом этапе, которых события из $\widetilde{A}_i$ предотвращают от отмены;
- множества $\hat{\hat{A}}$ отменяемых событий на $(i-1)$-ом этапе, для которых события из $\sharp^{i-1}(\widetilde{A}_i)$ являются обратной причиной.

Поскольку события из $\widetilde{A}_i$ уже присутствуют в конфигурации $C_i$ и не могут быть отменены впоследствии в ОПСС со свойством УПСЗ, то события из $\hat{A}_i$ становятся неотменяемыми, в силу пункта г) определения 3, в ОПСС со свойством УПСЗ. Так как события из $\sharp^{i-1}(\widetilde{A}_i)$ отсутствуют в $C_i$ и не могут произойти в дальнейшем, то события из $\hat{\hat{A}}$ становятся неотменяемыми, согласно пункту в) определения 3.

– На основе полученных множеств $E^i$ и $F^i$ строятся отношения $<^i$, $\sharp^i$, $\prec^i$, $\rhd^i$ и помечающая функция $l^i$ на $i$-ом этапе.

– Начальная конфигурация $C_0^i$ на $i$-ом этапе формируется из событий, оставшихся на $i$-ом этапе и принадлежащих начальной конфигурации на $(i-1)$-ом этапе, измененной посредством выполнения шага $(A_i \cup \underline{B_i})$.

Ниже приведены характерные свойства оператора удаления.

**Лемма 2.** *Для ОПСС* $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ *со свойством УПСЗ (СПСЗ), трассы* $t = (A_1 \cup \underline{B_1}) \ldots (A_n \cup \underline{B_n})$ $(C_0 \stackrel{A_1 \cup B_1}{\rightarrow} C_1 \ \ldots \ C_{n-1} \stackrel{A_n \cup B_n}{\rightarrow} C_n)$ $(n \geq 0)$ *в* $\mathcal{E}$ *и остаточной структуры* $\mathcal{E} \setminus t = (E^n, <^n, \sharp^n, l^n, F^n, \prec^n, \rhd^n, C_0^n)$ *верно:*

*(а)* $E^j \subseteq E^i$, $F^j \subseteq F^i$, $l^j \subseteq l^i$, $\nabla^j \subseteq \nabla^i$ $(\nabla \in \{<, \sharp, \prec, \rhd\})$ *для всех* $0 \leq i \leq j \leq n$;

*(б)* $\mathcal{E} \setminus t_i$ *— ОПСС со свойством УПСЗ (СПСЗ) для всех* $0 \leq i \leq n$;

*(в)* $B_i \subseteq F^{i-1}$ *для всех* $1 \leq i \leq n$;

*(г)* $A_i \subseteq E^{i-1}$ *для всех* $1 \leq i \leq n$;

*(д)* $\widetilde{A}_i \subseteq C_n$ *для всех* $1 \leq i \leq n$;

*(е)* $C_0^n = C_n \cap E^n$.

Доказательство этой леммы аналогично доказательству леммы 3 из статьи [23], но с учетом разницы определения множества предшественников по ПСЗ неотменяемых событий в шаге трассы в определении 5.

Проиллюстрируем применение оператора удаления.

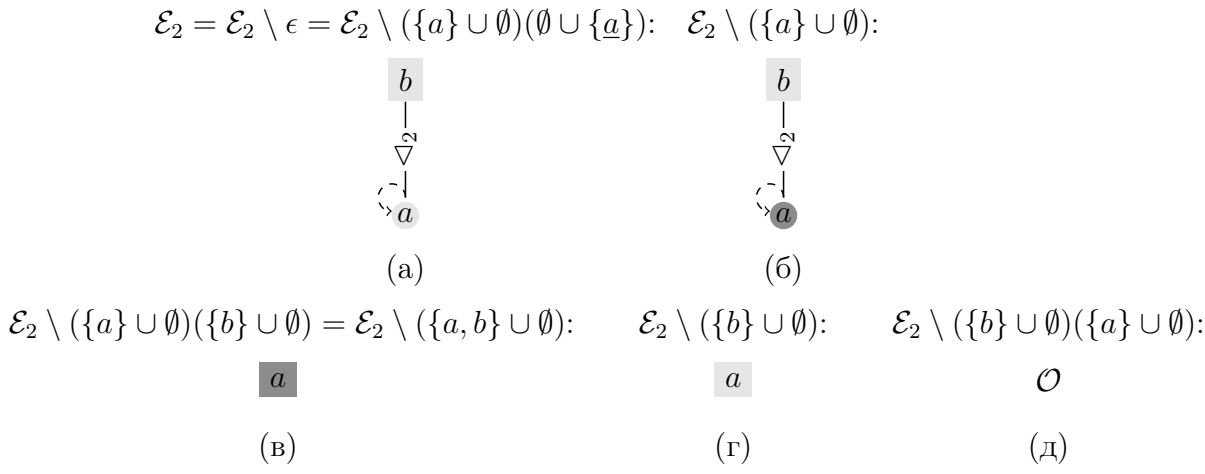



*Рис. 2.* Остаточные структуры для $\mathcal{E}_2$

**Пример 5.** *Рассмотрим обладающую свойством УПСЗ ОПСС $\mathcal{E}_2$ из примеров 2–4 с компонентами: $E_2 = \{a, b\}$; $<_2 = \emptyset$; $\sharp_2 = \emptyset$; $l_2$ — идентичная функция; $F_2 = \{a\}$; $\prec_2 = \{(a, \underline{a})\}$; $\rhd_2 = \{(b, \underline{a})\}$; $C_0^2 = \emptyset$. Знаем, что трассы в $\mathcal{E}_2$ — префиксы последовательностей:* $((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a\}\cup\emptyset)(\{b\}\cup\emptyset)$, $((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{a, b\}\cup\emptyset)$, $((\{a\}\cup\emptyset)(\emptyset\cup\{\underline{a}\}))^*(\{b\}\cup\emptyset)(\{a\} \cup \emptyset)$.

*Применяя оператор удаления к ОПСС $\mathcal{E}_2$ и её трассам, получаем следующие структуры:*

- $\grave{\mathcal{E}}_2 = \mathcal{E}_2 \setminus \epsilon = (E_2, <_2, \sharp_2, l_2, F_2, \prec_2, \rhd_2, C_0^2)$ *(см. рис. 2(а));*
- $\tilde{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (A_1 = \{a\} \cup \underline{B}_1 = \emptyset) = (\tilde{E} = E_2, \tilde{<} = <_2, \tilde{\sharp} = \sharp_2, \tilde{l} = l_2, \tilde{F} = F_2, \tilde{\prec} = \prec_2, \tilde{\rhd} = \rhd_2, \tilde{C}_0 = \{a\})$, *поскольку $(\widetilde{A}_1 \cup \sharp_2(\widetilde{A}_1)) = \emptyset$, благодаря тому, что $a \in F_2$, и $\tilde{C}_0 = ((C_0^2 = \emptyset) \cup (A_1 = \{a\})) \cap (\tilde{E} = \{a, b\}) = \{a\}$ (см. рис. 2(б));*
- $\hat{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (A_1 = \{a\} \cup \underline{B}_1 = \emptyset)(A_2 = \emptyset \cup \underline{B}_2 = \{\underline{a}\}) = \mathcal{E}_2$, *так как $(\widetilde{A}_2 \cup \dot{\sharp}(\widetilde{A}_2)) = \emptyset$, по причине того, что $A_2 = \emptyset$, и $((\tilde{C}_0 = \{a\}) \setminus B_2 = \{a\}) \cap \hat{E}_2 = \emptyset$ (см. рис. 2(а));*
- $\breve{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (A_1 = \{a\} \cup \underline{B}_1 = \emptyset)(A_2 = \{b\} \cup \underline{B}_2 = \emptyset) = (\breve{E} = \{a\}, \breve{<} = \emptyset, \breve{\sharp} = \emptyset; \breve{l} = l_2|_{\{a\}}; \breve{F} = \emptyset; \breve{\prec} = \emptyset; \breve{\rhd} = \emptyset, \breve{C}_0 = \{a\})$, *потому что $\widetilde{A}_2 = \{b\}$, благодаря тому, что $b \in A_2 \setminus \tilde{F}$, и $a \notin \breve{F}$, благодаря тому $(b, \underline{a}) \in \tilde{\rhd}$, а также $\breve{C}_0 = ((\tilde{C}_0 = \{a\}) \cup (A_2 = \{b\})) \cap (\breve{E} = \{a\}) = \{a\}$ (см. рис. 2(в));*
- $\check{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (A_1 = \{a, b\} \cup \underline{B}_1 = \emptyset) = \breve{\mathcal{E}}_2$, *поскольку $\widetilde{A}_1 = \{b\}$, так как $b \in A_1 \setminus F_2$, и $a \notin \check{F}$, так как $(b, \underline{a}) \in \rhd_2$, а также $\check{C}_0 = ((C_2^0 = \emptyset) \cup (A_1 = \{a, b\})) \cap (\check{E} = \{a\}) = \{a\} = \breve{C}_0$ (см. рис. 2(в));*
- $\dot{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (A_1 = \{b\} \cup \underline{B}_1 = \emptyset) = (\dot{E} = \{a\}, \dot{<} = \emptyset, \dot{\sharp} = \emptyset, \dot{l} = l_2|_{\{a\}}, \dot{F} = \emptyset, \dot{\prec} = \emptyset, \dot{\rhd} = \emptyset, \dot{C}_0 = \emptyset)$, *так как $\widetilde{A}_1 = \{b\}$, в силу $b \in A_1 \setminus F_2$, и $a \notin \dot{F}$, в силу $(b, \underline{a}) \in \rhd_2$, а*

*также* $\dot{C}_0 = ((C_2^0 = \emptyset) \cup (A_1 = \{b\})) \cap (\dot{E} = \{a\}) = \emptyset$ *(см. рис. 2(г));*

– $\ddot{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (A_1 = \{b\} \cup \underline{B}_1 = \emptyset)(A_2 = \{a\} \cup \underline{B}_2 = \emptyset) = (\ddot{E} = \emptyset, \ddot{<} = \emptyset, \ddot{\sharp} = \emptyset, \ddot{l} = \emptyset, \ddot{F} = \emptyset,$ $\ddot{\prec} = \emptyset, \ddot{\rhd} = \emptyset, \ddot{C}_0 = \emptyset)$, *потому что* $\widetilde{A}_2 = \{a\}$, *благодаря тому, что* $a \in A_2 \setminus \dot{F}$, *а также* $\ddot{C}_0 = ((\dot{C}_0 = \emptyset) \cup (A_2 = \{a\})) \cap (\dot{E} = \emptyset) = \emptyset$ *(см. рис. 2(д)).*

*Отметим, что оператор удаления создает одинаковые остаточные структуры после различных трасс. Например, легко видеть, что:*

$\grave{\mathcal{E}}_2 = \hat{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (((\{a\} \cup \emptyset)(\emptyset \cup \{\underline{a}\}))^*,$

$\tilde{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (((\{a\} \cup \emptyset)(\emptyset \cup \{\underline{a}\}))^*(\{a\} \cup \emptyset),$

$\breve{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (((\{a\} \cup \emptyset)(\emptyset \cup \{\underline{a}\}))^*(\{a\} \cup \emptyset)(\{b\} \cup \emptyset) =$

$\check{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (((\{a\} \cup \emptyset)(\emptyset \cup \{\underline{a}\}))^*(\{a, b\} \cup \emptyset),$

$\dot{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (((\{a\} \cup \emptyset)(\emptyset \cup \{\underline{a}\}))^*(\{b\} \cup \emptyset),$

$\ddot{\mathcal{E}}_2 = \mathcal{E}_2 \setminus (((\{a\} \cup \emptyset)(\emptyset \cup \{\underline{a}\}))^*(\{b\} \cup \emptyset)(\{a\} \cup \emptyset).$

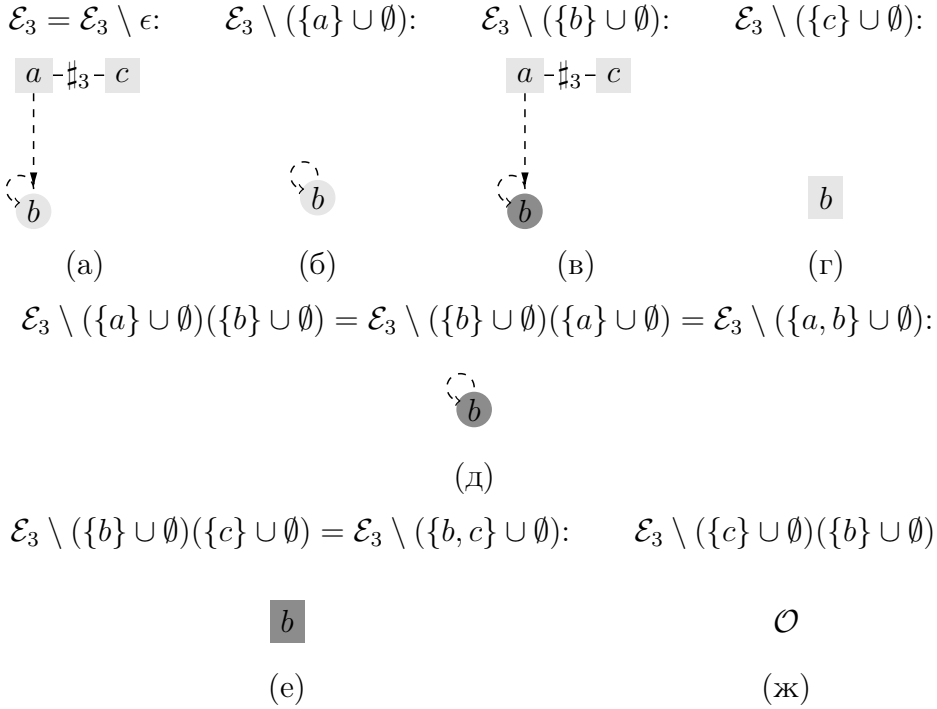



*Рис. 3.* Остаточные структуры для $\mathcal{E}_3$

*Теперь рассмотрим обладающую свойством УПСЗ ОПСС* $\mathcal{E}_3$ *из примеров 2–4 с компонентами:* $E_3 = \{a, b, c\}$; $<_3 = \emptyset$; $\sharp_3 = \{(a, c), (c, a)\}$; $l_3$ *— идентичная функция;* $F_3 = \{b\}$; $\prec_3 = \{(a, \underline{b}), (b, \underline{b})\}$; $\rhd_3 = \emptyset$ *и* $C_0^3 = \emptyset$. *Знаем, что трассы в* $\mathcal{E}_3$ *— это префиксы последовательностей:*

$(\{b\} \cup \emptyset)(\{c\} \cup \emptyset)$, $(\{c\} \cup \emptyset)(\{b\} \cup \emptyset)$, $(\{b, c\} \cup \emptyset)$,

$(\{a\} \cup \emptyset)((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)((\emptyset \cup \{\underline{b}\})(\{b\} \cup \emptyset))^*$,

$(\{b\} \cup \emptyset)(\{a\} \cup \emptyset)((\emptyset \cup \{\underline{b}\})(\{b\} \cup \emptyset))^*(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)$,

$(\{a, b\} \cup \emptyset)((\emptyset \cup \{\underline{b}\})(\{b\} \cup \emptyset))^*(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)$.

*Остаточные структуры для ОПСС $\mathcal{E}_3$ после некоторых её трасс показаны на рис. 3. А также имеем:*

$\mathcal{E}_3 \setminus (\{a\} \cup \emptyset) = \mathcal{E}_3 \setminus (\{a\} \cup \emptyset)((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^\star = \mathcal{E}_3 \setminus (\{b\} \cup \emptyset)(\{a\} \cup \emptyset)((\emptyset \cup \{\underline{b}\})(\{b\} \cup \emptyset))^*(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*$;

$\mathcal{E}_3 \setminus (\{a\} \cup \emptyset)(\{b\} \cup \emptyset) = \mathcal{E}_3 \setminus (\{a\} \cup \emptyset)((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)((\emptyset \cup \{\underline{b}\})(\{b\} \cup \emptyset))^* = \mathcal{E}_3 \setminus (\{b\} \cup \emptyset)(\{a\} \cup \emptyset)((\emptyset \cup \{\underline{b}\})(\{b\} \cup \emptyset))^*(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset) = \mathcal{E}_3 \setminus (\{a, b\} \cup \emptyset)((\emptyset \cup \{\underline{b}\})(\{b\} \cup \emptyset))^*(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)$.

*Поэтому на рис. 3 показаны остаточные структуры для ОПСС $\mathcal{E}_3$ после всех её трасс.*

*Далее рассмотрим обладающую свойством СПСЗ ОПСС $\mathcal{E}_4$ из примеров 2–4 с компонентами: $E_4 = \{a, b, c\}$; $<_4 = \emptyset$; $\sharp_4 = \{(a, c), (c, a)\}$; $l_4$ — идентичная функция; $F_4 = \{b\}$; $\prec_4 = \{(b, \underline{b})\}$; $\rhd_4 = \emptyset$; $C_0^4 = \emptyset$. Знаем, что трассы в $\mathcal{E}_4$ — это префиксы последовательностей:*

$((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)(\{x\} \cup \emptyset)(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)$,

$((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{x\} \cup \emptyset)((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)$,

$((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{x, b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)$.

*Далее пусть $x \neq x' \in \{a, c\}$. Остаточные структуры для ОПСС $\mathcal{E}_4$ после её некоторых трасс показаны на рис. 4. Также верно:*

$\mathcal{E}_4 \setminus \epsilon = \mathcal{E}_4 \setminus ((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^\star$;

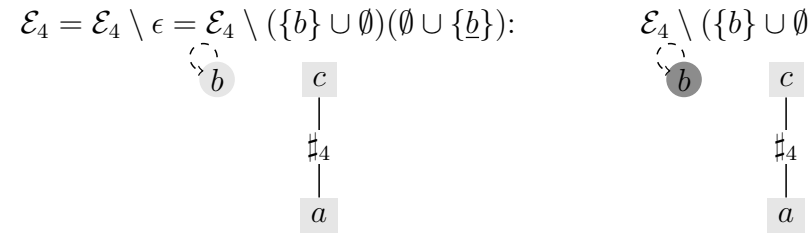
$\mathcal{E}_4 \setminus (\{b\} \cup \emptyset) = \mathcal{E}_4 \setminus (\{b\} \cup \emptyset)((\emptyset \cup \{\underline{b}\})(\{b\} \cup \emptyset))^\star$;

$\mathcal{E}_4 \setminus (\{x\} \cup \emptyset) = \mathcal{E}_4 \setminus ((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^\star(\{x\} \cup \emptyset)((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^\star = ((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^\star(\{b\} \cup \emptyset)(\{x\} \cup \emptyset)(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^* = \mathcal{E}_4 \setminus ((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{x, b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*$;

$\mathcal{E}_4 \setminus ((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)(\{x\} \cup \emptyset)(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset) = \mathcal{E}_4 \setminus ((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{x\} \cup \emptyset)((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset) = \mathcal{E}_4 \setminus ((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{x, b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\})((\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\}))^*(\{b\} \cup \emptyset)$.

*Поэтому на рис. 4 показаны остаточные структуры для ОПСС $\mathcal{E}_4$ после всех её трасс.*

$\diamond$




(а) (б)

$\mathcal{E}_4 \setminus (\{x\} \cup \emptyset)(\{b\} \cup \emptyset) = \mathcal{E}_4 \setminus (\{b\} \cup \emptyset)(\{x\} \cup \emptyset) = \mathcal{E}_4 \setminus (\{b, x\} \cup \emptyset)$:


(в)

$\mathcal{E}_4 \setminus (\{x\} \cup \emptyset) = \mathcal{E}_4 \setminus (\{x\} \cup \emptyset)(\{b\} \cup \emptyset)(\emptyset \cup \{\underline{b}\})$:


(г)

*Рис. 4.* Остаточные структуры для $\mathcal{E}_4$

Покажем, что остаточные структуры для ОПСС, обладающие свойством СПСЗ, являются инвариантом относительно эквивалентных трасс.

**Утверждение 1.** *Для ОПСС со свойством СПСЗ $\mathcal{E}$ и трасс $t, t' \in Trace(\mathcal{E})$ таких, что $[t] = [t']$, верно $\mathcal{E} \setminus t = \mathcal{E} \setminus t'$.*

**Пример 6.** *Рассмотрим обладающую свойством СПСЗ ОПСС $\mathcal{E}_4$ из примеров 2–5. Знаем, что $\{a\}$ — конфигурация в $\mathcal{E}_4$, $t = (\{a\} \cup \emptyset)$ и $t' = (\{b\} \cup \emptyset)(\{a\} \cup \emptyset)(\emptyset \cup \{\underline{b}\})$ — трассы в $\mathcal{E}_4$. По определению 3, получаем, что $last(t) = (C_0^4 \setminus \emptyset) \cup \{a\} = \{a\}$ и $last(t') = (((((C_0^4 \setminus \emptyset) \cup \{b\}) \setminus \emptyset) \cup \{a\}) \setminus \{b\}) \cup \emptyset = \{a\}$, т.е. $t \sim t'$. В примере 5 показано, что $\mathcal{E}_4 \setminus t = \mathcal{E}_4 \setminus t'$.*

*Мы оставляем читателю проверку остаточных структур для $\mathcal{E}_4$ после других эквивалентных трасс.*

*С другой стороны, приведенное выше утверждение неверно для обладающей свойством УПСЗ, но не свойством СПСЗ ОПСС $\mathcal{E}_2$ из примеров 2–5. Знаем, что $\{a, b\}$ — конфигурация в $\mathcal{E}_2$, $t = (\{a\} \cup \emptyset)(\{b\} \cup \emptyset)$ и $t' = (\{b\} \cup \emptyset)(\{a\} \cup \emptyset)$ — трассы в $\mathcal{E}_2$. В силу определения 3, имеем $last(t) = \{a, b\}$ и $last(t') = \{a, b\}$, т.е. $t \sim t'$. В примере 5 показано, что $\breve{\mathcal{E}}_2 = \mathcal{E}_2 \setminus t \neq \mathcal{E}_2 \setminus t' = \ddot{\mathcal{E}}_2$.* ◇

## 4. Семантика систем переходов для ОПСС

В этом разделе сначала приводятся базовые определения, касающиеся систем переходов, а затем для ОПСС $\mathcal{E}$ определяются отображения $TC(\mathcal{E})$ и $TR(\mathcal{E})$, которые строят два различных типа систем переходов.

На основе множества $L$ действий в ОПСС определим множество $\mathbb{L} := \mathbb{N}_0^L$ (множество мультимножеств на $L$ или функций из $L$ в множество неотрицательных целых чисел), которое будем использовать как множество меток в системах переходов.

Система переходов $\mathcal{T} = (S, \to, i)$, помеченная на множестве $\mathbb{L}$ меток, состоит их множества $S$ состояний, отношения перехода $\to \subseteq S \times \mathbb{L} \times S$ и начального состояния $i \in S$. Две системы переходов, помеченные на множестве $\mathbb{L}$, являются *изоморфными*, если существует биекция между их состояниями, сохраняющая отношение перехода и начальное состояние. Будем говорить, что отношение $R \subseteq S \times S'$ является *бисимуляцией* между системами переходов $\mathcal{T} = (S, \to, i)$ и $\mathcal{T}' = (S', \to', i')$, помеченными на $\mathbb{L}$, если $(i, i') \in R$ и для всех пар $(s, s') \in R$ и меток $\lambda \in \mathbb{L}$: если $(s, \lambda, s_1) \in \to$, то $(s', \lambda, s_1') \in \to'$ и $(s_1, s_1') \in R$ для некоторого состояния $s_1' \in S'$; а также если $(s', \lambda, s_1') \in \to'$, то $(s, \lambda, s_1) \in \to$ и $(s_1, s_1') \in R$ для некоторого состояния $s_1 \in S$. Две системы переходов, помеченные на множестве $\mathbb{L}$, являются *бисимуляционными*, если существует отношение бисимуляции между ними.

Определим понятие системы переходов, имеющей в качестве состояний конфигурации ОПСС.

**Определение 6.** *Для ОПСС $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$, помеченной на множестве $L$ действий,*

$TC(\mathcal{E})$ — конфигурационная система переходов $(Conf(\mathcal{E}), \rightharpoonup, C_0)$, *помеченная на множестве $\mathbb{L}$ меток,*

*где* $C \overset{M}{\rightharpoonup} C' \iff C \overset{(A \cup \underline{B})}{\to} C'$ *в* $\mathcal{E}$ *и* $M = l(A \cup \underline{B})$.

Объясним приведенное выше определение на примере.

**Пример 7.** *Рассмотрим ОПСС $\mathcal{E}_3$ (см. примеры 2–5). Из примера 2 знаем, что $\emptyset$, $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{b, c\}$ — конфигурации в $\mathcal{E}_3$. Там также приведены пояснения для переходов между конфигурациями. Используя определения 3 и 6, получаем конфигурационную систему переходов $TC(\mathcal{E}_3)$ (см. рис. 5).* ◇

Теперь рассмотрим определение системы переходов, имеющей в качестве состояний остаточные структуры для ОПСС.

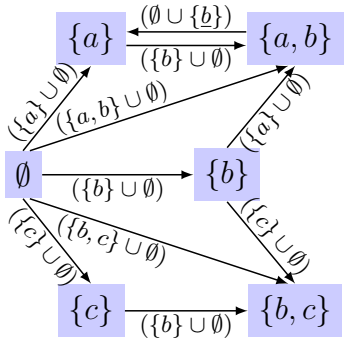



*Рис. 5.* Конфигурационная система переходов $TC(\mathcal{E}_3)$

**Определение 7.** *Для ОПСС* $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$, *помеченной на множестве* $L$ *действий,*

$TR(\mathcal{E})$ — остаточная система переходов $(Reach(\mathcal{E}), \rightharpoonup, \mathcal{E})$, *помеченная на множестве* $\mathbb{L}$ *меток,*

*где* $\mathcal{F} \overset{M}{\rightharpoonup} \mathcal{F}' \iff \mathcal{F}' = \mathcal{F} \setminus (A \cup \underline{B})$ *и* $M = l(A \cup \underline{B})$, *а также* $Reach(\mathcal{E}) = \{\mathcal{F} \mid \exists \mathcal{E}_0, \ldots, \mathcal{E}_k$ $(k \geq 0)$ *такие, что* $\mathcal{E}_0 = \mathcal{E} \setminus \epsilon$, $\mathcal{E}_k = \mathcal{F}$ *и* $\mathcal{E}_i \overset{l(A \cup \underline{B})}{\rightharpoonup} \mathcal{E}_{i+1}$ $(0 \leq i < k)\}$.

Проиллюстрируем это определение на примере.

**Пример 8.** *Рассмотрим ОПСС* $\mathcal{E}_3$ *из примеров 2–5. Используя определения 5 и 7, построим остаточную систему переходов* $TR(\mathcal{E}_3)$ *(см. рис. 6). Видим, что конфигурационная система переходов* $TC(\mathcal{E}_3)$ *(см. рис. 5) и остаточная система переходов* $TR(\mathcal{E}_3)$ *бисимуляционны, но не являются изоморфными.* ◇

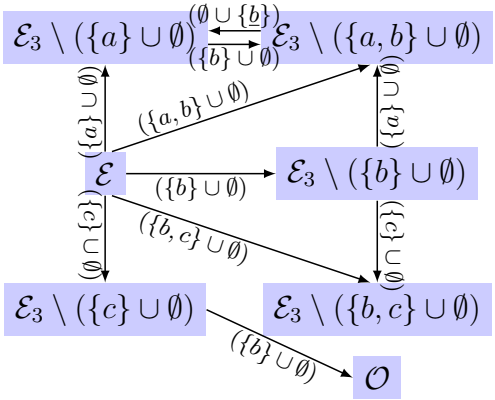



*Рис. 6.* Остаточная система переходов $TR(\mathcal{E}_3)$

**Теорема 1.** *Для помеченной на* $L$ *ОПСС* $\mathcal{E}$ *со свойством УПСЗ верно, что* $TC(\mathcal{E})$ *и* $TR(\mathcal{E})$ *бисимуляционно эквивалентны и не являются изоморфными в общем случае.*

Доказательство этой теоремы аналогично доказательству теоремы 1 из статьи [23], но с

учетом разницы определения множества предшественников по ПСЗ неотменяемых событий в шаге трассы в определении 5.

## 5. Теоретико-категорная характеризация отображений $TC(\cdot)$ и $TR(\cdot)$

В этом разделе сначала рассматриваются определения категорий моделей систем переходов и ОПСС, а затем устанавливается, можно ли отображения $TC(\mathcal{E})$ и $TR(\mathcal{E})$, где $\mathcal{E}$ — ОПСС со свойством УПСЗ, расширить до функторов между этими категориями.

Определим понятие морфизма между двумя системами переходов, помеченными на множестве $\mathbb{L}$ меток.

**Определение 8.** *Пусть* $\mathcal{T} = (S, \rightarrow, i)$ *и* $\mathcal{T}' = (S', \rightarrow', i')$ *— системы переходов, помеченные на множестве* $\mathbb{L}$ *меток. Функция* $\nu : S \to S'$ *— это* морфизм *из* $\mathcal{T}$ *в* $\mathcal{T}'$*, если верно:* $\nu(i) = i'$ *и для всех* $s, s_1 \in S$ *и для всех* $\lambda \in \mathbb{L}$ *выполняется: если* $(s, \lambda, s_1) \in \rightarrow$ *в* $\mathcal{T}$*, то* $(\nu(s), \lambda, \nu(s_1)) \in \rightarrow'$ *в* $\mathcal{T}'$*.*

Видим, что морфизм, определенный выше, представляет понятие симуляции одной системы переходов другой.

Теперь введем понятие морфизма между двумя ОПСС, помеченными на множестве $L$ действий.

**Определение 9.** *Пусть* $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ *и* $\mathcal{E}' = (E', <', \sharp', l', F', \prec', \rhd', C_0')$ *— ОПСС, помеченные на множестве* $L$ *действий. Функция* $\mu : E \to E'$ *— это* морфизм *из* $\mathcal{E}$ *в* $\mathcal{E}'$*, если верно:*

*1)* $\forall e \in E \ : \ \lfloor \mu(e) \rfloor_{<'} \subseteq \mu(\lfloor e \rfloor_<)$[ii]*;*

*2)* $\forall e, e' \in E \ : \ \mu(e) \ \sharp' \ \mu(e') \ \Rightarrow \ e \ \sharp \ e'$*;*

*3)* $\forall e \neq e' \in E \ : \ \mu(e) = \mu(e') \ \Rightarrow \ e \ \sharp \ e'$*;*

*4)* $l' \circ \mu = l$*;*

*5)* $\mu(F) \subseteq F'$*;*

*6)* $\forall u \in F \ : \ \llcorner \underline{\mu(u)} \lrcorner_{\prec'} \subseteq \mu(\llcorner \underline{u} \lrcorner_\prec)$*;*

*7)* $\forall e \in E, \ \forall u \in F \ : \ \mu(e) \rhd' \underline{\mu(u)} \ \Rightarrow \ e \rhd \underline{u}$*;*

*8)* $\mu(C_0) = C_0'$*.*

Покажем, что морфизм, определенный выше, представляет понятие симуляции поведения одной ОПСС поведением другой ОПСС.

[ii]Здесь и далее для подмножества $X \subseteq E$ будем использовать запись $\mu(X)$, чтобы обозначать множество $\{\mu(e) \mid e \in X\}$.

**Лемма 3.** *Пусть $\mathcal{E}$, $\mathcal{E}'$ — ОПСС, помеченные на множестве $L$ действий, и $\mu : \mathcal{E} \to \mathcal{E}'$ — морфизм. Если $C \stackrel{(A\subseteq B)}{\to} C_1$ в $\mathcal{E}$ для некоторых $C, C_1 \in Conf(\mathcal{E})$, то $\mu(C), \mu(C_1) \in Conf(\mathcal{E}')$ и $\mu(C) \stackrel{(\mu(A)\subseteq\mu(B))}{\to} \mu(C_1)$ в $\mathcal{E}'$.*

Определим категории систем переходов и ОПСС со свойством УПСЗ.

**Определение 10.** *Помеченные на $\mathbb{L}$ системы переходов (помеченные на $L$ ОПСС со свойством УПСЗ) и морфизмы между ними формируют категорию $\mathbf{TS}_{\mathbb{L}}$ ($\mathbf{RPES}_L$), в которой композиция двух морфизмов определяется как обычная композиция функций, а тождественный морфизм является тождественной функцией.*

**Лемма 4.** *Пусть $\mathbf{RPES}_L^0$ — подкатегория категории $\mathbf{RPES}_L$, содержащая в качестве объектов ОПСС вида $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, \emptyset)$ со свойством УПСЗ. Тогда категория $\mathbf{RPES}_L^0$ имеет копроизведения.*

**Утверждение 2.** *Отображение TC (TR) может быть расширено (не может быть расширено) до функтора из категории $\mathbf{RPES}_L$ в категорию $\mathbf{TS}_{\mathbb{L}}$.*

Этот результат показывает разницу между отображениями $TC(\mathcal{E})$ и $TR(\mathcal{E})$ для ОПСС $\mathcal{E}$ со свойством УПСЗ.

## 6. Заключение

В этой статье в контексте обратимых первичных структур событий, учитывающих причинно-следственные зависимости между событиями, была дана теоретико-категорная характеризация построений семантик в терминах систем переходов, основанных на конфигурациях и остаточных структурах. С этой целью, во-первых, была определена (шаговая) семантика рассматриваемой модели обратимых структур событий, которая основана на конфигурациях/трассах и которая строится из начальной конфигурации посредством добавления произошедших параллельных событий и/или посредством отмены ранее произошедших событий. Во-вторых, были исследованы свойства оператора удаления, который используется для построения остаточных структур, получаемых из заданной структуры событий посредством удаления из неё уже произошедших событий и конфликтующих с ними в ходе выполнения структуры. Есть надежда, что при разработке операционной семантики алгебраических исчислений параллельных процессов полученные здесь результаты могут быть столь же полезны для обратимых первичных структур событий, как и для традиционных (необратимых) (см. [9, 18] среди прочих).

В дальнейшем планируется расширить список рассматриваемых моделей, включив обратимые версии потоковых, расслоенных, обобщенных структур событий с симметричным и асимметричным конфликтом. Другой целью дальнейших исследований является изучение возможности получения изоморфизма, а не бисимуляции, между двумя типами семантик систем переходов за счёт обогащения модели обратимых первичных структур событий событиями, которые присутствуют в структуре, но которые не могут произойти из-за, например, отсутствия транзитивности/ацикличности в ПСЗ, наличия бесконечного количества предшественников по ПСЗ и т.д., как это было сделано для традиционных первичных структур событий в статье [7]. Там авторы смогли доказать, что наличие событий, которые не могут произойти, полезно при сравнительном анализе различных семантик, способствуя устранению несущественных несоответствий между исследуемыми моделями.

