**Российская академия наук**
**Сибирское отделение**
**Институт систем информатики**
**им. А. П. Ершова**

# МОЛОДАЯ ИНФОРМАТИКА

**Вып. 4**

**СБОРНИК ТРУДОВ**
**АСПИРАНТОВ И МОЛОДЫХ УЧЕНЫХ**

**Под редакцией**
**к.ф.-м.н. Н.С. Грибовской**

**Новосибирск 2014**

Сборник содержит статьи, представленные аспирантами и молодыми сотрудниками ИСИ СО РАН, по следующим направлениям: теоретические аспекты программирования, информационные технологии и информационные системы, системное программное обеспечение, прикладное программное обеспечение.

**Siberian Branch of the Russian Academy of Sciences**
**A. P. Ershov Institute of Informatics Systems**

# YOUNG INFORMATICS

**Issue 4**

**COLLECTION OF PAPERS**
**OF GRADUATE STUDENTS**
**AND YOUNG SCIENTISTS**

**Edited by**
**N.S. Gribovskaya, Ph.D.**

**Novosibirsk 2014**

The volume contains the papers presented by post-graduates and young researchers of A.P. Ershov Institute of Informatics Systems which concern the following research areas: theoretical aspects of programming, information technologies and systems, system software and application software.

## ПРЕДИСЛОВИЕ

Основной задачей сборника является стимулирование научной деятельности аспирантов и молодых сотрудников (до 35 лет) Института систем информатики СО РАН и их обучение качественному представлению научных работ. При обучении использовалось двухэтапное рецензирование работ, и в сборник включались те статьи, которые были доработаны с учетом рецензий. Работы принимались в рамках тематики института по следующим направлениям: теоретические аспекты программирования, информационные технологии и информационные системы, системное программное обеспечение, прикладное программное обеспечение.

Целью работы «**Логическая унификация бисимуляционных эквивалентностей для временных стабильных структур событий**» является исследование двух вариантов бисимуляционных эквивалентностей (сохраняющей историю бисимуляции и наследственной сохраняющей историю бисимуляции) и получение для них унифицированной логической характеризации в контексте временных стабильных структур событий. В настоящее время существует большое разнообразие эквивалентностных понятий, поэтому актуальна задача их унификации. Для решения данной задачи используются методы теории категорий, а именно следующие два наиболее известных метода, предложенные Джоулем, Нильсеном и Винскелем: метод открытых морфизмов и метод бисимуляции вычислений. В частности, в работе были построены два варианта бисимуляции вычислений, разработаны два варианта модальной логики и показано, что эти варианты логики являются характеристическими логиками для исследуемых эквивалентностей.

При верификации параллельных систем, поведение которых представлено в виде интерливинга параллельных действий, возникает проблема быстрого роста количества состояний. Развёртка модели безопасных сетей Петри – это метод построения конечного «истинно параллельного» представления поведения систем, способствующий их эффективной верификации. Однако со временем стало понятно, что развёртки моделей систем, имеющих сложную структурную и функциональную организацию, могут иметь значительные размеры. Поэтому в литературе было предложено понятие слитных процессов, представляющих собой редукцию развёрток безопасных сетей Петри. Временные сети Петри широко используются для моделирования параллельных систем реального времени. В работе «**Редукция развёрток безопасных временных сетей Петри**» расширяется и исследуется метод редукции развёртки в контексте слитных процессов для безопасных временных сетей Петри.

В работе «**Алгебраические решётки первичных структур событий**» изучаются взаимосвязи базовых отношений (причинной зависимости, параллелизма, альтернативного выбора (конфликта)) между событиями параллельных систем, представленных моделями первичных структур событий. В частности,

предлагаются и исследуются техники построения алгебраических решёток конфигураций (вычислений) структур событий.

Работа «**Основы SPARQL**» посвящена основным конструкциям языка запросов SPARQL 1.0, который используется в случае, когда данные представлены в формате RDF (Resource Description Format – формат данных в концепте Semantic Web). Этот язык используется для SPARQL Endpoint – веб-сервисов, содержащих RDF-СУБД, выполняющих SPARQL запросы. Здесь приводятся их синтаксис, семантика, формальное описание в терминах множеств основных элементов RDF данных (кроме blank nodes и RDF Schema). Однако такие элементы SPARQL, как blank nodes, сортировка ORDER BY, ограничение OFFSET и LIMIT результатов запроса и наличие повторений DISTINCT и REDUCE, именованные графы (GRAPH, FROM), RDF Collections в данной работе не рассматриваются. Кроме того, работа содержит неформальные описания трансляции строки запроса в некоторое промежуточное объектное представление, не зависящее от данных в RDF-СУБД, и алгоритма ее исполнения.