## Список литературы


1. Грибовская Н.С., Вирбицкайте И.Б.: Семантика систем переходов структур событий с отменяемыми событиями при сохранении причинно-следственной зависимости. Системная информатика, 2024, №24, стр. 59–90.
2. Arbach, Y., Karcher, D., Peters, K., Nestmann, U.: Dynamic causality in event structures. *Lecture Notes in Computer Science* **9039** (2015) 83–97. URL: https://doi.org/10.1007/978-3-319-19195-9_6
3. Aubert, C., Cristescu, I.: Contextual equivalences in configuration structures and reversibility. *J. Log. Algebr. Meth. Program.* 86(1), 77–106 (2017) URL: https://doi.org/10.1016/j.jlamp.2016.08.004
4. Baier, C., Majster-Cederbaum, M.: The connection between event structure semantics and operational semantics for TCSP. *Acta Informatica* 31 (1994). URL: https://doi.org/10.1007/BF01178923
5. Barylska, K., Gogolinska, A., Mikulski, L., Philippou, A., Piatkowski, M., Psara, K.: Formal translation from reversing Petri nets to coloured Petri nets. Lecture Notes in Computer Science, 13352, 172–186 (2022). URL: https://doi.org/10.1007/978-3-031-09005-9_12
6. Best, E., Gribovskaya, N., Virbitskaite, I.: Configuration- and residual-based transition systems for event structures with asymmetric conflict. Proc. SofSem 2017, *Lecture Notes in Computer Science* **10139** (2017) 132–146. URL: https://doi.org/10.1007/978-3-

319-51963-0_11

7. Best, E., Gribovskaya, N., Virbitskaite, I.: From event-oriented models to transition systems. In: Proc. Petri Nets 2018, *Lecture Notes in Computer Science* **10877** (2018) 117–139. URL: https://doi.org/10.1007/978-3-319-91268-4_7
8. Boudol, G.: Flow event structures and flow nets. *Lecture Notes in Computer Science* **469** (1990) 62–95. URL: https://doi.org/10.1007/3-540-53479-2_4
9. Boudol G., Castellani I.: Concurrency and atomicity. *Theoretical Computer Science* **59** (1988) 25–84. URL: https://doi.org/10.1016/0304-3975(88)90096-5
10. Crafa, S., Varacca, D., Yoshid, N.: Event structure semantics of parallel extrusion in the pi-calculus. *Lecture Notes in Computer Science* **7213** (2012) 225–239.
11. Cristescu, I., Krivine, J., Varacca, D.: A compositional semantics for the reversible pi-calculus. In: Proceedings of LICS 2013. IEEE Computer Society, pp. 388–397 (2013) URL: https://doi.org/10.1109/LICS.2013.45
12. Danos, V., Krivine, J.: Reversible communicating systems. In: Proceedings of CONCUR'04, volume 3170 of Lecture Notes in Computer Science, pp. 292–307, 2004. URL: https://doi.org/10.1007/978-3-540-28644-8_19
13. De Vos, A., De Baerdemacker, S., Van Rentergem, Y.: Synthesis of quantum circuits vs. Synthesis of classical reversible circuits. Synthesis Lectures on Digital Circuits and Systems. Morgan & Claypool Publishers, 2018. URL: https://doi.org/10.2200/S00856ED1V01Y201805DCS054
14. de Frutos Escrig, D., Koutny, M., Mikulski, L.: Reversing steps in Petri nets. In: Donatelli, S., Haar, S. (eds.) PETRI NETS 2019. *Lecture Notes in Computer Science* vol. 11522, pp. 171–191. Springer, Cham (2019). URL: https://doi.org/10.1007/978-3-030-21571-2 11
15. van Glabbeek, R.J.: On the expressiveness of higher dimensional automata. *Theoretical Computer Science* **356(3)** (2006) 265–290. URL: https://doi.org/10.1016/j.tcs.2006.02.012
16. van Glabbeek, R.J., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Inform. 37, 229–327 (2001) URL: https://doi.org/10.1007/s002360000041
17. van Glabbeek R.J., Plotkin G.D.: Configuration structures, event structures and Petri nets. *Theoretical Computer Science* **410(41)** (2009) 4111-4159. URL: https://doi.org/10.1016/j.tcs.2009.06.014

18. van Glabbeek, R.J., Vaandrager F.W.: Bundle event structures and CCSP. *Lecture Notes in Computer Science* **2761** (2003) 57–71. URL: https://doi.org/10.1007/978-3-540-45187-7_4
19. Gorrieri, R., Lanese, I.: Decidable Reversible Equivalences for Finite Petri Nets. CoRR abs/2506.11517 (2025) https://doi.org/10.48550/arXiv.2506.11517
20. Graversen, E., Phillips, I.C.C., Yoshida, N.: Towards a categorical representation of reversible event structures. J. Log. Algebraic Methods Program. 104: 16–59 (2019) URL: https://doi.org/10.1016/j.jlamp.2019.01.001
21. Graversen, E., Phillips, I.C.C., Yoshida, N.: Event structure semantics of (controlled) reversible CCS. J. Log. Algebraic Methods Program. 121: 100686 (2021) URL: https://doi.org/10.1016/j.jlamp.2021.100686
22. Graversen, E., Phillips, I.C.C., Yoshida, N.: Event structures for the reversible early internal $\pi$-calculus. *Journal of Logical and Algebraic Methods in Programming* **124** 100720.
23. Gribovskaya N., Virbitskaite I.: Comparative transition system semantics for cause-respecting reversible prime event structures. Electronic Proceedings in Theoretical Computer Science, vol. 386, pp. 112–126.
24. Gribovskaya N., Virbitskaite I.: Transition systems from asymmetric prime event structures with cause-respecting reversibility. To appear in International Journal of Foundations of Computer Science. URL: https://doi.org/10.1142/S0129054125460049
25. Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general Petri nets. *Theoretical Computer Science* **153** (1996) 129–170. URL: https://doi.org/10.1016/0304-3975(95)00120-4
26. Kari, J.: Reversible cellular automata: from fundamental classical results to recent developments. New Generation Comput. 36(3), 145–172 (2018) URL: https://doi.org/10.1007/s00354-018-0034-6
27. Katoen, J.-P.: Quantitative and qualitative extensions of event structures. PhD Thesis, Twente University, 1996.
28. Katoen, J.-P., Langerak, R., Latella, D.: Modeling systems by probabilistic process algebra: an event structures approach. *IFIP Transactions* **C-2** (1993) 253–268.
29. S. Kuhn, B. Aman, G. Ciobanu, A. Philippou, K. Psara, and I. Ulidowski. Reversibility in chemical reactions. In I. Ulidowski, I. Lanese, U. P. Schultz, and C. Ferreira, editors, Reversible Computation: Extending Horizons of Computing – Selected Results of

the COST Action IC1405, volume 12070 of Lecture Notes in Computer Science, pages 151–176. Springer, 2020. URL: https://doi.org/10.1007/978-3-030-47361-7_7

30. Langerak, R.: Bundle event structures: a non-interleaving semantics for LOTOS. Formal Description Techniques V. *IFIP Transactions* **C-10** (1993) 331–346.
31. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani J.-B.: Concurrent flexible reversibility. In Matthias Felleisen and Philippa Gardner, editors, Programming Languages and Systems – 22nd European Symposium on Programming, ESOP 2013, volume 7792 of Lecture Notes in Computer Science, pages 370–390. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-37036-6_21
32. Lanese, I., Mezzina, C.A., Stefani J.-B.: Reversibility in the higher-order $\pi$-calculus. Theoretical Computer Science, 625:25–84, 2016. URL: https://doi.org/10.1016/j.tcs.2016.02.019,
33. Lanese, I., Palacios, A., Vidal, G.: Causal-consistent replay debugging for message passing programs. In J. A. Pérez and N. Yoshida, editors, Formal Techniques for Distributed Objects, Components, and Systems – 39th IFIP WG 6.1 International Conference, FORTE 2019, volume 11535 of Lecture Notes in Computer Science, pages 167–184. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-21759-4_10
34. Loogen, R., Goltz, U.: Modelling nondeterministic concurrent processes with event structures. *Fundamenta Informatica* **14(1)** (1991) 39–74.
35. Majster-Cederbaum, M., Roggenbach, M.: Transition systems from event structures revisited. *Information Processing Letters* **67(3)** (1998) 119-124. URL: https://doi.org/10.1016/S0020-0190(98)00105-7
36. Medic, D., Mezzina, C. A., Phillips, I., Yoshida, N.: Towards a formal account for software transactional memory. In I. Lanese and M. Rawski, editors, Reversible Computation – 12th International Conference, RC 2020, volume 12227 of Lecture Notes in Computer Science, pages 255–263. Springer, 2020. URL: https://doi.org/10.1007/978-3-030-52482-1_16
37. Melgratti, H.C., Mezzina, C.A., Pinna, G.M.: A distributed operational view of reversible prime event structures. In: 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), 2021, pp. 1–13. URL: https://doi: 10.1109/LICS52264.2021.9470623.
38. Melgratti, H.C., Mezzina, C.A., Ulidowski, I.: Reversing P/T Nets. In:

COORDINATION 2019, volume 11533 of Lecture Notes in Computer Science. Springer, 2019 pp. 19–36. URL: https://doi.org/10.1007/978-3-030-22397-7_2

39. Nielsen M., Plotkin G., Winskel G.: Petri nets, event structures and domains. *Theoretical Computer Science*, **13**(1) (1981) 85–08. URL: https://doi.org/10.1016/0304-3975(81)90112-2
40. Nielsen M., Thiagarajan P.S.: Regular event structures and finite Petri nets: the conflict-free case. *Lecture Notes in Computer Science* **2360** (2002) 335–351. URL: https://doi.org/10.1007/3-540-48068-4_20
41. Philippou, A., Psara, K.: A collective interpretation semantics for reversing Petri nets. *Theor. Comput. Sci.* **924** (2022) 148–170. URL: https://doi.org/10.1016/j.tcs.2022.05.016
42. Philippou, A., Psara, K.: Reversible computation in cyclic Petri nets. CoRR abs/2010.04000 (2020) URL: https://arxiv.org/abs/2010.04000
43. Phillips, I.C.C., Ulidowski, I.: A hierarchy of reverse bisimulations on stable configuration structures. Math. Struct. Comput. Sci. 22, 333–372 (2012) URL: https://doi.org/10.1017/S0960129511000429
44. Phillips, I.C.C., Ulidowski, I.: Event identifier logic. Math. Struct. Comput. Sci. 24, 1–51 (2014) URL: https://doi.org/10.1017/S0960129513000510
45. Phillips, I.C.C., Ulidowski, I.: Reversibility and asymmetric conflict in event structures. Logic and Algebraic Methods in Programming 84(6), 781–805 (2015) URL: https://doi.org/10.1016/j.jlamp.2015.07.004
46. Pinna, G.M.: Reversing steps in membrane systems computations. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Zandron, C. (eds.) CMC 2017. LNCS, vol. 10725, pp. 245–261. Springer, Cham (2018). URL: https://doi.org/10.1007/978-3-319-73359-3 16
47. Ulidowski, I., Phillips, I.C.C., Yuen, S.: Reversing event structures. New Generation Computing 36(3), 281–306 (2018) URL: https://doi.org/10.1007/s00354-018-0040-8
48. Winskel, G.: Events in computation. PhD Thesis, University of Edinburgh, 1980.
49. Winskel, G.: Distributed probabilistic and quantum strategies. *Electronic Notes in Theoretical Computer Science* **298** (2013) 403–425. URL: https://doi.org/10.1016/j.entcs.2013.09.024

## Приложение А

Приведем базовые определения из теории категорий. Напомним, что категория $\mathcal{C}$ состоит из множества $Ob$, элементы которого называются объектами категории, и множества $Mor$, элементы которого называются морфизмами этой категории, при этом

- каждой упорядоченной паре объектов $A, B \in Ob$ сопоставлено некоторое множество морфизмов $Hom(A, B)$ из $Mor$, а каждому морфизму $f \in Mor$ соответствует единственная упорядоченная пара объектов $A, B \in Ob$ (обозначается $f : A \to B$);
- для любых двух морфизмов $f \in Hom(A, B)$ и $g \in Hom(B, C)$ задана операция композиции морфизмов $g \circ f \in Hom(A, C)$, обладающая ассоциативностью, то есть $f \circ (g \circ h) = (f \circ g) \circ h$ для всех морфизмов $h \in Hom(A, B)$, $g \in Hom(B, C)$ и $f \in Hom(C, D)$;
- для каждого объекта $A \in Ob$ задан тождественный морфизм $id_A \in Hom(A, A)$, действующий тривиально, то есть $f \circ id_A = id_B \circ f = f$ для любого морфизма $f \in Hom(A, B)$.

Будем говорить, что категория $\mathcal{C}$ имеет (конечные) копроизведения, если для любых двух объектов $A, B \in Ob$ существует объект $A \oplus B \in Ob$ и морфизмы $\pi_A : A \to A \oplus B$ и $\pi_B : B \to A \oplus B$ такие, что для любого объекта $C \in Ob$ с любой парой морфизмов $\lambda_A : A \to C$ и $\lambda_B : B \to C$ существует единственный морфизм $\lambda : A \oplus B \to C$ такой, что $\lambda \circ \pi_A = \lambda_A$ и $\lambda \circ \pi_B = \lambda_B$.

Для сравнения двух категорий и построения взаимосвязей между ними в теории категорий используется понятие функтора, то есть отображения, сохраняющего структуру категорий. Формально, функтор $F : \mathcal{C} \to \mathcal{C}'$ между двумя категориями $\mathcal{C}$ и $\mathcal{C}'$ – это отображение, которое каждому объекту $A \in Ob$ категории $\mathcal{C}$ ставит в соответствие объект $F(A) \in Ob'$ категории $\mathcal{C}'$, и каждому морфизму $f : A \to B$ $(A, B \in Ob)$ в категории $\mathcal{C}$ – морфизм $F(f) : F(A) \to F(B)$ в категории $\mathcal{C}'$, при этом $F(id_A) = id_{F(A)}$ и $F(g) \circ F(f) = F(g \circ f)$ для всех морфизмов $f : A \to B$ и $g : B \to C$ $(A, B, C \in Ob)$ в категории $\mathcal{C}$.

## Приложение Б

**Доказательство утверждения 1.** Поскольку $t$ $(t')$ — трасса в $\mathcal{E}$, то существует последовательность $C_0 \overset{A_1 \cup B_1}{\to} C_1 \ldots C_{n-1} \overset{A_n \cup B_n}{\to} C_n$ $(n \geq 0)$ $(C_0 \overset{X_1 \cup Y_1}{\to} C'_1 \ldots C'_{m-1} \overset{X_m \cup Y_m}{\to} C'_m$ $(m \geq 0))$ в $\mathcal{E}$. Так как равенство $[t] = [t']$ истинно, то $C_n = C'_m$.

*Предложение I.* Если $\mathcal{E}$ — ОПСС со свойством СПСЗ, то верно $F^j = F \cap E^j$ для каждого

$1 \leq j \leq n$.

*Доказательство.* Пусть $1 \leq j \leq n$. По определению 5, истинно $F^j = (F^{j-1} \cap E^j) \setminus (\hat{A}_j \cup \hat{\hat{A}}_j)$.

Сначала проверим, что множество $\hat{A}_j$ не пересекается со множеством $E^j$. Предположим обратное, т.е. существует событие $z$ из $E^j$ такое, что $z \in \hat{A}_j$. В силу определения 5, верно, что $z \in F^{j-1}$ и существует событие $a \in \widetilde{A}_j$ такое, что $a \rhd^{j-1} \underline{z}$. По лемме 2(а), получаем $a \rhd \underline{z}$. Поскольку ОПСС $\mathcal{E}$ обладает свойством СПСЗ, то имеем $z < a$. Далее, так как верно $a \in \widetilde{A}_j$, то возможны два случая:

- $a \in A_j \setminus F^{j-1}$. Благодаря лемме 2(г), $a$ принадлежит множеству $E^{j-1}$. В силу определения 5, получаем $z <^{j-1} a$, так как $z \in F^{j-1} \subseteq E^{j-1}$, т.е. $z \in \lfloor A_j \setminus F^{j-1} \rfloor_{<^{j-1}} \subseteq \widetilde{A}_j$, что противоречит принадлежности события $z$ множеству $E^j$.
- $a \in \lfloor A_j \setminus F^{j-1} \rfloor_{<^{j-1}}$, т.е. событие $a$ принадлежит множеству $E^{j-1}$ и существует событие $a' \in A_j \setminus F^{j-1}$ такое, что $a <^{j-1} a'$. Более того, по определению 5, имеем $z <^{j-1} a$, так как $z \in F^{j-1} \subseteq E^{j-1}$. Согласно лемме 2(б), $\mathcal{E}^{j-1}$ — ОПСС, обладающая свойством СПСЗ. В силу транзитивности $<^{j-1}$, получаем $z <^{j-1} a'$, т.е. $z \in \lfloor A_j \setminus F^{j-1} \rfloor_{<^{j-1}}$, что вновь противоречит принадлежности события $z$ множеству $E^j$.

Таким образом, множества $\hat{A}_j$ и $E^j$ не пересекаются.

Теперь покажем, что множество $\hat{\hat{A}}_j$ не пересекается со множеством $E^j$. Предположим обратное, т.е. существует событие $z \in E^j$ такое, что $z \in \hat{\hat{A}}_j$. По определению 5, это означает, что $z \in F^{j-1}$ и существует событие $a \in \sharp^{j-1}(\widetilde{A}_j)$ такое, что $a \prec^{j-1} \underline{z}$. Тогда, по лемме 2(а), получаем $a \prec \underline{z}$. Поскольку ОПСС $\mathcal{E}$ обладает свойством СПСЗ, то $z = a$, что противоречит принадлежности события $z$ множеству $E^j$.

Таким образом, показали, что $F^j = F^{j-1} \cap E^j$ для каждого $1 \leq j \leq n$. Это позволяет утверждать, что $F^j = F^0 \cap E^1 \cap \ldots \cap E^j$. Однако, благодаря лемме 2(а), знаем, что равенство $E^1 \cap \ldots \cap E^j = E^j$ истинно, т.е. верно $F^j = F \cap E^j$, так как $F^0 = F$. □

*Предложение II.* Для каждого $1 \leq j \leq m$ и $\widetilde{x} \in \widetilde{X}_j$ верно, что $\widetilde{x} \in \widetilde{A}_i$ для некоторого $1 \leq i \leq n$.

*Доказательство.* Из определения 5 известно, что $\widetilde{X}_j = (X_j \setminus F'^{j-1}) \cup \lfloor (X_j \setminus F'^{j-1}) \rfloor_{<'^{j-1}}$. Рассмотрим два случая:

- $\widetilde{x} \in X_j \setminus F'^{j-1}$. Докажем, что $\widetilde{x} \in A_i \setminus F^{i-1}$ для некоторого $1 \leq i \leq n$. Поскольку $\mathcal{E}$ — ОПСС со свойством СПСЗ, то, благодаря предложению I, имеем $F'^{j-1} = F \cap E'^{j-1}$. Тогда $\widetilde{x} \notin F$, так как, по лемме 2(г), верно $\widetilde{x} \in E'^{j-1}$. Более того, поскольку $\widetilde{x} \in \widetilde{X}_j$,

используя лемму 2(д), получаем $\widetilde{x} \in C'_m = C_n$. Согласно определению 3, существует $1 \leq i \leq n$ такое, что $\widetilde{x} \in A_i$. Так как $\widetilde{x} \notin F$, то имеем $\widetilde{x} \in A_i \setminus F^{i-1}$, в силу леммы 2(а).

- $\widetilde{x} \in \lfloor (X_j \setminus F'^{j-1}) \rfloor_{<'^{j-1}}$, т.е. $\widetilde{x} \in E'^{j-1}$ и $\widetilde{x} <'^{j-1} x$ для некоторого $x \in X_j \setminus F'^{j-1}$. Тогда, благодаря лемме 2(а), имеем $\widetilde{x} < x$. Используя предыдущий пункт, получаем, что $x \in A_i \setminus F^{i-1} \subseteq \widetilde{A}_i$ для некоторого $1 \leq i \leq n$. Более того, из леммы 2(г) следует $x \in E^{i-1}$, а из леммы 2(д) — $x \in C_n$. В силу леммы 1, конфигурация $C_n$ лево-замкнута относительно $<$. Тогда верно $\widetilde{x} \in C_n$, поскольку $\widetilde{x} < x$. Далее рассмотрим два случая:
  - $\widetilde{x} \in E^{i-1}$. Благодаря определению 5, получаем $\widetilde{x} <^{i-1} x$, потому что $\widetilde{x} < x$. Так как $x \in A_i \setminus F^{i-1}$, то справедливо $\widetilde{x} \in \lfloor (A_i \setminus F^{i-1}) \rfloor_{<^{i-1}} \subseteq \widetilde{A}_i$.
  - $\widetilde{x} \notin E^{i-1}$. Тогда имеем $\widetilde{x} \in E$ и $\widetilde{x} \notin E^{i-1}$. По определению 5, существует $1 \leq k \leq i-1$ такое, что $\widetilde{x} \in E^{k-1}$ и $\widetilde{x} \notin E^k$, т.е. $\widetilde{x} \in \widetilde{A}_k$ или $\widetilde{x} \in \sharp^{k-1}(\widetilde{A}_k)$. Предположим $\widetilde{x} \in \sharp^{k-1}(\widetilde{A}_k)$, т.е. существует событие $\widetilde{y} \in \widetilde{A}_k$ такое, что $\widetilde{x} \; \sharp^{k-1} \; \widetilde{y}$. В силу леммы 2(а), верно $\widetilde{x} \sharp \widetilde{y}$. Кроме того, из леммы 2(д) следует $\widetilde{y} \in C_n$. Это противоречит бесконфликтности конфигурации $C_n$, так как $\widetilde{x} \in C_n$. Таким образом, верно $\widetilde{x} \in \widetilde{A}_k$ для некоторого $1 \leq k \leq i-1 < n$. □

Проверим, что верно $E^n = E'^m$. Предположим обратное, то есть $E^n \neq E'^m$. Рассмотрим случай, когда существует событие $e \in E^n$ такое, что $e \notin E'^m$ (случай, когда $e \in E'^m$ и $e \notin E^n$ доказывается аналогично). Так как $e \in E$ и $e \notin E'^m$, то существует $1 \leq j \leq m$ такое, что $e \in (\widetilde{X}_j \cup \sharp^{j-1}(\widetilde{X}_j))$, по определению 5. Рассмотрим все возможные случаи:

- $e \in \widetilde{X}_j$. Благодаря предложению II, существует $1 \leq i \leq n$ такое, что $e \in \widetilde{A}_i$. Значит, по определению 5, верно $e \notin E^i$, что противоречит лемме 2(а), поскольку $e \in E^n$.
- $e \in \sharp^{j-1}(\widetilde{X}_j)$, т.е. $e \in E^{j-1}$ и существует событие $x \in \widetilde{X}_j$ такое, что $e \; \sharp^{j-1} \; x$. В силу леммы 2(а), имеем $e \sharp x$. По предложению II, существует $1 \leq i \leq n$ такое, что $x \in \widetilde{A}_i$. Отсюда с помощью леммы 2(г) и определения 5 получаем $x \in E^{i-1}$. Согласно лемме 2(а), верно $e \in E^{i-1}$, поскольку $e \in E^n$. Тогда, по определению 5, получаем $e \; \sharp^{i-1} \; x$, потому что $e \sharp x$. Следовательно, имеем $e \in \sharp^{i-1}(\widetilde{A}_i)$. Тогда верно $e \notin E^i$, что противоречит лемме 2(а), так как $e \in E^n$.

Таким образом, справедливо $E^n = E'^m$.

Покажем совпадение множеств $F^n$ и $F'^m$. Поскольку $\mathcal{E}$ — ОПСС со свойством СПСЗ, то, по предложению I, верно $F^n = F \cap E^n$ и $F'^m = F \cap E'^m$. Отсюда заключаем, что $F^n = F'^m$, поскольку множества $E^n$ и $E'^m$ совпадают.

Далее рассмотрим помечающие функции. В силу определения 5, известно, что $l^n =$

$l \mid_{E^1 \cap \ldots \cap E^n}$ и $l'^m = l \mid_{E'^1 \cap \ldots \cap E'^m}$. По лемме 2(а), истинно, что $E^n \subseteq E^i$ для всех $1 \leq i \leq n$ и $E'^m \subseteq E'^j$ для всех $1 \leq j \leq m$. Тогда имеем, что $l^n = l \mid_{E^n}$ и $l'^m = l \mid_{E'^m}$. Отсюда из равенства множеств $E^n$ и $E'^m$ получаем совпадение функций $l^n = l'^m$.

Аналогичным образом из равенства множеств $E^n = E'^m$ и $F^n = F'^m$ следует совпадение отношений $\nabla^n = \nabla'^m$, где $\nabla \in \{<, \sharp, \prec, \rhd\}$.

И, наконец, проверим совпадение начальных конфигураций. Из леммы 2(е) знаем, что $C_0^n = C_n \cap E^n$ и $C_0'^m = C'_m \cap E'^m$. Поскольку верно $C_n = C'_m$, то имеем $C_0^n = C_0'^m$, в силу совпадения множеств $E^n$ и $E'^m$.

Таким образом, справедливо $\mathcal{E} \setminus t = \mathcal{E} \setminus t'$. □

**Доказательство леммы 3.** Предположим, что $\mathcal{E} = (E, <, \sharp, l, F, \prec, \rhd, C_0)$ и $\mathcal{E}' = (E', <', \sharp', l', F', \prec', \rhd', C'_0)$ — ОПСС со свойством УПСЗ, помеченные на множестве $L$, и $\mu : \mathcal{E} \to \mathcal{E}'$ — морфизм между ними. По определению 9, $\mu$ — это функция из $E$ в $E'$ такая, что

1) $\forall e \in E : \ \lfloor \mu(e) \rfloor_{<'} \subseteq \mu(\lfloor e \rfloor_<)$;
2) $\forall e, e' \in E : \ \mu(e) \sharp' \mu(e') \ \Rightarrow \ e \sharp e'$;
3) $\forall e \neq e' \in E : \ \mu(e) = \mu(e') \ \Rightarrow \ e \sharp e'$;
4) $l' \circ \mu = l$;
5) $\mu(F) \subseteq F'$;
6) $\forall u \in F : \ \llcorner \underline{\mu(u)} \lrcorner_{\prec'} \subseteq \mu(\llcorner \underline{u} \lrcorner_\prec)$;
7) $\forall e \in E, \ \forall u \in F : \ \mu(e) \rhd' \underline{\mu(u)} \ \Rightarrow \ e \rhd \underline{u}$;
8) $\mu(C_0) = C'_0$.

*Предложение III.* Пусть $C_0 \xrightarrow{A_1 \cup \underline{B_1}} \ldots \xrightarrow{A_n \cup \underline{B_n}} C_n$ в $\mathcal{E}$. Тогда верно $\mu(C_0) \xrightarrow{\mu(A_1) \cup \underline{\mu(B_1)}} \ldots \xrightarrow{\mu(A_n) \cup \underline{\mu(B_n)}} \mu(C_n)$ в $\mathcal{E}'$.

*Доказательство.* Проведем доказательство индукцией по $n$.

$n = 0$ Тогда имеем $\mu(C_0) = C'_0$, по пункту 8) определения 9.

$n > 0$ Предположим, что $\mu(C_0) \xrightarrow{(\mu(A_1) \cup \underline{\mu(B_1)})} \ldots \xrightarrow{(\mu(A_{n-1}) \cup \underline{\mu(B_{n-1})})} \mu(C_{n-1})$ в $\mathcal{E}'$. Покажем, что $\mu(C_{n-1}) \xrightarrow{(\mu(A_n) \cup \underline{\mu(B_n)})} \mu(C_n)$ в $\mathcal{E}'$. Тогда, в силу определения 3, получаем $\mu(C_{n-1}) \in Conf(\mathcal{E}')$, а также $\mu(C_{n-1})$ — конечное и бесконфликтное множество. Кроме того, $\mu(C_{n-1})$ лево-замкнуто относительно $<$, по лемме 1. Далее, согласно определению 9, имеем, что $\mu(A_n) \subseteq E'$ и $\mu(B_n) \subseteq \mu(F) \subseteq F'$, так как $B_n \subseteq F$. Осталось показать, что шаг $(\mu(A_n) \cup \underline{\mu(B_n)})$ возможен из конфигурации $\mu(C_{n-1})$ в $\mathcal{E}'$. Проверим справедливость пунктов а)-г) определения 3.

а) Так как шаг $(A_n \cup \underline{B_n})$ возможен из конфигурации $C_{n-1}$ в $\mathcal{E}$, то $A_n \cap C_{n-1} = \emptyset$, $B_n \subseteq C_{n-1}$ и $(C_{n-1} \cup A_n)$ — конечное и бесконфликтное множество. Предположим $\mu(A_n) \cap \mu(C_{n-1}) \neq \emptyset$, т.е. существуют $a \in A_n$ и $x \in C_{n-1}$ такие, что $\mu(a) = \mu(x)$. Так как имеем $(A_n \cap C_{n-1}) = \emptyset$, то верно $a \neq x$. По пункту 3) определения 9, получаем $a \sharp x$, что противоречит бесконфликтности множества $(C_{n-1} \cup A_n)$. Таким образом, имеем $\mu(A_n) \cap \mu(C_{n-1}) = \emptyset$. Вложение $\mu(B_n) \subseteq \mu(C_{n-1})$ очевидным образом следует из вложения $B_n \subseteq C_{n-1}$. Конечность множества $(\mu(C_{n-1}) \cup \mu(A_n))$ обеспечивается конечностью множества $(C_{n-1} \cup A_n)$. Осталось проверить бесконфликтность этого множества. Предположим обратное, т.е. существуют $z,\ z' \in (\mu(C_{n-1}) \cup \mu(A_n))$ такие, что $z \sharp' z'$. Тогда существуют $a, b \in (C_{n-1} \cup A_n)$ такие, что $\mu(a) = z$ и $\mu(b) = z'$. По пункту 2) определения 9, получаем $a \sharp b$, что противоречит бесконфликтности множества $(C_{n-1} \cup A_n)$. Таким образом, $(\mu(C_{n-1}) \cup \mu(A_n))$ — бесконфликтное множество.

б) Пусть $e \in \mu(A_n)$, $e' \in E'$ и $e' <' e$. Тогда существует $a \in A_n$ такое, что $e = \mu(a)$. Кроме того, имеем $e' \in \lfloor e \rfloor_{<'}$, по определению 2. Благодаря пункту 1) определения 9, верно $e' \in \mu(\lfloor a \rfloor_<)$, т.е. справедливо $e' = \mu(e_1)$ для некоторого события $e_1 \in E$ такого, что $e_1 < a$. Поскольку шаг $(A_n \cup \underline{B_n})$ возможен из конфигурации $C_{n-1}$ в $\mathcal{E}$, то событие $e_1$ принадлежит множеству $C_{n-1} \setminus B_n$, а значит, событие $e'$ принадлежит множеству $\mu(C_{n-1} \setminus B_n) \subseteq \mu(C_{n-1})$. Осталось проверить, что событие $e'$ не принадлежит множеству $\mu(B_n)$. Предположим обратное, т.е. существует событие $b \in B_n$ такое, что $e' = \mu(b)$. Однако, поскольку верно $e' = \mu(e_1)$, то, благодаря пункту 3) определения 9, получаем: либо $e_1 = b$, что противоречит принадлежности события $e_1$ множеству $(C_{n-1} \setminus B_n)$, либо $e_1 \sharp b$, что противоречит бесконфликтности множества $C_{n-1}$, с учетом $B_n \subseteq C_{n-1}$. Таким образом, событие $e'$ принадлежит множеству $(\mu(C_{n-1}) \setminus \mu(B_n))$.

в) Пусть $e \in \mu(B_n)$, $e' \in E'$ и $e' \prec' \underline{e}$. Тогда существует $a \in B_n$ такое, что $e = \mu(a)$. Кроме того, имеем $e' \in \llcorner e \lrcorner_{\prec'}$, по определению 2. Используя пункт 6) определения 9, получаем $e' \in \mu(\llcorner a \lrcorner_\prec)$, т.е. $e' = \mu(e_1)$ для некоторого $e_1$ такого, что $e_1 \prec \underline{a}$. Поскольку шаг $(A_n \cup \underline{B_n})$ возможен из конфигурации $C_{n-1}$ в $\mathcal{E}$, то $e_1 \in (C_{n-1} \setminus (B_n \setminus \{a\})) \subseteq C_{n-1}$. Так как верно $e' = \mu(e_1)$, то имеем $e' \in \mu(C_{n-1})$. Осталось убедиться в том, что $e' \notin (\mu(B_n) \setminus \mu(\{a\}))$. Предположим обратное, т.е. $e' = \mu(b)$ для некоторого события $b \in B_n$, неравного событию $a$. Из равен-

ства $e' = \mu(e_1)$ следует $\mu(e_1) = \mu(b)$. Благодаря пункту 3) определения 9, получаем: либо $e_1 = b$, что противоречит принадлежности события $e_1$ множеству $C_{n-1} \setminus (B_n \setminus \{a\})$, либо $e_1 \sharp b$, что противоречит бесконфликтности множества $C_{n-1}$, с учетом $B_n \subseteq C_{n-1}$. Следовательно, событие $e' = \mu(e_1)$ принадлежит множеству $\mu(C_{n-1}) \setminus (\mu(B_n) \setminus \mu(\{a\}))$.

г) Пусть $e \in \mu(B_n)$, $e' \in E'$ и $e' \rhd' \underline{e}$. Тогда существует $a \in B_n$ такое, что $e = \mu(a)$. Проверим два возможных варианта:

– $e' \notin \mu(E)$. Очевидно, что $e' \notin (\mu(C_{n-1}) \cup \mu(A_n))$, так как $C_{n-1},\ A_n \subseteq E$.

– $e' \in \mu(E)$. Это означает, что $e' = \mu(e_1)$ для некоторого $e_1 \in E$. Благодаря пункту 7) определения 9, верно $e_1 \rhd \underline{a}$, поскольку $a \in B_n \subseteq F$. Так как шаг $(A_n \cup \underline{B_n})$ возможен из конфигурации $C_{n-1}$ в $\mathcal{E}$, то имеем $e_1 \notin (C_{n-1} \cup A_n)$. Следовательно, получаем $e' = \mu(e_1) \notin \mu(C_{n-1} \cup A_n) = (\mu(C_{n-1}) \cup \mu(A_n))$.

Отсюда заключаем, что шаг $(\mu(A_n) \cup \underline{\mu(B_n)})$ возможен из $\mu(C_{n-1})$ в $\mathcal{E}'$ и приводит в конфигурацию $Y = (\mu(C_{n-1}) \setminus \mu(B_n)) \cup \mu(A_n)$. Проверим справедливость $Y = \mu(C_n)$. Очевидно, имеем $\mu(C_n) = \mu((C_{n-1} \setminus B_n) \cup A_n) = \mu(C_{n-1} \setminus B_n) \cup \mu(A_n)$. Покажем, справедливость $\mu(C_{n-1} \setminus B_n) = \mu(C_{n-1}) \setminus \mu(B_n)$.

Пусть $d \in \mu(C_{n-1} \setminus B_n)$, т.е существует событие $c \in C_{n-1} \setminus B_n$ такое, что $\mu(c) = d$. Так как имеем $c \in C_{n-1}$, то верно $d = \mu(c) \in \mu(C_{n-1})$. Предположим $d \in \mu(B_n)$, т.е. существует событие $b \in B_n$ такое, что $\mu(b) = d$. Поскольку событие $c$ принадлежит множеству $(C_{n-1} \setminus B_n)$, то события $c$ и $b$ не могут быть равными. Тогда по пункту 3) определения 9, верно $c \sharp b$, что противоречит бесконфликтности множества $C_{n-1}$, с учетом $B_n \subseteq C_{n-1}$. Значит, имеем $d \in (\mu(C_{n-1}) \setminus \mu(B_n))$.

Теперь пусть $d \in (\mu(C_{n-1}) \setminus \mu(B_n))$, т.е. $d \in \mu(C_{n-1})$ и $d \notin \mu(B_n)$. Так как $d$ принадлежит множеству $\mu(C_{n-1})$, то существует событие $c \in C_{n-1}$ такое, что $\mu(c) = d$. Предположим, что верно $c \in B_n$. Тогда имеем $\mu(c) \in \mu(B_n)$, что противоречит $d = \mu(c)$. Следовательно, верно $c \in (C_{n-1} \setminus B_n)$. Это влечёт принадлежность события $d$ множеству $\mu(C_{n-1} \setminus B_n)$.

Отсюда заключаем, что $Y = \mu(C_n)$. □

Возьмём две произвольные конфигурации $C, C' \in Conf(\mathcal{E})$ такие, что $C \xrightarrow{A \cup \underline{B}} C'$ в $\mathcal{E}$. Так как верно $C \in Conf(\mathcal{E})$, то, по определению 3, существуют множества $A_i \subseteq E$ и $B_i \subseteq F$ $(1 \leq i \leq n-1)$ такие, что $C_{i-1} \xrightarrow{A_i \cup \underline{B_i}} C_i$ и $C_{n-1} = C$. Тогда получаем, что $C_0 \xrightarrow{A_1 \cup \underline{B_1}} \ldots \xrightarrow{A_{n-1} \cup \underline{B_{n-1}}} C \xrightarrow{A \cup \underline{B}} C'$ в $\mathcal{E}$. По предложению III, имеем, что $\mu(C_0) \xrightarrow{\mu(A_1) \cup \underline{\mu(B_1)}} \ldots \xrightarrow{\mu(A_{n-1}) \cup \underline{\mu(B_{n-1})}}$

$\mu(C) \stackrel{\mu(A)\cup\mu(B)}{\longrightarrow} \mu(C')$ в $\mathcal{E}'$. Из пункта 8 определения 9 знаем, что $\mu(C_0) = C_0'$. Тогда, по определению 3, верно $\mu(C), \mu(C') \in Conf(\mathcal{E}')$. □

**Доказательство леммы 4**. Для доказательства достаточно построить копроизведение для двух произвольных ОПСС $\mathcal{E}_i = (E_i, <_i, \sharp_i, l_i, F_i, \prec_i, \rhd_i, \emptyset)$ $(i = 1, 2)$. Построим структуру $\mathcal{E} = \mathcal{E}_1 \oplus \mathcal{E}_2$ как дизъюнктивное объединение следующим образом: $\mathcal{E} = \mathcal{E}_1 \oplus \mathcal{E}_2 = (E, <, \sharp, l, F, \prec, \rhd, \emptyset)$, где

- $E = \{(i, e) \mid i \in \{1, 2\}, e \in E_i\}$ и $F = \{(i, e) \mid i \in \{1, 2\}, e \in F_i\}$;
- для всех $(i, e), (j, e') \in E$ пусть $(i, e) < (j, e')$ тогда и только тогда, когда $i = j$ и $e <_i e'$;
- для всех $(i, e), (j, e') \in E$ пусть $(i, e) \sharp (j, e')$ тогда и только тогда, когда $i \neq j$ или $(i = j$ и $e \sharp_i e')$;
- для всех $(i, e) \in E$ функция пометки сохраняется, т.е. $l((i, e)) = l_i(e)$;
- для всех $(i, e) \in E$ и $(j, e') \in F$ отношение $\prec$ определено следующим образом: $(i, e) \prec \underline{(j, e')}$ тогда и только тогда, когда $i = j$ и $e \prec_i \underline{e'}$;
- для всех $(i, e) \in E$ и $(j, e') \in F$ пусть $(i, e) \rhd \underline{(j, e')}$ тогда и только тогда, когда $i \neq j$ или $(i = j$ и $e \rhd_i \underline{e'})$.

Учитывая, что $\mathcal{E}_i$ $(i = 1, 2)$ — это ОПСС со свойством УПСЗ, с помощью определения 2 нетрудно видеть, что построенная структура $\mathcal{E}_1 \oplus \mathcal{E}_2$ является объектом категории $\mathbf{RPES}_L^0$.

Определим два проектирующих отображения $\pi_i : \mathcal{E}_i \to \mathcal{E}_1 \oplus \mathcal{E}_2$ $(i = 1, 2)$ по следующему правилу: $\pi_i(e) = (i, e)$ для всех $e \in E_i$ $(i = 1, 2)$. Проверим, что $\pi_i$ $(i = 1, 2)$ действительно являются морфизмами категории $\mathbf{RPES}_L^0$.

1) Пусть $e \in E_i$. Тогда $\lfloor \pi_i(e) \rfloor_< = \lfloor (i, e) \rfloor_< = \pi_i(\lfloor e \rfloor_{<_i})$, по определению отношения $<$.
2) Пусть $e, e' \in E_i$ и при этом $\pi_i(e) \sharp \pi_i(e')$. Это означает, что верно $(i, e) \sharp (i, e')$. Далее, по определению отношения $\sharp$, получаем $e \sharp_i e'$.
3) Теперь предположим, что существуют $e \neq e' \in E_i$ такие, что $\pi_i(e) = \pi_i(e')$. По определению отображения $\pi_i$, имеем $(i, e) = (i, e')$, что противоречит $e \neq e'$. Значит, наше предположение неверно.
4) По определению функции $l$, верно, что $l \circ \pi_i(e) = l((i, e)) = l_i(e)$ для любого $e \in E_i$.
5) Пусть $e \in F_i$. Тогда имеем $\pi_i(e) = (i, e) \in F$, по определению множества $F$.
6) Пусть $u \in F_i$. Рассмотрим множество $\llcorner \underline{\pi_i(u)} \lrcorner_\prec$. Очевидно, что $\pi_i(u) = (i, u)$ и, кроме того, $\llcorner \underline{(i, u)} \lrcorner_\prec = \pi_i(\llcorner \underline{(u)} \lrcorner_{\prec_i})$, по определению отношения $\prec$.
7) Предположим, что $e \in E_i$ и $u \in F_i$ такие, что $\pi_i(e) \rhd \underline{\pi_i(u)}$, т.е. $(i, e) \rhd \underline{(i, u)}$. Отсюда,

благодаря определению отношения $\rhd$, получаем $e \ \rhd_i \ \underline{u}$.

8) Так как начальные конфигурации — пустые множества, то верно $\pi_i(\emptyset) = \emptyset$.

Для завершения доказательства необходимо для любого объекта $\mathcal{E}'$ и пары морфизмов $\lambda_i : \mathcal{E}_i \to \mathcal{E}'$ $(i = 1, 2)$ категории $\mathbf{RPES}_L^0$ показать существование и единственность морфизма $\lambda : \mathcal{E} = \mathcal{E}_1 \oplus \mathcal{E}_2 \to \mathcal{E}'$ такого, что $\lambda \circ \pi_i = \lambda_i$ $(i = 1, 2)$. Для любого события $(i, e) \in E$ определим отображение $\lambda((i, e)) = \lambda_i(e) \in E'$, которое очевидным образом удовлетворяет последнему равенству для $i = 1, 2$. Единственность отображения следует из построения проекций $\pi_i$. Проверим, что $\lambda$ является морфизом категории $\mathbf{RPES}_L^0$.

1) Пусть $(i, e) \in E$. Тогда верно $\lfloor \lambda((i, e)) \rfloor_{<'} = \lfloor \lambda_i(e) \rfloor_{<'} \subseteq \lambda_i(\lfloor e \rfloor_{<_i})$, поскольку $\lambda_i$ является морфизом. Так как $\lambda_i = \lambda \circ \pi_i$, то $\lambda_i(\lfloor e \rfloor_{<_i}) = \lambda \circ \pi_i(\lfloor e \rfloor_{<_i})$. По построению структуры $\mathcal{E}$, выполняется $\pi_i(\lfloor e \rfloor_{<_i}) = \lfloor (i, e) \rfloor_<$, т.е. $\lfloor \lambda((i, e)) \rfloor_{<'} \subseteq \lambda(\lfloor (i, e) \rfloor_<)$.
2) Пусть $(i, e), (j, e') \in E$ и $\lambda((i, e)) \ \sharp' \ \lambda((j, e'))$, т.е. $\lambda_i(e) \ \sharp' \ \lambda_j(e')$. Если $i \neq j$, то $(i, e) \ \sharp \ (j, e')$, по определению отношения $\sharp$ в $\mathcal{E}$. Когда $i = j$, то, поскольку $\lambda_i$ является морфизмом, заключаем, что верно $e \ \sharp_i \ e'$. А это, вновь в силу определения отношения $\sharp$ в $\mathcal{E}$, позволяет сделать вывод, что справедливо $(i, e) \ \sharp \ (j, e')$.
3) Выберем произвольным образом два события $(i, e) \neq (j, e') \in E$ такие, что $\lambda((i, e)) = \lambda((j, e'))$, т.е. $\lambda_i(e) = \lambda_j(e')$. Если $i \neq j$, то, по определению отношения $\sharp$ в $\mathcal{E}$, получаем $(i, e) \ \sharp \ (j, e')$. Когда $i = j$, поскольку $\lambda_i$ является морфизмом, то верно $e \ \sharp_i \ e'$. Тогда, вновь, в силу определения отношения $\sharp$ в $\mathcal{E}$, получаем $(i, e) \ \sharp \ (j, e')$.
4) Так как $\lambda_i$ — морфизм, то имеем $l' \circ \lambda_i = l_i$. Отсюда, по определению функций $\lambda$ и $l$, верно, что $l' \circ \lambda((i, e)) = l'(\lambda_i(e)) = l_i(e) = l((i, e))$ для любого $(i, e) \in E$.
5) Нетрудно заметить, что выполняется следующее: $\lambda(F) = \lambda(\{(i, e) \mid e \in F_i, \ i = 1, 2\}) = \{\lambda_i(e) \mid e \in F_i, \ i = 1, 2\} = \lambda_1(F_1) \cup \lambda_2(F_2) \subseteq F'$, так как $\lambda_i$ является морфизом.
6) Пусть $(i, e) \in F$. Тогда верно $\llcorner \underline{\lambda((i, e))} \lrcorner_{\prec'} = \llcorner \underline{\lambda_i(e)} \lrcorner_{\prec'} \subseteq \lambda_i(\llcorner e \lrcorner_{\prec_i})$, поскольку $\lambda_i$ является морфизом. Так как $\lambda_i = \lambda \circ \pi_i$, то справедливо $\lambda_i(\llcorner e \lrcorner_{\prec_i}) = \lambda \circ \pi_i(\llcorner e \lrcorner_{\prec_i})$. Однако, по построению структуры $\mathcal{E}$, верно $\pi_i(\llcorner e \lrcorner_{\prec_i}) = \llcorner (i, e) \lrcorner_<$.
7) Выберем произвольным образом два события $(i, e) \in E$ и $(j, e') \in F$ такие, что $\lambda(i, e) \rhd' \underline{\lambda(j, e')}$, т.е. $\lambda_i(e) \rhd' \ \underline{\lambda_j(e')}$. Если $i = j$, т.е. $\lambda_i(e) \rhd' \ \underline{\lambda_i(e')}$, то, так как $\lambda_i$ — морфизм, известно, что $e \ \rhd_i \ \underline{e'}$. Это означает, что $(i, e) \rhd \underline{(i, e')}$. Пусть $i \neq j$. Тогда, согласно определению отношения $\rhd$ в $\mathcal{E}$, верно $(i, e) \rhd \underline{(j, e')}$.
8) И, наконец, имеем $\lambda(\emptyset) = \lambda(\emptyset) = \emptyset$. □

**Доказательство утверждения 2.**

Пусть $\mathcal{E}_1 = (E_1, <_1, \sharp_1, l_1, F_1, \prec_1, \rhd_1, C_0^1)$ и $\mathcal{E}_2 = (E_1, <_2, \sharp_2, l_2, F_2, \prec_2, \rhd_2, C_0^2)$ — ОПСС со свойством УПСЗ и $\mu : \mathcal{E}_1 \to \mathcal{E}_2$ — морфизм категории $\mathbf{RPES}_L$.