В работе «**Автоматическая генерация тестов для проверки распараллеливающих и векторизующих преобразований циклов в компиляторе**» рассматривается проблема автоматической генерации тестовых программ для тестирования компилятора в области многопоточных расширений языка C++. Произведен краткий обзор существующих решений по генерации, а также обоснован применяемый в работе подход на основе параметрической контекстно-свободной грамматики. Описан процесс создания грамматик для тестового генератора, позволяющих проверить распараллеливающие и векторизующие преобразования циклов для таких расширений языка как OpenMP, Cilk Plus и GFX-offload. Работа также описывает процесс внедрения данных тестовых генераторов в процесс тестирования компилятора и его практические результаты.

Введение понятия приоритетов в модели сетей Петри (СП) позволяет увеличить их выразительную мощность до универсальной. Известны два типа приоритетов переходов сети: статические и динамические. В отличие от статических приоритетов, которые задаются в синтаксисе СП, динамические зависят от ее поведения, т.е. при функционировании СП одному и тому же переходу могут быть сопоставлены разные приоритеты. Статические приоритеты были введены во временные сети Петри (ВСП) французскими учеными в 2007 г. Они же построили конечную абстракцию поведения ВСП в терминах графа зон. В работе «**Альтернативная характеризация понятия зоны временных сетей Петри с динамическими приоритетами**» было введено понятие динамических приоритетов в контексте ВСП, изучены свойства и особенности поведения временных сетей Петри с динамическими приоритетами (ВСПсДП) и приведена альтернативная характеризация понятия зоны для ВСПсДП.

## PREFACE

The purpose of the volume is to stimulate research activity of post-graduates and young researchers of A.P. Ershov Institute of Informatics Systems and to train them in qualitative presentation of scientific papers. Training has been based on two-stage paper reviewing, and the volume contains only those papers which were improved according to reviewers' remarks. To be accepted, the paper should cover one of the research topics of the Institute, such as theoretical aspects of programming, information technologies and systems, system software and application software.

The intention of the paper "**A Logical Characteristic of Bisimulation Equivalences for Timed Stable Event Structures**" is to study two variants of bisimulation equivalences (history preserving bisimulation and hereditary history preserving bisimulation) in the setting of timed stable event structures and to provide a uniform logical characteristic for them. For this purpose, the authors use two popular category-theoretic approaches initiated by Joyal, Nielsen, and Winskel – open maps and path bisimilarity. In particular, in this paper two variants of path bisimilarity are constructed and two logics of path assertions are treated. Finally, it is shown that these logics are characteristic for the above-mentioned timed bisimulation equivalences.

The "state space explosion problem" arises in verification of concurrent systems whose behaviours have the form of interleavings. In order to cope with the problem within the framework of Petri net models, an unfolding method for constructing a finite non-interleaving representation of the net behaviours has been proposed in the literature. However, unfoldings of structurally and functionally complicated Petri net models may be very large. Therefore, to reduce the size of net unfoldings, a merge process technique has been put forward by Khomenko et al. in 2005. Time Petri nets are used for modelling and analysis of real-time and concurrent systems. In the paper "**An Unfolding Reduction of Safe Time Petri Nets**", a method of merge processes is extended and investigated in the context of safe time Petri nets.

The intention of the paper "**Constructing Algebraic Lattices for Prime Event Structures**" is to study the interconnections of basic relations (causal dependency, concurrency, and alternative choice (conflict)) between events of concurrent systems represented as prime event structures. In particular, the author proposes techniques for constructing algebraic lattices of configurations of the model under consideration.

The paper "**Foundations of SPARQL**" is devoted to basic constructions of the query language SPARQL 1.0 which is used when data are presented in the RDF format (Resource Description Format is a format of data in the concept of Semantic Web). This language is used in SPARQL Endpoint – web services containing RDF databases that perform SPARQL queries. The paper describes their syntax, semantics and formal description in terms of sets of the basic elements of RDF data (except for blank nodes and RDF Schema). Some elements of SPARQL, such as RDF Collections, blank nodes, a sorting ORDER BY, constraints LIMIT and OFFSET on query results, repetitions DISTINCT and REDUCE, and named graphs GRAPH and FROM, are not discussed in the paper. However, the author provided informal descriptions of query string translation in some intermediate object representation independent of the RDF-database, and the algorithm of its execution.

The paper "**Automated Test Generation for Checking Parallelization and Vectorization of Loops in the Compiler**" describes the existing approaches to test generation and focuses on the use of parametric context-free grammars in test generation for parallel extensions of C++ language like OpenMP, Cilk Plus and GFX-offload. Much attention is given to the process of grammar creation and to practical results.

There are static and dynamic priorities in the Petri net. Static priorities are given to transitions in the Petri net syntax whereas dynamic priorities are specified depending on on current net markings, i.e. transitions may change their priorities in the Petri net behavior. It is well known that static priorities strictly extend the expressiveness of time Petri nets (TPNs) – bounded TPNs with static priorities (PrTPNs) can be considered equivalent to timed automata, in terms of weak timed bisimularity. A finite abstraction of PrTPNs' state space represented as a zone graph was constructed by Bernard Berthomieu et al. in 2007. In the paper "**An Alternative Characterization of Zones for Time Petri Nets with Dynamic Priorities**" the author extends TPNs with dynamic priorities and presents an alternative characterization of their zones, to be useful for verification of the model under consideration.

**О.В. Арабаджи, Н.С. Грибовская**

# ЛОГИЧЕСКАЯ УНИФИКАЦИЯ БИСИМУЛЯЦИОННЫХ ЭКВИВАЛЕНТНОСТЕЙ ДЛЯ ВРЕМЕННЫХ СТАБИЛЬНЫХ СТРУКТУР СОБЫТИЙ*

## 1. ВВЕДЕНИЕ

Большое многообразие моделей, предложенных в теории параллелизма, требует их систематизации и унификации. Часто отношения между моделями могут быть охарактеризованы в терминах поведенческих эквивалентностей, которые обычно используются для сравнения поведения систем, а также упрощения их структуры. Существует большое разнообразие эквивалентностных понятий, взаимосвязи между которыми хорошо изучены. Наиболее известными являются три подхода – трассовый [7], тестовый [6, 10] и бисимуляционный [13]. При первом подходе сравниваются только языки, порождаемые системами. Данный подход очень удобен для анализа последовательных систем, но он приводит к потере информации о недетерминированном выборе (конфликтующих событиях), т.е. информации о ветвящейся структуре процесса. При бисимуляционном подходе две системы считаются эквивалентными, если внешний наблюдатель не может обнаружить различий в поведении этих систем с учетом точек недетерминированного выбора. При тестовом подходе поведение системы исследуется посредством набора тестов, в этом случае два процесса считаются эквивалентными, если они могут или должны проходить один и тот же набор тестов.

Для унификации поведенческих эквивалентностей в последние годы часто используются методы теории категорий. Самый популярный из них называется методом открытых морфизмов и предполагает построение категории изучаемой модели $\mathbb{M}$, выделение в этой категории подкатегории вычислений $\mathbb{P}$, относительно которой вводится понятие открытого морфизма, и, наконец, определение абстрактной $\mathbb{P}$-бисимуляции в терминах существования «моста» открытых морфизмов между моделями. Этот метод уже позволил дать унифицированное определение для широкого спектра поведенческих эквивалентностей в контексте различ-

*Данная работа выполнена при поддержке DFG (грант No BE 1267/14-1) и РФФИ (грант No 14-01-91334)

ных параллельных моделей [9–12]. В частности, в статье [1] была представлена характеризация на базе открытых морфизмов для различных поведенческих эквивалентностей в контексте временных стабильных структур событий. Другой теоретико-категорный подход основан на абстрактной бисимуляции вычислений, являющейся отношением на «морфизмах вычислений» (т. е. морфизмах из объектов подкатегории $\mathbb{P}$ вычислений в объекты категории $\mathbb{M}$). В работе [8] была дана логическая характеризация (на основе специального языка модальной логики) этой абстрактной бисимуляции, а также установлено ее совпадение с бисимуляцией открытых морфизмов, что позволяет строить характеристические логики для поведенческих эквивалентностей в контексте различных параллельных моделей. В частности, в статье [2] представлена логическая унификация различных поведенческих эквивалентностей в контексте первичных временных структур событий.