Расширим отображение $TC$ до функтора, а именно определим, как это отображение преобразует морфизмы категории $\mathbf{RPES}_L$ в морфизмы категории $\mathbf{TS}_L$. Определим отображение $TC(\mu)$ следующим образом: пусть $TC(\mu)(C) = \mu(C) \in Conf(\mathcal{E}_2)$ для всех $C \in Conf(\mathcal{E}_1)$. Проверим, что $TC(\mu)$ является морфизмом из $TC(\mathcal{E}_1)$ в $TC(\mathcal{E}_2)$ в категории $\mathbf{TS}_{\mathbb{L}}$.

Очевидно, что $TC(\mu)(C_0^1) = \mu(C_0^1) = C_0^2$, по пункту 8) определения 9.

Пусть конфигурации $C, C' \in Conf(\mathcal{E}_1)$ выбраны так, что $C \stackrel{M}{\rightharpoonup} C'$. По определению отношения $\rightharpoonup$, имеем $C \stackrel{A \cup \underline{B}}{\longrightarrow} C'$ в $\mathcal{E}_1$ для некоторых множеств $A \subseteq E_1$ и $B \subseteq F_1$ таких, что $M = l_1(A \cup \underline{B})$. Другими словами, шаг $A \cup \underline{B}$ возможен из конфигурации $C$ и его выполнение приводит к конфигурации $C' = (C \setminus B) \cup A$. Используя лемму 3, получаем, что $\mu(C), \mu(C') \in Conf(\mathcal{E}_2)$ и $\mu(C) \stackrel{\mu(A) \cup \mu(B)}{\longrightarrow} \mu(C')$ в $\mathcal{E}_2$. Поскольку отображение $\mu$ является морфизмом категории $\mathbf{RPES}_L$, то $l_2 \circ \mu = l_1$ по пункту 4) определения 9. Следовательно, верно $M = l_1(A \cup \underline{B}) = l_2 \circ \mu(A \cup \underline{B}) = l_2(\mu(A) \cup \underline{\mu(B)})$. Однако, в соответствии с определением отношения $\rightharpoonup$, это означает, что $TC(\mu)(C) = \mu(C) \stackrel{M}{\rightharpoonup} \mu(C') = TC(\mu)(C')$. Таким образом, $TC(\mu)$ — морфизм в категории $\mathbf{TS}_L$.

Заметим, что при таком определении $TC(\mu)$ естественным образом сохраняются тождественные морфизмы и композиции морфизмов, а значит, $TC$ — функтор.

Рассмотрим ОПСС $\mathcal{E}^* = (E^*, <^*, \sharp^*, l^*, F^*, \prec^*, \rhd^*, C_0^*)$ со свойством УПСЗ, где $E^* = \{a, b\}$, $<^* = \emptyset$, $\sharp^* = \emptyset$, $l^*$ — идентичная функция, $F^* = \emptyset$, $\prec^* = \emptyset$, $\rhd^* = \emptyset$ и $C_0^* = \emptyset$. Очевидно, что для ОПСС $\mathcal{E}^*$ и $\mathcal{E}_2$ из примера 2 не существует ни одного морфизма категории $\mathbf{TS}_{\mathbb{L}}$, который бы отображал систему переходов $TR(\mathcal{E}^*)$ в систему переходов $TR(\mathcal{E}_2)$. Однако отображение $\eta : \mathcal{E}^* \to \mathcal{E}_2$ такое, что $\eta(a) = a$ и $\eta(b) = b$, является морфизмом категории $\mathbf{RPES}_L$. Значит, отображение $TR$ нельзя расширить до функтора между категориями $\mathbf{RPES}_L$ и $\mathbf{TS}_{\mathbb{L}}$. □

УДК 004.8

# Современные тенденции в развитии нейронных сетей

*Насибулов Илья Андреевич (Институт систем информатики СО РАН, Новосибирский государственный университет),*

*Насибулов Егор Андреевич (Институт систем информатики СО РАН, Новосибирский государственный университет)*

В последние 30 лет нейронные сети являются одним из наиболее бурно развивающихся направлений искусственного интеллекта. Они широко применяются в обработке звука и изображения, медицине, задачах анализа и генерации контента и других. Это стало возможным благодаря значительному росту вычислительных мощностей, возможности обработки больших объёмов данных и развитию теории нейросетей.

В данной работе приведён анализ развития алгоритмов обучения и архитектур нейросетей от их зарождения до современного состояния. Были выделены наиболее активно развивающиеся направления, такие как большие языковые модели, сети-гиганты и мультимодальные модели. Также упомянуто перспективное направление развитие, связанное с сетями Колмогорова-Арнольда.

***Ключевые слова:*** *искусственный интеллект, нейронная сеть, машинное обучение, глубокое обучение, свёрточные нейронные сети, языковые модели, сети-трансформеры, сети Колмогорова-Арнольда.*

## 1. Введение

Подход к искусственному интеллекту (*ИИ*) в XXI веке значительно изменился. Термин ввёл Джон Маккарти [1], определяя ИИ как машинные вычисления, способные решать задачи, предназначенные для человека и даже имитировать поведение человека. Термин подразумевал под собой следующее: искусственный — эта часть от машины, а интеллект — человеческая часть, способная решать задачи, в том числе и когнитивные, и выдавать себя за человека. Подход к реализации такого ИИ описывался принципом, что ИИ должен самовоспроизводиться с помощью несложных инструкций кода [2]. Несмотря на то, что в определении ИИ отсутствуют технические детали реализации, подразумевая, что способов

реализаций может быть несколько, в последнее время ИИ всё чаще ассоциируется с нейросетями [3], особенно глубокого обучения.

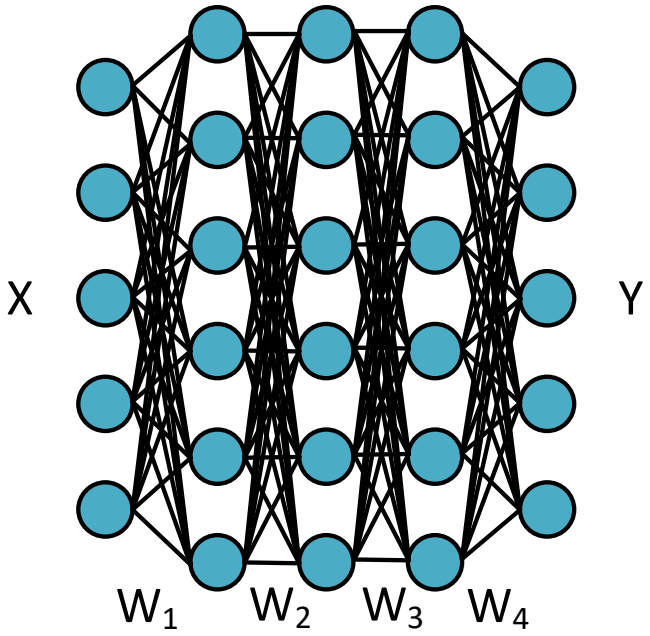



Рисунок 1. Представление нейросети в виде вычислительного графа. $X$ — входные данные, $Y$ — выходные данные, $W$ — матрицы весов.

Первые нейросети задумывались как вычислительные модели, основанные на имитации искусственными нейронами биологических [4], способные обучаться на примерах и применять полученную информацию для решения аналогичных задач. Развивая эту идею, можно воспринимать нейросети как вычислительные графы [5], на рёбрах которых происходят несложные арифметические действия (рис. 1). Таким образом, решение задач с помощью нейросетей технически относится к моделированию. Перспективы создания ИИ с помощью нейросетей послужили выделению отдельного направления в науке, получившего название общий искусственный интеллект [6]. Несмотря на то, что масштабирование нейросетей позволило получить ряд значимых результатов, chatGPT и аналогичные архитектуры всё ещё не могут полноценно заменить живых специалистов, что показал опыт ведущих наукоёмких компаний. Тем не менее, спектр задач, которые решаются с применением нейросетей, довольно обширен.

Одной из типичных задач является анализ изображений. С ним связаны задачи от распознавания рукописного текста до распознавания лиц. Одной из задач обеспечения безопасности является распознавание номеров машин с помощью видеонаблюдения с последующим выявлением нарушений правил дорожного движения [7]. Анализ изображений

находит себя и в медицине [8, 9] — по симптомам и соответствующим снимкам пациентов нейросети помогают соответствующим специалистам поставить диагноз. Есть применение и в аграрной области — если анализировать снимки полей с урожаем, например, поля пшеницы, то можно выявить потенциально больные колосья для последующей обработки посевов. Или же можно получить достаточно много информации по сорту будущего посева и потенциальному процессу роста по снимку семян или листов растений [10].

Нейросети активно применяются в задачах компьютерной лингвистики, психологии и изучения мышления [11, 12]. Есть подходы, позволяющие использовать нейросети для переводов различных текстов и анализа аргументации [13]. В числе успешно решаемых задач — переводы узкоспециализированных технических текстов с определёнными лексическими оборотами [14, 15]. Также ведутся работы в направлении переводов с малораспространённых языков различных народов России и не только [16].

Своё применение нейросети находят и в биоинформатике. Например, анализ последовательностей белков можно частично упростить [17, 18], что в дальнейшем облегчает расшифровку последовательностей аминокислот в организме.

Ещё одним применением нейросетей является генерация контента, такого как текст, изображения, звук и т.д. Свою нишу нейросети постепенно занимают и в генерации программного кода [19]. По данному вопросу существуют две точки зрения исследователей. Одни считают, что через декаду лет программисты как таковые будут нужны не в качестве специалистов по написанию кода, а скорее операторов нейросетей и ИИ для формирования конечного программного продукта. Контраргументом является то, что код генерируется на основе уже существующих материалов, заложенных в нейросети на этапе обучения. Большая часть сгенерированного кода не является кодом высокого уровня в смысле оптимального по используемым ресурсам ЭВМ и их производных. Большинство программного кода написано программистами низкой квалификации, так называемыми младшими программистами, и в условиях ограниченности по времени разработки, что несёт в себе изначальные изъяны, которым обучается нейросеть и, в дальнейшем, при генерации нового кода считает эти изъяны допустимой нормой. Первые считают, что для исправления таких проблем и нужно будет «оперирование» нейросетей, т. е. дальнейшее улучшение с помощью них же сгенерированного изначального кода. Обе позиции имеют место быть, однако сложно спорить с тем, что нейросети уже как минимум сдают ЕГЭ по разным предметам на проходной балл для поступления в ВУЗы. Это активно обсуждалось в прессе и образовательных учреждениях [20].

Так называемые из-за своего размера сети-гиганты объединяют в той или иной степени все вышеперечисленные возможности. Актуальные на 2025 г. архитектуры сетей-гигантов содержат миллиарды параметров. ChatGPT и его аналоги, в числе которых Гигачат от Сбербанка, YandexGPT от Яндекса и китайская Deepseek уже используются широкими слоями населения. Возможности этих нейросетей обширны, и есть публикации в прессе о случаях, когда пользователи предпочитают общаться с нейросетями вместо своих коллег и друзей. С тех пор как тест Тьюринга был пройден нейросетями, уровень доверия к нейросетям растёт, в том числе в таких важных сферах, как здравоохранение [21]. Тем не менее, не стоит забывать, что алгоритмы нейросетей не являются абсолютно надёжными и в большинстве моделей недоступны для верификации. Таким образом, ошибки и галлюцинации нейросетей становятся непредсказуемыми и опасными. В последние годы интерес к исследованию данной проблемы держится на устойчиво высоком уровне как с технической [22, 23, 24], так и с этической стороны [25, 26, 27].

Целью данной работы является дать краткий обзор развития нейронных сетей от первых персептронов до сетей Колмогорова-Арнольда (*KAN*). К сожалению, все подобные обзоры [28, 29, 30] быстро устаревают в связи с бурным развитием области и появлением новых направлений, таких как сети Колмогорова-Арнольда [31]. Несмотря на то, что KAN появились сравнительно недавно и большого количества результатов с их использованием ещё не было получено на момент написания данной статьи, архитектура уже зарекомендовала себя как перспективный инструмент для решения нелинейных задач.

## 2. Зарождение нейросетей

Первой нейросетью считается искусственный нейрон, предложенный У. Маккалаком и У. Питтсом [32]. Они же и ввели понятие искусственной нейронной сети (*ИНС*). Нейрон имеет строение аналогичное сумматору, однако имеются только логические сигналы 0 и 1. В качестве функции активации была выбрана пороговая функция Хевисайда. Это служит неким аналогом активации человеческих нейронов в нервной системе. Таким образом, мы получаем

$$s = w \cdot x + b, z = g(s),$$

Где $x$ и $w$ — вектора входных данных и весов соответственно, $x \in Z_{\{0,1\}}^{N}$, $b$ — смещение, $z$ — выход, $g$ — активационная функция, в данном случае

$$g = \begin{cases} 1, \text{если } s > a, \\ 0, \text{если } s \leq a, \end{cases}$$

где $a > 0$ — порог активации. Если учесть факт, что при формировании нейросетей нейроны объединяются в нейросетевые слои, то вышеописанные уравнения можно записать в матричном виде

$$s = Wx + b, z = g(s),$$

где W — матрица весов.

Первой ИНС, которая способна была решить задачу классификации и широко применялась на практике, была нейросеть Ф. Розенблатта [33], так называемый персептрон. Нейросеть интересна наличием одного скрытого слоя нейронов в ней. Скрытыми слоями называются слои между входными данными и выходным слоем. Веса и смещения задаются случайным образом из {-1, 0, 1}, а в качестве функции активации используется sign. Таким образом, получаем

$$s_1 = W_1 x + b_1, y = g_0(s_1),$$

$$s_2 = W_2 y + b_2, z = g_1(s_2),$$

где $z = g_1(s) = sign(s), x \in Z_{\{0,1\}}^N$. $y$ является выходом первого слоя и одновременно входными данными для второго слоя.

М. Минский и С. Паперт в 1969 году в своей работе [5] рассматривают персептроны Розенблатта в качестве вычислительных графов, что открывает возможности для применения нейросетей в математическом моделировании. Тем не менее, никакого выхода на новые практические задачи продемонстрировано в работе не было, и на какое-то время интерес исследователей к этой области ИИ угас.

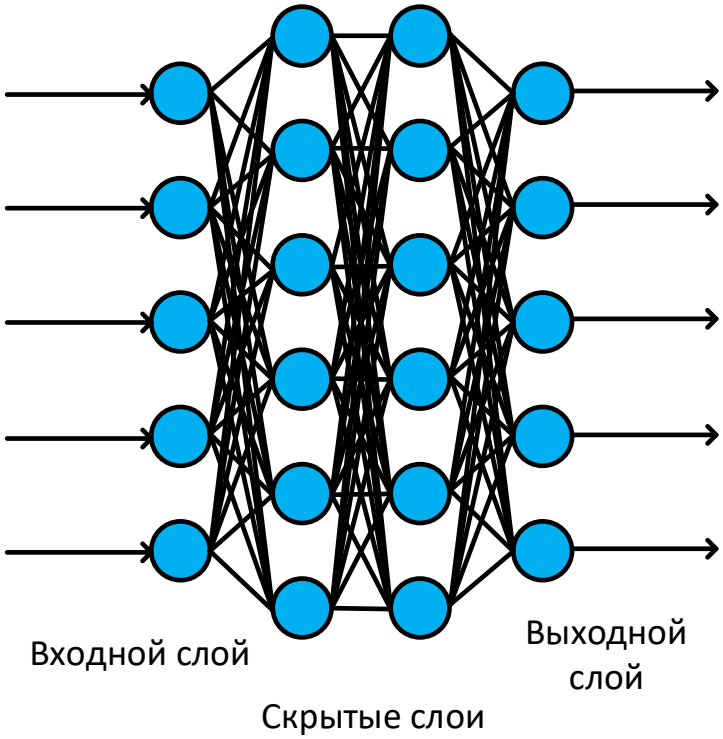



Рисунок 2. Многослойный персептрон. Все нейроны предыдущего слоя соединены со всеми нейронами следующего слоя, и только с ними.

## 3. Нейронные сети со скрытыми слоями

В 1986 вышла работа Д. Румельхарта [34], в которой был продемонстрирован потенциал многослойных персептронов Розенблатта (рис. 2) с некоторыми улучшениями:

$$s_1 = W_1 x + b_1, y = \tilde{g}_0(s_1),$$

$$s_2 = W_2 y + b_2, z = \tilde{g}_1(s_2),$$

где $\tilde{g}_i(s) = \tilde{g}_{sg}(s) = \frac{1}{1+e^{-x}}$ является сигмоидальной функцией или $\tilde{g}_i(s) = \tilde{g}_{th}(s) = th(s)$ гиперболический тангенс, $i = 0, 1$. Входной слой уже не ограничен исключительно логическими сигналами, а принимает вещественные значения $x \in \mathbb{R}^N$. В обучении нейросетей в данной работе активно применяется метод обратного распространения ошибки, который предложили и развили А. Галушкин и П. Вербос в [35, 36, 37, 38].

Для развития нейронных сетей критически важными являются теоремы о суперпозиции Колмогорова-Арнольда [39] и универсальная теорема аппроксимации [40]. Согласно первой, $\forall f \in \mathbb{C}[0,1]^d$ существует $d(2d+1)$ функций одного аргумента $\phi_{ij} \in \mathbb{C}[0,1]$ таких, что $f$ может быть представлена в виде

$$f(x_1,\dots,x_d)=\sum_{i=1}^{2d+1}\chi_i\left(\sum_{j=1}^{d}\phi_{ij}(x_j)\right)$$

для некоторых $\chi_i \in \mathbb{C}[0,1]$, зависящих от $f$.

Универсальная теорема аппроксимации представляет собой адаптацию теоремы суперпозиции Колмогорова-Арнольда к области нейронных сетей и говорит, что искусственная нейронная сеть прямого распространения с одним скрытым слоем может аппроксимировать любую непрерывную функцию многих переменных с любой точностью, при условии, что сеть имеет в скрытом слое достаточное число нейронов $N$, имеющих сигмоидальную функцию активации $s_{sg}$.

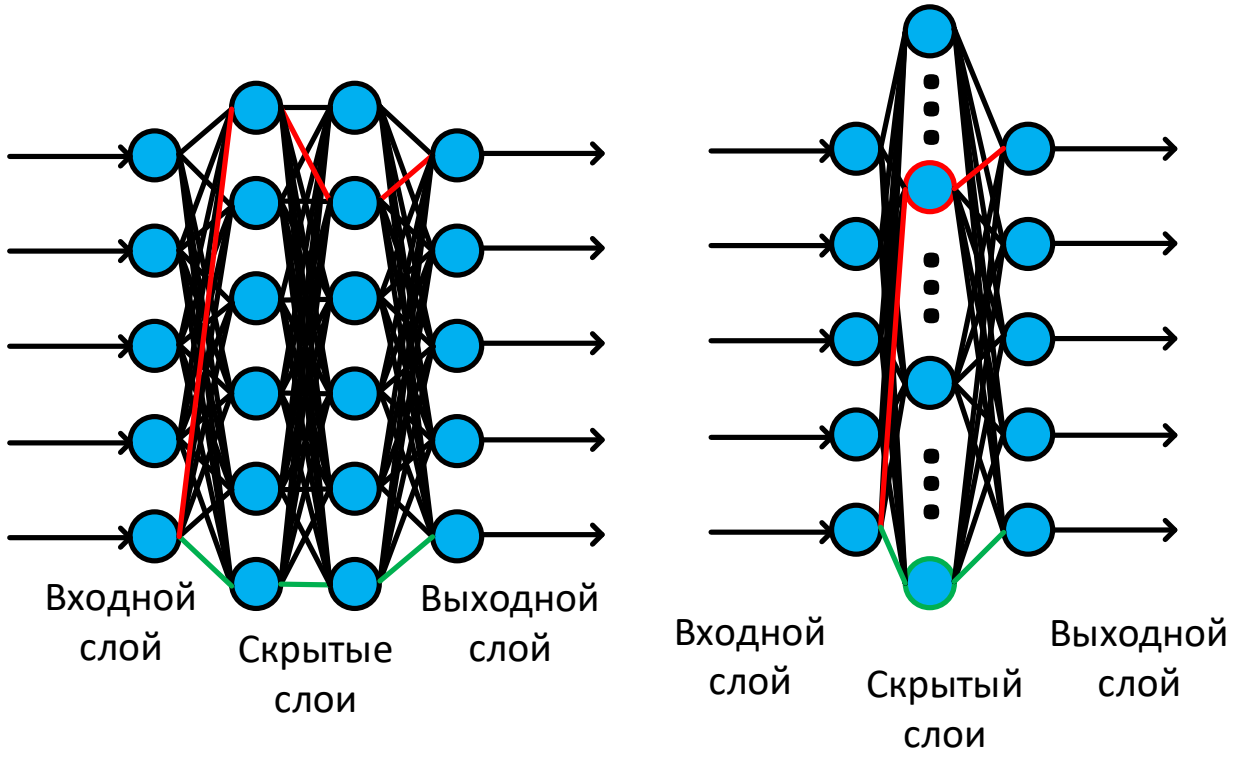



Рисунок 3. Многослойная нейронная сеть (слева) и эквивалентная ей сеть с одним скрытым слоем (справа). Цветом выделены эквивалентные пути, дающие одинаковый вклад в выходной слой, в разных архитектурах.

Действительно, если каждый уникальный путь через нейроны от входного к выходному слою (рис. 3) представить в виде нейрона этого единственного слоя, получим полную эквивалентность с точки зрения результата работы нейросетей как функционалов. Сам факт

существования такого гомеоморфизма оказался крайне полезным для доказательства ряда теорем.

Важной для развития рекуррентных нейронных сетей является теорема о полной тьюринговости, доказанная Х. Зигельманом и Э. Сонтагом [41]: Любые машины Тьюринга могут моделироваться полностью связанными рекуррентными сетями, созданными из нейронов с сигмоидальными функциями активации, при условии, что сеть имеет достаточное число нейронов в скрытом слое $M$ и достаточное число шагов временной памяти $K$.

В работе К. Хорника [42] 1991 года значительно расширено понимание возможностей нейронных сетей. Была обобщена универсальная теорема аппроксимации для случая произвольных нелинейных функций активации. Это позволило сделать следующие фундаментальные выводы: нейросети способны к аппроксимации, а свойства последней определяются архитектурой; выбор конкретной функции активации менее критичен, чем считалось ранее; для построения эффективных нейронных сетей может быть использован широкий класс функций активации.

Ещё одним доказанным в 1992 году результатом является универсальная аппроксимационная теорема рекуррентных нейронных сетей, доказанная К. Фунахаши и Ю. Накамурой [43]: Любая нелинейная динамическая система может быть аппроксимирована рекуррентной нейронной сетью с любой точностью, без ограничений на компактность пространства состояний системы, при условии, что сеть имеет достаточное число нейронов в скрытом слое.

Т. Чоу и Х. Ли в 2000 году расширили универсальную аппроксимационную теорему рекуррентных нейронных сетей на случай неавтономных нелинейных обыкновенных дифференциальных уравнений [44].

Несмотря на то, что появились теоремы, раскрывающие область применимости нейронных сетей для решения различных задач машинного обучения, на практике возможности ИНС с одним скрытым слоем достаточно ограничены и не подходят для решения большого класса задач. Однослойная нейросеть, которую можно построить для любой многослойной сети, крайне плохо обучается и годится только в качестве абстрактной математической модели. Попытки же добавления скрытых слоёв нейронов не приносят значительного продвижения из-за проблемы затухания градиента [45]. В процессе обучения поправки из выходного слоя просто не доходят до входа сети.

## 4. Глубокие (многослойные) нейросети

Вторая половина 2000-х годов приносит плоды, благодаря которым интерес к нейросетям снова разжигается в научном сообществе. Работы Д. Хинтона и Р. Салахутдинова [46] предлагают некоторые способы обучения нейросетей со многими скрытыми слоями, однако на практике эти способы получились затратными в плане вычислений и неустойчивыми для сетей с более чем 3–5 слоями. Для решения проблемы обучения нейросетей со многими скрытыми слоями оказались принципиальными два фактора. Один из них — подобрать правильную функцию активации, что сделал в 2010 году В. Наир, предложив использовать функцию ReLU (*Rectified Linear Unit*) [47]. Функция ReLU представляет собой $g_{ReLU}(s) = \max(0, s)$. Вторым фактором стал способ начальной инициализации весов нейросетей. К. Глорот и Й. Бенжио в 2010 году предложили [48] дисперсию инициализирующего шума находить по формуле

$$Var(w) = \frac{2}{N_{in} + N_{out}},$$

где $N_{in}$ и $N_{out}$ — число искусственных нейронов в предыдущем и следующем нейрослое соответственно. Эти факторы дали старт для очередного быстрого развития нейросетей, а как сопутствующий результат, сформировали понятие глубокой нейронной сети — нейросети, содержащей 2 и более скрытых нейрослоёв (рис. 3).

## 5. Свёрточные нейросети

Отдельного упоминания стоит история развития свёрточных нейросетей, так как они показывали и показывают себя как один из самых эффективных инструментов для анализа изображений. История создания свёрточных нейросетей уходит в 50-е — 60-е годы, когда работы таких исследователей, как Д. Хебба [4] и других нейрофизиологов показали, что зрительная кора головного мозга для распознавания объектов имеет отдельный вид нейронов, которые реагируют на определённые шаблоны (*pattern*) в получаемых сигналах — линии, углы, движение.

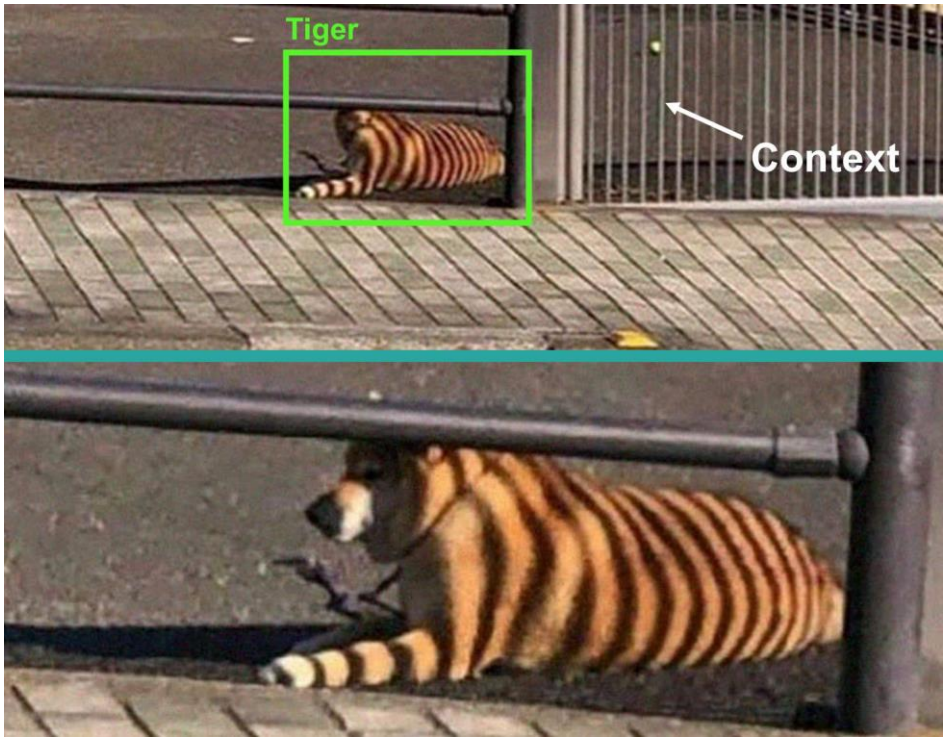



Рисунок 4. Завирусившееся в интернете изображение, на котором IntelexVision вместо собаки видит тигра, не учитывая контекст в виде забора, дающего полосатую тень.

В связи с этим принцип работы нейросетей основан на поиске тех или иных признаков объекта, который подаётся в качестве входных данных. Этот процесс называется поиском шаблонов в данных. Однако входные данные могут содержать персональный контекст, который может искажать исходные признаки объекта. Ярким примером такого искажения является следующее изображение (рис. 4), на котором человек определит собаку, однако нейросеть, ранее обученная решать задачу выявления животных на изображении и классифицировать выявленное животное, относит данную собаку к тигру.

Подобные случаи неверного истолкования признаков входных объектов доказывают важность персонального контекста и в целом обработки данных при работе с нейросетями, а не только выявления самой эффективной архитектуры и получения лучших метрик в выходных данных.

Первые свёрточные нейросети работали на следующих принципах: к изображению применяется операция свёртки; для обнаружения шаблонов используются соответствующие фильтры; изображение обрабатывается слой за слоем с целью извлечения более сложных шаблонов; для обучения сетей используется метод обратного распространения ошибки.

Первой практически работающей архитектурой свёрточной нейросети стала LeNet, которую в 1989 году разработал Я. Лекун с соавторами [49] для распознавания написанных от руки цифр почтового индекса, предоставляемых почтовой службой США. Работа нейросети основана на принципе разделения весов, когда несколько нейронов или групп нейронов используют одни и те же веса для снижения количества уникальных параметров в модели и оптимизации процесса обучения. К 1998 году идеи коллектива выливаются в архитектуру LeNet-5 [50], состоящей из чередования свёрточных слоёв и слоёв субдискретизации, завершая выход нейросети двумя полносвязными свёрточными слоями.

В 2010 году Д. К. Кирешан и Ю. Шмидхубер публикуют препринт [51], в котором реализованная архитектура нейросети добивается рекордных на то время показателей точности определения рукописных символов эталонных тестов MNIST. Для достижения цели была спроектирована нейросеть, содержащая 9 скрытых слоёв, а для её обучения использовался графический процессор, что позволило снизить время обучения сети.

М. Цейлер с соавторами публикуют работу [52], в которой предложили использование слоя деконволюции. Если рассмотреть его на примере входного изображения $y_i$ с $K_0$ входными каналами $y_1, y_2, .., y_{K_0}$, то каждый из этих каналов представляется в виде линейной суммы $K_1$ скрытых карт признаков $z_k^i$, свёрнутых фильтрами $f_{k,c}$:

$$\sum_{k=1}^{K_1} z_k^i \oplus f_{k,c} = y_c^i.$$

Если изображение $y_c^i$ имеет размер $N_r \times N_c$, а фильтры имеют размер $H \times H$, то карты скрытых объектов имеют размер $(N_r + H - 1) \times (N_c + H - 1)$.

В 2012 году **AlexNet** — **с**вёрточная нейронная сеть, разработанная командой исследователей под руководством А. Крижевского [53] — одерживает революционную победу в конкурсе ImageNet LSVRC-2012, где показывает впечатляющий результат с ошибками топ-1 и топ-5 в 37,5% и 17,0% против 45,7% и 25,7% у конкурентов, усреднявших показания двух классификаторов, обученных на Фишеровских векторах [54]. Ключевыми особенностями AlexNet стали применение функции активации ReLU, эффективное использование графического процессора для обучения и применение техники Dropout для борьбы с переобучением, предложенная ранее тем же коллективом [55]. Техника основана на выключении в течение одной итерации обучения части

нейронов скрытых слоёв с определённым шансом с последующим их возвращением в конце итерации с последующей нормировкой весов нейронов.

М. Лин с соавторами в 2013 году [56] предложили ввести слой глобальной усредняющей субдискретизации, что в дальнейшем позволило создавать полносвязные свёрточные нейросети без полносвязных слоёв в конце сети. Также было показано, что вставка многослойных персептронов между свёрточными слоями усиливает свёрточные свойства.

В 2014 году К. Симонян и Э. Зиссерман публикуют работу [57], в которой описывают так называемую VGG-сеть. Архитектура предлагает использовать вместо тяжёлых свёрточных слоёв размером 5x5 и более свёрточные слои размером 3x3 с увеличением их количества — сама нейросеть включала 19 нейрослоёв. Как оказалось, такой подход оказывается крайне эффективным за счёт уменьшения числа параметров и уменьшения тяжеловесных арифметических операций.

К. Сегеди с коллегами в 2014 году публикуют работу [58], где предлагают к рассмотрению архитектурный принцип под названием Inception и конкретную нейросеть с 22-мя нейрослоями GoogLeNet, основанную на данной архитектуре и успешно применяемую в задачах классификации и обнаружения. В архитектуре Inception ключевым принципом является использование ядер размера 1x1, что является переосмыслением идей Лин [56].

В том же 2014 году Л. Сифре защищает кандидатскую диссертацию [59], посвящённую получению шаблонов изображений, стабильных относительно трансляции и поворота посредством каскада вейвлет-преобразований по пространственным и угловым координатам. Для этого он вводит Depthwise Separable свёрточный слой нейронов, в котором также являются важными ядра размера 1x1.

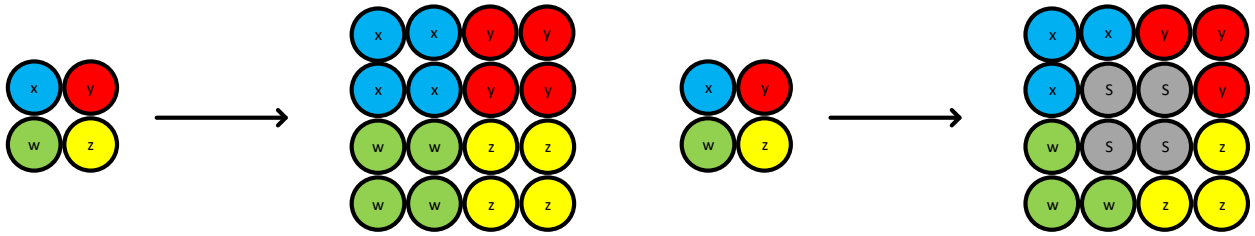



Рисунок 5. Пример операции пространственного расширения карты признаков. Слева — простейшая схема с дублированием существующих, справа — с применением интерполяции и созданием синтетических признаков. В данном примере $S = \frac{x+y+z+w}{4}$

Развивая идеи [56], в 2014 году Дж. Лонг с соавторами в своей работе [60] предложили концепцию полносвёрточной сети для задач, результатом в которых является изображение такого же размера, как исходное. Предлагается использовать операцию пространственного расширения карты признаков (*Upsampling*) для решения проблемы несбалансированных наборов данных (рис. 5). В простейшем случае операция представляет собой метод копирования существующих предметов, а может применяться и интерполяция для создания новых синтетических примеров в классе на основе существующих. В дальнейшем подход с применением интерполяции доработали и нейрослой стал называться слоем транспонированной свёртки [61].

Рисунок 6. Появление дефектов в случае использования только деконволюции (сверху), для сравнения — использование деконволюции в сочетании с расширением карты признаков (снизу). Источник: [62].

В работе [62] А. Одена изучает пиксельные дефекты, возникающие при анализе изображений с использованием нейросетей с пространственным расширением карты признаков. Коллектив приходит к выводу, что стандартный подход к созданию изображений с помощью деконволюции имеет значительные успехи, но также имеет некоторые концептуально простые проблемы, которые приводят к появлению дефектов в создаваемых изображениях (рис. 6). Тщательное же продумывание архитектуры нейросетей с использованием операций расширения карты признаков и слоя транспонированной свёртки может оказаться лучшим решением, чем уже существовавшие решения в виде свёрточных нейросетей.

В 2015 году С. Иоффе с командой разрабатывают пакетную нормализацию данных (*Batch Normalization*) при передаче между слоями [63]. Метод основан на том, что нормализация становится частью архитектуры нейросети и выполняется для каждого обучающего мини-пакета, что позволяет использовать более быстрые темпы обучения сетей и снизить требования к инициализации исходных данных, а в некоторых случаях не применять технику борьбы с переобучением Dropout.

О. Роннебергер с коллегами разрабатывают в 2015 году подход к построению архитектуры нейросети и её обучению в задаче сегментации изображений [64], позволяющий эффективно использовать имеющиеся выборки данных. Архитектура основана на двух частях — сжимающей, активно использующей свёрточные слои для извлечения шаблонов из входных данных, и расширяющей, использующей операцию пространственного расширения карты признаков. Связи между этими двумя частями архитектуры позволяют добиться эффективного обучения на относительно меньших наборах входных данных и лучшей сегментации изображений, что и позволило выиграть ISBI cell tracking challenge 2015 с большим отрывом относительно конкурентов (свёрточных нейросетей с подвижным окном).

Ф. Ю и В. Колтун в своей работе [65] предлагают использовать архитектурный модуль нейросети, содержащий операцию разреженной свёртки. Коллектив показывает, что данный модуль можно применять для систематического объединения многомасштабной контекстной информации без потери разрешения, что выливается в повышение точности современных систем семантической сегментации.

В конце 2015 года выходит работа К. Хэ с коллегами [66], в которой предлагается к рассмотрению архитектура Residual Network (*ResNet*) — свёрточная нейросеть с остаточными блоками. Переформулировав слои как обучающие остаточные функции со ссылкой на входные слои, коллектив делает упор на глубину сети, сравнивая до 152 слоёв с 16–19 в архитектуре VGG. Несмотря на количество слоёв, ResNet является сетью с меньшей сложностью. Совокупность остаточных блоков даёт погрешность в 3,57% на эталонном наборе тестов ImageNet. Также данная архитектура побеждает на ILSVRC 2015 в задачах классификации.

Х. Хан и Б. Енер в 2018 году на конференции представляют работу [67], в которой используют слой вейвлет-деконволюции для спектрального разложения временных рядов вместо предобработки сигналов в свёрточных нейросетях, что позволяет уменьшить количество параметров обучения и повысить интерпретируемость классификатора временных рядов. Данный метод позволил уменьшить ошибку в распознавании телефонных сигналов на 4% до 18,1%.

С. Фудзиеда с коллегами представили в своей работе [68] использование вейвлет-преобразований непосредственно в нейросети в качестве вейвлет-нейрослоёв субдискретизации. Такой подход позволяет использовать спектральную информацию, обычно теряющуюся в обычных свёрточных нейросетях, для эффективного решения задач классификации текстур и аннотации 2D-изображений.

В 2019 году П. Лю с коллегами предлагает использование [69] мультивейвлет-свёрточной нейросети, архитектура которой включает встройку вейвлет-преобразований в нейросеть для уменьшения карт признаков и увеличить поле восприятия соответствующих фильтров. Также данную архитектуру можно применять для восстановления карт объектов с высоким разрешением с использованием обратных вейвлет-преобразований в архитектуре.

## 6. Глубокие рекуррентные нейросети

В конце 1980-х — 1990-х годах М. Джордан и Дж. Элман внесли фундаментальный вклад в развитие рекуррентных нейронных сетей (*RNN*). В 1990 году Элман в своей работе [70] предлагает модель рекуррентной ИНС с обратной связью и наличием одношагового временного контекста как обобщение высказанных идей различными исследователями в 86-90-х годах, первым из которых был Джордан, говоривший про временной контекст в техническом отчёте [71]:

$$h_k = g_h(W_h x_k + U_h h_{k-1} + b_h),$$

$$y_k = g_y(W_y h_k + b_y),$$

где $k$ — дискретное время, $h_k$ — вектор скрытого состояния нейросети в момент времени $k$, а $U_h h_{k-1}$ отвечает за обратную связь и временной контекст.

В 1997 году Джордан предложил модификацию сети Элмана [70] в работе [72] по изучению коартикуляционных явлений в речи, которая заключается в том, что контекст решения определяется выходом сети, а не скрытым слоем:

$$h_k = g_h(W_h x_k + U_h y_{k-1} + b_h),$$

$$y_k = g_y(W_y h_k + b_y),$$

Такие сети как в [72] и [70] называются SimpleRNN и имеют некоторые проблемы. Однако рекуррентные сети, состоящие из стандартных рекуррентных ячеек, не способны обрабатывать долгосрочные зависимости: по мере увеличения разрыва между соответствующими входными данными становится трудно получить информацию о соединении. А сигналы об ошибках, поступающие в обратном направлении во времени, имеют тенденцию либо усиливаться, либо исчезать [73].