Одной из наиболее популярных моделей параллельных недетерминированных процессов с семантикой истинного параллелизма является модель структур событий, предложенная Винскелем [14]. Основное достоинство этой модели – явное задание базовых отношений (причинной зависимости, параллелизма и недетерминированного выбора) на событиях. Известны различные классы моделей структур событий: первичные [14], стабильные [15], потоковые [4], расслоенные [9] и т.д., однако они значительно различаются по своей выразительной мощности. Наиболее сильным классом является класс стабильных структур событий.

Цель данной работы – получить унифицированную логическую характеризацию для сохраняющей историю бисимуляции и наследственной сохраняющей историю бисимуляции в контексте временных стабильных структур событий. Для достижения поставленной цели используется теоретико-категорная характеризация поведенческих эквивалентностей, описанная в [1], и идея логической характеризации абстрактной бисимуляции вычислений из [8].

## 2. ВРЕМЕННЫЕ СТРУКТУРЫ СОБЫТИЙ

В данном разделе вводятся основные понятия и определения, касающиеся временных стабильных структур событий.

**Определение 1.** *(Помеченной) структурой событий* называется набор $S = (E, Con, \vdash, l)$, где

- $E$ – множество событий;

- $Con$ – непустое множество конечных подмножеств в $E$, называемое *предикатом совместимости*, удовлетворяющее условию: $X \subseteq Y \in Con \Rightarrow X \in Con$;
- $\vdash \subseteq Con \times E$ – *отношение инициации*, удовлетворяющее условию: $X \vdash e \wedge X \subseteq Y \in Con \Rightarrow Y \vdash e$;
- $l\colon E \to Act$ – помечающая функция, сопоставляющая событиям действия из конечного множества действий $Act$.

(Помеченная) структура событий $S$ называется *стабильной*, если верно $X \vdash e, Y \vdash e$ и $X \cup Y \cup \{e\} \in Con \Rightarrow X \cap Y \vdash e$. Множество стабильных структур событий обозначим через $\mathcal{S}$.

Под состоянием отдельного процесса системы, представленной структурой событий, понимается множество событий, которые успели выполниться на текущем этапе. Назовем это состояние конфигурацией.

**Определение 2.** Пусть $S$ – структура событий. Конечное подмножество событий $C \subseteq E$ называется *конфигурацией* в $S$, если $C \in Con$ и для каждого $e \in Con$ существуют $e_1, \ldots, e_n \in C$ такие, что $e_n = e$ и $\{e_1, \ldots, e_{i-1}\} \vdash e_i$ для всех $1 \leq i \leq n$.

**Определение 3.** Для структуры событий $S$ и ее конфигурации $C$ отношение *причинной зависимости* на событиях в $C$ определяется так: $e_1 \leq_C e_2 \Longleftrightarrow \forall C' \in \mathcal{C}(S) \circ e_2 \in C' \wedge C' \subseteq C \Rightarrow e_1 \in C'$.

Определяемая таким образом причинная зависимость на событиях конфигурации является частичным порядком, как это было показано в [15].

Теперь рассмотрим временное расширение стабильных структур событий или *временные стабильные структуры событий* [3]. Зафиксируем множества, с которым будем работать в дальнейшем. Пусть $\mathbb{R}$ – множество неотрицательных действительных чисел и

$$Interv = \{[d_1, d_2] \mid d_1, d_2 \in \mathbb{R}, d_1 \leq d_2\}.$$

**Определение 4.** *(Помеченной) временной структурой событий* называется набор $TS = (S, D)$, где

- $S$ – структура событий;
- $D\colon E_S \to Interv$ – временная функция, сопоставляющая событиям временные интервалы срабатывания.

Множество временных стабильных структур событий обозначим через $\mathcal{TS}$, пустую временную структуру событий — через $\mathcal{O}$.

**Определение 5.** Пусть $TS = (S, D)$ – временная стабильная структура событий. Пара $TC = (C, T)$, состоящая из конфигурации $C$ в $S$ и

функции $T\colon C \to R$ называется *временной конфигурацией* в $TS$, если выполнены следующие условия:

(i) $T(e) \in D(e)$ для всех $e \in C$;

(ii) $e_1 \leq_C e_2 \Rightarrow T(e_1) \leq T(e_2)$ для всех $e_1, e_2 \in C$.