Для решения проблем SimpleRNN в 1997 году З. Хохрайтер предлагает архитектуру LSTM — Long Short Term Memory [74], или же долгая краткосрочная память. Они улучшили запоминающую способность стандартной рекуррентной ячейки, введя в нее шлюзы (*gates*). После этой новаторской работы LSTM были модифицированы и популяризированы многими исследователями. Варианты включают LSTM без шлюза забывания, LSTM с шлюзом забывания и LSTM с подключением через глазок. Обычно термин "ячейка LSTM" обозначает LSTM со шлюзом забывания [73].

Ф. Морин и Й. Бенгиа в 2005 году в своей работе про языковое моделирование распознавателей речи [75] предлагают использование иерархической декомпозиции условных вероятностей отношений языковых конструкций к определённым классам семантической иерархии WordNet, что по сути является модификацией нейрослоя Softmax. Это позволяет эффективно решать задачи классификации в компьютерной лингвистике с десятками тысяч классов.

А. Гравес и Ю. Шмидбухер в своей работе 2005 года [76] совершенствуют архитектуру LSTM и представляют архитектуру сети Bidirectional LSTM. Тестируя различные архитектуры нейросетей, исследователи делают вывод о том, что двунаправленные сети превосходят однонаправленные, а архитектура LSTM в целом превосходит обычные RNN сети.

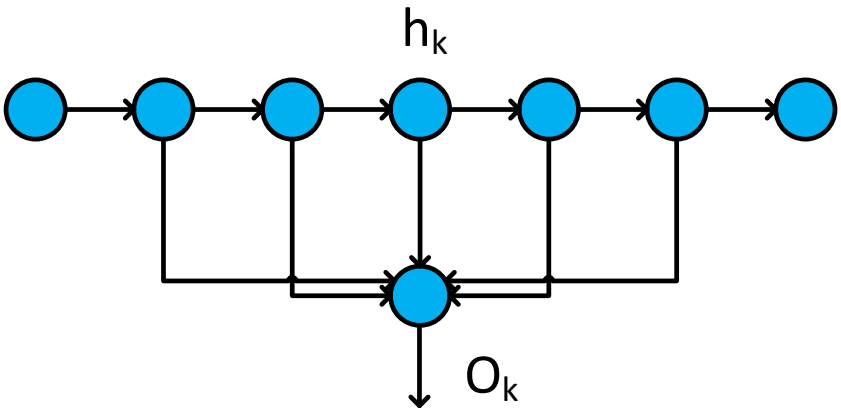



Рисунок 7. Слой внимания (*Attention layer*). Выходное значение формируется как взвешенное среднее выхода.

В 2013 году Гравес [77] предложил реализацию слоя внимания (*Attention layer*). Выход слоя внимания является взвешенным средним выхода рекуррентного слоя (рис. 7).

С. Ши с соавторами в 2015 году представляют в работе [78] свёрточную LSTM нейросеть — инновационную архитектуру, объединяющую преимущества свёрточных и LSTM сетей для краткосрочного прогнозирования осадков. Данная работа демонстрирует

успешное применение глубокого обучения в метеорологии и показывает эффективные методы обработки пространственно-временных данных.

В 2015 году Н. Кальхбреннер в работе [79] представляет архитектуру Grid LSTM, расширяющую возможности LSTM для работы с входными данными, имеющими сетчатую структуру. Отличия от традиционной долгой кратковременной памяти заключается в связи ячеек между нейросетевыми уровнями, а также в пространственно-временных данных. Превосходство архитектуры показано на наборе эталонных тестов предсказания символов Википедии, в задаче перевода текста с китайского на английский. На бенчмарке MNIST для определения рукописных символов архитектура показала конкурентоспособный процент ошибки.

К. Лаурент с коллегами в своей работе 2015 года показывает как пакетная нормализация позволяет значительно сократить время обучения сети [80]. Коллектив отмечает, что хотя эта техника может привести к более быстрой сходимости критериев обучения, как такового преимущества при решении задач языкового моделирования и распознавания речи не даёт.

Д. Амодей с соавторами [81] добиваются ускорения работы нейросети для распознавания английской и китайской речи в своей работе. Ключом к повышению эффективности является использование архитектур нейросетей, которые позволяют применение схемы формирования пакетов для графических процессоров. Применение данной схемы, называющейся диспетчеризацией пакетов (*Batch Dispatch*), позволило добиться эффективной работы программного пакета в реальном времени на серверах разработки.

В 2016 году Дж. Л. Ба публикует работу [82], в которой метод пакетной нормализации модифицируется в метод нормализации слоя. Отличием служит применение адаптивных смещений нейронов до применения элементов нелинейности. Также схожие операции происходят и во время обучения и тестирования нейросети. Такой подход эффективен для стабилизации динамики скрытых состояний в рекуррентных нейросетях и может значительно сократить время, требуемое для обучения сети.

А. Васвани с коллегами представляют в своей работе [83] архитектуру Transformer, которая с помощью механизма многоголового внимания (*Multi-head attention*) может эффективно моделировать зависимости между данными без использования рекуррентных или свёрточных нейрослоёв, заменяя их массивами ячеек внимания. Данная архитектура показала значительные улучшения показателей в задачах обработки естественного языка и эффективность обучения в условиях ограниченности обучающих данных. Однако у

данной сети есть недостаток в виде ограниченности операционного контекста — всего несколько десятков токенов.

М. Дехгани и С. Гувс в 2018 году представляют архитектуру Universal Transformer, которая является обобщением архитектуры сети-трансформера с использованием рекуррентных последовательностей, что позволяет преодолеть ограничения сетей-трансформеров во многих простых задачах [84].

Работа Дж. Девлина с соавторами 2018 года [85] представляет модель BERT (*Bidirectional Encoder Representations from Transformers*), которая предлагает новый подход к предварительному обучению языковых моделей, учитывающий левый и правый контекст во всех слоях. Архитектура показала свою эффективность в задачах обработки естественного языка [86], а операционный контекст в несколько сотен токенов позволяет преодолеть недостатки LSTM.

В 2019 году Ц. Даи с коллегами представляет в [87] архитектуру нейросети Transformer-XL, которая позволяет изучать зависимости за пределами фиксированной длины без нарушения временной согласованности. Улучшение составляет на 80% в случае RNN, и 450% для обычных нейросетей архитектуры Transformer.

Современное состояние архитектур рекуррентных нейронных сетей позволяет решать широкий класс задач [88, 89, 90].

# 7. Современные тенденции глубокого обучения

## 7.1 Обработка естественного языка

Одним из современным направлений развития является улучшение архитектуры больших языковых моделей. Развитие рекуррентных нейронных сетей, ячеек долгой краткосрочной памяти и архитектуры трансформеров позволило развиваться таким семействам нейросетей как вышеописанная BERT [85], модель от Google Brain Team, названная T5 [91, 92], или ChatGPT [93]. Принципиальным вкладом T5 в развитие нейросетей является унифицирование задачи обработки естественного языка в единую схему преобразования текста. В отличие от существовавших до неё моделей, в том числе BERT, T5 обрабатывает входной текст (*энкодер*) и на его основе генерирует выходной текст (*декодер*).

Развитие ChatGPT демонстрирует впечатляющий прогресс в области искусственного интеллекта и обработки естественного языка. Первой архитектурой для ChatGTP была GPT-1, появившаяся в 2018 году и представлявшая из себя архитектуру трансформера, умеющего

генерировать простые тексты, отвечать на несложные вопросы, завершать предложения, генерировать небольшие описания на основе описания характеристик. Но GPT-1 имела недостатки в виде ограниченной длины генерации текстов, слабое понимание контекста, сложности с решением сложных задач, несвязное ведение диалога с пользователем. По-настоящему революционной стала архитектура GPT-3, которая колоссально увеличила количество параметров модели — 175 миллиардов, значительно улучшила качество генерации и уменьшила выборки данных для обучения. В 2022 году появился сам ChatGPT, расширивший возможности пользовательского интерфейса, а на текущий момент активно используется архитектура GPT-4o в комбинации с GPT-5, вышедшей в релиз 7 августа 2025 года.

В 2022 году Google AI представляет архитектуру Pathways Language Model (*PaLM*) [94]. Нейросеть представляет собой языковую модель-трансформер с 540 миллиардами параметров, а в обучении использовалась Pathways, новая система ML, которая обеспечивает высокоэффективное обучение в нескольких модулях тензорных процессоров Google. PaLM обладает широкими возможностями в решении многоязычных задач и генерации исходного кода. В работе [94] продемонстрировано превосходство модели над GPT-3.

Нейросети-трансформеры нашли своё применение и в задаче определения белков, где позволили улучшить существующие модели. Дж. Джампер с коллегами из DeepMind представили AlphaFold — первую нейросеть, способную определять последовательности белков с атомной точностью [95]. Успехи были подтверждены на конкурсе 14th Critical Assessment of protein Structure Prediction (*CASP14*). AlphaFold открыла новую эру в предсказании белковых структур, значительно ускорив процесс, который ранее занимал месяцы и годы расчётных и экспериментальных исследований.

П. Левис с коллегами в 2020 году представляют универсальный рецепт тонкой настройки моделей генерации с расширенным поиском (*RAG*), которые объединяют предварительно обученную параметрическую и непараметрическую память для генерации языка [96]. Анализируя данный метод настройки в широком спектре задач обработки естественного языка, команда делает вывод, что модели RAG генерируют более конкретный, разнообразный и основанный на фактах язык, чем базовая версия модели seq2seq, основанная только на параметрах. Данный принцип впоследствии стал использоваться многими коммерческими решениями, по крайней мере специальными версиями моделей из семейства архитектур, таких как GPT, DALL-E, Claude, Gemini, Copilot и другие.

## 7.2 Компьютерное зрение

А. Досовитский с соавторами в работе [97] рассуждают о том, что архитектура сетей-трансформеров стала стандартом для задач обработки естественного языка, а в компьютерном зрении применение архитектуры остается ограниченным. Механизм внимания применяется либо совместно со свёрточными сетями, либо используется для замены определённых компонентов свёрточных сетей при сохранении их общей структуры. Команда демонстрирует, что зависимость от свёрточных нейросетей необязательна, и чистый преобразователь, применяемый непосредственно к последовательностям фрагментов изображений, хорошо справляется с задачами классификации изображений.

Р. Ромбах с соавторами представляют в 2021–2022 годах скрытую диффузионную модель Stable Diffusion для генерации изображений по текстовому описанию [98]. Команда улучшает диффузионные модели, основанные на декомпозиции процесса формирования изображения на последовательное применение автоэнкодеров (*энкодер* + *декодер*) с шумоподавлением, с помощью использования скрытого пространства мощных предварительно обученных автоэнкодеров и внедрения в архитектуру модели уровней перекрёстного внимания. Архитектура стала первой высокопроизводительной моделью с открытым кодом в области генеративного ИИ.

Конкурирующей архитектурой является DALL-E 2 — генеративная модель от OpenAI, представленная в 2022 году, способная создавать реалистичные изображения и произведения искусства на основе текстовых описаний. К сожалению, OpenAI не публикует полные научные статьи о DALL-E 2 в открытом доступе, поэтому преимущества в виде более высокого качества финальных изображений, лучшего понимания контекста запроса, меньшее число дефектов при генерации и более стабильные результаты в сравнении со Stable Diffusion можно считать субъективными, однако из рассмотрения исключить саму нейросеть не позволяют. Ещё одним коммерческим конкурентом является Midjourney от одноимённой компании. Нейросети приписывают преимущество при генерации изображений в художественном стиле, однако закрытый характер разработки не позволяет полноценно провести анализ модели.

В области компьютерного зрения NVIDIA в 2022 году предлагают свою разработку StyleGAN v4 — генеративно-состязательную сеть (*GAN*), предназначенную для создания фотореалистичных изображений лиц и других объектов. Архитектура включает основанный на грамматике обучения основанной лексике (*G2L2*) подход к изучению композиционного и обоснованного представления значений языка на основе обоснованных данных, таких как парные изображения и тексты. В ходе работы сети слова сопоставляются с кортежем

синтаксического типа и нейросимволической семантической составляющей. Модель продолжает развиваться, открывая новые возможности для создания фотореалистичного контента.

В области компьютерного зрения можно отметить CLIP (*Contrastive Language-Image Pre-training*) [99]. В 2021 году А. Радфорт с коллегами демонстрируют, что предварительное обучение определения соответствия подписей и изображений является эффективным и масштабируемым способом изучения представлений изображений с нуля на основе набора данных из 400 миллионов пар (изображение, текстовое описание), собранных из Интернета. Модель нетривиально подходит для большинства задач и часто конкурирует с полностью контролируемыми базовыми показателями без необходимости какого-либо обучения для конкретного набора данных.

В 2023 году Meta AI представляет разработку Segment Anything Model [100], предназначенную для автоматической сегментации объектов на изображениях. Модель была обучена для обработки как текстовых запросов, так и изображений, при этом её важной особенностью является способность выполнять инструкции в условиях отсутствия примеров (*zero-shot*) для новых изображений и задач. Производительность при этих условиях впечатляет — часто результат не уступает или даже превосходит полученный при наличии полностью размеченных аналогичных примеров в обучающей выборке.

## 7.3 Вопросы архитектуры и масштабирования нейросетей

В 2020 году Д. Лепихин с соавторами обозначают проблему, связанную с масштабированием нейросетей [101]. Несмотря на надёжное повышение качества моделей с масштабированием, растёт сложность вычислений, программирования и эффективного распараллеливания. Для преодоления этих проблем команда использует архитектурное решение Mixture of Experts. С помощью модуля Gshard, состоящего из набора облегченных API-интерфейсов аннотаций и расширения для компилятора XLA, команда добивается расширения многоязычной модели нейронного машинного перевода архитектуры Transformer до 600 миллиардов параметров, используя автоматическое сегментирование. Модель обучалась 4 дня на 2048 тензорных процессорах и показала высокое качество перевода.

В 2019 году М. Тан и К. В. Ле публикуют работу про семейство архитектур EfficientNet [102], переосмысляющую масштабирование нейросетей. Тщательный баланс глубины, ширины и разрешения сети при масштабировании может привести к повышению производительности. Основываясь на этом наблюдении, команда предлагает новый метод,

который равномерно масштабирует все параметры глубины, ширины и разрешения с использованием простого, но высокоэффективного комплексного коэффициента. Также Тан и Ле используют поиск по нейронной архитектуре для разработки новой базовой сети и её масштабирования с целью получения нового семейства моделей, называемых EfficientNet.

В 2019 году Э. Ховард с командой представляют семейство архитектур MobileNetV3 [103]. Команда представляет новое поколение нейронных мобильных сетей, адаптированное к процессорам мобильных телефонов с помощью аппаратного обеспечения для поиска сетевой архитектуры (*NAS*), дополненного алгоритмом NetAdapt, а затем усовершенствованного за счет новых достижений в архитектуре. Данное решение представляет собой эффективное решение для задач компьютерного зрения на мобильных устройствах, предлагая баланс между точностью и производительностью.

Дж. Расли с соавторами публикуют в 2020 году статью про библиотеку DeepSpeed от Microsoft [104] для предназначенную для масштабирования и оптимизации обучения глубоких нейронных сетей на графических и тензорных процессорах. Одна из частей библиотеки — параллельный оптимизатор ZeRO, который значительно снижает ресурсы, необходимые для распараллеливания модели. При этом ZeRO увеличивает количество параметров, которые можно обучить. DeepSpeed предоставляет возможности для обучения моделей с триллионами параметров.

## 7.4 Обучение с подкреплением

В 2018 году DeepMind представили нейросеть AlphaZero, представляющую собой архитектуру, способную достигать сверхчеловеческого уровня игры в различные стратегические игры на основе обучения с подкреплением — метода машинного обучения, при котором нейросеть учится принимать оптимальные решения через взаимодействие с окружающей средой, получая обратную связь в виде наград или наказаний [105]. В отличие от AlphaGo [106], обучавшейся в том числе на размеченных данных игр профессионалов, AlphaZero, имея в своём арсенале только правила игры, за 24 часа достигла сверхчеловеческого уровня игры в шахматы и сёги, а также в го, и в каждом случае убедительно побеждала программы-чемпионки мира.

В 2019 году О. Винялс с соавторами публикуют работу [107] про нейросеть AlphaStar от DeepMind, достигшей уровня игры Грандмастеров в StarCraft II. Для обучения использовался многоагентный алгоритм обучения с подкреплением, который использует данные как из игр с участием людей, так и других нейросетей в рамках разнообразной лиги постоянно адаптирующихся стратегий и контрстратегий, каждая из которых представлена глубокими

нейронными сетями. Серия онлайн-игр против игроков-людей показала, что рейтинг AlphaStar был на уровне гроссмейстера для всех трех рас StarCraft и превышал рейтинг 99,8% официально зарегистрированных игроков-людей.

В 2020 году выходит статья Дж. Шритвайзера с соавторами про нейросеть MuZero от DeepMind [108], обобщающую идеи предыдущих разработок и концентрирующуюся на разработке победных стратегий без знаний про игру, полагаясь на динамику изменения окружения. MuZero при итеративном применении предсказывает величины, непосредственно относящиеся к планированию: вознаграждение, политику выбора действий и функцию ценности. При оценке в Го, шахматах и сёги, без какого-либо знания правил игры, MuZero соответствовал сверхчеловеческой производительности алгоритма AlphaZero, который был применён вместе с правилами игры.

В 2022 году DeepMind представляет нейросеть AlphaTensor [109], использующую методы обучения с подкреплением для автоматического открытия новых алгоритмов умножения матриц. Архитектура основана на AlphaZero и предлагает существенные улучшения в эффективности умножения матриц для различных размеров, в том числе улучшение алгоритма Штрассена для матриц размера $4 \times 4$.

## 7.5 Мультимодальные модели

Мультимодальными нейросетями называются системы искусственного интеллекта, способные обрабатывать несколько типов данных одновременно: текст, изображения, аудио, видео и другие форматы информации. Формально к ним можно отнести много нейросетей из больших языковых моделей, сетей для компьютерного зрения и других, но акцент на мультимодальности стали делать с 2022–2023 года.

В 2022 году статья С. Рида с соавторами представляет разработку DeepMind мультимодального агента Gato [110], работа которого выходит за рамки текстового вывода. Gato работает как мультимодальная, многозадачная и многоцелевая универсальная нейросеть. Одна и та же сеть с одинаковыми весами может играть в игры, подписывать изображения, общаться в чате, складывать блоки с помощью настоящей руки робота и многое другое в зависимости от контекста.

Нейросеть Flamingo — ещё одна разработка DeepMind — освещена в статье Дж.-Б. Алайрака с соавторами [111]. Flamingo представляет собой семейство моделей визуального языка, обладающих возможностью быстро адаптироваться к новым задачам, используя всего несколько примеров с аннотациями. Предлагаются архитектурные инновации, позволяющие объединить мощные предварительно обученные визуальные и

языковые модели, обрабатывать последовательности произвольно чередующихся визуальных и текстовых данных и легко использовать изображения или видео в качестве входных данных. Благодаря своей гибкости сети Flamingo могут обучаться на крупномасштабных мультимодальных веб-ресурсах, содержащих произвольно чередующийся текст и изображения, что является ключевым фактором для обеспечения их возможностями обучения в режиме реального времени.

В 2023 году DeepMind представляет семейство сетей Gemini [112]. Семейство мультимодальных моделей Gemini обладает замечательными возможностями для понимания изображений, аудио, видео и текста. Новые возможности семейства Gemini в области кросс-модального мышления и понимания языка позволят использовать их в самых разнообразных случаях.

Р. Сан с коллегами публикует обзор на основе разработки OpenAI генеративной модели Sora, предназначенная для создания высококачественных видео на основе текстовых описаний [113]. Команда делает вывод, что продукт является важной вехой на пути к созданию общего искусственного интеллекта.

Развитие нейросетей-трансформеров и других современных архитектур заслуживает отдельного обзора [30]. В данной работе мы не будем подробно рассматривать их развитие, поскольку оно связано в большей степени с их масштабируемостью, нежели с новыми концептуальными идеями.

## 7.6 Сети Колмогорова-Арнольда

Теорема суперпозиции Колмогорова-Арнольда легла в основу так называемых сетей Колмогорова-Арнольда. В отличие от многослойных персептронов, лежащих в основе всех современных ИНС, KAN имеют фиксированные функции активации на узлах и обучаемые функции активации на ребрах (рис. 8). Каждый функция активации является одномерной функцией, параметризованной в виде сплайна. Это изменение позволяет KAN превосходить MLP с точки зрения точности и интерпретируемости в ряде задач [31]. KAN являются многообещающей альтернативой MLP, открывая возможности для дальнейшего совершенствования современных моделей глубокого обучения, которые в значительной степени зависят от MLP.

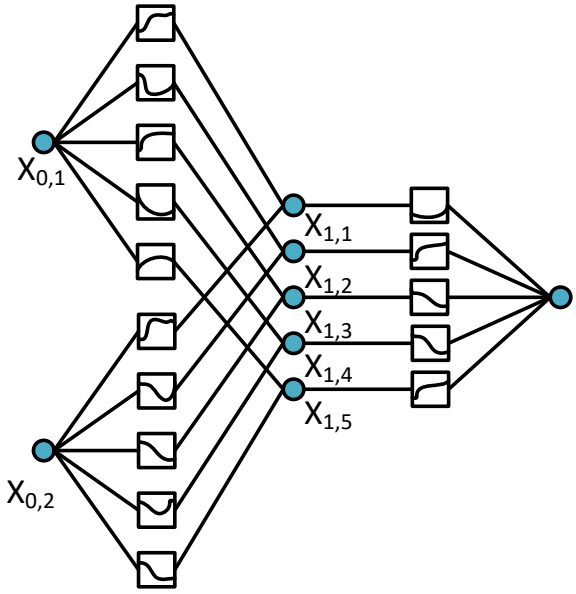



Рисунок 8. Схема архитектуры сети Колмогорова-Арнольда. В отличие от MLP, активационные функции присутствуют как в нейронах, так и в рёбрах, их связывающих.

Для задач аппроксимации при помощи KAN существует теорема Майорова-Пинкуса [114]:

Существует аналитическая вещественная строго монотонно возрастающая функция активации $\sigma$, удовлетворяющая следующему свойству. $\forall f \in C[0,\ 1]^d$и $\varepsilon > 0$ существуют вещественные константы $d_i,\ c_{ij},\ \theta_{ij},\ \gamma_i$ и вектора $\boldsymbol{w}^{ij} \in \mathbb{R}$, для которых

$$\left| f(x) - \sum_{i=1}^{6d+3} d_i \sigma \left( \sum_{j=1}^{3d} c_{ij} \sigma\left(w^{ij} \cdot x + \theta_{ij}\right) + \gamma_i \right) \right| < \varepsilon$$

$\forall x \in [0,\ 1]^d$.

Согласно теореме Майорова-Пинкуса, существует класс функций, требующих большого размера нейронной сети, причём с ростом точности масштабируемость растёт экспоненциально. Также теорема определяет нижнюю границу сложности аппроксимации.

С. А. Немковым [115] было обосновано, что для задач со сложными нелинейными зависимостями KAN являются предпочтительной архитектурой в сравнении с MLP. Это связано с тем, что MLP аппроксимируют результат кусочно-линейными функциями, что приводит к необоснованному росту числа параметров от каждого сегмента. В то же время KAN обладают индуктивным сдвигом в сторону разложения сложных зависимостей на

одномерные гладкие функции. В ряде задач это помогает не только достигнуть результата, но и получить дополнительную информацию для анализа.

## 8. Заключение

Основой современных нейронных сетей является многослойный персептрон. Входные сигналы могут иметь любое числовое значение, а итоговое значение в узле получается суммированием произведений входных сигналов с весами рёбер. Также в каждом узле присутствует активационная функция, которая превращает линейный функционал в произвольный, добавляя элементы нелинейности.

Структура организации искусственных нейронов, связей между ними и способы обработки информации внутри сети может быть гораздо сложнее, чем полносвязные слои нейронов. Для разных задач лучше подходят разные архитектуры и, зачастую, выбор архитектуры обусловлен какой-то эвристикой или даже просто эмпирически.

Среди современных тенденций развития наибольший интерес представляют большие языковые модели, сети-трансформеры, мультимодальные модели, среди представителей которых наиболее известны chatGPT, BERT, Cursor, DeepSeek и другие.

Также развивается альтернатива многослойным персептронам — сети Колмогорова-Арнольда, содержащие не только фиксированные активационные функции в узлах, но и обучаемые активационные функции на рёбрах.

## Список литературы

1. McCarthy J., Minsky M., Rochester N., Shannon C. A proposal for the Dartmouth summer research project on artificial intelligence : technical report. Hanover : Dartmouth College, 1956. 13 p.
2. von Neumann J., Burks A. W. Theory of Self-Reproducing Automata. Urbana : University of Illinois Press, 1966. 322 p.
3. Müller V. C., Bostrom N. Future progress in artificial intelligence: a survey of expert opinion // Fundamental Issues of Artificial Intelligence. Cham : Springer International Publishing, 2016. P. 555–572.
4. Hebb D. O. The Organization of Behavior. New York : John Wiley & Sons, 1949. 335 p.
5. Minsky M., Papert S. Perceptrons: An Introduction to Computational Geometry. Cambridge, MA : MIT Press, 1969. 258 p.
6. Artificial General Intelligence / ed. by B. Goertzel, C. Pennachin. Berlin ; Heidelberg : Springer, 2007. 445 p.
7. Garibotto G. A binocular license plate reader for high precision speed measurement // Journal of Intelligent Transportation Systems. 2001. Vol. 6, no. 1. P. 35–48.

8. Мишинов С. В., Русских Н. Е., Строганов М. С. и др. Использование подходов машинного обучения для воссоздания утраченной части костей черепа // Российский нейрохирургический журнал им. проф. А. Л. Поленова. 2023. Т. 15, спец. вып. 1. С. 17. EDN YJJXPW.
9. Мишинов С. В., Гутт А. А., Пушкина Е. В. и др. Опыт применения подходов машинного обучения в нейрохирургии // Третий Сибирский нейрохирургический конгресс : сб. тезисов / под ред. Д. А. Рзаева. Новосибирск : ООО «Семинары, Конференции и Форумы», 2022. С. 56–57. EDN DOXLVA.
10. Дорошков А. В., Арсенина С. И., Пшеничникова Т. А. и др. Применение компьютерного анализа микроизображений листа для оценки характеристик опушения пшеницы Triticum aestivum L. // Информационный вестник ВОГиС. 2009. Т. 13, № 1. С. 218–226. EDN KUXGKZ.
11. Sidorova E. A., Akhmadeeva I. R., Kononenko I. S. et al. Argument extraction based on the indicator approach // Pattern Recognition and Image Analysis. 2023. Vol. 33, no. 3. P. 498–505.
12. Кожаринов А. С., Кириченко Ю. А., Афанасьев И. В. и др. Методы анализа когнитивных искажений и концепция автоматизированной интеллектуальной системы их детектирования // Нейрокомпьютеры: разработка, применение. 2022. Т. 24, № 4. С. 39–74.
13. Сидорова Е. А., Загорулько Ю. А., Кононенко И. С. и др. Подход к построению датасета для задачи извлечения аргументативных отношений // Двадцать первая Национальная конференция по искусственному интеллекту с международным участием КИИ-2023 (Смоленск, 16–20 октября 2023 г.) : труды конференции : в 2 т. Т. 1. Смоленск : Принт-Экспресс, 2023. С. 211–222.
14. Loukachevitch N., Manandhar S., Baral E. et al. Nerel-bio: a dataset of biomedical abstracts annotated with nested named entities // Bioinformatics. 2023. Vol. 39, no. 4.
15. Bruches E., Mezentseva A., Batura T. A system for information extraction from scientific texts in Russian // Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2021). Cham : Springer, 2022. P. 234–245. (Communications in Computer and Information Science ; vol. 1620).
16. Loukachevitch N., Artemova E., Batura T. et al. Nerel: a Russian information extraction dataset with rich annotation for nested entities, relations, and Wikidata entity links // Language Resources and Evaluation. 2023. Vol. 57, no. 3.
17. St Laurent G., Savva Y. A., Maloney R. et al. Genome-wide analysis of A-to-I RNA editing by single-molecule sequencing in Drosophila // Nature Structural & Molecular Biology. 2013. Vol. 20, no. 11. P. 1333–1339. EDN RXCSLX.
18. Вяткин Ю. В., Антонец Д. В., Шабурова Е. В. и др. Языковые модели в изучении белков // 11-я Московская конференция по вычислительной молекулярной биологии MCCMB’23 : материалы конференции. Москва : ИППИ РАН, 2023. EDN AAFHRE.
19. Borg M., Hewett D., Hagatulah N. et al. Echoes of AI: investigating the downstream effects of AI assistants on software maintainability. 2025.

20. Костарева И. Нейросеть успешно сдала ЕГЭ: что это значит для системы образования? [Электронный ресурс]. 2023. URL: (дата обращения: …).
21. Peng C., Yang X., Chen A. et al. A study of generative large language model for medical research and healthcare // npj Digital Medicine. 2023. Vol. 6. Art. 210.
22. Nechesov A. V., Kondratyev D. A., Sviridenko D. I. et al. Conceptual framework for trustworthy artificial intelligence: combining large language models with formal logic systems // System Informatics. 2025. No. 27. P. 93–118.
23. Kalai A. T., Nachum O., Vempala S. S. et al. Why language models hallucinate : technical report. San Francisco : OpenAI, 2025. Sept.
24. Abbasi Yadkori Y., Kuzborskij I., Stutz D. et al. Mitigating LLM hallucinations via conformal abstention. 2024.
25. Li X., Dong X. L., Lyons K. B. et al. Truth finding on the deep web: is the problem solved? // Proceedings of the VLDB Endowment. 2012. Vol. 6, no. 2. P. 97–108.
26. Yao L. et al. Online truth discovery on time series data // Proceedings of the SIAM International Conference on Data Mining. Philadelphia, PA : Society for Industrial and Applied Mathematics, 2018. P. 162–170.
27. Li Y. et al. On the discovery of evolving truth // Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’15). New York : ACM, 2015. P. 675–684.
28. Андриевский Б. Р., Балашов М. В., Бахтадзе Н. Н. и др. Теория управления: дополнительные главы : учеб. пособие. М. : ЛЕНАНД, 2019. 512 с.
29. Макаренко А. В. Глубокие нейронные сети: зарождение, становление, современное состояние // Проблемы управления. 2020. № 2. С. 3–19.
30. Xu P., Zhu X., Clifton D. A. Multimodal learning with transformers: a survey. 2023.
31. Liu Z., Wang Y., Vaidya S. et al. KAN: Kolmogorov–Arnold networks. 2025.
32. McCulloch W., Pitts W. A logical calculus of the ideas immanent in nervous activity // Bulletin of Mathematical Biophysics. 1943. Vol. 5. P. 115–133.
33. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain // Psychological Review. 1958. Vol. 65, no. 6. P. 386–408.
34. Parallel distributed processing: explorations in the microstructures of cognition / ed. by D. E. Rumelhart, J. L. McClelland. Cambridge, MA : MIT Press, 1986. Vol. 1–2.
35. Галушкин А. И. Синтез многослойных систем распознавания образов. М. : Энергия, 1974. 368 с.
36. Werbos P. J. Beyond regression: new tools for prediction and analysis in the behavioral sciences : PhD thesis. Cambridge, MA : Harvard University, 1974.
37. Барцев С. И., Охонин В. В. Адаптивные сети обработки информации : препринт № 59Б. Красноярск : Ин-т физики СО АН СССР, 1986. 45 с.

38. Rumelhart D. E., Hinton G. E., Williams R. J. Learning internal representations by error propagation // Parallel Distributed Processing. Cambridge, MA : MIT Press, 1986. Vol. 1. P. 318–362.
39. Арнольд В. И. О представлении функций нескольких переменных в виде суперпозиции функций меньшего числа переменных // Математика, её преподавание, приложения и история. 1958. Т. 3. С. 41–61.
40. Cybenko G. Approximation by superpositions of a sigmoidal function // Mathematics of Control, Signals and Systems. 1989. Vol. 2, no. 4. P. 303–314.
41. Siegelmann H., Sontag E. Turing computability with neural nets // Applied Mathematics Letters. 1991. Vol. 4, no. 6. P. 77–80.
42. Hornik K. Approximation capabilities of multilayer feedforward networks // Neural Networks. 1991. Vol. 4, no. 2. P. 251–257.
43. Funahashi K., Nakamura Y. Approximation of dynamical systems by continuous time recurrent neural networks // Neural Networks. 1993. Vol. 6, no. 6. P. 801–806.
44. Chow T., Li X. Modeling of continuous time dynamical systems with input by recurrent neural networks // IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. 2000. Vol. 47, no. 4. P. 575–578.
45. Bengio Y., Simard P., Frasconi P. Learning long-term dependencies with gradient descent is difficult // IEEE Transactions on Neural Networks. 1994.
46. Hinton G., Salakhutdinov R. Reducing the dimensionality of data with neural networks // Science. 2006. Vol. 313, no. 5786. P. 504–507.
47. Nair V., Hinton G. Rectified linear units improve restricted Boltzmann machines // Proceedings of the International Conference on Machine Learning. 2010. P. 807–814.
48. Glorot X., Bengio Y. Understanding the difficulty of training deep feedforward neural networks // Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. 2010. Vol. 9. P. 249–256.
49. Lecun Y., Boser B., Denker J. et al. Backpropagation applied to handwritten zip code recognition // Neural Computation. 1989. Vol. 1, no. 4. P. 541–551.
50. Lecun Y., Bottou L., Bengio Y. et al. Gradient-based learning applied to document recognition // IEEE Intelligent Signal Processing. 1998. P. 306–351.
51. Ciresan D., Meier U., Gambardella L. et al. Deep big simple neural nets excel on handwritten digit recognition. arXiv preprint. 2010. arXiv:1003.0358.
52. Zeiler M. D., Krishnan D., Taylor G. W. Deconvolutional networks // Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. San Francisco, CA, 2010. P. 2528–2535.
53. Krizhevsky A., Sutskever I., Hinton G. Imagenet classification with deep convolutional neural networks // Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS’12). 2012. Vol. 1. P. 1097–1105.

54. Sänchez J., Perronnin F. High-dimensional signature compression for large-scale image classification // 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2011. P. 1665–1672.
55. Hinton G., Srivastava N., Krizhevsky A. et al. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint. 2012. arXiv:1207.0580.
56. Lin M., Chen Q., Yan S. Network in network. arXiv preprint. 2013. arXiv:1312.4400.
57. Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint. 2014. arXiv:1409.1556.
58. Szegedy C., Liu W., Jia Y. et al. Going deeper with convolutions. arXiv preprint. 2014. arXiv:1409.4842.
59. Sifre L. Rigid-motion scattering for image classification : PhD thesis. Palaiseau : École Polytechnique, CMAP, 2014.
60. Long J., Shelhamer E., Darrell T. Fully convolutional networks for semantic segmentation. arXiv preprint. 2014. arXiv:1411.4038.
61. Dumoulin V., Visin F. A guide to convolution arithmetic for deep learning. arXiv preprint. 2016. arXiv:1603.07285.
62. Odena A., Dumoulin V., Olah C. Deconvolution and checkerboard artifacts // Distill. 2016. Vol. 1.
63. Ioffe S., Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint. 2015. arXiv:1502.03167.
64. Ronneberger O., Fischer P., Brox T. U-net: convolutional networks for biomedical image segmentation. arXiv preprint. 2015. arXiv:1505.04597.
65. Yu F., Koltun V. Multi-scale context aggregation by dilated convolutions. arXiv preprint. 2015. arXiv:1511.07122.
66. He K., Zhang X., Ren S. et al. Deep residual learning for image recognition. arXiv preprint. 2015. arXiv:1512.03385.
67. Khan H., Yener B. Learning filter widths of spectral decompositions with wavelets // Proceedings of the Neural Information Processing Systems (NIPS) Conference. 2018. P. 4601–4612.
68. Fujieda S., Takayama K., Hachisuka T. Wavelet convolutional neural networks. arXiv preprint. 2018. arXiv:1805.08620.
69. Liu P., Zhang H., Lian W. et al. Multi-level wavelet convolutional neural networks. arXiv preprint. 2019. arXiv:1907.03128.
70. Elman J. Finding structure in time // Cognitive Science. 1990. Vol. 14, no. 2. P. 179–211.
71. Jordan M. I. Serial order: a parallel distributed processing approach : report 8604. San Diego : Institute for Cognitive Science, University of California, 1986.
72. Jordan M. Serial order: a parallel distributed processing approach // Advances in Psychology. 1997. No. 121. P. 471–495.
73. Yu Y., Si X., Hu C. et al. A review of recurrent neural networks: LSTM cells and network architectures // Neural Computation. 2019. Vol. 31, no. 7. P. 1235–1270.

74. Hochreiter S., Schmidhuber J. Long-short term memory // Neural Computation. 1997. Vol. 9, no. 8. P. 1735–1780.
75. Morin F., Bengio Y. Hierarchical probabilistic neural network language model // Proceedings of the AISTATS Conference. 2005. P. 246–252.
76. Graves A., Schmidhuber J. Framewise phoneme classification with bidirectional LSTM networks // International Joint Conference on Neural Networks. 2005. P. 2047–2052.
77. Graves A. Generating sequences with recurrent neural networks. arXiv preprint. 2013. arXiv:1308.0850.
78. Shi X., Chen Z., Wang H. et al. Convolutional LSTM network: a machine learning approach for precipitation nowcasting. arXiv preprint. 2015. arXiv:1506.04214.
79. Kalchbrenner N., Danihelka I., Graves A. Grid long short-term memory. arXiv preprint. 2015. arXiv:1507.01526.
80. Laurent C., Pereyra G., Brakel P. et al. Batch normalized recurrent neural networks. arXiv preprint. 2015. arXiv:1510.01378.
81. Amodei D., Anubhai R., Battenberg E. et al. Deep Speech 2: end-to-end speech recognition in English and Mandarin. arXiv preprint. 2015. arXiv:1512.02595.
82. Ba J. L., Kiros J. R., Hinton G. E. Layer normalization. arXiv preprint. 2016. arXiv:1607.06450.
83. Vaswani A., Shazeer N., Parmar N. et al. Attention is all you need. arXiv preprint. 2017. arXiv:1706.03762.
84. Dehghani M., Gouws S., Vinyals O. et al. Universal transformers. arXiv preprint. 2018. arXiv:1807.03819.
85. Devlin J., Chang M. W., Lee K. et al. BERT: pretraining of deep bidirectional transformers for language understanding. arXiv preprint. 2018.
86. Sidorova E., Akhmadeeva I., Kononenko I. et al. The role of indicators in argumentative relation prediction // Computational Linguistics and Intellectual Technologies. Papers from the Annual International Conference “Dialogue”. 2023. P. 477–485.
87. Dai Z., Yang Z., Yang Y. et al. Transformer-XL: attentive language models beyond a fixed-length context. arXiv preprint. 2019.
88. Devlin J., Chang M., Lee K. et al. BERT: pre-training of deep bidirectional transformers for language understanding // Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis, Minnesota : Association for Computational Linguistics, 2019. P. 4171–4186.
89. Dosovitskiy A., Beyer L., Kolesnikov A. et al. An image is worth 16×16 words: transformers for image recognition at scale // International Conference on Learning Representations. 2020.
90. Chami I., Abu-El-Haija S., Perozzi B. et al. Machine learning on graphs: a model and comprehensive taxonomy // The Journal of Machine Learning Research. 2022. Vol. 1. P. 3840–3903.