Множество конфигураций в $TS$ обозначим через $\mathcal{C}(TS)$, а множество временных конфигураций – через $\mathcal{TC}(TS)$. Назовем $(\emptyset, \emptyset)$ *начальной временной конфигурацией* в $TS$.

Динамика моделируемого процесса заключается в изменении его состояния, которое происходит в соответствии с *отношением перехода* из одного состояния в другое. Пусть $TS$ – временная стабильная структура событий и $TC = (C, T), TC_1 = (C_1, T_1) \in \mathcal{TC}(TS)$. Будем говорить, что $TC$ переходит в $TC_1$ (обозначается $TC \to TC_1$), если $C \subseteq C_1$ и $T = T_1 \mid_C$. Определим *ограничение* временной структуры событий $TS$ на $TC = (C, T)$ как $TS\lceil TC = (S\lceil C, T)$, где $S\lceil C = (C, \mathcal{P}(C), \vdash \cap (\mathcal{P}(C) \times C), l \mid_C)$ и $\mathcal{P}(C)$ – множество подмножеств $C$. Заметим, что $TS\lceil TC \in \mathcal{TS}$.

Далее введем понятие временного частичного упорядоченного множества, которое является временной стабильной структурой событий, моделирующей конечный детерминированный процесс.

**Определение 6.** *(Помеченным) временным частично упорядоченным множеством* называется конечная временная стабильная структура событий $TP = (E, Con, \vdash, l, D)$, где $D\colon E \to \mathbb{R}$ и $(E, D) \in \mathcal{TC}(TP)$.

Обозначим через $\mathcal{TP}$ множество всех временных частично упорядоченных множеств. Заметим, что $TS\lceil TC \in \mathcal{TP}$ для любой $TS \in \mathcal{TS}$ и $TC \in \mathcal{TC}(TS)$.

Пусть $TP = (E, Con, \vdash, l, D)$ и $TP' = (E', Con', \vdash', l', D') \in \mathcal{TP}$. Назовем $TP$ *приращением* $TP'$, если существует биекция $f\colon E \to E'$ такая, что $e_1 \leq_E e_2 \Leftarrow f(e_1) \leq_{E'} f(e_2)$, $l(e_1) = l'(f(e_1))$, $D(e_1) = D'(f(e_1))$ для всех $e_1, e_2 \in E$. Будем говорить, что $TP$ и $TP'$ *изоморфны* (обозначается $TP \simeq TP'$), если существует биекция $f\colon E \to E'$ такая, что $e_1 \leq_E e_2 \iff f(e_1) \leq_{E'} f(e_2)$, $l(e_1) = l'(f(e_1))$, $D(e_1) = D'(f(e_1))$ для всех $e_1, e_2 \in E$.

**Пример 1.** В качестве примера рассмотрим временную стабильную структуру событий

$TS_0 = (E, Con, \vdash, l, D)$, где

- $E = \{e_1, e_2, e_3, e_4\}$;
- $Con = \{\{e_1, e_2\}, \{e_1, e_3\}, \{e_1, e_4\}, \{e_3, e_4\}\}^*$;

*В данном примере указаны только максимальные множества совместимых событий.

- $\emptyset \vdash e_1, \emptyset \vdash e_3, \{e_1\} \vdash e_2, \{e_1\} \vdash e_4, \{e_3\} \vdash e_4$;
- $l(e_1) = (e_4) = a, l(e_2) = b, l(e_3) = c$;
- $D(e_1) = [0,1], D(e_2) = [1,3], D(e_3) = [2,3], D(e_4) = [3,3]$.

Временные конфигурации в $TS_0$:
$\mathcal{TC}(TS_0) = \{(\emptyset,\emptyset), (\{e_1\},T_1), (\{e_3\},T_2), (\{e_1,e_2\},T_3), (\{e_1,e_3\},T_4),$
$(\{e_1,e_4\},T_5), (\{e_3,e_4\},T_6) \mid T_1(e_1), T_3(e_1), T_4(e_1), T_5(e_1) \in [0,1];$
$T_3(e_2) \in [1,3]; T_2(e_3), T_4(e_3), T_6(e_3) \in [2,3]; T_5(e_4), T_6(e_4) \in [3,3]\}$.

## 3. БИСИМУЛЯЦИИ

В данном разделе вводятся понятия временных вариантов сохраняющей историю бисимуляции и наследственной сохраняющей историю бисимуляции.