91. Raffel C., Shazeer N., Roberts A. et al. Exploring the limits of transfer learning with a unified text-to-text transformer. 2023.
92. Miftahova A., Pugachev A., Skiba A. et al. NamedEntityRangers at SemEval-2022 task 11: transformer-based approaches for multilingual complex named entity recognition // Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022). 2022. P. 1570–1575. URL: https://aclanthology.org/2022.semeval-1.216.pdf
93. Brown T. B., Mann B., Ryder N. et al. Language models are few-shot learners. 2020.
94. Chowdhery A., Narang S., Devlin J. et al. PaLM: scaling language modeling with pathways. 2022.
95. Jumper J., Evans R., Pritzel A. et al. Highly accurate protein structure prediction with AlphaFold // Nature. 2021. Vol. 596, no. 7873. P. 583–589.
96. Lewis P., Perez E., Piktus A. et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. 2021.
97. Dosovitskiy A., Beyer L., Kolesnikov A. et al. An image is worth 16×16 words: transformers for image recognition at scale. 2021.
98. Rombach R., Blattmann A., Lorenz D. et al. High-resolution image synthesis with latent diffusion models. 2022.
99. Radford A., Kim J. W., Hallacy C. et al. Learning transferable visual models from natural language supervision. 2021.
100. Kirillov A., Mintun E., Ravi N. et al. Segment anything. 2023.
101. Lepikhin D., Lee H., Xu Y. et al. GShard: scaling giant models with conditional computation and automatic sharding. 2020.
102. Tan M., Le Q. V. EfficientNet: rethinking model scaling for convolutional neural networks. 2020.
103. Howard A., Sandler M., Chu G. et al. Searching for MobileNetV3. 2019.
104. Rasley J., Rajbhandari S., Ruwase O. et al. DeepSpeed: system optimizations enable training deep learning models with over 100 billion parameters. 2020. P. 3505–3506.
105. Silver D., Hubert T., Schrittwieser J. et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. 2017.
106. Silver D., Huang A., Maddison C. J. et al. Mastering the game of Go with deep neural networks and tree search // Nature. 2016. Vol. 529, no. 7587. P. 484–489.
107. Vinyals O., Babuschkin I., Czarnecki W. M. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning // Nature. 2019. Vol. 575, no. 7782. P. 350–354.
108. Schrittwieser J., Antonoglou I., Hubert T. et al. Mastering Atari, Go, chess and shogi by planning with a learned model // Nature. 2020. Vol. 588, no. 7839. P. 604–609.
109. Fawzi A., Balog M., Huang A. et al. Discovering faster matrix multiplication algorithms with reinforcement learning // Nature. 2022. Vol. 610, no. 7930. P. 47–53.
110. Reed S., Zolna K., Parisotto E. et al. A generalist agent. 2022.

111. Alayrac J.-B., Donahue J., Luc P. et al. Flamingo: a visual language model for few-shot learning. 2022.
112. Gemini Team, Anil R., Borgeaud S. et al. Gemini: a family of highly capable multimodal models. 2025.
113. Sun R., Zhang Y., Shah T. et al. From Sora what we can see: a survey of text-to-video generation. 2024.
114. Maiorov V., Pinkus A. Lower bounds for approximation by MLP neural networks // Neurocomputing. 1999. Vol. 25, no. 1–3. P. 81–91.
115. Немков С. А. Интерпретируемое моделирование смешения красок с помощью KAN // ИТНОУ: Информационные технологии в науке, образовании и управлении. 2025. № 1–2(24–25). С. 37–42.

УДК 004.4'4

# Алгоритм восстановления позиций выражений в исходном коде Cloud Sisal программ

*Гордеев Д. С. (Институт систем информатики им. А.П. Ершова СО РАН)*

В статье предложен алгоритм восстановления позиций выражений в исходном коде программ на языке Cloud Sisal. Актуальность исследования обусловлена важностью точного сопоставления элементов абстрактного синтаксического дерева с фрагментами исходного текста для построения инструментов разработки, таких как редактор исходного кода, визуальный отладчик, средства диагностики ошибок. Предлагаемый подход решает проблему неполной информации о позициях в выходных данных синтаксических анализаторов, модификация которых затруднительна. В работе описан разработанный трехфазный алгоритм, включающий этапы восстановления последовательности лексем, вычисления позиций лексем и вычисления позиций вершин абстрактного синтаксического дерева. Асимптотическая оценка времени выполнения алгоритма линейно зависит от объема входных данных и не превышает O(n), где n — количество символов в исходной программе.

***Ключевые слова:*** *CPPS, Cloud Sisal, внутреннее представление, абстрактное синтаксическое дерево, AST, лексема, синтаксический анализатор.*

## 1. Введение

В лаборатории конструирования и оптимизации программ ИСИ СО РАН ведется разработка системы облачного параллельного программирования CPPS [4], являющейся интегрированной средой разработки и исполнения программ на языке функционального программирования Cloud Sisal [5]. Одним из ключевых компонентов CPPS является визуальный отладчик, который реализует визуальное изображение внутреннего представления Cloud Sisal программ [3]. Внутреннее представление IR является иерархическим атрибутированным ориентированным ациклическим графом с портами. Данное представление состоит из вершин, содержащих входные и выходные порты. Вершины IR соответствуют функциям и операциям, входные порты соответствуют аргументам, а выходные порты соответствуют результатам вычислений. Для повышения удобства визуальной отладки требуется синхронизация между визуальным представлением

IR графа и исходным текстом Cloud Sisal программы. Возникает задача вычисления позиций выражений в исходном тексте по соответствующим им вершинам IR графа. Решение данной задачи необходимо для реализации таких функций, как: выделение выражений в исходном коде при выделении вершин в IR графе, отображение визуальных эффектов для фрагментов исходного текста программы при распространении данных по дугам IR графа, динамическая установка и снятие точек останова в визуальном редакторе IR графа, динамическая установка и снятие точек останова редакторе исходного кода. Поскольку вершины IR тесно связаны с узлами абстрактного синтаксического дерева AST [1], исходная задача сводится к задаче вычисление соответствия между узлами дерева AST и их позициями в исходном тексте Cloud Sisal программы.

Задача точного сопоставления элементов абстрактного синтаксического дерева с фрагментами исходного текста программ является актуальной и рассматривается в не только в работах, связанных с визуализацией программ. В работе об автоматической верификации C-программ на основе смешанной аксиоматической семантики [10] приведено описание протокола обратной трансляции из промежуточного C-kernel-представления на исходный язык C-light. При трансляции из C-light в C-kernel модифицированные конструкции обогащаются метаинформацией. Эта метаинформация включает данные о примененных при трансляции правилах и о позициях данных конструкций в исходной программе, включая номера строк. Таким образом, данный протокол обратной трансляции является примером решения задачи о поддержке обратного отображения программы из промежуточного представления в исходное. Недостатком данного протокола обратной трансляции является необходимость модификации транслятора для поддержки работы с метаинформацией. В работе о системе IF1-Viewer [8], предназначенной для отображения графовых представлений Sisal-программ в виде промежуточной формы IF1, реализована функциональность подсветки строк в исходном коде Sisal-программы при нажатии на соответствующий узел графового представления в графическом редакторе. Недостатком системы IF1-Viewer является подсветка строк в целом без детализации до конкретных столбцов или позиций в строке, а также отсутствие устойчивости к небольшим модификациям исходного кода, таким как добавление комментариев и скобок. В системе HASKEU [7], предназначенной для разработки программ на функциональном языке Haskell, реализована функциональность одновременных графического и текстового представлений Haskell-программ, а также функциональность синхронизации между данными представлениями. Недостатком данной работы является отсутствие детального описания реализации синхронизации между графическим и текстовым представлениями Haskell-программ. В работах [2, 6] описаны

компоненты трансляции и компиляции программ на языках SISAL 3.*, которые применяются для визуализации внутреннего представления Cloud Sisal программ [3], однако возвращаемая информация о позициях выражений в исходном тексте программы является неполной.

В данной работе описывается трехфазный алгоритм восстановления позиций вершин дерева AST в исходном тексте. В разделе 2 описана формальная постановка задачи, приведены обозначения и основные термины. В разделе 3 описан алгоритм, состоящий из трёх этапов: восстановления последовательности лексем слова из соответствующего дерева AST, восстановление позиций лексем во входном слове с помощью посимвольного сканирования входного слова и восстановление позиций вершин AST с помощью восстановленных позиций лексем. Для каждого из этапов приведён псевдокод. В разделе 4 приведены доказательства утверждений о том, что алгоритм всегда завершается, имеет линейную сложность относительно длины входного слова, и для всех лексем слова вычисляются позиции во входном слове. В разделе 4 также приводится обсуждение применимости предложенного алгоритма в случае возможности модифицировать синтаксический анализатор, возвращающий AST.

## 2. Постановка задачи

Пусть задан контекстно-свободный язык L. Пусть G это однозначная контекстно-свободная грамматика, порождающая язык L. Пусть w это слово (исходный текст программы) из языка L. Пусть SA это синтаксический анализатор, построенный для грамматики G. Пусть AST это абстрактное синтаксическое дерево, построенное для слова w анализатором SA. Пусть $V_{AST}$ это множество вершин из дерева AST. Позицией подстроки в тексте (лексемы или выражения) будем называть четвёрку натуральных чисел ($l_S$, $c_S$, $l_E$, $c_E$) из $N^4$, где $l_S$ и $c_S$ это номера строки и столбца начала подстроки соответственно, $l_E$ и $c_E$ это номера строки и столбца окончания подстроки соответственно. Требуется построить отображение F: $V_{AST} \rightarrow N^4$ , которое каждой вершине дерева AST сопоставляет позицию в исходном тексте w.

## 3. Описание алгоритма

Входные данные алгоритма: слово w из языка L и дерево AST = SA(w).

Выходные данные алгоритма: отображение $F : V_{AST} \rightarrow N^4.V_{AST}$

Алгоритм состоит из трех этапов:

1) Восстановление последовательности лексем слова w из дерева AST.

2) Восстановление позиций лексем в слове w.

3) Восстановление позиций вершин дерева AST.

## 3.1. Восстановление последовательности лексем

Вход: Корень дерева AST.

Выход: Упорядоченный список лексем, соответствующих листьям дерева AST.

Алгоритм представляет собой рекурсивный обход дерева в глубину, в процессе которого посещаются все листовые узлы. Лексемы, хранящиеся в этих узлах, последовательно добавляются в выходной список в порядке их обхода. Данный порядок соответствует порядку следования лексем в исходном тексте для корректно построенного AST.

```
function ExtractLexemes(AstNode node) → Lexeme[]
begin
   if node is leaf then
      begin
         return [node.lexeme]
      end
   else
      begin
         lexemes = []
         for childNode in node.children
            lexemes.append(ExtractLexemes(childNode))
         return lexemes
      end
end
```

## 3.2. Восстановление позиций лексем

Вход: Исходный текст w, список лексем Lexeme[].

Выход: Отображение P: N→ $N^4$, сопоставляющее каждой лексеме из L ее позицию в w.

Алгоритм выполняет сканирование слова w и для каждой лексемы l находит ее первое вхождение в тексте, начиная с текущей позиции сканирования. Алгоритм руководствуется следующими правилами:

**а) Обработка комментариев:** Алгоритм распознает и пропускает однострочные и многострочные комментарии, определенные в грамматике G. При подсчете номеров строк и столбцов длина комментариев учитывается, но сами комментарии в список лексем не входят.

**б) Поиск вхождений:** Для лексемы l алгоритм ищет подстроку, совпадающую с l, начиная с текущей позиции в тексте. После успешного сопоставления текущая позиция сканирования перемещается на символ, следующий за окончанием лексемы l, и начинается поиск следующей лексемы.

**в) Обработка лишних скобок:** Алгоритм корректно обрабатывает области текста, не соответствующие ни одной лексеме из входного списка пропуская их. Это обеспечивает устойчивость к форматированию исходного кода.

Приведённый ниже псевдокод алгоритма опирается на семантику конструкции switch-case, при которой выполняется только одна case-ветвь, и, соответственно, не требуется добавлять инструкцию break для каждой case-ветви. Заметим, что в switch из цикла while только две case-ветви не увеличивают индекс i, указывающий на текущий символ в слове w. Первая такая case-ветвь (*) соответствует переходу в состояние обработки потенциально избыточных открывающих скобок, в котором начинается накапливание позиций открывающих скобок. Вторая такая case-ветвь (**) соответствует возврату из состояния открывающих скобок, при котором используются позиции только самых правых открывающих скобок. При этом соответствующие лексемы открывающих скобок уже обработаны в процессе накапливания позиций открывающих скобок.

```
function BuildLexemePositions(string w, Lexeme[] lexemes) → (N → N^4)
P = {}
current_line = 1
current_column = 1
lexeme_index = -1
mode = text
sub_mode = none
foreach lexeme in lexemes
begin
   lexeme_index += 1
   length = len(lexeme)
   tokenScanCompleted = false
   while (tokenScanCompleted = false && i < len(w))
      switch (mode, sub_mode)
         case (text, _) when w[i..i+2] = "//":
            mode = single_line_comment; current_column += 2; i += 2;
```

```
case (text, _) when w[i..i+2] = “/*”:
   mode = multi_line_comment; current_column += 2; i += 2;
case (text, _) when w[i..i+2] = “\r\n”:
   current_line += 1; current_column = 0; i += 1;
case (text, none) when w[i..i+len] = lexeme:
   tokenScanCompleted = true; current_column += len; i += len;
   position = (current_line, current_column, current_line, current_column + len)
   P = P U (lexeme_index, position)
case (text, none) when w[i..i+1] = “(”: // (*)
   j = lexeme_index
   while (lexemes[j] = “(”) begin  j += 1 end
   lb_count = j – lexeme_index
   fixedSizeQueue.setSize(count)
   sub_mode = lb
case (text, none):
   current_column += 1; i += 1;
case (text, lb) when lb_count > 0 and w[i..i+1] = “(” :
   tokenScanCompleted = true
   fixedSizeQueue ← (current_line, current_column, current_line, current_column + len)
   lb_count -= 1
   current_column += 1; i += 1;
case (text, lb) when lb_count > 0:
   current_column += 1; i += 1;
case (text, lb) when lb_count = 0 and w[i..i+len] = lexeme:
   fixedSizeQueue ← (current_line, current_column, current_line, current_column + len)
   current_column += 1; i += 1;
case (text, lb) when lb_count = 0 and w[i..i+1] = “(”: // (**)
   for(int k = 0; k < fixedSizeQueue.Size; k++)
      P = P U (lexeme_index + k – fixedSizeQueue.Size, fixedSizeQueue[k])
   sub_mode = none
case (text, lb) when lb_count = 0:
   current_column += 1; i += 1;
case (multi_line, _) when w[i..i+2] = “*/”:
   mode = text
```

```
            current_column += 2; i += 2;
        case (multi_line_comment, _) when w[i..i+2] = "\r\n":
            current_column = 0; i += 1; current_line += 1
        case (multi_line_comment, _):
            current_column += 1; i += 1
        case (single_line_comment, _) when w[i..i+2] = "\r\n":
            mode = text
            current_column = 0; i += 1; current_line += 1
        case (single_line_comment, _):
            current_column += 1; i += 1
    end
    return P
```

## 3.3. Восстановление позиций вершин дерева AST

Вход: Дерево AST, отображение P.

Выход: Отображение F: $V_{AST} \rightarrow N^4$

Алгоритм представляет собой рекурсивный обход AST. Для каждой вершины $v \in V_{AST}$ позиция вычисляется как объединение позиций всех ее дочерних вершин. Обход вершин AST производится в том же порядке, что и в алгоритме из раздела 3.1, при этом значение index соответствует количеству обработанных лексем из слова w.

```
procedure BuildAstNodePosition(AstNode node, N → N^4 P, integer index = 0, F = {})
begin
if node is leaf then
    index += 1
    F = F U {(node, P(index))}
else
    begin
        start_index = index
        for childNode in node.children
            BuildAstNodePosition(childNode, P, index, F)
        first = P(start_index + 1)
        last = P(index)
        position = (first.start_line, first.start_column, last.end_line, last.end_column)
```

```
        F = F U {(node, position)}
    end
end
```

# 4. Обсуждение

Предложенный трехфазный алгоритм представляет решение для случая, когда существующий синтаксический анализатор возвращает дерево AST с недостаточной информацией о позициях выражений в исходном тексте программы или без неё и недоступен для модификации. Если существует возможность использовать лексический анализатор отдельно от синтаксического, то вместо рекурсивного обхода дерева AST на первом этапе предложенного алгоритма, на котором восстанавливается последовательность лексем, возможно использовать вызов лексического анализатора. Далее, если лексический анализатор доступен для модификации, то второй этап предложенного алгоритма возможно исключить, если модифицировать лексический анализатор так, чтобы для каждой извлекаемой лексема вычислялась и сохранялась её позиция в исходном слове w. Далее, если синтаксический анализатор доступен для модификации, третий этап предложенного алгоритма можно исключить, если модифицировать синтаксический анализатор так, чтобы при построении дерева AST каждая вершина снабжалась информацией о позиции соответствующего выражения в исходном слове w, наследуемой от позиций порождающих его лексем. Если же модификация синтаксического или лексического анализаторов невозможна или трудоёмка, то предложенный трехфазный алгоритм представляет собой полное и эффективное решение поставленной задачи.

## 4.1. Свойства алгоритма

Предложенный трехфазный алгоритм обладает следующими свойствами.

**Утверждение 1.** Алгоритм всегда завершается.

*Доказательство:*

Этап 1: Обход конечного дерева AST гарантированно завершается, так как количество узлов $|V_{AST}|$ конечно, и каждый узел посещается ровно один раз.

Этап 2: Длина исходного текста |w|=n. Каждая итерация цикла либо увеличивает индекс i, который указывает на позицию сканирования слова w, либо соответствует переключению в режим обработки открывающих скобок, либо обратному переключению из режима обработки открывающих скобок. Причём, если группа открывающих скобок разделена только пробельными символами или комментариями, то для такой группы переключение в

режим обработки открывающих скобок произойдёт ровно один раз. Таким образом, количество итераций, не увеличивающих индекс i не превышает 2/3*n в случае максимально возможного количества открывающих скобок. И, поскольку $i \leq n$, алгоритм завершится за не более чем 2n шагов.

Этап 3: Аналогично этапу 1.

**Утверждение 2.** Сложность алгоритма составляет O(n), где n — длина слова w.

*Доказательство:*

Этап 1: Сложность обхода дерева в глубину равна $O(|V_{AST}|)$, где $V_{AST}$ множество вершин. Известно, что количество вершин в дереве AST пропорционально количеству лексем для многих однозначных грамматик [9]. При этом количество лексем не превышает количество символов в слове w. Соответственно, сложность обхода дерева AST равна O(n).

Этап 2: Длина исходного текста $|w|=n$. Каждая итерация цикла либо увеличивает индекс i, который указывает на позицию сканирования слова w, либо соответствует переключению в режим обработки открывающих скобок, либо обратному переключению из режима обработки открывающих скобок. Причём, если группа открывающих скобок разделена только пробельными символами или комментариями, то для такой группы переключение в режим обработки открывающих скобок произойдёт ровно один раз. Таким образом, количество итераций, не увеличивающих индекс i не превышает 2/3*n в случае максимально возможного количества открывающих скобок. И, поскольку $i \leq n$, то сложность алгоритма равна O(n).

Этап 3: Аналогично этапу 1.

**Утверждение 3.** Все лексемы из списка Lexeme[] получают позиции этапе 2.

*Доказательство:* Поскольку слово w принадлежит языку L, а список лексем получен из AST для w, последовательность лексем в L с точностью до пробельных символов и комментариев совпадает с последовательностью лексем в w. Следовательно, этап 2, пропуская комментарии и пробелы, гарантированно находит каждую следующую лексему, так как поиск начинается с текущей позиции и продолжается с позиции за текущей лексемой, поиск продолжается только в правом направлении, и все лексемы из восстановленной последовательности существуют в слове w.

# 5. Заключение

В работе представлен алгоритм восстановления позиций узлов AST в исходном коде. Алгоритм всегда завершается, имеет линейную сложность и не требует модификации существующего синтаксического анализатора. Для визуальной отладки программ на языке Cloud Sisal в рамках системы CPPS, где вершины IR наследуют структуру AST, и, соответственно, позиции выражений в исходном тексте, предложенный алгоритм полностью решает задачу соответствия изображений вершин внутреннего представления IR и языковых выражений в исходном тексте программы на языке Cloud Sisal. Это соответствие даёт возможность включать и выключать точки останова, не только используя визуальные изображения портов, дуг и вершин, а также фрагменты исходного текста, такие как имена переменных или параметров. Также это соответствие отрывает возможность добавлять отладочные визуальные эффекты непосредственно к изображению исходного текста Cloud Sisal программы в визуальном отладчике.

## Список литературы

1. Ахо А. В., Лам М. С., Сети Р., Ульман Д. Д. Компиляторы: принципы, технологии и инструментарий, 2-е изд. М.: Вильямс, 2008. 1184 с.
2. Глуханков М. П., Дортман П. А., Павлов А. А., Стасенко А. П. Транслирующие компоненты системы функционального программирования SFP // Современные проблемы конструирования программ. — Новосибирск: ИСИ СО РАН, 2002. — Вып. 9. — С. 69–87.
3. Гордеев Д. С. Визуализация внутреннего представления программ на языке Cloud SISAL, Научная визуализация, 2016. – Том 8, № 2. – С. 98–106.
4. Касьянов В. Н., Гордеев Д. С., Золотухин Т. А., Касьянова Е.В., Кондратьев Д.А. Система облачного параллельного программирования CPPS: визуализация и верификация Cloud Sisal программ // под ред. В. Н. Касьянова. Новосибирск: ИПЦ НГУ, 2020. 256 с.
5. Касьянов В. Н., Касьянова Е. В. Язык программирования Cloud Sisal, – Новосибирск, 2018. – 45 с. – (Препринт/РАН, Сиб. отд-ние, ИСИ; N181).
6. Стасенко А. П., Пыжов К. А., Идрисов Р. И. Компилятор в системе функционального программирования SFP // Вестник Новосибирского государственного университета. Серия: Информационные технологии. – 2008. – Т. 6, № 3. – С. 135-146.
7. Alam A., Bush V. HASKEU: An editor to support visual and textual programming in tandem // Proceeding of the 2016 SAI Computing Conference (SAI). London, UK, July 13-15, 2016. IEEE. pp. 805-814. DOI: https://doi.org/10.1109/sai.2016.7556071

8. Chen H., Shirazi B., Thrane S., Marquis J. IF1-Viewer: A Visual Tool for Graphical Display and Execution of SISAL Programs // Proceeding of 13th IEEE Annual International Phoenix Conference on Computers and Communications. Phoenix, USA, April 12-15, 1994. IEEE. pp. 206-212. DOI: https://doi.org/10.1109/PCCC.1994.504116
9. Grune D., Jacobs C. J. H. Parsing Techniques: A Practical Guide (Second Edition). Springer, 2008. — 662 p.

УДК 004.82, 004.42, 004.021

# Язык спецификации дискретных динамических систем, ориентированных на знания, структурированные в онтологиях

*Ануреев И.С. (Институт систем информатики СО РАН)*

В статье рассматривается язык ABML (Attribute-Based Modeling Language), предназначенный для спецификации и прототипирования дискретных динамических систем, ориентированных на знания, структурированные в онтологиях. Язык позволяет формально описывать как онтологические модели систем, так и правила их функционирования, включая динамическое изменение структуры знаний и состояний объектов.

ABML реализован как лексическое расширение диалекта Common Lisp (SBCL) и опирается на минимальный, но выразительный концептуальный базис, включающий объекты, атрибуты и типы объектов. Особое внимание уделяется разделению объектов на изменяемые и константные, а также механизмам типизации, основанным на атрибутах.

В работе подробно описаны средства языка для задания типов, создания и модификации объектов, сопоставления с образцом и вычисления атрибутов. Ключевым элементом ABML является механизм атрибутных замыканий, позволяющий формализовать контекстно-зависимые вычисления атрибутов и моделировать динамику систем в дискретном времени.

Практическая применимость языка демонстрируется на примере моделирования сушилки для рук, для которой построена онтология, а также описаны правила инициализации и функционирования системы. Представленный подход показывает, что ABML может служить удобным инструментом для онтологического моделирования интеллектуальных, информационных и программных систем.

*Ключевые слова*: онтологии, атрибуты, онтологические модели, графы знаний, атрибутные замыкания, ABML, дискретные динамические системы

## 1. Введение

Современные информационные и программные системы все чаще проектируются как сложные дискретные динамические системы, функционирование которых опирается на структурированные знания. Такие знания, как правило, представлены в виде онтологий, определяющих понятия предметной области, их свойства и отношения. В этой связи воз-

никает потребность в формальных языках, способных единообразно описывать как структуру знаний, так и динамику изменения состояний системы во времени.

Существующие языки онтологического моделирования, такие как OWL и связанные с ним формализмы, в первую очередь ориентированы на декларативное представление знаний и логический вывод. Однако они ограничены в средствах описания поведения систем и моделирования их функционирования как последовательности дискретных шагов. С другой стороны, традиционные языки программирования обладают мощными вычислительными возможностями, но не всегда обеспечивают адекватную поддержку онтологического уровня моделирования и явной работы со знаниями.

В данной работе предлагается язык ABML, который разрабатывается как средство онтологического моделирования дискретных динамических систем. Основной целью языка является объединение онтологического подхода с возможностями процедурного и функционального программирования, что позволяет описывать как структуру системы, так и правила ее функционирования в рамках единого формализма.

ABML построен как лексическое расширение диалекта Common Lisp – языка SBCL. Выбор Lisp-подобного языка обусловлен его высокой выразительностью, развитой системой макросов и удобством встраивания предметно-ориентированных языков. Это позволяет реализовать ABML с минимальным числом новых конструкций, сохраняя при этом возможность использования всех средств базового языка.

Концептуальный фундамент ABML основан на трех ключевых понятиях: объектах, атрибутах и типах объектов. Объекты используются для представления элементов системы, атрибуты – для задания их свойств и отношений, а типы объектов – для онтологической классификации. Существенной особенностью языка является различие между изменяемыми и константными объектами, что позволяет явно контролировать семантику изменений состояний системы.

Важным элементом ABML являются механизмы сопоставления с образцом и атрибутных замыканий. Сопоставление с образцом обеспечивает удобное средство задания условий и правил поведения системы, а атрибутные замыкания позволяют формализовать вычисление атрибутов в фиксированном контексте, что особенно важно при моделировании динамики и зависимостей между компонентами системы.

Для демонстрации возможностей языка в статье рассматривается пример моделирования сушилки для рук. На этом примере показывается, как с помощью ABML можно

построить онтологическую модель системы, задать ее начальное состояние и формально описать правила функционирования в виде дискретных тактов. Тем самым иллюстрируется применимость языка для моделирования реальных технических и информационных систем.

В оставшейся части статьи последовательно рассматриваются основные компоненты предлагаемого подхода. В разделе 2 вводится базис языка ABML и формулируются его ключевые концепции. Раздел 3 посвящён системе типов языка, включая базовые типы и типы объектов, основанные на атрибутах. В разделе 4 описываются объекты, их создание и принципы работы с изменяемыми и константными экземплярами. Раздел 5 рассматривает механизмы работы с атрибутами и способы доступа и изменения их значений. В разделе 6 вводятся средства сопоставления с образцом, используемые для задания правил функционирования систем. Раздел 7 посвящён атрибутным замыканиям и их роли в моделировании динамики. В разделе 8 описывается онтология сушилки для рук как пример онтологической модели системы. В разделах 9 и 10 рассматриваются запуск и функционирование сушилки в терминах ABML. В разделе 11 проведен анализ родственных работ. В заключительном разделе подводятся итоги работы и обсуждаются направления дальнейших исследований.

## 2. Базис языка ABML

Язык *ABML (Attribute-Based Modeling Language)* предназначен для прототипирования дискретных динамических систем, ориентированных на знания, структурированные в онтологиях. Он позволяет специфицировать как онтологии таких систем, так и правила функционирования этих систем, меняющие как саму онтологию, так и ее содержимое. Язык является лексическим расширением SBCL (Steel Bank Common Lisp) [21] – популярного диалекта Common Lisp. Мы выбираем язык из семейства Common Lisp в качестве основы ABML как благодаря его хорошо развитым возможностям по встраиванию предметно-ориентированных языков, так и для того, чтобы иметь возможность использовать при необходимости напрямую лисповские средства.

ABML вводит в SBCL лишь небольшой набор дополнительных функций. Хотя большинство этих функций являются макросами, мы далее для универсальности будем использовать термин функция.

Мы проектируем ABML как язык для онтологического моделирования дискретных ди-

намических систем (далее – *систем*), в том числе информационных и программных систем.

ABML основан на минимальном концептуальном фундаменте, состоящем из трех базовых понятий – *объекты*, *атрибуты* и *типы объектов*:

- *Объекты* являются базовыми единицами для моделирования элементов и подсистем системы. Существуют два вида объектов: *изменяемые* (mutable) и *константные* (constant). Изменение атрибутов *изменяемого объекта* (добавление, удаление или изменение атрибута) сохраняет идентичность объекта, тогда как любое изменение атрибута *константного объекта* приводит к созданию нового константного объекта.
- *Атрибуты* определяют свойства и отношения объектов. Каждый атрибут имеет *имя*, *значение* и *тип*, который ограничивает множество допустимых значений.
- *Типы объектов* классифицируют группы объектов, обладающих общими характеристиками, и соответствуют понятиям в онтологии.

## 3. Типы

Типы в ABML делятся на *базовые типы* и основанные на атрибутах *типы объектов*. ABML поддерживает следующие базовые типы:

- `lispt` – множество значений Lisp;
- `symbol` – множество символов Lisp;
- `atom` – множество атомов Lisp;
- `string` – множество строк Lisp;
- `int` – множество целых чисел;
- `nat` – множество натуральных чисел (включая `0`);
- `real` – множество вещественных чисел;
- `(listt` $t$`)` – списки элементов типа $t$;
- `(uniont` $t_1$ ... $t_n$`)` – объединение типов $t_1$, ..., $t_n$;
- `(enumt` $v_1$ ... $v_n$`)` – тип, состоящий из значений $v_1$, ..., $v_n$;
- `(funt` $t_1$ ... $t_n$ $t$`)` – функции из $t_1 \times \ldots \times t_n$ в $t$;
- `any` – объединение всех базовых типов и типов объектов;
- `bool` – синоним `any`, подчеркивающий, что `nil` интерпретируется как ложь, а любое значение, отличное от `nil`, – как истина.

*Типы объектов* делятся на *типы изменяемых объектов* и *типы константных объек-*

*тов.*

*Тип изменяемых объектов* $t''$ определяется как `(mot` $ad_1$ `...` $ad_r$`)`, где декларации атрибутов $ad_j$ задают ограничения на значения атрибутов изменяемых объектов этого типа.

Пусть $a'$, $t'$, $t'_1$, $t'_2$ и $v'$ – значения выражений $a$, $t$, $t_1$, $t_2$ и $v$ соответственно.

Существует четыре вида деклараций атрибутов:

1. `:av` $a$ $v$ – объявляет атрибут $a'$ со значением $v'$. Значение этого атрибута для любого значения (называемого экземпляром) типа $t''$ всегда равно $v'$.
2. `:at` $a$ $t$ – объявляет атрибут $a'$ с типом $t'$. Значение этого атрибута для любого экземпляра типа $t''$ должно принадлежать типу $t'$. На месте $t$ может быть лямбда-выражение с одним аргументом, выступающее в роли характеристической функции: если на значение атрибута функция возвращает значение, отличное от `nil`, то такое значение атрибута считается допустимым.
3. `:atv` $a$ $t$ $v$ – объявляет атрибут $a'$ с типом $t'$ и значением по умолчанию $v'$. Помимо ограничения, описанного в пункте (2), это объявление присваивает значение $v'$ атрибуту $a'$ во всех создаваемых экземплярах типа $t''$, если иное значение не было задано при создании экземпляра.
4. `:amap` $t_1$ $t_2$ – объявляет множество значений типа $t'_1$ в качестве атрибутов, значения которых принадлежат типу $t'_2$. Значение любого атрибута $a$ в экземпляре типа $t''$ должно принадлежать $t'_2$, если $a$ является элементом $t'_1$.

*Тип константных объектов* определяется как `(cot` $ad_1$ `...` $ad_r$`)`, где объявления атрибутов $ad_j$ аналогичным образом задают ограничения на значения атрибутов константных объектов.

Для краткости обозначения

```
1 mot
2 cot
```

используются как сокращения для стандартных определений типов объектов

```
1 (mot)
2 (cot)
```

В ABML новые типы могут определяться с помощью конструкции `(typedef` $n$ $t$`)`, которая объявляет тип с именем $n$ как синоним типа $t$. Для удобства используются сокращенные формы

```
(mot n ad_1 ... ad_r)
(cot n ad_1 ... ad_r)
```

вместо эквивалентных определений

```
(typedef n (mot ad_1 ... ad_r))
(typedef n (cot ad_1 ... ad_r))
```

## 4. Объекты

Объекты могут появляться в модели системы только через специальные функции, которые порождают экземпляры типов объектов.

*Для изменяемых объектов* функция генерации экземпляров имеет вид

```
(mo t ad_1 ... ad_r),
```

где допускаются только объявления атрибутов вида `:av` $a$ $v$. Эта функция создает новый изменяемый объект $o$ типа $t$ и присваивает ему атрибуты и их значения в соответствии с объявлениями атрибутов $ad_j$. Объект $o$ также наследует все атрибуты со значениями по умолчанию, определенные в типе $t$.

Генерация экземпляров типов объектов и последующие изменения их атрибутов и значений этих атрибутов подчиняется двум принципам.

*Принцип ограниченности* гласит, что присваиваемые атрибутам объекта значения должны удовлетворять ограничениям деклараций атрибутов типа, экземпляром которого этот объект является.

*Принцип открытости* утверждает, что экземпляр любого типа объектов может содержать атрибуты, явно не объявленные в этом типе, причем значения таких необъявленных атрибутов ничем не ограничены.

ABML включает предопределенные функции для работы с изменяемыми объектами и типами изменяемых объектов:

- `(uid` $o$`)` – возвращает уникальный идентификатор объекта $o$. Этот уникальный идентификатор является натуральным числом, однозначно идентифицирующем этот объект – никакие два сгенерированных экземпляра типа изменяемых объектов не могут иметь одинакового идентификатора;
- `(imax` $t$`)` – возвращает количество экземпляров типа $t$;

- (otype $o$) – возвращает тип объекта $o$, т. е. (otype $o$) = $t$;
- (is-instance $o$ $t$) – проверяет, является ли объект $o$ экземпляром типа $t$;
- (attributes $o$) – возвращает список непустых атрибутов объекта $o$. Атрибут считается непустым, если его значение отличается от nil.

*Для константных объектов* функция генерации экземпляров имеет вид

```
(co t ad_1 ... ad_r).
```

Все, что описано выше для изменяемых объектов, также применимо и к константным объектам, за исключением того, что для них и их типов не определены функции uid и imax, соответственно.

Для удобства используются сокращенные формы

```
(mo ad_1 ... ad_r)
(co ad_1 ... ad_r)
```

вместо эквивалентных определений

```
(mo mot ad_1 ... ad_r)
(co cot ad_1 ... ad_r)
```

## 5. Атрибуты

Для добавления новых деклараций атрибутов к типам объектов используется функция

```
(att t ad_1 ... ad_r),
```

которая добавляет декларации атрибутов $ad_1$, ..., $ad_r$ к типу $t$.

Для получения значения атрибута $a$ объекта $o$ используется функция (aget $o$ $a$). Для установки значения $v$ атрибута $a$ объекта $o$ применяется функция (aset $o$ $a$ $v$).

Эти функции также работают со списками (listt), где индексы трактуются как атрибуты. В этом случае индекс не должен превышать длину списка при использовании aset и должен быть строго меньше длины списка при использовании aget (так как индексация списков начинается с 0). В противном случае возвращается ошибка.

Они поддерживают также работу с атрибутами на любом уровне вложенности

```
(aget o a_1 ... a_n)
(aset o a_1 ... a_n v)
```

В этом случае, сначала вычисляется атрибут $a_1$, затем вычисляется атрибут $a_2$ на значении $v_1$ атрибута $a_1$ и т. д.

Для вложенного вычисления атрибутов также применяются эквивалентные записи

```
(aget o (aseq a1 ... an))
(aset o (aseq a1 ... an) v)
```

с использованием формы (aseq $a_1$ ... $a_n$).

Если список атрибутов явно не задан, вместо формы (aseq ...) используется форма (aseql $e$). В этом случае список атрибутов вычисляется как значение выражения $e$.

Также имеется сокращенная форма

```
(aset o :av a1 v1 :av ... :av an vn)
```

эквивалентная вложенной форме

```
(aset (... (aset o a1 v1) ...) an vn)
```

представляющей последовательные применения функции aset.

Поведение функции aset зависит от того, применяется ли она к изменяемому или константному объекту. Для изменяемых объектов функция обновляет значение указанного атрибута без изменения самого объекта. В отличие от этого, при применении к константным объектам создается новый константный объект, идентичный исходному, за исключением обновленного значения атрибута. Для списков функция ведет себя так же, как и для константных объектов.

Функция acall является атрибутно-ориентированным вариантом функции aget и трактует атрибут как функцию:

- (acall $a$ $o$) эквивалентна (aget $o$ $a$);
- (acall $a$ $o$ $v_1$ ... $v_s$) применяет функцию с $s$ аргументами, хранящуюся в атрибуте $a$ объекта $o$, к аргументам $v_1$, ..., $v_s$.

## 6. Сопоставление с образцом

Язык ABML имеет развитые средства сопоставления с образцом, основанные на сопоставителях (matchers) вида

```
(match c1 ... cr)
(nmatch c1 ... cr)
```

которые состоят из последовательности клозов сопоставления $c_j$ и реализуют чередование сопоставления с образцом и действий, выполняемых при успешном или неуспешном сопоставлении.

Клозы сопоставления делятся на три категории: *клозы атрибутов*, *клозы выражений* и *клозы действий*.

Пусть $e'$, $a'$, $v'$ и $t'$ обозначают значения $e$, $a$, $v$ и $t$, соответственно.

*Клозы атрибутов* выполняют сопоставление значений атрибутов. ABML поддерживает три вида клозов атрибутов:

1. `:av` $e$ $a$ $v$ – сопоставление успешно, если $e'$ является объектом и его атрибут $a'$ имеет значение $v'$.
2. `:at` $e$ $a$ $t$ – сопоставление успешно, если $e'$ является объектом и значение его атрибута $a'$ принадлежит типу $t'$.
3. `:ap` $e$ $a$ $p$ – сопоставление успешно, если $e'$ является объектом. Параметру $p$, называемую параметром сопоставителя, присваивается значение $e'$.

Формы `(aseq` $a_1$ ... $a_n$`)` и `(aseql` $e$`)` также могут использоваться вместо одиночных атрибутов.

*Клозы выражений* выполняют сопоставление значений выражений. ABML поддерживает три вида клозов выражений:

1. `:v` $e$ $v$ – сопоставление успешно, если $e'$ равно $v'$.
2. `:t` $e$ $t$ – сопоставление успешно, если $e'$ принадлежит типу $t'$.
3. `:p` $e$ $p$ – сопоставление всегда успешно. Параметру сопоставителя$p$ присваивается значение $e'$.

*Клозы действий* задают действия, выполняемые при успешном или неуспешном сопоставлении. ABML поддерживает два вида клозов действий:

1. `:do` $e_1$ ... $e_m$ – последовательно вычисляет выражения $e_j$. Эти выражения могут использовать параметры сопоставителя. После вычисления выражений сопоставление продолжается.
2. `:exit` $e_1$ ... $e_m$ – последовательно вычисляет выражения $e_j$. Эти выражения могут использовать параметры сопоставителя. После вычисления выражений сопоставитель завершает работу, возвращая последнее вычисленное значение.

Сопоставители помимо возвращения значения, также возвращают признак того, успешно ли прошло сопоставление или нет. Поэтому их можно использовать на месте клозов

атрибутов и выражений, обеспечивая таким образом вложенные сопоставления.

Сопоставитель `match` последовательно вычисляет клозы сопоставления, входящие в него по следующим правилам:

1. Если очередной клоз является клозом атрибутов, клозом выражений или сопоставителем, и успешно сопоставляется, то переходим к вычислению следующего клоза.
2. Если очередной клоз является клозом атрибутов, клозом выражений или сопоставителем, сопоставление терпит неудачу, и оставшаяся последовательность клозов содержит `exit`-клоз, то вычисляем ближайший `exit`-клоз и завершаем работу сопоставителя с признаком успешного сопоставления.
3. Если очередной клоз является клозом атрибутов, клозом выражений или сопоставителем, сопоставление терпит неудачу, и оставшаяся последовательность клозов не содержит `exit`-клозов, то завершаем работу сопоставителя с признаком неудачного сопоставления.
4. Если очередной клоз является `do`-клозом, то вычисляем его и переходим к вычислению следующего клоза.
5. Если очередной клоз является `exit`-клозом, то пропускаем его и переходим к вычислению следующего клоза.
6. Если клозов больше нет, то завершаем работу сопоставителя с признаком неудачного сопоставления.

Сопоставитель `nmatch` также как и `match` последовательно вычисляет клозы сопоставления, но действует противоположным образом:

1. Если очередной клоз является клозом атрибутов, клозом выражений или сопоставителем, сопоставление терпит неудачу, то переходим к вычислению следующего клоза.
2. Если очередной клоз является клозом атрибутов, клозом выражений или сопоставителем, и успешно сопоставляется, и оставшаяся последовательность клозов содержит `exit`-клоз, то вычисляем ближайший `exit`-клоз и завершаем работу сопоставителя с признаком успешного сопоставления.
3. Если очередной клоз является клозом атрибутов, клозом выражений или сопоставителем, успешно сопоставляется, и оставшаяся последовательность клозов не содержит `exit`-клозов, то завершаем работу сопоставителя с признаком неудачного сопоставления.

4. Если очередной клоз является `do`-клозом, то вычисляем его и переходим к вычислению следующего клоза.
5. Если очередной клоз является `exit`-клозом, то пропускаем его и переходим к вычислению следующего клоза.
6. Если клозов больше нет, то завершаем работу сопоставителя с признаком неудачного сопоставления.

## 7. Атрибутные замыкания

В дополнение к способам вычисления значений атрибутов, описанным выше, ABML предоставляет механизм связывания атрибутов со значениями (экземплярами) любых типов и вычисления этих атрибутов в фиксированном контексте с использованием *атрибутных замыканий*.

*Атрибутное замыкание* задает:

- вычисляемый атрибут,
- конкретный экземпляр типа, для которого вычисляется этот атрибут,
- а также конечное множество дополнительных параметров вместе с их значениями (называемое контекстом вычисления атрибута), которые влияют на вычисление атрибута.

Замыкания атрибутов представляются в виде константных объектов.

Константный объект $ac$ называется *атрибутным замыканием* относительно атрибута $a$ и типа $t$, если выполняются следующие условия:

- `(aget` $ac$ `"attribute") =` $a$
- `(aget` $ac$ `"instance") =` $i$, где $i$ является экземпляром типа $t$.

Остальные атрибуты объекта $ac$ образуют контекст вычисления атрибута $a$.

Способ вычисления атрибутных замыканий задается декларацией атрибутного замыкания одного из следующих видов:

```
(aclosure ac :attribute a :type t :instance i s1 s2 s3 :do e1 ... er)
(aclosure ac :attribute a :type t :instance i s1 s2 s3
  :match c1 ... cr)
(aclosure ac :attribute a :type t :instance i s1 s2 s3
  :nmatch c1 ... cr)
```

где $s_1$, $s_2$ и $s_3$ имеют вид

```
1 :a_1 p_1 ... :a_n p_n
2 :ap w_1 b_1 q_1 ... :ap w_m b_m q_m
3 :p u_1 t_1 ... :p u_k t_k
```

соответственно.

Результат вычисления атрибутного замыкания $ac$ для атрибута $a$ и типа $t$ определяется $\lambda$-функцией `(lambda (`$ac$`) `$b$`)`, где тело $b$ имеет вид

```
1 (match :ap ac "instance" i :ap ac a_1 p_1 ... :ap ac a_n p_n
2   :ap w_1 b_1 q_1 ... :ap w_m b_m q_m :p u_1 t_1 ... :p u_k t_k :do e_1 ... e_r)
3
4 (match :ap ac "instance" i :ap ac a_1 p_1 ... :ap ac a_n p_n
5   :ap w_1 b_1 q_1 ... :ap w_m b_m q_m :p u_1 t_1 ... :p u_k t_k c_1 ... c_r)
6
7 (match :ap ac "instance" i :ap ac a_1 p_1 ... :ap ac a_n p_n
8   :ap w_1 b_1 q_1 ... :ap w_m b_m q_m :p u_1 t_1 ... :p u_k t_k
9   (nmatch c_1 ... c_r))
```

соответственно. Здесь выражения $e_1$, ..., $e_r$, $c_1$, ..., $c_r$ могут зависеть от параметров $i$, $ac$, $p_1$, ..., $p_n$, $q_1$, ..., $q_m$, $t_1$, ..., $t_k$.

Часть `:instance` $i$ $s_1$ $s_2$ $s_3$ декларации атрибутного замыкания называется *префиксом декларации*. Элементы префикса могут как переставляться (при этом соответствующим образом переставляются элементы в $\lambda$-функции), так и опускаться. Часть декларации, следующая за префиксом, называется *телом декларации*.

Декларация атрибутного замыкания задает способ вычисления, а само вычисление выполняется функцией `(eval-aclosure` $ac$`)`. Напомним, что вычисление атрибутного замыкания $ac$ эквивалентно вычислению значения атрибута `(aget` $ac$ `"attribute")`.

Помимо функции `(eval-aclosure` $ac$`)` над атрибутными замыканиями определены следующие функции:

- `(clear-aclosure` $ac$`)` – удаляет все атрибуты у атрибутного замыкания $ac$ кроме `"attribute"` и `"instance"`, т. е. контекст вычисления атрибута в $ac$;
- `(update-eval-aclosure` $ac$ `...)` – сначала выполняет `(aset` $ac$ `...)`, модифицируя значения атрибутов замыкания $ac$, а затем `(update-eval-aclosure` $ac'$`)` для моди-

фицированного замыкания $ac'$;

- `(clear-update-eval-aclosure` $ac$ `...)` – сначала выполняет `(clear-aclosure` $ac$`)`, удаляя контекст вычисления атрибута в замыкании $ac$, а затем `(update-eval-aclosure` $ac'$ `...)` для модифицированного замыкания $ac'$.

В следующих разделах будет рассмотрен такой пример дискретной динамической системы как сушилка для рук и для нее на языке ABML будет построена онтология (онтологическая модель) и правила первого запуска этой системы и ее дальнейшего функционирования.

## 8. Онтология сушилки для рук

Онтология (или онтологическая модель) сушилки для рук определяется тремя типами изменяемых объектов.

Тип `"system"` определяет сушилку как систему, состоящую из сенсора и контроллера:

```
(mot "system"
  :at "controller" "controller"
  :at "sensor" "sensor")
```

Тип `"sensor"` описывает сенсор через его состояние, моделирующее замечены руки или нет:

```
(mot "sensor"
  :at "state" (enumt "detected" "not detected"))
```

Тип `"controller"` моделирует контроллер, определяя такие его компоненты как

- связанный с ним сенсор `"sensor"`;
- состояние `"state"`, в котором находится контроллер (режим его работы);
- константы `"maximum drying time"` и `"cooling time"`, характеризующие максимальное время непрерывной работы сушилки и время охлаждения сушилки, заданные для простоты числом тактов работы контроллера;
- локальные часы `"local clock"`, подсчитывающие количество тактов, которые контроллер непрерывно находился в состоянии сушки или состоянии охлаждения.

Он задается следующим образом:

```
(mot "controller"
```

```
  :at "sensor" "sensor"
  :at "local clock" nat
  :at "state" (enumt "waiting" "drying" "cooling")
  :av "maximum drying time" 100000
  :av "cooling time" 1000)
```

Конкретная сушилка для рук (точнее ее состояние в определенной момент времени) может, например, быть задана (порождена) следующим образом:

```
(match
  :p (mo "sensor" :av "state" "not detected") sen
  :p (mo "controller"
    :av "sensor" sen
    :av "local clock" 0
    :av "state" "waiting") cont
  (mo "system" :av "controller" cont :av "sensor" sen))
```

Это выражение возвращает экземпляр типа "system".

Заметим, что конечные наборы экземпляров типов можно рассматривать как граф знаний, в котором вершины помечены этими экземплярами и значениями базовых типов, а дуги помечены именами атрибутов и ведут от объекта, для которого вычисляется атрибут к значению этого атрибута.

В данном примере, граф знаний, соответствующий состоянию сушилки для рук, определенному выше, имеет вид как на Рис.1.

## 9. Запуск сушилки

Запуск сушилки моделируется декларацией атрибутного замыкания для атрибута "init" и типа "system":

```
(aclosure ac :attribute "init" :type "system" :instance i
  :match
  :ap i "sensor" sen :ap i "controller" cont
  :do
    (aset sen "state" "not detected")
```

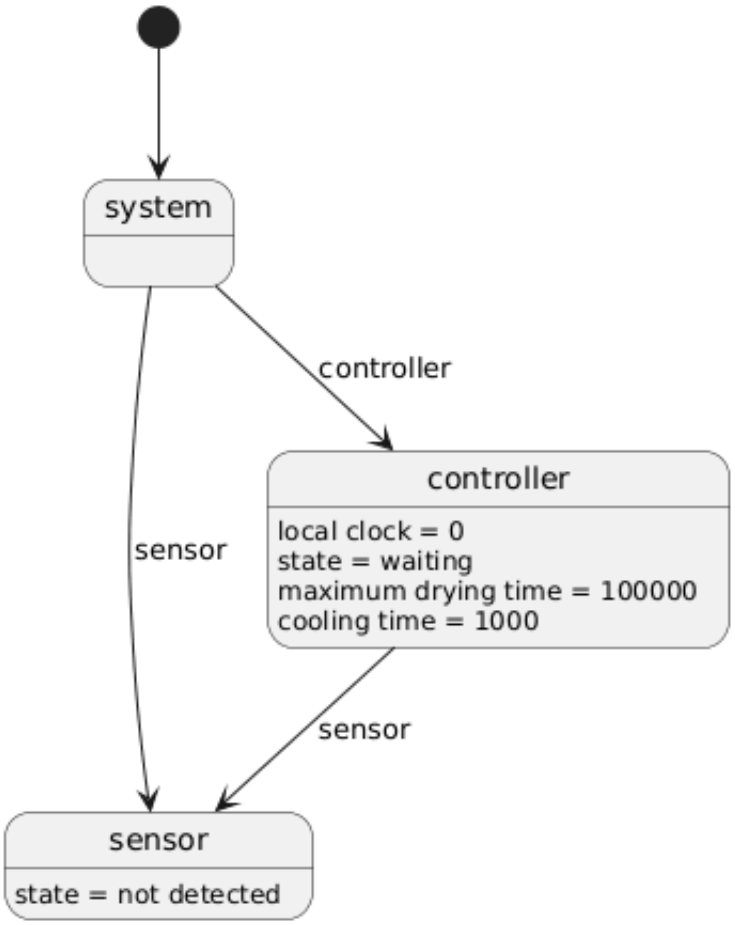



Рис. 1: Граф знаний для состояния сушилки для рук

```
    (aset cont :av "sensor" sen :av "state" "waiting" :av "local
       clock" 0))
```

Эта декларация присваивает начальные значения атрибутам компонент *sen* и *cont* системы *i*.

## 10. Функционирование сушилки

Функционирование сушилки также моделируется через декларации атрибутных замыканий.

Декларации для атрибута "step" определяет один такт работы системы и ее компонент.

Для системы целиком она определяется как

```
(aclosure ac :attribute "step" :type "system" :instance i :do
  (update-eval-aclosure ac :av "instance" (aget i "sensor"))
  (update-eval-aclosure ac :av "instance" (aget i "controller")))
```

Сначала выполняется такт сенсора с получением данных из окружающей среды, а затем такт контроллера на полученных данных.

Шаг для сенсора состоит в получении случайных данных и моделируется с помощью функции `random-list-element`, выбирающей случайное значение из списка:

```
(aclosure ac :attribute "step" :type "sensor" :instance i
  :do (aset i "state" (random-list-element
    (list "detected" "not detected"))))
```

Шаг контроллера состоит в определении его состояния и запуска соотвествующего режима функционирования (ожидание, сушка, пассивное охлаждение) в данном состоянии:

```
(aclosure ac :attribute "step" :type "controller" :instance i
  :match :ap i "state" s (nmatch
    :v s "waiting"
    :exit (update-eval-aclosure ac :av "attribute" "waiting")
    :v s "drying"
    :exit (update-eval-aclosure ac :av "attribute" "drying")
    :v s "cooling"
    :exit (update-eval-aclosure ac :av "attribute" "cooling")))
```

Режим ожидания задается следующей декларацией:

```
(aclosure ac :attribute "waiting" :type "controller" :instance i
  :match :av i (aseq "sensor" "state") "detected"
    :do (aset i :av "state" "drying" :av "local clock" 0))
```

В этом режиме отслеживается срабатывание датчика и переход контроллера в этом случае в состояние сушки с обнулением локального времени.

В режиме сушки, задаваемом декларацией

```
(aclosure ac :attribute "drying" :type "controller" :instance i
  :nmatch
    :v (< (aget i "local clock")
      (aget i "maximum drying time")) T
    :exit (aset i :av "state" "cooling" :av "local clock" 0)
    :av i (aseq "sensor" "state") "not detected"
```

```
    :exit (aset i "state" "waiting")
    :do (aset i "local clock" (+ (aget i "local clock") 1)))
```

сначала выполняется проверка не превышен ли лимит непрерывной сушки. Если лимит превышен, контроллер переходит в режим пассивного охлаждения с обнулением локального времени. В противном случае, проверяется состояние датчика и если он ничего не обнаруживает, то контроллер переходит в состояние ожидания. Если ни одно из выше проверяемых условий не выполнено, то увеличивается время локальных часов на 1 (один такт). Заметим, что в случае перехода в состояние ожидания время не сбрасывается в ноль, так как для этого состояния время локальных часов не учитывается.

Декларация, моделирующая режим пассивного охлаждения, определяется аналогичным образом:

```
(aclosure ac :attribute "cooling" :type "controller" :instance i
  :nmatch
    :v (< (aget i "local clock")
      (aget i "cooling time")) T
    :exit (aset i "state" "waiting")
    :do (aset i "local clock" (+ (aget i "local clock") 1)))
```

Таким образом, мы построили как модель состояний такой системы как сушилка для рук, так и модель функционирования этой системы в терминах онтологии.

## 11. Родственные работы

Исследования, посвящённые формальному описанию знаний и динамики систем, ведутся в нескольких взаимосвязанных направлениях, включая онтологическое моделирование, языки спецификации динамических и реактивных систем, предметно-ориентированные языки (DSL), а также подходы, основанные на графах знаний и атрибутных вычислениях [18, 20, 24]. Язык ABML находится на пересечении этих направлений, объединяя элементы онтологий, типизированных объектных моделей и механизмов описания поведения.

**Онтологические языки и представление знаний.** Наиболее распространённым формализмом для представления онтологий является семейство языков, основанных на дескриптивных логиках, прежде всего OWL (Web Ontology Language) [5, 7, 9]. Эти языки

обеспечивают строгую семантику, поддержку логического вывода и широко применяются в задачах семантического веба и интеграции знаний [8, 18]. Однако OWL и родственные ему формализмы ориентированы преимущественно на статическое описание знаний и обладают ограниченными возможностями для моделирования динамики и изменения состояний объектов во времени [2].

Для расширения онтологического подхода в сторону описания поведения разрабатывались различные онтологические модели процессов и событий, включая OWL-S и SOSA/SSN [18, 20, 23, 24]. Эти модели позволяют описывать действия, события и наблюдения, но, как правило, не предоставляют формального механизма исполнения или пошагового моделирования динамических систем. В отличие от них, ABML изначально ориентирован на моделирование дискретной динамики и допускает явное описание шагов функционирования системы.

**Языки спецификации динамических и реактивных систем.** Значительный пласт родственных работ связан с языками спецификации динамических, реактивных и киберфизических систем [6]. Классическими примерами являются языки временной логики, такие как LTL и CTL [19], а также формализмы на основе автоматов и систем переходов [4]. Эти подходы широко используются для верификации свойств систем, однако они слабо приспособлены для непосредственного описания сложных структур знаний и онтологий.

Языки спецификации, такие как Event-B и TLA+ [11, 16], предлагают строгие математические средства для описания состояний и переходов, но требуют значительных усилий для моделирования предметной области на уровне объектов и атрибутов. В отличие от перечисленных формализмов, ABML ориентирован на знание-центричный подход, в котором онтологическая структура системы и динамика её функционирования описываются в рамках единого атрибутного формализма.

**Объектно-ориентированные и атрибутно-ориентированные модели.** Многие идеи ABML перекликаются с объектно-ориентированным моделированием и индустриальными языками моделирования, такими как UML и SysML [17]. В этих языках объекты, атрибуты и состояния играют центральную роль, однако формальная семантика большинства их конструкций либо задаётся неявно, либо выходит за рамки стандартов, а средства исполнения моделей, как правило, носят ограниченный или инструментально-зависимый характер (см, например, [3, 15, 22]).

Атрибутно-ориентированные подходы к моделированию рассматривались, в частности, в контексте систем правил и продукционных систем, где вычисление значений атрибутов определяется набором явно заданных зависимостей и условий [12]. В таких системах вычисление значений атрибутов может зависеть от контекста и состояния других объектов. ABML развивает эти идеи, вводя формализованный механизм атрибутных замыканий, который позволяет явно задавать контекст вычисления и связывать его с конкретным экземпляром типа.

**Предметно-ориентированные языки и Lisp-подобные системы.** Разработка ABML как расширения Common Lisp тесно связана с традицией создания предметно-ориентированных языков (DSL) [13]. Lisp и его диалекты исторически используются для создания языков моделирования и спецификации благодаря мощной макросистеме и однородному синтаксису [1].

Существуют Lisp-ориентированные системы для представления знаний и онтологий, такие как Loom и OCML, которые предоставляют средства описания понятий и отношений. Однако они, как правило, либо ориентированы на логический вывод, либо не поддерживают явное моделирование дискретной динамики. ABML отличается тем, что сочетает декларативное описание структуры знаний с процедурным описанием поведения.

**Графы знаний и вычисления на графах.** В последние годы активно развиваются подходы, основанные на графах знаний, где информация представляется в виде вершин и дуг с семантической интерпретацией [10]. Графы знаний используются в интеллектуальных системах, анализе данных и моделировании сложных взаимосвязей. В этом контексте модель ABML может интерпретироваться как граф знаний, в котором объекты и значения образуют вершины, а атрибуты — помеченные рёбра.

Отличительной особенностью ABML является то, что вычисления и изменения состояния системы формализуются как преобразования такого графа знаний во времени. Это сближает ABML с подходами, основанными на трансформациях графов [14], но при этом сохраняет удобство атрибутного и типизированного моделирования.

**Итоги сравнения.** Таким образом, существующие родственные работы либо фокусируются на статическом представлении знаний, либо на формальной спецификации динамики без явной онтологической структуры. Язык ABML занимает промежуточную позицию,

предлагая унифицированный формализм для онтологического моделирования и описания дискретной динамики систем. Его атрибутно-ориентированный подход и механизм атрибутных замыканий позволяют выразить широкий класс моделей, что отличает его от большинства существующих решений.

## 12. Заключение

В работе представлен язык ABML, предназначенный для спецификации дискретных динамических систем, ориентированных на знания, структурированные в онтологиях. Язык объединяет онтологическое моделирование и формальное описание поведения систем в рамках единого, компактного и выразительного формализма.

Основным достоинством ABML является минимальный, но универсальный концептуальный базис, включающий объекты, атрибуты и типы объектов. Разделение объектов на изменяемые и константные позволяет явно задавать семантику изменений и облегчает моделирование эволюции состояний системы. Атрибутно-ориентированная типизация обеспечивает гибкий механизм задания ограничений и классификации объектов, соответствующий онтологическому подходу.

Развитые средства работы с атрибутами, включая вложенный доступ, массовое обновление и интерпретацию атрибутов как функций, делают язык удобным для описания сложных структур знаний. Механизмы сопоставления с образцом позволяют компактно и наглядно формулировать правила функционирования систем, а также реализовывать условные переходы между состояниями.

Ключевым элементом языка является механизм атрибутных замыканий, который обеспечивает контекстно-зависимое вычисление атрибутов и служит основой для моделирования дискретной динамики. Использование атрибутных замыканий позволяет рассматривать поведение системы как последовательность вычислений атрибутов, что хорошо согласуется с онтологической интерпретацией модели в виде графа знаний.

Пример моделирования сушилки для рук наглядно демонстрирует практическую применимость ABML. В рамках одного языка удалось задать онтологию системы, ее начальное состояние и правила функционирования, описывающие поведение сенсора и контроллера во времени. Это подтверждает, что ABML может использоваться для прототипирования и анализа поведения реальных технических, информационных и программных систем.

В перспективе язык ABML может быть расширен средствами верификации, анализа свойств моделей и интеграции с внешними онтологическими и логическими инструментами. Такой подход делает ABML перспективным средством для исследования и разработки интеллектуальных систем, основанных на знаниях и онтологиях.

## Список литературы


1. Alneami A., Mc Kevitt P. On Lisp: Advanced Techniques for Common Lisp. Paul Graham // Artificial Intelligence Review. — 1999. — Vol. 13, no. 3. — P. 239–241.
2. Baader F., Horrocks I., Sattler U. Description logics as ontology languages for the semantic web // Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday. — Springer, 2005. — P. 228–248.
3. Clark T., Warmer J. Object modeling with the OCL: the rationale behind the Object Constraint Language. — Springer, 2002.
4. Clarke E. M. J., Grumberg, O., Peled, DA: Model Checking. — 1999.
5. (Comet-) atomic 2020: On symbolic and neural commonsense knowledge graphs / Hwang J. D., Bhagavatula C., Le Bras R., Da J., Sakaguchi K., Bosselut A., and Choi Y. // Proceedings of the AAAI conference on artificial intelligence. — 2021. — Vol. 35. — P. 6384–6392.
6. Compositional model checking of consensus protocols via interaction-preserving abstraction / Gu X., Cao W., Zhu Y., Song X., Huang Y., and Ma X. // 2022 41st International Symposium on Reliable Distributed Systems (SRDS) / IEEE. — 2022. — P. 82–93.
7. Consortium W. W. W. et al. OWL 2 web ontology language document overview. — 2012.
8. A derived information framework for a dynamic knowledge graph and its application to smart cities / Bai J., Lee K. F., Hofmeister M., Mosbach S., Akroyd J., and Kraft M. // Future Generation Computer Systems. — 2024. — Vol. 152. — P. 112–126.
9. Design ontology supporting model-based systems engineering formalisms / Lu J., Ma J., Zheng X., Wang G., Li H., and Kiritsis D. // IEEE Systems Journal. — 2021. — Vol. 16, no. 4. — P. 5465–5476.
10. Ehrlinger L., Wöß W. Towards a definition of knowledge graphs. // SEMANTiCS (Posters, Demos, SuCCESS). — 2016. — Vol. 48, no. 1-4. — P. 2.

11. Farrell M., Monahan R., Power J. F. Building specifications in the Event-B institution // Logical Methods in Computer Science. — 2022. — Vol. 18.
12. Forgy C. L. Rete: A fast algorithm for the many pattern/many object pattern match problem // Readings in artificial intelligence and databases. — Elsevier, 1989. — P. 547–559.
13. Fowler M. Domain-specific languages. — Pearson Education, 2010.
14. Fundamentals of algebraic graph transformation / Ehrig H., Ehrig K., Prange U., and Taentzer G. — Springer, 2006.
15. Kleppe A. G., Warmer J. B., Bast W. MDA explained: the model driven architecture: practice and promise. — Addison-Wesley Professional, 2003.
16. Lamport L. Specifying systems. — Addison-Wesley Boston, 2002. — Vol. 388.
17. Merging OMG standards in a general modeling, transformation, and simulation framework. / Schneider V., Yupatova A., Dulz W., and German R. // SimuTools. — 2015. — P. 299–301.
18. Ontologies in digital twins: A systematic literature review / Karabulut E., Pileggi S. F., Groth P., and Degeler V. // Future Generation Computer Systems. — 2024. — Vol. 153. — P. 442–456.
19. Pnueli A. The temporal logic of programs // 18th annual symposium on foundations of computer science (sfcs 1977) / ieee. — 1977. — P. 46–57.
20. Representing Time-Continuous Behavior of Cyber-Physical Systems in Knowledge Graphs / Gill M. S., Jeleniewski T., Gehlhoff F., and Fay A. // arXiv preprint arXiv:2506.13773. — 2025.
21. Rhodes C. SBCL: A sanely-bootstrappable Common Lisp // Workshop on Self-sustaining Systems / Springer. — 2008. — P. 74–86.
22. Rumpe B. Agile modeling with UML: Code generation, testing, refactoring. — Springer, 2017.
23. The SSN ontology of the W3C semantic sensor network incubator group / Compton M., Barnaghi P., Bermudez L., Garcia-Castro R., Corcho O., Cox S., Graybeal J., Hauswirth M., Henson C., Herzog A., et al. // Journal of Web Semantics. — 2012. — Vol. 17. — P. 25–32.
24. Наместников А.М. Применение онтологического подхода в задаче генерации событийных данных с помощью имитационных моделей // Онтология проектирования. — 2023. — Vol. 13, no. 2 (48). — P. 243–253.

УДК 004.451.2, 004.82, 004.021, 519.6, 004.42

# Операционная семантика операторов передачи управления в языке C на языке ABML

*Ануреев И.С. (Институт систем информатики СО РАН)*

В работе рассматривается онтологический подход к заданию операционной семантики операторов передачи управления языка программирования C. В качестве формального средства используется предметно-ориентированный язык ABML, ранее предложенный для спецификации дискретных динамических систем, ориентированных на знания, структурированные в онтологиях. Показано, что операционную семантику фрагментов языков программирования, заданную в терминах систем переходов, можно интерпретировать как динамическую систему и формализовать средствами ABML.

В статье вводится онтология операторов передачи управления языка C, включающая операторы goto, break, continue и return, а также онтологии конструкций, реагирующих на передачу управления, таких как помеченные операторы, блоки и оператор switch. Для этих онтологических моделей задается операционная семантика в виде атрибутных замыканий, вычисляемых относительно агентов и окружения.

Особое внимание уделяется адаптации языка ABML к задачам задания операционной семантики, включая уточнение понятия атрибутного замыкания, введение стадий вычисления и явное моделирование контекста выполнения. Предложенный подход обеспечивает модульность, расширяемость и наглядность спецификации семантики.

Полученные результаты демонстрируют применимость онтологического моделирования для формального описания семантики языков программирования и создают основу для дальнейшего расширения подхода на другие конструкции языка C, а также на анализ и верификацию программ.

*Ключевые слова*: операционные семантики, онтологии языков программирования, модели языков программирования, атрибутные замыкания, ABML, операторы передачи управления

## 1. Введение

Формальное задание семантики языков программирования является одной из ключевых задач теории программирования и формальных методов. Операционная семантика, описывающая поведение программ через последовательность элементарных шагов выполнения, традиционно задается с помощью систем переходов, абстрактных машин или пра-

вил вывода. Такие описания, хотя и обладают высокой точностью, часто оказываются слабо структурированными, трудно расширяемыми и плохо приспособленными для повторного использования при анализе различных фрагментов языка.

В последние годы заметный интерес вызывает применение онтологического подхода к моделированию программных систем. Онтологии позволяют явно фиксировать структуру предметной области, типы сущностей и отношения между ними, что делает модели более прозрачными и пригодными для автоматизированной обработки. В контексте языков программирования это открывает возможность представлять синтаксические и семантические конструкции языка в виде онтологических моделей, а правила их функционирования – в виде формализованных механизмов изменения знаний.

В работе [63] был предложен язык ABML (Attribute-Based Modeling Language), предназначенный для спецификации и прототипирования дискретных динамических систем, ориентированных на знания, структурированные в онтологиях. Язык ABML объединяет онтологическое моделирование с элементами функционального и процедурного программирования и предоставляет средства для задания объектов, атрибутов, типов, сопоставления с образцом и атрибутных замыканий. В предыдущей работе было показано, что ABML может эффективно использоваться для моделирования динамики реальных технических и информационных систем.

Настоящая статья развивает данный подход и рассматривает возможность применения ABML для задания операционной семантики языков программирования. Основная идея заключается в том, что систему переходов, лежащую в основе операционной семантики, можно рассматривать как дискретную динамическую систему, а значит, описывать ее средствами ABML. Однако специфика языков программирования – наличие контекста выполнения, стеков, областей видимости, передачи управления – требует адаптации базовых механизмов языка.

В качестве предмета исследования выбран фрагмент языка C, связанный с операторами передачи управления. Эти операторы играют важную роль в управлении потоком выполнения программы и существенно усложняют формальное описание семантики из-за нелокальных переходов, взаимодействия с блоками, циклами и оператором switch. В статье предлагается онтологическая модель операторов передачи управления и связанных с ними конструкций, а также формальное задание их операционной семантики на языке ABML.

Целью работы является демонстрация того, что онтологический подход в сочетании с языком ABML позволяет получить модульное, расширяемое и формально точное описание операционной семантики операторов передачи управления языка C. Полученные результаты могут служить основой для дальнейшего расширения модели, а также для исследований в области анализа, верификации и интерпретации программ.

Статья имеет следующую структуру. В разделе 2 рассматривается адаптация языка ABML к задачам задания операционной семантики языков программирования и вводятся необходимые уточнения базовых понятий. В разделе 3 описываются этапы построения операционной семантики в терминах ABML. Раздел 4 посвящен онтологии операторов передачи управления языка C, а в разделе 5 вводится онтология конструкций, связанных с передачей управления. В разделе 6 описываются модели агентов и окружения, используемые для задания контекста выполнения. Разделы 7 и 8 содержат формальное описание операционной семантики операторов передачи управления и связанных с ними конструкций. В разделе 9 проведен анализ родственных работ. В заключении подводятся итоги работы и обсуждаются направления дальнейших исследований.

## 2. Адаптация ABML для разработки операционных семантик

В работе [63] был предложен предметно-ориентированный язык ABML, предназначенный для спецификации и прототипирования дискретных динамических систем, ориентированных на знания, структурированные в онтологиях. Основная идея применения этого языка для спецификации операционной семантики языков программирования заключается в следующем. Если задавать операционную семантику языков программирования с помощью систем переходов, то такие системы переходов можно рассматривать как дискретные динамические системы и, таким образом, можно применить язык ABML для спецификации таких систем.

Однако эти системы имеют свои особенности, и чтобы учитывать их, требуется некоторые модификации языка ABML. Опишем ниже эти модификации.

Понятие атрибутного замыкания переопределяется следующим образом. Константный объект $ac$ называется *атрибутным замыканием* относительно атрибута $a$ и типа $t$, если выполняются следующие условия:

- (aget $ac$ "attribute") = $a$;
- (aget $ac$ "instance") = $i$, где $i$ является экземпляром типа $t$;

- `"agent"` – экземпляр типа изменяемых объектов `"agent"`. Агенты выполняют двоякую роль. С одной стороны, агенты хранят в своих атрибутах знания, необходимые для выполнения конструкций языка программирования (например, значения переменных, типы переменных, распределение памяти и т. п.). С другой стороны, агентов можно рассматривать как отдельных исполнителей, специфика которых определяется хранимым в них знанием (например, для определения потоков, процессов и т. п.). Обязательным атрибутом агентов является атрибут `"value"`, хранящий последнее вычисленное в этом агенте значение;
- `"env"` – экземпляр типа изменяемых объектов `"env"`. Окружение во-первых, хранит информацию, общую для всех агентов, а во-вторых, позволяет агентам обмениваться этой информацией через атрибуты окружения. В окружении также реализуется обобщение концепции стека, широко используемой в языках программирования для откладывания вычислений (например, элементом стека может быть функция с аргументами из ее текущего вызова). Реализация основана на двух обязательных атрибутах окружения. Атрибут `"agents"` типа `(listt "agent")` хранит список всех действующих агентов. Атрибут `"aclosures"` типа `(cot :amap "agent" (listt cot))` моделирует стек для каждого агента, хранящий отложенные атрибутные замыкания для этого агента. Декларации типов для окружения и агентов задаются пользователем отдельно для каждого языка программирования.

Остальные атрибуты объекта $ac$ образуют контекст вычисления атрибута $a$. Также в контекст вычисления добавляется обязательный атрибут `"stage"`, значения которого характеризуют отдельные стадии вычисления атрибутного замыкания. Конкретный набор этих стадий зависят от типа экземпляра, для которого вычисляется атрибут в замыкании $ac$. Например, для условного оператора можно выделить 3 стадии: вычисление условия, выбор ветви и вычисление выбранной ветви. Использование данного атрибута способствует модульности операционной семантики, а также в совокупности с механизмом стеков, описанным ниже, упрощает обработку исключительных ситуаций (появляющихся как результат выполнения операторов передачи управления, операторов порождения исключений и т. п.), которые могут встретиться на каждой стадии.

Таким образом, к обязательным атрибутам атрибутных замыканий `"attribute"` и `"instance"` добавляются еще два атрибута `"agent"` и `"env"`. В частности, эти атрибуты также будут сохраняться при выполнении функции `(clear-aclosure` $ac$`)`, которая удаля-

ет атрибуты из контекста вычисления.

В язык ABML добавляются следующие функции работы со стеками окружения:

- push-aclosure – добавляет атрибутное замыкание *ac* в стек соответствующего агента, связанного с *ac*. Она имеет следующую семантику:

```
(defun push-aclosure (ac)
  (match :ap ac "agent" a :ap ac "env" e
    :ap e "agents" al
    :ap e (aseq "aclosures" a) cl
    :do (aset e "aclosures" a (cons ac cl))
    :v (not (member a al)) T
    :do (aset e "agents" (cons a al))))
```

- (pop-aclosure *ac*) – удаляет атрибутное замыкание *ac* из стека соответствующего агента, возвращая его в качестве значения этой функции. Она имеет следующую семантику:

```
(defun pop-aclosure (ac)
  (match :ap ac "env" e :ap e "agents" al
    :v (not (null al)) T :exit nil
    :p (car al) a :ap e (aseq "aclosures" a) cl
    :v (not (null cl)) T :exit nil
    :do (aset e "aclosures" a (cdr cl)) (car cl)))
```

- (peek-aclosure *ac*) – читает атрибутное замыкание *ac* из стека соответствующего агента, возвращая его в качестве значения этой функции. Она имеет следующую семантику:

```
(defun peek-aclosure (ac)
  (match :ap ac "env" e :ap e "agents" al
    :v (not (null al)) T :exit nil
    :p (car al) a :ap e (aseq "aclosures" a) cl
    :v (not (null cl)) T :exit nil
    :do (car cl)))
```

Имеются также сокращения

```
(update-push-aclosure ac ...)
clear-update-push-aclosure ac ...)
```

для часто используемых операций

```
(push-aclosure (aset ac ...))
(update-push-aclosure (clear-aclosure ac))
```

В префикс декларации атрибутного замыкания добавляется элемент `:value` $p$, который связывает с переменной $p$ значение атрибута `"value"` агента `(aget` $ac$ `"agent")`, связанного с атрибутным замыканием $ac$. Таким образом, декларация атрибутного замыкания принимает один из следующих видов:

```
(aclosure ac :attribute a :type t :instance i :value p s1 s2 s3
  :do e1 ... er)
(aclosure ac :attribute a :type t :instance i :value p s1 s2 s3
  :match c1 ... cr)
(aclosure ac :attribute a :type t :instance i :value p s1 s2 s3
  :nmatch c1 ... cr)
```

где $s_1$, $s_2$ и $s_3$ имеют вид

```
:a1 p1 ... :an pn
:ap w1 b1 q1 ... :ap wm bm qm
:p u1 t1 ... :p uk tk
```

соответственно.

Результат вычисления атрибутного замыкания $ac$ для атрибута $a$ и типа $t$ определяется $\lambda$-функцией `(lambda (`$ac$`)` $b$`)`, где тело $b$ имеет вид

```
(match :ap ac "instance" i :ap ac (aseq "agent" "value") p
  :ap ac a1 p1 ... :ap ac an pn :ap w1 b1 q1 ... :ap wm bm qm
  :p u1 t1 ... :p uk tk :do e1 ... er (next-aclosure ac))

(match :ap ac "instance" i :ap ac (aseq "agent" "value") p
  :ap ac a1 p1 ... :ap ac an pn :ap w1 b1 q1 ... :ap wm bm qm
  :p u1 t1 ... :p uk tk c1 ... cr (next-aclosure ac))
```

```
8
9  (match :ap ac "instance" i :ap ac (aseq "agent" "value") p
10   :ap ac a1 p1 ... :ap ac an pn :ap w1 b1 q1 ... :ap wm bm qm
11   :p u1 t1 ... :p uk tk (nmatch c1 ... cr) (next-aclosure ac))
```

соответственно. Здесь выражения $e_1$, ..., $e_r$, $c_1$, ..., $c_r$ могут зависеть от параметров $i$, $ac$, $p$, $p_1$, ..., $p_n$, $q_1$, ..., $q_m$, $t_1$, ..., $t_k$.

По-прежнему элементы префикса декларации атрибутного замыкания могут как переставляться, так и опускаться.

Функция (next-aclosure $ac$) определяет какое атрибутное замыкание требуется взять из стеков окружения и выполнить после того, как завершится выполнение замыкания $ac$. Эта функция определяется пользователем в зависимости от специфики языка программирования, для которого строится операционная семантика. В этой статье мы используем следующее определение:

```
1  (defun next-aclosure (ac)
2    (match :ap ac "value" v
3      :v (not (is-instance v "stop next aclosure")) T :exit v
4      :ap ac "agent" a :ap ac "env" e :ap e (aseq "aclosures" a) st
5      :v (null st) T :exit (eval-aclosure (car st))
6      :ap e "agents" al
7      :v (not (null al)) T :exit nil
8      :p (car al) a1 :ap e (aseq "aclosures" a1) st1
9      :v (null st1) T :exit (aset e "aclosures" a (cdr cl))
10       (eval-aclosure (car st1))
11     :do (aset e "agents" a (cdr al)) next-aclosure (ac)))
```

Оно выбирает первый элемент стека агента, связанного с замыканием $ac$, а если стек пуст, то первый элемент первого агента из списка агентов в окружении, для которого стек не пуст. Если стеки для всех агентов пусты, то эта функция ничего не делает.

Для моделирования ситуации, когда next-aclosure ничего не делает, и, таким образом, ABML-программа завершает свою работу, используется значение типа "stop next aclosure" из атрибута "value" агента, связанного с замыканием $ac$. Этот тип определяется следующим образом:

```
(cot "stop next aclosure" :at "type" string :at "aclosure" (cot))
```

Атрибут "type" хранит тип остановки программы (например, "error"), а атрибут "aclosure" хранит замыкание, на котором произошла остановка.

## 3. Этапы построения операционной семантики на языке ABML

Подход к построению операционной семантики фрагмента языка программирования на языке ABML состоит из следующих шагов:

1. Построить онтологию фрагмента языка программирования как набор типов объектов языка ABML, соответствующих синтаксическим и семантическим конструкциям этого фрагмента.
2. Определить типы для агентов и окружения.
3. Задать операционную семантику фрагмента как набор атрибутных замыканий относительно атрибута "opsem" и всех типов, определенных на шаге 1, которые соответствуют исполняемым конструкциям фрагмента. В этом случае, вычисление атрибута "opsem" для онтологической модели исполнимой конструкции через атрибутное замыкание соответствует вычислению операционной семантики этой конструкции относительно агента, окружения и других параметров, задаваемых этим атрибутным замыканием.

## 4. Онтология операторов передачи управления

Онтология операторов передачи управления задается следующим набором типов:

```
(typedef "jump statement" (uniont
  "goto1" "continue" "break" "return1" "return1"))
(mot "goto1" :at 1 "identifier")
(mot "continue")
(mot "break")
(mot "return")
(mot "return1" :at 1 "expression")
```

Индексы 1, 2, 3 и т. д. в именах типов показывают позиции аргументов. Типы "continue", "break" и "return", соответствующие операторам `continue`, `break` и `return` без аргументов, не имеют атрибутов. Атрибутом 1 типа "goto" является метка оператора `goto`, а

атрибутом 1 типа "return1" является выражение, связанное с оператором return. Тип "jump statement" моделирует все виды операторов передачи управления и определяется как их объединение.

## 5. Онтология операторов, связанных с операторами передачи управления

Определим также операторы и выражения, которые реагируют на передачу управления, разбив их на группы.

Первую группу операторов составляют помеченные операторы. Для упрощения операционной семантики мы рассматриваем в качестве помеченных операторов различные виды меток без следующих за ними операторов. Эта группа операторов моделируется следующими типами:

```
(typedef "labeled statement" (uniont
  "label1" "case1" "default"))
(mot "label1" :at 1 "label name")
(mot "case1" :at 1 "constant expression")
(mot "default")
```

Типы "label1" и "case1", соответствующие обычных меткам и case-меткам, имеют один атрибут 1 со значениями типов "label name" и "constant expression", представляющих метки и константные выражения, соответственно. Тип "default" моделирует default-метки.

Вторую группу составляют операторы блока. Их модели определяются следующим образом:

```
(mot "{1}"
  :at 1 (listt (uniont "declaration" "statement"))
  :at "variables" (listt "variable")
  :at "label position" (cot :amap "label name" nat))
```

Атрибут "variables" хранит список переменных, объявляемых на верхнем уровне в списке операторов блока. Это знание нужно для того, чтобы при переходе к телу этого оператора для переменных с теми же именами, как переменные из этого списка, сохранять ячейки памяти, связанные с ними, поскольку эти переменные будут прятаться при объявлениях

соответствующих переменных внутри тела, а при выходе из тела цикла восстанавливать старые связи между переменными и ячейками.

Атрибут "variable location" в этом типе хранит отображение меток, встречающихся в списке операторов блока, в их позиции в этом списке.

Третью группу образуют онтологические модели операторов `switch`, представленные типом "switch(1)2":

```
(mot "switch(1)2" :at 1 "expression" :at 2 (listt "statement")
  :at "variables" (listt "variable"))
```

Атрибуты 1 и 2 имеют типы "expression" и (listt "statement") и задают управляющее выражение и тело этих операторов.

Четвертую группу образуют операторы итерации, включающих операторы `while`, операторы `do-while` и два вида операторов `for` без и с декларацией переменных. Эта группа моделируется следующими типами:

```
(typedef "iteration statement" (uniont "while(1)2"
  "do1while(2)" "for(1;2;3)4" "for(var1;2;3)4"))

(mot "while(1)2" :at 1 "expression" :at 2 "statement")

(mot "do1while(2)" :at 1 "statement" :at 2 "expression")

(mot "for(1;2;3)4" :at 1 "expression" :at 2 "expression"
  :at 3 "expression" :at 4 "statement")

(mot "for(var1;2;3)4" :at 1 "declaration" :at 2 "expression"
  :at 3 "expression" :at 4 "statement"
  :at "variables" (listt "variable"))
```

Атрибут "variables" в типе "for(var1;2;3)4" имеет тот же смысл, что и в типе "switch". Он хранит список переменных, объявляемых в атрибуте 1. Смысл остальных аргументов данных типов легко определяется их положением в имени типа.

Пятую группу составляют вызовы функций. Они моделируются следующим типом:

```
(mot "1(2)" :at 1 "expression" :at 2 (listt "expression"))
```

Экземпляры этого типа связаны только с онтологическими моделями операторов `return`.

## 6. Модели агентов и окружения

Поскольку у нас последовательное вычисление и вычислитель только один, тип для окружения не имеет атрибутов:

```
(mot "env")
```

Агент для языка Си хранит достаточно много информации, но для нашего подмножества языка Си достаточно двух атрибутов:

```
(mot "agent"
  :at "location" (cot :amap "variable" "location")
  :at "value" "c value")
```

Предопределенный атрибут `"value"` имеет тип `"c value"`, который строится как объединение всех типов значений языка Си.

Атрибут `"location"` сопоставляет переменным программы связанные с ними ячейки памяти и имеет тип `(cot :amap "variable""location")`.

Тип `"location"` ячеек памяти определяется следующим образом:

```
(mot "location" :at "value" "c value")
```

Он имеет атрибут `"value"`, который хранит значение, приписанное ячейке памяти.

## 7. Операционная семантика моделей операторов передачи управления

**Оператор break.** Операционная семантика оператора `break` (более точно его онтологической модели) задается следующим атрибутным замыканием:

```
(aclosure ac :attribute "opsem" :type "break"
  :p (pop-aclosure ac) ac1 :nmatch
  :v (null ac1) T :exit (error ac "break")
  :do (match
    :ap ac1 "stage" st :do (nmatch
      :v (equal st "exiting 1(2)")
```

```
7       :exit (error ac "break")
8       :v (equal st "exiting while(1)2")
9       :v (equal st "exiting do1while(2)")
10      :v (equal st "exiting for(1;2;3)4")
11      :exit
12      :v (equal st "exiting for(var1;2;3)4")
13      :v (equal st "exiting switch(1)2")
14      :exit (eval-aclosure ac1)
15      :v (equal st "exiting {1}")
16      :exit (push-aclosure ac) (eval-aclosure ac1)
17      :do (eval-aclosure ac))))
```

Тело этого атрибутного замыкания моделирует передачу управления, осуществляемую оператором `break`, через другие операторы.

Строка `2` сохраняет в параметре `ac1` верхний элемент стека, связанного с текущим агентом (`aget ac "agent"`), удаляя этот элемент из стека в текущем окружении (`aget ac "env"`).

В строке `3` выполняется проверка, а не пуст ли этот стек (пустота стека соответствует значению `nil` параметра `ac1`). Если стек пуст, то выдается ошибка, так как это означает, что не встретился оператор, который должен был поймать переход по оператору `break`.

В строке `5` в параметре `st` сохраняется значение текущей стадии вычисления атрибутного замыкания `ac1`.

В строке `6` проверяется, является ли эта стадия стадией выхода `"exiting 1(2)"` из вычисления вызова функции. Если является, то выдается ошибка, так как такая ситуация тоже невозможна. Заметим, что единственное знание о других операторах и выражениях, которым владеет оператор передачи управления (в данном случае оператор `break`) – это знание имен стадий этих операторов, которые реагируют на передачу управления.

В строках `8-10` проверяется, является ли эта стадия стадией выхода `"exiting while(1)2"`, `"exiting do1while(2)"` и `"exiting for(1;2;3)4"` из операторов `while`, `do-while` и `for` без декларации переменных, соответственно. Если является, то передача управления завершается в строке `11`. А поскольку эти стадии являются признаками завершения соответствующих операторов, то, в соответствии с семантикой функции

`next-aclosure` управление передается следующему замыканию из стека замыканий (в частности, если после этих циклов имеется еще оператор, то управление передается ему).

В строках 12-13 проверяется, является ли эта стадия стадией выхода `"exiting for(var1;2;3)4"` и `"exiting switch(1)2"` из операторов `for` с декларацией переменных и `switch`, соответственно. Если является, то передача управления завершается в строке 14. Но, в отличие от предыдущих случаев, перед выходом из этих операторов выполняются действия, связанные с восстановлением старых ячеек памяти для переменных, а именно вычисляется атрибутное замыкание `ac1`.

В строке 15 проверяется, является ли эта стадия стадией выхода из блока. Если является, то передача управления продолжается после блока, что обеспечивается выражением `(push-aclosure ac)`, но перед этим также выполняются действия, связанные с восстановлением старых ячеек памяти для переменных, а именно вычисляется выражение `(eval-aclosure ac1)`. Заметим, что чтобы обеспечить такой порядок вычисления этих выражений, первое выражение откладывает вычисление `ac`, помещая его в стек. Поэтому сначала вычислится `ac1`, а потом восстановиться из стека и вычислиться `ac`.

Строка 17 соответствует любой другой стадии и любому другому оператору. В этом случае передача управления продолжается.

**Оператор continue.** Операционная семантика оператора `continue` задается во многом аналогичным образом:

```
1  (aclosure ac :attribute "opsem" :type "continue"
2    :p (peek-aclosure ac) ac1 :nmatch
3    :v (null ac1) T :exit (error ac "continue")
4    :do (match
5      :ap ac1 "stage" st :do (nmatch
6        :v (equal st "exiting 1(2)")
7        :exit (error ac "continue")
8        :v (equal st "exiting while(1)2")
9        :exit (update-eval-aclosure ac1 :stage "executing 1")
10       :v (equal st "exiting do1while(2)")
11       :exit (update-eval-aclosure ac1 :stage "executing 2")
12       :v (equal st "exiting for(1;2;3)4")
```

```
      :v (equal st "exiting for(var1;2;3)4")
      :exit (update-eval-aclosure ac1 :stage "executing 3")
      :v (equal st "exiting switch(1)2")
      :v (equal st "exiting {1}")
      :exit (pop-aclosure ac) (push-aclosure ac)
        (eval-aclosure ac1)
      :do (pop-aclosure ac) (eval-aclosure ac))))
```

Но при этом имеется несколько отличий для этого оператора.

Во-первых, для стадии `"exiting for(var1;2;3)4"` не нужно восстанавливать старые ячейки памяти для переменных, так как мы не выходим из оператора `for`.

Во-вторых, стадия `"exiting switch(1)2"` обрабатывается также, как и завершающая стадия для блока, поскольку передача управления продолжается.

В-третьих, для стадий, связанных с завершением операторов итерации управление передается на соответствующие стадии этих операторов, с которых их выполнение продолжается.

**Оператор goto.** Операционная семантика оператора `goto` задается следующим образом:

```
(aclosure ac :attribute "opsem" :type "goto1"
  :p (pop-aclosure ac) ac1 :nmatch
  :v (null ac1) T :exit (error ac "goto1")
  :do (match :ap ac1 "stage" st :do (nmatch
    :v (equal st "exiting 1(2)")
    :exit (error ac "goto")
    :v (equal st "exiting for(var1;2;3)4") T
    :v (equal st "exiting switch(1)2") T
    :exit (push-aclosure ac) (eval-aclosure ac1)
    :v (equal st "exiting {1}") T
    :exit (match :ap i 1 lab :ap ac1 "instance" i1
      :ap i1 "label position" lp
      :v (member lab (attributes lp)) T
      :do (match :ap i1 2 sts :ap (aget lp lab) k
```

```
15          :do (push-aclosure ac1)
16            (clear-update-eval-aclosure ac1
17              :stage "iteration" :av "current" (+ k 1)
18              :av "bound" (length sts)
19              :av "statements" sts))
20        :exit (push-aclosure ac) (eval-aclosure ac1))
21      :do (eval-aclosure ac))))
```

Для стадий `"exiting for(var1;2;3)4"` и `"exiting switch(1)2"` (строки 7 и 8) передача управления сопровождается восстановлением старых значений переменных.

Для стадии завершения блока (строка 10) в строке 11 в параметрах `lab` и `i1` сохраняются метка оператора `goto` и блок, до конца которого произошла передача управления, а в строке 12 в параметре `lp` сохраняется отображение меток, которые встречаются в операторе блока на верхнем уровне, в их позиции в списке операторов блока.

В строке 13 выполняется проверка, принадлежит ли метка `lab` меткам, которые встречаются в операторе блока на верхнем уровне.

Если проверка выполняется (строка 14), то параметрам `sts` и `k` присваиваются список операторов блока и позиция метки `lab` в этом списке. Затем запускается выполнение блока с оператора в `k + 1` позиции. Это делается на стадии `"iteration"` вычисления блока. Контекст вычисления атрибута `"opsem"` на этой стадии дополнительно включает атрибуты `"current"`, `"bound"` и `"statements"`, хранящие позицию вычисляемого в текущий момент оператора из списка операторов блока, число операторов в списке и сам список, соответственно.

Если проверка не выполняется (строка 20), то восстанавливаются старые ячейки памяти для переменных, определенных в блоке, блок завершается и передача управления продолжается.

**Оператор return без аргумента.** Операционная семантика этого оператора проще, чем предыдущих:

```
(aclosure ac :attribute "opsem" :type "return" :match
  :p (pop-aclosure ac) ac1 :v (not (null ac1)) ac1
  :do (match :ap ac1 "stage" st :do (nmatch
    :v (equal st "exiting for(var1;2;3)4") T
```

```
5       :v (equal st "exiting switch(1)2") T
6       :v (equal st "exiting {1}") T
7       :exit (push-aclosure ac) (eval-aclosure ac1)
8       :av (equal st "exiting 1(2)") T
9       :exit (eval-aclosure ac1)
10      :do (eval-aclosure ac)))
11    :exit (error ac "return"))
```

Для случаев, представленных в строках 4-6 также происходит восстановление старых ячеек памяти для переменных в строке 7 с продолжением передачи управления.

Единственный случай, когда передача управления завершается – это выход из вызова функции (строка 8).

**Оператор return с аргументом.** Операционная семантика для оператора `return`, возвращающего значение, разбивается на несколько стадий.

Для начальной стадии (она всегда имеет имя `nil`) имеем следующее определение:

```
1 (aclosure ac :attribute "opsem" :type "return1" :instance i :do
2   (update-push-aclosure ac "stage" "storing return value")
3   (clear-update-eval-aclosure ac "instance" (aget i 1)))
```

В строке 2 в стек окружения откладывается стадия `"storing return value"`, которая потом перехватит возвращаемое функцией значение и передаст его за пределы вызова функции.

В строке 3 запускается вычисление выражения, связанного с оператором `return` через атрибут `1`. Заметим, что если при вычислении выражение произойдет исключительная ситуация (например, будет послан сигнал в Си), то отложенная стадия `"storing return value"` не будет вычисляться, так как будет удалена из стека механизмом просачивания исключительной ситуации подобно тому, как просачиваются операторы передачи управления.

Стадия `"storing return value"` определяется следующим образом:

```
1 (aclosure ac :attribute "opsem" :type "return1"
2   :stage "storing return value" :value v :do
3   (update-eval-aclosure ac :stage "propagation"
```

```
4     :av "return value" v))
```

В строке 2 в параметре v сохраняется вычисленное значение выражения, связанного с оператором return. В строках 3-4 запускается стадия "propagation", которая просачивает это значение до выхода из вызова функции.

Стадия "propagation" определяется в основном аналогично единственной стадии оператора return без аргумента:

```
1  (aclosure ac :attribute "opsem" :type "return1"
2    :stage "propagation" :match
3    :p (pop-aclosure ac) ac1 :v (not (null ac1)) ac1
4    :do (match :ap ac1 "stage" st :do (nmatch
5      :v (equal st "exiting for(var1;2;3)4") T
6      :v (equal st "exiting switch(1)2") T
7      :v (equal st "exiting {1}") T
8      :exit (push-aclosure ac) (eval-aclosure ac1)
9      :av (equal st "exiting 1(2)") T
10     :exit
11       (update-push-aclosure ac :stage "returning value")
12       (eval-aclosure ac1)
13     :do (eval-aclosure ac)))
14   :exit (error ac "return1"))
```

Единственное отличие заключается в том, что после выхода из вызова функции (строка 12) выполняется еще одна стадия "returning value" оператора goto, которая делает значение выражения последним вычисленным значением, помещая его в атрибут value текущего агента.

Операционная семантика этой последней стадии выполнения оператора goto определяется следующим образом:

```
1  (aclosure ac :attribute "opsem" :type "return1"
2    :stage "returning value" :do (aget ac "return value"))
```

Эта стадия просто возвращает значение атрибута "return value" из контекста вычисления атрибута opsem в качестве значения этого атрибута.

## 8. Операционная семантика моделей операторов, связанных с операторами передачи управления

Выполнение оператора `switch` разбивается на 8 стадий, каждая из которых моделирует отдельный аспект его выполнения: сохранение контекста, вычисление управляющего выражения, сопоставление меток (несколько стадий) и восстановление окружения. Ниже приведено формальное описание этих стадий на языке ABML.

На нулевой стадии осуществляется переход к стадии `"entering switch(1)2"` (строка 2), которая перед выполнением оператора `switch` сохраняет ячейки памяти, связанные с переменными, объявляемыми в этом операторе на верхнем уровне:

```
1 (aclosure ac :attribute "opsem" :type "switch(1)2" :do
2   (update-eval-aclosure ac :stage "entering switch(1)2"))
```

Стадия `"entering switch(1)2"` моделируется следующим образом:

```
1 (aclosure ac :attribute "opsem" :type "switch(1)2"
2   :stage "entering switch(1)2" :instance i :agent a
3   :ap i "variables" vars :ap (mo) varlocs :do
4     (update-push-aclosure ac :stage "executing 1")
5     (dolist (var vars varlocs)
6       (aset varlocs var (aget a "location" var))))
```

В строке 3 выполняется присваивание параметру `vars` списка переменных, объявленных на верхнем уровне, и инициализация параметра `varlocs` пустым изменяемым объектом.

В строке 4 откладывается в стек выполнение стадии `"executing 1"`, которое вычисляет управляющее выражение оператора `switch`, хранящееся в атрибуте `1`.

После выполнения строк 5-6 параметр `varlocs` хранит отображение переменных из списка `vars` в ячейки памяти, связанные с ними до начала выполнения оператора `switch`. Значение этого параметра возвращается в качестве значения атрибутного замыкания.

Стадия `"executing 1"` определяется следующим образом:

```
1 (aclosure ac :attribute "opsem" :type "switch(1)2"
2   :stage "executing 1" :instance i :value v :do
3     (update-push-aclosure ac
4       :stage "exiting switch(1)2"
```

```
5       :av "variable context" v)
6     (update-push-aclosure ac :stage "executing 2")
7     (update-eval-aclosure ac :instance (aget i 1)))
```

В строке 2 в параметре v сохраняется значение переменной varlocs, вычисленной на предыдущей стадии.

После выполнения строк 3-5 в стек окружения сохраняется стадия "exiting switch(1)2", которая восстанавливает старые ячейки памяти, связанные с переменными, при завершении выполнения оператора switch.

В строке 6 в стек сохраняется стадия "exiting switch(1)2", отвечающая за выполнения тела оператора switch.

В строке 7 вычисляется управляющее выражение оператора switch.

Операционная семантика стадии "executing 2" определяется следующим образом:

```
1 (aclosure ac :attribute "opsem" :type "switch(1)2"
2   :stage "executing 2" :instance i :agent a :value v
3   :ap i 2 sts :do
4   (update-eval-aclosure ac :stage "nomatch"
5     :av "current" 0 :av "bound" (length sts)
6     :av "statements" sts :av "pattern" v))
```

В строке 2 в параметре v сохраняется значение управляющего выражения, вычисленного на предыдущей стадии.

В строке 2 в параметре sts сохраняется список операторов, составляющих тело оператора switch.

Строки 4-6 запускают выполнение стадии "nomatch", которая осуществляет последовательный просмотр операторов списка в случае, если еще не найдено сопоставление с case-меткой. Эта стадия использует параметры "current", "bound", "statements" и "pattern", хранящие позицию текущего оператора, число операторов в списке, список операторов и вычисленное значение управляющего выражения.

Стадия "nomatch" определяется следующим образом:

```
1 (aclosure ac :attribute "opsem" :type "switch(1)2"
2   :stage "nomatch" :ap ac "current" j :ap ac "bound" k
3   :ap ac "statements" sts :ap "pattern" p :match
```

```
4   :v (< j k) T :p (nth j sts) st :do
5     (nmatch
6       :v (is-instance st "case1") T
7       :exit (match :v (= (aget st 1) p) T
8         :do (update-eval-aclosure ac :stage "match"
9           :av "current" (+ j 1))
10        :exit (update-eval-aclosure ac
11          :av "current" (+ j 1)))
12      :v (is-instance st "default") T
13      :exit (update-eval-aclosure ac :stage "match"
14        :av "current" (+ j 1))
15      :do (update-eval-aclosure ac :av "current" (+ j 1))))
```

В строке 4 выполняется проверка перебраны ли все операторы из списка. Если да, то стадия завершается. Также в этой строке параметру `st` присваивается текущий оператор в качестве значения.

В строке 6 рассмотрен случай, когда текущий оператор является `case`-оператором. В этом случае в строке 7 выполняется проверка, совпадает ли метка `case`-оператора со значением управляющего выражения. Если совпадает, то в строках 8-11 выполняется переход к стадии `"match"`, которая продолжает проход по операторам списка, зная, что сопоставление уже произошло.

В строке 12 рассмотрен случай, когда текущий оператор является `default`-оператором. В этом случае также выполняется переход к стадии `"match"`.

В строке 15 рассмотрен случай, когда текущий оператор не является ни `case`-оператором, ни `default`-оператором.

Стадия `"nomatch"` задается следующим атрибутным замыканием:

```
1 (aclosure ac :attribute "opsem" :type "switch(1)2"
2   :stage "match" :ap ac "current" j :ap ac "bound" k
3   :ap ac "statements" sts :match
4   :v (< j k) :do
5     (update-push-aclosure ac :av "current" (+ j 1))
6     (clear-update-eval-aclosure ac :instance (nth j sts)))
```

В строке 6 выполняется текущий оператор, а в строке 5 осуществляется переход к следующему оператору.

И последняя стадия "exiting switch(1)2" определяется следующим образом:

```
1 (aclosure ac :attribute "opsem" :type "switch(1)2"
2   :stage "exiting switch(1)2" :agent a
3   :ap "variable context" varlocs :do
4     (dolist (var (attributes varlocs))
5       (aset a "location" var (aget varlocs var))))
```

В строке 3 в параметре `varlocs` сохраняется старое отображение переменных в связанные с ними ячейки памяти.

При выполнении строк 4-5 эти старые связи переменных и ячеек памяти становятся актуальными и сохраняются в текущем агенте.

Заметим, что при определении операционной семантики оператора `switch` рассматривалось только нормальное последовательное выполнение этого оператора. Это справедливо и для других операторов. Этого достаточно, поскольку распространение (propagation) и обработка исключительных ситуаций обеспечивается операторами, связанными с порождением таких ситуаций (операторами передачи управления, операторами порождения исключений и т. п.).

## 9. Родственные работы

Исследования в области формальной семантики языков программирования имеют длительную историю и охватывают широкий спектр подходов – от классических структурной операционной семантики и операционной семантики малого шага до денотационных, аксиоматических и гибридных формализаций [26, 45]. В последние годы наблюдается устойчивый интерес к развитию формальных моделей семантики промышленных языков программирования, прежде всего языка C и C-подобных языков, что обусловлено их широким применением в системном программировании и критически важных программных компонентах [18, 21, 36].

**Операционная семантика языка C и его подмножеств.** Язык C традиционно рассматривается как один из наиболее сложных объектов для формальной семантики из-за низкоуровневой модели памяти и неопределённого поведения. Формальные семантики C и

его подмножеств (например, Clight в CompCert) стали предметом активного исследования, как в классических работах по механизации семантики [12], так и в современных обзорах и улучшениях семантических моделей [47, 60]. Верифицированный компилятор CompCert является краеугольным камнем исследований корректности компиляции C-программ на промышленном уровне [18, 34] и продолжает развиваться [32, 54]. Дополнительные проекты, такие как Checked C, предлагают более безопасные расширения C с формальными семантическими моделями [1, 14, 15, 22, 35]. Семантики C в виде операционных семейств используются для доказательства корректности оптимизаций и анализа поведения в случае неопределенного поведения [48, 55].

**Формальные семантики C-подобных языков.** Помимо C, значительное число исследований посвящено семантике C-подобных языков. Например, для Rust разработаны исчерпывающие операционные семантики, включая формальные модели владения и заимствования [5, 27, 59]. LLVM IR как объединенное промежуточное представление также изучается с точки зрения формальных семантик [10, 23, 33, 37, 61, 61], что важно для семантики оптимизаций и трансформаций.

**Онтологические и ориентированные на знания подходы к семантике.** Использование онтологий и моделей знания становится всё более распространённым в семантических исследованиях. Онтологии формализуют концепты, отношения и правила в предметной области, что улучшает семантическую интерпретацию сложных систем [19, 24]. Хотя большинство работ по онтологиям фокусируются на семантике естественных языков или интеграции данных, методы онтологического моделирования также применялись для описания программных систем и представления семантики языков программирования [2, 41, 42].

Современные подходы к формальным семантикам взаимодействуют с логикой и построением онтологий [25, 52], обеспечивая основу для семантической интеграции и доказательств верификации.

Некоторые подходы используют методы онтологий совместно с логическими выводами и переносом знаний для динамических систем, что близко к концепции онтологической семантики исполнения [3, 7, 8, 28].

**Атрибутные грамматики и расширенные модели исполнения** Исследования атрибутных грамматик расширили классические синтаксические грамматики, включая семантические атрибуты и контекстно-чувствительные преобразования [16, 40, 50]. Эти модели стали основой для дескриптивных семантик, где семантика операций определяется не только формальными правилами, но и дополнительной структурной информацией о контексте исполнения [13, 40, 50].

Атрибутные грамматики остаются важным инструментом для определения семантики, особенно в расширенных моделях исполнения, где семантика контекста учитывается более гибко [6, 29, 55].

Современные расширения включают поддержку динамического контекста, агентных систем и взаимодействия с внешней средой [13, 51, 53].

**Связь с задачами анализа и верификации программ.** Формальные семантики активно используются в статическом анализе, доказательстве корректности и проверке оптимизаций [4, 9, 11, 17, 31, 38, 39, 43, 49, 57, 58, 62].

Семантика LLVM IR, а также моделей C и Rust, обеспечивает основу для автоматической верификации и анализа системного программного обеспечения [10, 20, 56].

Онтологическая операционная семантика обещает более лёгкую интеграцию с системами логического вывода, поскольку онтологии являются стандартным формализмом для представления знаний и могут напрямую связываться с автоматизированными анализаторами [30, 44, 46].

**Сравнение с настоящей работой.** В отличие от большинства исследований, где семантика операторов передачи управления задаётся в рамках фиксированной модели состояний, настоящая работа предлагает интерпретацию операционной семантики как динамики онтологических моделей, обеспечивая модульность и концептуальную целостность спецификации. Это позволяет естественно выразить нелокальную передачу управления и восстановление контекста исполнения, что остаётся сложной задачей для традиционных подходов.

## 10. Заключение

В данной работе был предложен онтологический подход к заданию операционной семантики операторов передачи управления языка C с использованием предметно-ориенти-

рованного языка ABML. Показано, что системы переходов, традиционно применяемые для определения операционной семантики, естественным образом интерпретируются как дискретные динамические системы и могут быть формализованы в терминах онтологического моделирования.

В рамках работы была построена онтология операторов передачи управления, охватывающая операторы goto, break, continue и return, а также онтологии конструкций, реагирующих на передачу управления, включая помеченные операторы, блоки и оператор switch. Для этих онтологических моделей была задана операционная семантика в виде атрибутных замыканий относительно атрибута opsem, вычисляемых в контексте агентов и окружения.

Существенным результатом является адаптация языка ABML к задачам задания семантики языков программирования. Уточнение понятия атрибутного замыкания, введение стадий вычисления и явное моделирование контекста выполнения позволили выразить сложные аспекты семантики, такие как нелокальная передача управления и восстановление состояния после завершения операторов.

Предложенный подход обладает рядом преимуществ по сравнению с традиционными формализациями. Он обеспечивает модульность спецификаций, возможность повторного использования онтологических моделей, а также естественную расширяемость при добавлении новых конструкций языка. Кроме того, онтологическое представление создает предпосылки для интеграции с инструментами анализа знаний, логического вывода и верификации.

В дальнейшем представляется перспективным расширение разработанной модели на другие конструкции языка C, включая выражения, функции и механизмы обработки исключений, а также применение подхода к другим языкам программирования. Отдельный интерес представляет использование онтологической операционной семантики для анализа свойств программ, построения интерпретаторов и разработки формальных средств верификации.

## Список литературы

1. Achieving safety incrementally with Checked C / Ruef A., Lampropoulos L., Sweet I., Tarditi D., and Hicks M. // International Conference on Principles of Security and Trust / Springer International Publishing Cham. — 2019. — P. 76–98.

2. Adamo G., Villa F. Ontology of Descriptions and Observations for Integrated Modelling (ODO-IM). — 2024.
3. Angele J., Kifer M., Lausen G. Ontologies in F-logic // Handbook on ontologies. — Springer, 2009. — P. 45–70.
4. Appel A. W. Program logics for certified compilers. — Cambridge University Press, 2014.
5. Atkey R. e. a. Semantic Foundations for Rust Ownership // Journal of Functional Programming. — 2022.
6. Author A. Advanced Attribute Grammars: Contextual Semantics // Journal of Formal Languages. — 2021.
7. Babaei Giglou H., D'Souza J., Auer S. LLMs4OL: Large language models for ontology learning // International Semantic Web Conference / Springer. — 2023. — P. 408–427.
8. Balaban M. The F-logic approach for description languages // Annals of Mathematics and Artificial Intelligence. — 1995. — Vol. 15, no. 1. — P. 19–60.
9. Barroso P., Pereira M., Ravara A. Leroy and Blazy Were Right: Their Memory Model Soundness Proof // Verified Software. Theories, Tools and Experiments.: 14th International Conference, VSTTE 2022, Trento, Italy, October 17–18, 2022, Revised Selected Papers / Springer Nature. — 2023. — Vol. 13800. — P. 20.
10. Beck C., Chen H., Zdancewic S. Vellvm: Formalizing the Informal LLVM: (Experience Report) // NASA Formal Methods Symposium / Springer. — 2025. — P. 91–99.
11. Beringer L., Appel A. W. Abstraction and subsumption in modular verification of C programs // Formal Methods in System Design. — 2021. — Vol. 58, no. 1. — P. 322–345.
12. Blazy S., Leroy X. Mechanized Semantics for the Clight Subset of the C Language // Journal of Automated Reasoning. — 2009. — Vol. 43. — P. 263–288.
13. Bridging MDE and AI: a systematic review of domain-specific languages and model-driven practices in AI software systems engineering / Rädler S., Berardinelli L., Winter K., Rahimi A., and Rinderle-Ma S. // Software and Systems Modeling. — 2025. — Vol. 24, no. 2. — P. 445–469.
14. Checked C: Making C safe by extension / Elliott A. S., Ruef A., Hicks M., and Tarditi D. // 2018 IEEE Cybersecurity Development (SecDev) / IEEE. — 2018. — P. 53–60.
15. Checkedcbox: Type directed program partitioning with checked c for incremental spatial memory safety / Li L., Bhattar A., Chang L., Zhu M., and Machiry A. // arXiv preprint arXiv:2302.01811. — 2023.

16. Ciccaglione M., Caliandro P., Pellegrini A. Tahr: The Generative Attribute Grammar Framework // arXiv preprint arXiv:2512.01872. — 2025.
17. Cohen J. M., Wang Q., Appel A. W. Verified erasure correction in Coq with MathComp and VST // International Conference on Computer Aided Verification / Springer. — 2022. — P. 272–292.
18. CompCert: A Formally Verified C Compiler. — `https://compcert.org`. — 2025.
19. The computer science ontology: A comprehensive automatically-generated taxonomy of research areas / Salatino A. A., Thanapalasingam T., Mannocci A., Birukou A., Osborne F., and Motta E. // Data Intelligence. — 2020. — Vol. 2, no. 3. — P. 379–416.
20. Crux, a precise verifier for rust and other languages / Pernsteiner S., Diatchki I. S., Dockins R., Dodds M., Hendrix J., Ravich T., Redmond P., Scott R., and Tomb A. // arXiv preprint arXiv:2410.18280. — 2024.
21. A Formal Semantics of C with Applications : Rep. / Technical Report ; executor: Ellison C., Roșu G. : 2010. — Access mode: `https://iris-project.org/pdfs/2025-icfp-osiris.pdf`.
22. A formal model of Checked C / Li L., Liu Y., Postol D., Lampropoulos L., Van Horn D., and Hicks M. // Journal of Computer Security. — 2023. — Vol. 31, no. 5. — P. 581–614.
23. Formalizing the LLVM intermediate representation for verified program transformations / Zhao J., Nagarakatte S., Martin M. M., and Zdancewic S. // Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. — 2012. — P. 427–440.
24. Guizzardi G., Guarino N. Explanation, semantics, and ontology // Data & Knowledge Engineering. — 2024. — Vol. 153. — P. 102325.
25. Harper R. Semantic Foundations for Programming Languages. — MIT Press, 2023.
26. Hoare C. A. R., He J. Unifying Theories of Programming. — Prentice Hall, 1998.
27. Kan S. e. a. An Executable Operational Semantics for Rust // arXiv preprint arXiv:1804.07608. — 2018. — Access mode: `https://arxiv.org/abs/1804.07608`.
28. Kifer M., Lausen G., Wu J. Logical foundations of object-oriented and frame-based languages // Journal of the ACM (JACM). — 1995. — Vol. 42, no. 4. — P. 741–843.
29. Knuth D. E. Semantics of Context-Free Languages // Mathematical Systems Theory. — 1968.
30. Koopmann P. Explaining Reasoning Results for Description Logic Ontologies // Joint

Proceedings of the 20th and 21st Reasoning Web Summer Schools (RW 2024 & RW 2025) / Schloss Dagstuhl–Leibniz-Zentrum für Informatik. — 2025. — P. 6–1.

31. Krebbers R. A formal C memory model for separation logic // Journal of Automated Reasoning. — 2016. — Vol. 57, no. 4. — P. 319–387.
32. Krebbers R., Leroy X., Wiedijk F. Formal C semantics: CompCert and the C standard // International Conference on Interactive Theorem Proving / Springer. — 2014. — P. 543–548.
33. Lee J. A Validated Semantics for LLVM IR. — PhD thesis, IST UT Lisbon. — 2024. — Access mode: `https://web.ist.utl.pt/nuno.lopes/students/Juneyoung_Lee_PhD.pdf`.
34. Leroy X. Formal certification of a compiler back-end or: programming a compiler with a proof assistant // Conference record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — 2006. — P. 42–54.
35. Leroy X. Formal verification of a realistic compiler // Communications of the ACM. — 2009. — Vol. 52, no. 7. — P. 107–115.
36. Leroy X., Blazy S. From Mechanized Semantics to Verified Compilation: the Clight Semantics of CompCert // FASE 2024. — Springer. — 2024.
37. Li L., Gunter E. L. K-LLVM: a relatively complete semantics of LLVM IR // 34th European Conference on Object-Oriented Programming (ECOOP 2020) / Schloss Dagstuhl–Leibniz-Zentrum für Informatik. — 2020. — P. 7–1.
38. Mansky W. Bringing Iris into the Verified Software Toolchain // arXiv preprint arXiv:2207.06574. — 2022.
39. Mansky W., Du K. An Iris instance for verifying CompCert C programs // Proceedings of the ACM on Programming Languages. — 2024. — Vol. 8, no. POPL. — P. 148–174.
40. Michaelson D., Nadathur G., Van Wyk E. A modular approach to metatheoretic reasoning for extensible languages // ACM Transactions on Programming Languages and Systems. — 2025. — Vol. 47, no. 3. — P. 1–56.
41. Na Nongkhai L., Wang J., Mendori T. Development and evaluation of adaptive learning support system based on ontology of multiple programming languages // Education Sciences. — 2025. — Vol. 15, no. 6. — P. 724.
42. Nongkhai L. N., Wang J., Mendori T. Developing an Ontology of Multiple Programming Languages from the Perspective of Computational Thinking Education. // International Association for Development of the Information Society. — 2022.
43. Park S. H., Pai R., Melham T. A formal CHERI-C semantics for verification // International

Conference on Tools and Algorithms for the Construction and Analysis of Systems / Springer. — 2023. — P. 549–568.

44. Pileggi S. F. Ontology in hybrid intelligence: A concise literature review // Future Internet. — 2024. — Vol. 16, no. 8. — P. 268.
45. Plotkin G. D. A Structural Approach to Operational Semantics. — University of Edinburgh, 1970.
46. Qureshi H. M., Faber W. Evaluating Datalog Tools for Meta-reasoning over OWL 2 QL // Theory and Practice of Logic Programming. — 2024. — Vol. 24, no. 2. — P. 368–393.
47. Ramel D. Modernizing C for Security: the Checked C Initiative // ADTmag. — 2016.
48. Rasband V. e. a. Formalizing Undefined Behavior in C // ACM SIGPLAN Notices. — 2020.
49. RefinedC: automating the foundational verification of C code with refined ownership types / Sammler M., Lepigre R., Krebbers R., Memarian K., Dreyer D., and Garg D. // Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. — 2021. — P. 158–174.
50. Ringo N., Kramer L., Van Wyk E. Nanopass attribute grammars // Proceedings of the 16th ACM SIGPLAN International Conference on Software Language Engineering. — 2023. — P. 70–83.
51. The rise of agentic AI: A review of definitions, frameworks, architectures, applications, evaluation metrics, and challenges / Bandi A., Kongari B., Naguru R., Pasnoor S., and Vilipala S. V. // Future Internet. — 2025. — Vol. 17, no. 9. — P. 404.
52. Roșu G., Serbanuta T. K Framework and Formal Semantics // Journal of Functional Programming. — 2010.
53. A survey of ai agent protocols / Yang Y., Chai H., Song Y., Qi S., Wen M., Li N., Liao J., Hu H., Lin J., Chang G., et al. // arXiv preprint arXiv:2504.16736. — 2025.
54. Thibault J. e. a. SECOMP: Formally Secure Compilation of Compartmentalized C Programs // arXiv preprint arXiv:2401.16277. — 2024. — Access mode: `https://arxiv.org/abs/2401.16277`.
55. Towler C. e. a. Operational Semantics for Low-Level Languages // Journal of Programming Languages. — 2023.
56. Van Oorschot D., Huisman M., Şakar Ö. First steps towards deductive verification of LLVM IR // International Conference on Fundamental Approaches to Software Engineering / Springer. — 2024. — P. 290–303.

57. Vst-a: A foundationally sound annotation verifier / Zhou L., Qin J., Wang Q., Appel A. W., and Cao Q. // Proceedings of the ACM on Programming Languages. — 2024. — Vol. 8, no. POPL. — P. 2069–2098.
58. VST-Floyd: A separation logic tool to verify correctness of C programs / Cao Q., Beringer L., Gruetter S., Dodds J., and Appel A. W. // Journal of Automated Reasoning. — 2018. — Vol. 61, no. 1. — P. 367–422.
59. Wang F. e. a. KRust: A Formal Executable Semantics of Rust // arXiv preprint arXiv:1804.10806. — 2018. — Access mode: https://arxiv.org/abs/1804.10806.
60. Wils S., Jacobs B. Certifying C Program Correctness with Respect to CompCert with VeriFast // arXiv preprint arXiv:2110.11034. — 2021. — Access mode: https://arxiv.org/abs/2110.11034.
61. Zakowski Y. e. a. Modular, Compositional, and Executable Formal Semantics for LLVM IR. — 2021. — Access mode: https://dl.acm.org/doi/10.1145/3473572.
62. Zhao Y., Sanan D. Rely-guarantee Reasoning about Concurrent Memory Management: Correctness, Safety and Security // arXiv preprint arXiv:2309.09997. — 2023.
63. Ануреев И.С. Язык спецификации дискретных динамических систем, ориентированных на знания, структурированные в онтологиях // Системная информатика. — 2025. — no. 29. — P. 137–158.

УДК 004.451.2, 004.82, 004.021, 519.6, 004.42

# Операционная семантика выражений в языке Rust на языке ABML

*Бодин Е.В. (Институт систем информатики СО РАН)*

*Ануреев И.С. (Институт систем информатики СО РАН)*

В статье рассматривается формальное описание операционной семантики выражений языка программирования Rust с использованием предметно-ориентированного языка моделирования ABML. Основное внимание уделяется динамическим аспектам вычислений, включая управление памятью, владение, заимствование и проверку конфликтов доступа на этапе выполнения.

Предлагаемый подход опирается на онтологическое представление синтаксических и семантических сущностей Rust, что позволяет единообразно описывать выражения, блоки и структуры данных как элементы единой вычислительной модели. В отличие от традиционных формализаций, модель явно включает метаданные безопасности, необходимые для воспроизведения механизмов ownership и borrow checking.

Особенностью работы является использование иерархической модели памяти, позволяющей корректно описывать частичное заимствование структур и доступ к их полям. Это обеспечивает более точную динамическую семантику по сравнению с плоскими моделями памяти и демонстрирует соответствие формальных правил реальному поведению программ на Rust.

Полученная операционная семантика является исполняемой и может служить основой для анализа программ, прототипирования интерпретаторов и дальнейших исследований в области формальной верификации языков с управляемой безопасностью памяти.

*Ключевые слова*: операционные семантики, онтологии языков программирования, модели языков программирования, атрибутные замыкания, ABML, Rust, управление памятью, онтологическое моделирование

## 1. Введение

Современные языки программирования системного уровня все чаще ориентируются на строгие гарантии безопасности памяти, которые должны обеспечиваться без значительного снижения производительности. Язык Rust представляет собой один из наиболее успешных примеров такого подхода, сочетая низкоуровневый контроль над ресурсами с

формально заданными правилами владения и заимствования. Эти правила традиционно проверяются на этапе компиляции, однако их точная семантическая интерпретация требует аккуратного формального описания.

Формальная операционная семантика играет ключевую роль в понимании поведения программ, анализе корректности и построении инструментов верификации. Для языков с нетривиальной моделью памяти, таких как Rust, стандартные подходы, основанные на простых состояниях вида «память—окружение», оказываются недостаточными. В частности, они не позволяют напрямую выразить ограничения, связанные с одновременным доступом к данным, жизненным циклом значений и частичным заимствованием составных объектов.

В данной работе предлагается использовать онтологический подход к заданию операционной семантики, реализованный с помощью языка ABML [21]. В рамках этого подхода вычислительное состояние рассматривается как совокупность объектов и их атрибутов, а динамика исполнения описывается через вычисление и обновление этих атрибутов. Такой взгляд позволяет естественным образом интегрировать в модель дополнительные семантические слои, в том числе метаданные, отвечающие за безопасность памяти.

ABML ранее применялся для формального описания семантики языковых конструкций, включая операторы передачи управления в языке C [20]. Эти исследования показали, что онтологическое моделирование обеспечивает модульность, расширяемость и исполняемость семантики. Настоящая статья развивает данный подход применительно к языку Rust, фокусируясь не на управляющих конструкциях, а на выражениях и связанных с ними механизмах работы с памятью.

В статье формализуются базовые синтаксические сущности Rust, включая выражения, объявления переменных и блоки, а также вводится иерархическая модель локаций памяти. Особое внимание уделяется операционной семантике заимствования, разыменования и присваивания, а также алгоритму динамической проверки конфликтов, моделирующему поведение borrow checker. Рассматриваемые примеры демонстрируют, как предложенная модель воспроизводит как корректные сценарии доступа к данным, так и ситуации, приводящие к ошибкам выполнения.

Таким образом, целью работы является построение исполняемой операционной семантики выражений Rust на основе ABML, которая не только отражает ключевые свойства языка, но и может служить фундаментом для дальнейших исследований в области фор-

мальных методов, анализа программ и разработки инструментов поддержки Rust.

Статья организована следующим образом. В разделе 2 вводится онтология выражений и типов данных языка Rust, описываются базовые синтаксические сущности и их семантические роли в вычислительной модели. В разделе 3 рассматриваются модели агентов и окружения, используемые для представления состояния вычислений, включая иерархическую модель памяти и метаданные безопасности. Раздел 4 посвящен формальному заданию операционной семантики выражений Rust на языке ABML, включая объявления переменных, вычисление базовых выражений, присваивание и доступ к полям структур. Также в нем подробно рассматривается операционная семантика заимствования, разыменования и универсальное правило проверки заимствований, моделирующее поведение borrow checker. В разделе 5 приводятся иллюстративные и комплексные примеры выполнения программ, демонстрирующие работу иерархической проверки конфликтов и частичного заимствования структур. Далее следует раздел «Родственные работы», в котором проводится сопоставление предложенного подхода с существующими формализациями семантики Rust и других языков программирования. В заключении подводятся итоги работы и обсуждаются направления дальнейших исследований.

## 2. Онтология выражений и типов данных языка Rust

В данном разделе вводится онтология выражений, констант и типов данных. В онтологическом подходе к спецификации операционной семантики языков программирования [20] онтология конструкций языка Rust описывается набором типов языка ABML, полное описание которого можно найти в [21].

### 2.1. Имена

В этом подразделе описываются типы для разных видов имен, используемых в Rust-программах:

```
(typedef "name" (uniont symbol string))
(typedef "variable" "name")
(typedef "field name" "name")
(typedef "struct name" "name")
```

Таким образом, все имена моделируются лисповскими строками.

## 2.2. Константы и значения

В этом подразделе описываются типы для разных видов констант языка Rust, а также значений, которые могут возвращать выражения на этом языке:

```
(typedef "constant" (uniont "i32 value" "bool value"))

(typedef "i32 value" int)
(typedef "bool value" (enumt "true" "false"))
(typedef "()" (enumt "()"))

(typedef "value" (uniont "constant" "()" "reference"))

(mot "reference" :at "location" "location"
  :at "lifetime" "lifetime")

(typedef "location" (uniont "simple location" "struct location"
  "field location"))
(mot "simple location")
(mot "struct location" :amap "field name" "location")
(mot "field location")

(mot "lifetime")
```

Экземпляры типов `"i32 value"` и `"bool value"` являются моделями значений типов `i32` и `bool` языка Rust. Язык Rust имеет и другие примитивные типы, но мы для простоты в этой статье ограничиваемся только этими двумя. Остальные типы моделируются аналогичным образом.

Тип `"()"` моделирует значение `()` в языке Rust типа `unit`.

Тип `"location"` моделирует локации (адреса, ячейки памяти) в языке Rust. Мы выделяем 3 подтипа локаций – локации, связанные с переменными примитивных типов; локации, связанные с переменными типа структуры и локации, связанные с полями структуры. В языке Rust имеются и другие составные типы помимо структур, например, кортежи, но мы для простоты ограничиваемся только структурами, поскольку моделирование значе-

ний других составных типов делается аналогичным образом.

Тип "lifetime" моделирует ссылки, которые связаны с локациями и имеют время жизни.

Тип "lifetime" определяет значения, которые описывают время жизни для ссылок.

## 2.3. Типы данных

В этом подразделе собраны типы языка ABML, моделирующие типы языка Rust. Для простоты мы ограничиваемся небольшим набором типов, но это набор несложно расширить:

```
(typedef "type" (uniont "i32" "bool" "unit" "struct name"
   "struct type"
  "&T1" "&mutT1"))

(typedef "i32" (enumt "i32"))
(typedef "bool" (enumt "bool"))
(typedef "unit" (enumt "unit"))
(cot "struct type" :amap "field name" "type")
(cot "&T" :at "type" "type")
(cot "&mutT" :at "type" "type")
```

Типы "i32" и "bool" моделируют типы i32 и bool языка Rust.

Тип "unit" моделирует тип unit языка Rust.

Тип "struct type" моделирует типы структур, задавая их поля и типы этих полей.

Типы "&T1" и "&mutT1" моделируют типы для обычных и мутабельных ссылок.

## 2.4. Выражения

Модели выражений языка Rust, рассматриваемые в этой статье, на языке ABML определяются следующим набором типов:

```
(typedef "expression" (uniont "variable" "constant" "1.2" "&1"
  "&mut1" "*1" "1+2"))

(mot "1.2" :at 1 "expression" :at 2 "field name")
(mot "&1" :at 1 "expression")
```

```
(mot "&mut1" :at 1 "expression")
(mot "*1" :at 1 "expression")
(mot "1+2" :at 1 "expression" :at 2 "expression")

(mot "1=2" :at 1 "identifier" :at 2 "expression"
  :at "type" "type")
(mot "let1:=2" :at 1 "identifier" :at 2 "expression"
  :at "type" "type")
(mot "let mut1:=2" :at 1 "identifier" :at 2 "expression"
  :at "type" "type")
(mot "{1}" :at 1 (listt "expression"))
```

Они моделируют небольшой, но достаточный набор выражений для описания основных концепций операционной семантики языка Rust. Заметим, что для моделей выражений `assign`, `let` и `let mut` языка Rust, мы считаем, что тип атрибута `1` известен и хранится в атрибуте `"type"`.

## 3. Модели агентов и окружения

Состояние программы в ABML моделируется агентом, который оперирует знаниями о памяти и окружении. Для языка Rust множество агентов определяется следующим типом:

```
(mot "agent"
  :at "location" (cot :amap "variable" "location")
  :at "mutability" (cot :amap "variable"
    (enumt "mutable" "immutable"))
  :at "location value" (cot :amap "location" "value")
  :at "location type" (cot :amap "location" "type")
  :at "borrows" (cot :amap "location" (listt "borrow"))
  :at "lifetimes" (cot :amap "lifetime" (listt "location"))
  :at "value" "value"
```

Атрибут `"location"` связывает переменные программы с локациями.

Атрибут `"mutability"` определяет, является ли эта связь мутабельной или нет.

Атрибуты `"location value"` и `"location type"` задают значения, хранящиеся в лока-

циях и типы этих значений, соответственно.

Атрибут "borrows" определяет стеки заимствований для локаций.

Атрибут "lifetimes" описывает, как локации распределяются по времени жизни.

Атрибут "value" – это стандартный атрибут языка ABML, который хранит последнее вычисленное значение.

Заимствования и время жизни определяются следующим образом:

```
(mo "borrow" :at "lifetime" "lifetime"
  :at "kind" (enumt "free", "shared", "unique"))

(mo "lifetime")
```

## 4. Операционная семантика моделей выражений

В данном разделе задается исполняемая семантика конструкций Rust в виде атрибутных замыканий [21]. В отличие от традиционных интерпретаторов, семантика на языке ABML описывает не просто изменение значений, а трансформацию базы знаний агента, включая обновление метаданных безопасности.

Семантика Rust разбивается на семантику мест (возвращает локацию), семантику r-значений (возвращает значение) и семантику операторов (изменяет содержимое агента), которые задаются атрибутными замыканиями для атрибутов "place", "rvalue" и "statement", соответственно.

### 4.1. Семантика мест

Семантика мест определяется для выражений типов "variable", "1.2" и "*1".

Для моделей переменных семантика определяется следующим образом:

```
(aclosure ac :attribute "place" :type "variable" :do
  (update-push-aclosure ac :av "stage" "returning value")
  (update-push-aclosure ac :av "stage" "checking location"))

(aclosure ac :attribute "place" :type "variable"
  :stage "checking location" :instance i :agent a
  :ap a (aseq "location" i) loc :match
    :v (not (null loc)) T
```

```
    :do loc
    :exit (co "stop next aclosure" :av "type" "error" :av
      "aclosure" ac))

(aclosure ac :attribute "place" :type "variable"
  :stage "returning value" :value loc :do loc)
```

Таким образом, возвращается локация, связанная с переменной, в том случае, если такая связь есть. В противном случае, выполнение программы останавливается и выдается ошибка.

Семантика операции доступа к полю структуры задаются аналогичным образом:

```
(aclosure ac :attribute "place" :type "1.2" :do
  (update-push-aclosure ac :av "stage" "returning value")
  (update-push-aclosure ac :av "stage" "checking field location")
  (update-push-aclosure ac :av "stage" "evaluating 1")

(aclosure ac :attribute "place" :type "1.2" :stage "evaluating 1"
  :instance i :ap i 1 v1 :do
    (clear-update-eval-aclosure ac :attribute "place"
      :instance v1)))

(aclosure ac :attribute "place" :type "1.2"
  :stage "checking field location" :value v1 :agent a :instance i
  :ap i 2 v2 :ap v1 (aseq "fields" v2) loc :match
    :v (not (null loc)) T
    :do loc
    :exit (co "stop next aclosure" :av "type" "error" :av
      "aclosure" ac))

(aclosure ac :attribute "place" :type "1.2"
  :stage "returning value" :value loc :do loc)
```

Семантика операции * определяется следующими замыканиями:

```
(aclosure ac :attribute "place" :type "*1" :do
  (update-push-aclosure ac :av "stage" "returning value")
  (update-push-aclosure ac :av "stage" "evaluating 1"))

(aclosure ac :attribute "place" :type "*1" :stage "evaluating 1"
  :instance i :ap i 1 v1 :do
    (clear-update-eval-aclosure ac :attribute "rvalue"
      :instance v1))

(aclosure ac :attribute "place" :type "*1"
  :stage "returning value" :agent a :value ref
  :ap ref "location" loc :ap a (aseq "location value" loc) v :do
    v)
```

## 4.2. Семантика r-значений

Семантика r-значений определяется для выражений типов "value", "place", "&1", "&mut1", "&*1", "&mut*1", "*1" и "1+2".

Тип "place" определяется как объединение следующих типов:

```
(typedef "place" (uniont "variable" "1.2"))
```

Семантика значений задается следующим образом:

```
(aclosure ac :attribute "rvalue" :type "value" :do
  (update-push-aclosure ac :av "stage" "returning value")

(aclosure ac :attribute "rvalue" :type "value"
  :stage "returning value" :instance i :do i)
```

Следующие атрибутные замыкания задают семантику типа "place":

```
(aclosure ac :attribute "rvalue" :type "place" :do
  (update-push-aclosure ac :av "stage" "returning value")
  (update-push-aclosure ac :av "stage" "checking location")
  (update-push-aclosure ac :av "stage" "evaluating location"))
```

```
5
6  (aclosure ac :attribute "rvalue" :type "place"
7    :stage "evaluating location" :do
8      (update-eval-aclosure ac :attribute "place"))
9
10 (aclosure ac :attribute "rvalue" :type "place"
11   :stage "checking location" :agent a :value loc
12   :ap a (aseq "borrows" loc) st :ap (car st) "kind" kd :match
13     :av (or (equal kd "free") (equal kd "shared")
14       (equal kd "unique"))
15     :do loc
16     :exit (co "stop next aclosure" :av "type" "error" :av
         "aclosure" ac))
17
18 (aclosure ac :attribute "rvalue" :type "place"
19   :stage "returning value" :agent a :value loc
20   :ap a (aseq "location value" loc) v :do v)
```

Для операции &p семантика определяется следующим образом:

```
1  (aclosure ac :attribute "rvalue" :type "&1" :do
2    (update-push-aclosure ac :av "stage" "returning value")
3    (update-push-aclosure ac :av "stage" "checking location")
4    (update-push-aclosure ac :av "stage" "evaluating location")
5
6  (aclosure ac :attribute "rvalue" :type "&1"
7    :stage "evaluating location" :instance i :ap i 1 v :do
8      (clear-update-eval-aclosure ac :attribute "place"
9        :instance v)))
10
11 (aclosure ac :attribute "rvalue" :type "&1"
12   :stage "checking location" :agent a :value loc
13   :ap a (aseq "borrows" loc) st :ap (car st) "kind" kd :match
```

```
    :av (or (equal kd "free") (equal kd "shared"))
    :do loc
    :exit (co "stop next aclosure" :av "type" "error" :av
      "aclosure" ac)))

(aclosure ac :attribute "rvalue" :type "&1"
  :stage "returning value" :agent a :value loc
  :v (mo "lifetime") lt :ap a (aseq "borrows" loc) bor :do
    (aset a
      :av (aseq "borrows" loc)
        (cons (mo "borrow" :av "lifetime" lt :av "kind" "shared")
          (aseq a "borrows" loc))
      :av (aseq "lifetimes" lt) (list loc))
    (co "reference" :av "location" loc :av "lifetime" lt))
```

Условием выполнимости этой операции является тот факт, что локация для места `p` не должна быть эксклюзивной. Заметим, что мы не утверждаем, что она должна быть свободной или разделяемой, поскольку хотим иметь расширяемую модель.

Семантика для операции `&mut p` определяется аналогичным образом:

```
(aclosure ac :attribute "rvalue" :type "&mut1" :do
  (update-push-aclosure ac :av "stage" "returning value")
  (update-push-aclosure ac :av "stage" "checking location")
  (update-push-aclosure ac :av "stage" "evaluating location")

(aclosure ac :attribute "rvalue" :type "&mut1"
  :stage "evaluating location" :instance i :ap i 1 v :do
    (clear-update-eval-aclosure ac :attribute "place"
      :instance v)))

(aclosure ac :attribute "rvalue" :type "&mut1"
  :stage "checking location" :agent a :value loc :instance i
  :ap a (aseq "borrows" loc) st :ap (car st) "kind" kd :match
```

```
    :av (and (or (equal kd "free") (equal kd "unique"))
      (clear-update-eval-aclosure ac :attribute "mutable"
      :instance (aget i 1))
    :do loc
    :exit (co "stop next aclosure" :av "type" "error" :av
       "aclosure" ac)))

(aclosure ac :attribute "rvalue" :type "&mut1"
  :stage "returning value" :agent a :value loc
  :v (mo "lifetime") lt :ap a (aseq "borrows" loc) bor :do
    (aset a
      :av (aseq "borrows" loc)
        (cons (mo "borrow" :av "lifetime" lt :av "kind" "unique")
          (aseq a "borrows" loc))
      :av (aseq "lifetimes" lt) (list loc))
    (co "reference" :av "location" loc :av "lifetime" lt))
```

Только условия выполнимости другие: локация для места `p` должна быть эксклюзивной, а само место мутабельным. Мутабельность места определяется набором атибутных замыканий для атрибута `"mutable"`.

Для операции `&*p` семантика определяется следующим образом:

```
(aclosure ac :attribute "rvalue" :type "&*1" :do
  (update-push-aclosure ac :av "stage" "returning value")
  (update-push-aclosure ac :av "stage" "checking location")
  (update-push-aclosure ac :av "stage" "evaluating location")

(aclosure ac :attribute "rvalue" :type "&*1"
  :stage "evaluating location" :instance i :ap i 1 v :do
    (clear-update-eval-aclosure ac :attribute "place"
      :instance v)))

(aclosure ac :attribute "rvalue" :type "&*1"
```

```
  :stage "checking location" :agent a :value loc :instance i
  :ap a (aseq "borrows" loc) st :ap (car st) "kind" kd :match
    :av (is-instance (aget i 1) "&mutT1")
    :do loc
    :exit (co "stop next aclosure" :av "type" "error" :av
      "aclosure" ac)))

(aclosure ac :attribute "rvalue" :type "&*1"
  :stage "returning value" :agent a :value loc
  :v (mo "lifetime") lt :ap a (aseq "borrows" loc) bor :do
    (aset a
      :av (aseq "borrows" loc)
        (cons (mo "borrow" :av "lifetime" lt :av "kind" "shared")
          (aseq a "borrows" loc))
      :av (aseq "lifetimes" lt) (list loc))
    (co "reference" :av "location" loc :av "lifetime" lt))
```

Для операции `&mut*p` семантика определяется аналогичным образом:

```
(aclosure ac :attribute "rvalue" :type "&mut*1" :do
  (update-push-aclosure ac :av "stage" "returning value")
  (update-push-aclosure ac :av "stage" "checking location")
  (update-push-aclosure ac :av "stage" "evaluating location")

(aclosure ac :attribute "rvalue" :type "&mut*1"
  :stage "evaluating location" :instance i :ap i 1 v :do
    (clear-update-eval-aclosure ac :attribute "place"
      :instance v)))

(aclosure ac :attribute "rvalue" :type "&mut*1"
  :stage "checking location" :agent a :value loc :instance i
  :ap a (aseq "borrows" loc) st :ap (car st) "kind" kd :match
    :av (and (equal kd "unique")
```

```
      (is-instance (aget i 1) "&mutT1"))
    :do loc
    :exit (co "stop next aclosure" :av "type" "error" :av
      "aclosure" ac)))

(aclosure ac :attribute "rvalue" :type "&mut*1"
  :stage "returning value" :agent a :value loc
  :v (mo "lifetime") lt :ap a (aseq "borrows" loc) bor :do
    (aset a
      :av (aseq "borrows" loc)
        (cons (mo "borrow" :av "lifetime" lt :av "kind" "unique")
          (aseq a "borrows" loc))
      :av (aseq "lifetimes" lt) (list loc))
    (co "reference" :av "location" loc :av "lifetime" lt))
```

Семантика операции * определяется следующими замыканиями:

```
(aclosure ac :attribute "rvalue" :type "*1" :do
  (update-push-aclosure ac :av "stage" "returning value")
  (update-push-aclosure ac :av "stage" "evaluating 1"))

(aclosure ac :attribute "rvalue" :type "*1" :stage "evaluating 1"
  :instance i :ap i 1 v1 :do
    (clear-update-eval-aclosure ac :attribute "rvalue"
      :instance v1))

(aclosure ac :attribute "rvalue" :type "*1"
  :stage "returning value" :agent a :value ref
  :ap ref "location" loc :ap a (aseq "location value" loc) v :do
    v)
```

Операция + определяется следующим образом:

```
(aclosure ac :attribute "rvalue" :type "1+2" :do
  (update-push-aclosure ac :av "stage" "returning value")
```

```
  (update-push-aclosure ac :av "stage" "evaluating 2")
  (update-push-aclosure ac :av "stage" "evaluating 1")

(aclosure ac :attribute "rvalue" :type "1+2"
  :stage "evaluating 1" :instance i :ap i 1 v1 :do
    (clear-update-eval-aclosure ac :attribute "rvalue"
      :instance v1)))

(aclosure ac :attribute "rvalue" :type "1+2"
 :stage "evaluating 2" :value v1 :instance i :ap i 2 v2 :do
    (clear-update-eval-aclosure ac :attribute "rvalue"
      :instance v2 :av "v1" v1)))

(aclosure ac :attribute "rvalue" :type "1+2"
  :stage "returning value" :ap "v1" v1 :value v2 :do (+ v1 v2))
```

Подобным образом определяются и другие бинарные операции.

## 4.3. Семантика операторов

Семантика операторов определяется для выражений типов "1=2", "let1=2", "letmut1=2" и "{1}". При определении семантики для этих выражений характерными являются стадии "updating agent" и "checking location". Первая определяет, как модифицируется агент, а вторая – условия выполнимости операции.

Операция "1=2" определяется следующим образом:

```
(aclosure ac :attribute "statement" :type "1=2" :do
  (update-push-aclosure ac :av "stage" "updating agent")
  (update-push-aclosure ac :av "stage" "evaluating 2")
  (update-push-aclosure ac :av "stage" "checking location")
  (update-push-aclosure ac :av "stage" "evaluating 1")

(aclosure ac :attribute "statement" :type "1=2"
  :stage "evaluating 1" :instance i :ap i 1 v1 :do
```

```
    (clear-update-eval-aclosure ac :attribute "place"
      :instance v1)))

(aclosure ac :attribute "statement" :type "1=2"
  :stage "checking location" :value loc :agent a
  :ap a (aseq "borrows" loc) st :ap (car st) "kind" kd :match
    :av (or (equal kd "free") (equal kd "unique"))
    :do loc
    :exit (co "stop next aclosure" :av "type" "error" :av
       "aclosure" ac))

(aclosure ac :attribute "statement" :type "1=2"
 :stage "evaluating 2" :value v1 :instance i :ap i 2 v2 :do
    (clear-update-eval-aclosure ac :attribute "rvalue"
      :instance v2 :av "v1" v1)))

(aclosure ac :attribute "statement" :type "1=2"
  :stage "updating agent" :ap "v1" v1 :value v2 :do
    (aset a "location value" v1 v2))
```

Условие выполнимости операции требует, чтобы локация, являющаяся результатом вычисления значения атрибута 1, была или свободной, или эксклюзивной, а место 1 мутабельным.

Семантика операции "let1=2" имеет вид:

```
(aclosure ac :attribute "statement" :type "let1=2" :do
  (update-push-aclosure ac :av "stage" "updating agent")
  (update-push-aclosure ac :av "stage" "evaluating 2"))

(aclosure ac :attribute "statement" :type "let1=2"
 :stage "evaluating 2" :value v1 :instance i :ap i 2 v2 :do
    (clear-update-eval-aclosure ac :attribute "rvalue"
      :instance v2))
```

```
(aclosure ac :attribute "statement" :type "let1=2"
  :stage "updating agent" :instance i :ap i 1 x :value v2
  :v (mo "simple location") loc :do
    (aset a :av (aseq "location" x) loc
      :av (aseq "mutability" x) "immutable"
      :av (aseq "location value" loc) v2
      :av (aseq "borrows" loc)
        (list (mo "borrow" :av "kind" "free"))))
```

Для простоты, мы рассмотрели только случай, когда переменная, являющаяся значением атрибута 1 имеет примитивный тип. Случай структуры определяется аналогичным образом.

Операция "letmut1=2" определяется аналогичным образом:

```
(aclosure ac :attribute "statement" :type "letmut1=2" :do
  (update-push-aclosure ac :av "stage" "updating agent")
  (update-push-aclosure ac :av "stage" "evaluating 2"))

(aclosure ac :attribute "statement" :type "letmut1=2"
 :stage "evaluating 2" :value v1 :instance i :ap i 2 v2 :do
    (clear-update-eval-aclosure ac :attribute "rvalue"
      :instance v2))

(aclosure ac :attribute "statement" :type "letmut1=2"
  :stage "updating agent" :instance i :ap i 1 x :value v2
  :v (mo "simple location") loc :do
    (aset a :av (aseq "location" x) loc
      :av (aseq "mutability" x) "mutable"
      :av (aseq "location value" loc) v2
      :av (aseq "borrows" loc)
        (list (mo "borrow" :av "kind" "free"))))
```

Отличается только модификация агента.

Особенностью семантики блока является его возможность определять локальные времена жизни и исключать их при выходе из блока:

```
(aclosure ac :attribute "statement" :type "{1}" :agent a
  :ap a "lifetimes" lfs :do
    (update-push-aclosure ac :av "stage" "restoring lifetime"
      :av "lifetime list" (attributes lfs))
    (update-push-aclosure ac :av "stage" "evaluating 1"))

(aclosure ac :attribute "statement" :type "{1}"
  :stage "evaluating 1" :instance i :ap i 1 stl :do
    (update-eval-aclosure ac :stage "body iteration"
      :av "current" 0 :av "statements" stl
      :av "bound" (length stl)))

(aclosure ac :attribute "statement" :type "{1}"
  :stage "body iteration" :ap "current" k :ap "statements" stl
  :ap "bound" n :match
    :v (< k n) T
    :do
      (update-eval-aclosure ac :av "current" (+ k 1))
      (clear-update-eval-aclosure ac :instance (nth k stl)))

(aclosure ac :attribute "statement" :type "{1}"
  :stage "restoring lifetime" :agent a :ap "lifetime list" lfl
  :match :v (mo) lfs
    (dolist (lf lfl) (aset lfs lf (aget a "lifetimes" lf)))
    (aset a "lifetimes" lfs)
```

## 5. Верификация правил безопасности на примерах

В данном разделе рассматривается работа интерпретатора ABML на классических сценариях языка Rust. Особое внимание уделяется тому, как динамическая семантика агента воспроизводит статические проверки компилятора, обеспечивая безопасность работы с па-

мятью.

## 5.1. Конфликты заимствования

Основная концепция безопасности Rust заключается в запрете одновременного существования разделяемого доступа (чтения) и возможности модификации данных. Это предотвращает неопределенное поведение и гонки данных. Рассмотрим пример, нарушающий эти правила:

```
let mut x = 5;
let y = &x;        // Shared borrow (разделяемое заимствование)
*x = 10;           // ОШИБКА: x уже заимствован переменной y
```

Выполним интерпретацию этих операций в модели на ABML.

Первая инструкция устанавливает следующие связи в атрибутах агента:

```
loc1 := (mo "simple location")

(aget a "location" x) = loc1
(aget a "mutability" x) = "mutable"
(aget a "location value" loc1) = 5
(aget a "borrows" loc1) = (list (mo "borrow" :av "kind" "free"))
```

где `loc1` – новая локация.

Вторая инструкция модифицирует агента следующим образом:

```
loc1 := (mo "simple location")
loc2 := (mo "simple location")
lt1 := (mo "lifetime")

(aget a "location" x) = loc1
(aget a "location" y) = loc2
(aget a "mutability" x) = "mutable"
(aget a "mutability" y) = "immutable"
(aget a "location value" loc1) = 5
(aget a "location value" loc2) =
  (co "reference" :av "location" loc1 :av "lifetime" lt1)
```

```
(aget a "borrows" loc1) =
  (list (mo "borrow" :av "lifetime" lt1 :av "kind" "shared")
    (mo "borrow" :av "kind" "free"))
(aget a "borrows" loc2) = (list (mo "borrow" :av "kind" "free"))
(aget a "lifetimes" lt1) = (list loc1)
```

где `loc2` – новая локация, и `lt1` – новое время жизни.

Третья инструкция начинает с вычисления `*x`. В этом случае `x` вычисляется как `rvalue` и возвращает число `5`, а вычисление `*x` требует ссылки. Поэтому вычисление операционной семантики завершается с ошибкой.

## 5.2. Частичное заимствование структур

Одним из ключевых преимуществ онтологического подхода является возможность точного моделирования частичного доступа к компонентам сложных данных. В отличие от систем с «плоской» памятью, блокирующих объект целиком, иерархическая модель ABML позволяет агенту анализировать доступ на уровне отдельных полей.

```
let mut point = Point { x: 1, y: 2 };
let r = &point.x;   // Заимствуем только поле .x
point.y = 10;       // OK: поле .y доступно для записи
```

В данном примере поле `x` заимствовано для чтения, что запрещает его изменение. Однако поле `y` остается свободным и может быть изменено.

Выполним интерпретацию этих операций в модели на ABML.

Первая инструкция устанавливает следующие связи в атрибутах агента:

```
locx := (mo "field location")
locy := (mo "field location")
locpoint := (mo "struct location" :av "x" locx :av "y" locy)

(aget a "location" point) = locpoint
(aget a "mutability" point) = "mutable"
(aget a "location value" locx) = 1
(aget a "location value" locy) = 2
(aget a "location value" loc_point) =
```

```
  (mo "struct location" :av "x" locx :av "y" locy)
(aget a "borrows" locx) = (list (mo "borrow" :av "kind" "free"))
(aget a "borrows" locy) = (list (mo "borrow" :av "kind" "free"))
(aget a "borrows" locpoint) = (list (mo "borrow" :av "kind"
  "free"))
```

Вторая инструкция модифицирует агента следующим образом:

```
locx := (mo "field location")
locy := (mo "field location")
loc_point := (mo "struct location" :av "x" locx :av "y" locy)
locr := (mo "field location")
lt1 := (mo "lifetime")

(aget a "location" point) = locpoint
(aget a "location" r) = locr
(aget a "mutability" point) = "mutable"
(aget a "mutability" r) = "immutable"
(aget a "location value" locx) = 1
(aget a "location value" locy) = 2
(aget a "location value" loc_point) =
  (mo "struct location" :av "x" locx :av "y" locy)
(aget a "location value" locr) =
  (co "reference" :av "location" locx :av "lifetime" lt1)
(aget a "borrows" locx) =
  (list (mo "borrow" :av "lifetime" lt1 :av "kind" "shared")
    (mo "borrow" :av "kind" "free"))
(aget a "borrows" locy) = (list (mo "borrow" :av "kind" "free"))
(aget a "borrows" locpoint) = (list (mo "borrow" :av "kind"
  "free"))
(aget a "lifetimes" lt1) = (list locx)
```

Третья инструкция также выполняется, так как заимствование наложено только на `locx`, а не на `locpoint` целиком:

```
locx := (mo "field location")
locy := (mo "field location")
loc_point := (mo "struct location" :av "x" locx :av "y" locy)
locr := (mo "field location")
lt1 := (mo "lifetime")

(aget a "location" point) = locpoint
(aget a "location" r) = locr
(aget a "mutability" point) = "mutable"
(aget a "mutability" r) = "immutable"
(aget a "location value" locx) = 1
(aget a "location value" locy) = 10
(aget a "location value" loc_point) =
  (mo "struct location" :av "x" locx :av "y" locy)
(aget a "location value" locr) =
  (co "reference" :av "location" locx :av "lifetime" lt1)
(aget a "borrows" locx) =
  (list (mo "borrow" :av "lifetime" lt1 :av "kind" "shared")
    (mo "borrow" :av "kind" "free"))
(aget a "borrows" locy) = (list (mo "borrow" :av "kind" "free"))
(aget a "borrows" locpoint) = (list (mo "borrow" :av "kind"
  "free"))
(aget a "lifetimes" lt1) = (list locx)
```

Этот пример наглядно демонстрирует точность онтологического моделирования: агент «понимает», что заимствование части объекта не эквивалентно блокировке всего объекта, что полностью соответствует семантике разделенного заимствования (split borrowing) в Rust.

## 6. Родственные работы

Исследования в области формальной семантики языков программирования, и в особенности языка Rust, в последние годы привлекают значительное внимание научного сооб-

щества. Это связано с нетривиальной моделью памяти Rust, основанной на концепциях владения, заимствования и строгих гарантиях отсутствия гонок данных. В результате появилось множество работ, направленных на формализацию как статических, так и динамических аспектов языка.

Одной из первых работ, целенаправленно описывающих динамическое поведение Rust, является исполняемая операционная семантика RustSEM [3]. В данной работе авторы предлагают формальную модель исполнения программ Rust, в которой явно представлены механизмы владения и заимствования. Семантика задается в виде системы переходов состояний и ориентирована на воспроизведение поведения реальных программ, включая ситуации, приводящие к ошибкам доступа к памяти. Подход RustSEM демонстрирует возможность динамической проверки корректности работы с заимствованиями, однако модель в значительной степени опирается на плоское представление памяти.

Схожую цель преследует работа KRust [11], в которой формальная семантика Rust реализована в рамках K-фреймворка. Использование K позволяет автоматически получать исполняемый интерпретатор и инструменты анализа на основе формального описания семантики. Авторы показывают, что предложенная модель корректно воспроизводит основные элементы языка, включая перенос владения (move), заимствования и мутабельность. Семантика KRust была сопоставлена с тестами официального компилятора Rust, что подтверждает ее практическую применимость.

Отдельное направление исследований связано с формализацией заимствований через символические модели. В работах по проверке корректности заимствований посредством символьной семантики [6, 7] предлагается формализм LLBC (Low-Level Borrow Calculus), который служит промежуточным уровнем между высокоуровневым Rust и низкоуровневыми моделями памяти. Авторы доказывают корректность символьной семантики по отношению к операционной, что позволяет использовать модель для формальной верификации свойств программ, связанных с безопасностью памяти.

Дальнейшим развитием этого направления является инструмент Aeneas [8], ориентированный на верификацию программ Rust путем перевода в функциональные представления. В данной работе операционная семантика Rust редуцируется к чистой семантике, в которой управление памятью и адресами заменяется абстрактными понятиями владения и займов. Такой подход облегчает доказательство функциональной корректности, но абстрагируется от многих деталей реального исполнения.

Фундаментальной работой в области формального обоснования Rust является проект RustBelt [15]. В нем авторы вводят формальный язык $\lambda$Rust и задают его операционную семантику, что позволяет строго доказать безопасность ключевых механизмов Rust, включая отсутствие использования после освобождения (use-after-free) и гонок данных. В отличие от исполняемых семантик, RustBelt ориентирован прежде всего на доказательство теоретических свойств языка, а не на моделирование конкретных сценариев выполнения программ. В RustBelt предлагается логическая семантика для подмножества Rust и доказываются ключевые свойства безопасности памяти [10, 12, 15]. Хотя исходная версия RustBelt была опубликована ранее, последующие работы существенно расширили этот подход, включая поддержку relaxed memory и unsafe-кода [1, 12, 13].

Развитие RustBelt привело к созданию RustHornBelt и RefinedRust, которые объединяют логические и типовые методы верификации и обеспечивают высокую степень автоматизации доказательств [9, 14, 16]. Эти работы демонстрируют, как операционная семантика Rust может быть связана с логическими моделями и доказательными системами.

Параллельно развиваются практико-ориентированные средства верификации, такие как Verus и Flux, использующие расширенные типовые системы и SMT-решатели для доказательства свойств программ [5, 18, 19]. Эти инструменты опираются на формальные семантические модели Rust, хотя и не всегда задают их явно в операционной форме.

Отдельный класс исследований посвящен aliasing-моделям и динамической семантике заимствований, включая Stacked Borrows и его расширения, которые уточняют допустимые сценарии доступа к памяти во время выполнения [4, 17]. Эти модели оказывают существенное влияние на современные формализации Rust и интерпретацию unsafe-кода.

Существуют также работы, фокусирующиеся на концептуальном и когнитивном анализе модели владения Rust. Так, в [2] предлагается концептуальная модель ownership-типов, которая формализует интуитивные представления о владении и заимствовании и сопоставляет их с реальными правилами языка. Хотя данная работа не задает операционную семантику напрямую, она вносит существенный вклад в понимание связи между статическими и динамическими аспектами Rust.

На этом фоне предложенная в настоящей статье операционная семантика выражений Rust на языке ABML занимает промежуточное положение между теоретическими и практико-ориентированными подходами. В отличие от большинства существующих моделей, она использует онтологическое представление вычислительного состояния и иерар-

хическую модель памяти, что позволяет естественно выразить частичное заимствование и динамическую проверку конфликтов доступа. Таким образом, работа дополняет существующие исследования, предлагая альтернативный, онтологически обогащенный способ формализации семантики Rust.

## 7. Заключение

В статье была представлена операционная семантика выражений языка Rust, формализованная с использованием онтологического и атрибутно-ориентированного подхода, реализованного в языке ABML. Предложенная модель ориентирована на точное воспроизведение динамического поведения программ с учетом ключевых гарантий безопасности памяти, характерных для Rust.

Одним из основных результатов работы является введение иерархической модели памяти, в которой локации могут представлять как целые структуры, так и их отдельные поля. Такая организация памяти позволяет корректно моделировать частичное заимствование и отражает реальные правила доступа к данным, применяемые в языке Rust. В сочетании с метаданными живучести, мутабельности и активных заимствований эта модель обеспечивает строгую динамическую проверку конфликтов.

Разработанная операционная семантика описана в виде исполняемых правил, что отличает ее от чисто декларативных формализаций. Это делает возможным использование модели не только для теоретического анализа, но и для экспериментального исполнения программ, трассировки вычислений и исследования граничных случаев поведения механизмов ownership и borrow checking.

Сравнение с предыдущими работами, использующими ABML для описания семантики других языков и конструкций, показывает универсальность выбранного подхода. Онтологическое представление синтаксических и семантических сущностей позволяет постепенно расширять модель, добавляя новые конструкции языка Rust без радикального пересмотра уже существующих определений.

В перспективе предложенная семантика может быть расширена для поддержки более сложных элементов языка, таких как функции, замыкания, обобщенные типы и параллельные вычисления. Кроме того, она может служить основой для построения формальных инструментов анализа и верификации программ на Rust, а также для сопоставления динамической семантики с результатами статического анализа компилятора.

В целом, работа демонстрирует, что онтологический и атрибутно-ориентированный подход в сочетании с ABML является эффективным средством формализации языков программирования с развитой моделью безопасности памяти и открывает новые возможности для исследований в области формальных семантик.

## Список литературы

1. Batty M. J. The C11 and C++ 11 concurrency model : Ph. D. thesis ; University of Cambridge, UK. — 2015.
2. Crichton W., Gray G., Krishnamurthi S. A grounded conceptual model for ownership types in rust // Proceedings of the ACM on Programming Languages. — 2023. — Vol. 7, no. OOPSLA2. — P. 1224–1252.
3. An Executable Operational Semantics for Rust with the Formalization of Ownership and Borrowing / Kan S., Chen Z., Sanan D., Lin S.-W., and Liu Y. // arXiv preprint arXiv:1804.07608. — 2018.
4. Exploring C semantics and pointer provenance / Memarian K., Gomes V. B., Davis B., Kell S., Richardson A., Watson R. N., and Sewell P. // Proceedings of the ACM on Programming Languages. — 2019. — Vol. 3, no. POPL. — P. 1–32.
5. Flux: Liquid types for rust / Lehmann N., Geller A. T., Vazou N., and Jhala R. // Proceedings of the ACM on Programming Languages. — 2023. — Vol. 7, no. PLDI. — P. 1533–1557.
6. Ho S., Fromherz A., Protzenko J. Sound Borrow-Checking for Rust via Symbolic Semantics // Proceedings of the ACM on Programming Languages. — 2024. — Vol. 8, no. ICFP. — P. 426–454.
7. Ho S., Fromherz A., Protzenko J. Sound Borrow-Checking for Rust via Symbolic Semantics (Long Version) // arXiv preprint arXiv:2404.02680. — 2024.
8. Ho S., Protzenko J. Aeneas: Rust verification by functional translation // Proceedings of the ACM on Programming Languages. — 2022. — Vol. 6, no. ICFP. — P. 711–741.
9. Iris from the ground up / Jung R., Krebbers R., Jourdan J.-H., Bizjak A., Birkedal L., and Dreyer D. // Submitted to JFP. — 2017.
10. Jung R. Understanding and evolving the Rust programming language : Phd dissertation ; Saarländische Universitäts-und Landesbibliothek. — 2020.

11. Krust: A formal executable semantics of rust / Wang F., Song F., Zhang M., Zhu X., and Zhang J. // 2018 International Symposium on Theoretical Aspects of Software Engineering (TASE) / IEEE. — 2018. — P. 44–51.
12. MoSeL: A general, extensible modal framework for interactive proofs in separation logic / Krebbers R., Jourdan J.-H., Jung R., Tassarotti J., Kaiser J.-O., Timany A., Charguéraud A., and Dreyer D. // Proceedings of the ACM on Programming Languages. — 2018. — Vol. 2, no. ICFP. — P. 1–30.
13. A promising semantics for relaxed-memory concurrency / Kang J., Hur C.-K., Lahav O., Vafeiadis V., and Dreyer D. // ACM SIGPLAN Notices. — 2017. — Vol. 52, no. 1. — P. 175–189.
14. Refinedrust: A type system for high-assurance verification of Rust programs / Gäher L., Sammler M., Jung R., Krebbers R., and Dreyer D. // Proceedings of the ACM on Programming Languages. — 2024. — Vol. 8, no. PLDI. — P. 1115–1139.
15. RustBelt: Securing the foundations of the Rust programming language / Jung R., Jourdan J.-H., Krebbers R., and Dreyer D. // Proceedings of the ACM on Programming Languages. — 2017. — Vol. 2, no. POPL. — P. 1–34.
16. RustHornBelt: a semantic foundation for functional verification of Rust programs with unsafe code / Matsushita Y., Denis X., Jourdan J.-H., and Dreyer D. // Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. — 2022. — P. 841–856.
17. Stacked borrows: an aliasing model for Rust / Jung R., Dang H.-H., Kang J., and Dreyer D. // Proceedings of the ACM on Programming Languages. — 2019. — Vol. 4, no. POPL. — P. 1–32.
18. Verus: Verifying rust programs using linear ghost types / Lattuada A., Hance T., Cho C., Brun M., Subasinghe I., Zhou Y., Howell J., Parno B., and Hawblitzel C. // Proceedings of the ACM on Programming Languages. — 2023. — Vol. 7, no. OOPSLA1. — P. 286–315.
19. Verus: verifying Rust programs using linear ghost types (extended version) / Lattuada A., Hance T., Cho C., Brun M., Subasinghe I., Zhou Y., Howell J., Parno B., and Hawblitzel C. // arXiv preprint arXiv:2303.05491. — 2023.
20. Ануреев И.С. Операционная семантика операторов передачи управления в языке C на языке ABML // Системная информатика. — 2025. — no. 29. — P. 159–188.
21. Ануреев И.С. Язык спецификации дискретных динамических систем, ориентирован-

ных на знания, структурированные в онтологиях // Системная информатика. — 2025. — no. 29. — P. 137–158.