**Определение 7.** Пусть $TS$ и $TS'$ – временные стабильные структуры событий.

- *Временная сохраняющая историю бисимуляция (thp-бисимуляция)* между $TS$ и $TS'$ – это отношение $\mathcal{B}$, состоящее из троек $(TC, f, TC')$ (где $TC$ – временная конфигурация в $TS$, $TC'$ – временная конфигурация в $TS'$, а $f\colon TS\lceil TC \to TS'\lceil TC'$ – изоморфизм), такое, что $((\emptyset,\emptyset),\emptyset,(\emptyset,\emptyset)) \in \mathcal{B}$ и $\forall\,(TC, f, TC') \in \mathcal{B}$ выполнено:
  1) если $TC \to TC_1$ в $TS$, то
  $TC' \to TC'_1$ в $TS'$ и $(TC_1, f_1, TC'_1) \in \mathcal{B}$, где $f \subseteq f_1$,
  для некоторой временной конфигурации $TC'_1 \in \mathcal{TC}(TS')$ и некоторого изоморфизма $f_1$;
  2) если $TC' \to TC'_1$ в $TS'$, то
  $TC \to TC_1$ в $TS$ и $(TC_1, f_1, TC'_1) \in \mathcal{B}$, где $f \subseteq f_1$,
  для некоторой временной конфигурации $TC_1 \in \mathcal{TC}(TS)$ и некоторого изоморфизма $f_1$;
- *thp*-бисимуляция $\mathcal{B}$ между $TS$ и $TS'$ называется *временной наследственной сохраняющей историю бисимуляцией (thhp-бисимуляцией)*, если для всех троек вида $(TC, f, TC') \in \mathcal{B}$ выполнено:
  1) если $TC_1 \to TC$ в $TS$, то
  $TC'_1 \to TC'$ в $TS'$ и $(TC_1, f_1, TC'_1) \in \mathcal{B}$ для некоторой временной конфигурации $TC'_1 \in \mathcal{TC}(TS')$ и некоторого изоморфизма $f_1$ такого, что $f_1 \subseteq f$;
  2) если $TC'_1 \to TC'$ в $TS'$, то
  $TC_1 \to TC$ в $TS$ и $(TC_1, f_1, TC'_1) \in \mathcal{B}$ для некоторой времен-

ной конфигурации $TC_1 \in \mathcal{TC}(TS)$ и некоторого изоморфизма $f_1$ такого, что $f_1 \subseteq f$;

- Временные стабильные структуры событий $TS$ и $TS'$ называются *thp-бисимуляционными (thhp-бисимуляционными)* $\Longleftrightarrow$ между ними существует *thp*-бисимуляция (*thhp*-бисимуляция).

## 4. ЭЛЕМЕНТЫ ТЕОРИИ КАТЕГОРИЙ

Одним из наиболее важных понятий теории категорий [14] являются *открытые морфизмы*. Как было показано в статьях [8, 11] метод открытых морфизмов позволяет унифицировать понятие бисимуляции на различных категориях моделей. Для заданной категории открытые морфизмы определяются по специально выделенной подкатегории и являются своего рода функциями подобия между объектами категории. Пусть $\mathbb{M}$ – категория, а $\mathbb{P}$ – подкатегория категории $\mathbb{M}$.

**Определение 8.** Морфизм $m\colon X \to Y$ в категории $\mathbb{M}$ называется $\mathbb{P}$-*открытым* $\Longleftrightarrow$ для любых морфизмов $f\colon P_1 \to P_2$ в $\mathbb{P}$, $p\colon P_1 \to X$ и $q\colon P_2 \to Y$ в $\mathbb{M}$ таких, что $m \circ p = q \circ f$, существует морфизм $h\colon P_2 \to X$ в $\mathbb{M}$ такой, что $p = h \circ f$ и $q = m \circ h$.

Заметим, что объекты категории $\mathbb{M}$ и $\mathbb{P}$-открытые морфизмы образуют подкатегорию в категории $\mathbb{M}$, так как тождественные морфизмы и композиции $\mathbb{P}$-открытых морфизмов являются $\mathbb{P}$-открытыми морфизмами.

Теперь приведем определение абстрактной бисимуляции открытых морфизмов.

**Определение 9.** $X$ и $Y$ в категории $\mathbb{M}$ называются $\mathbb{P}$-*бисимуляционными* тогда и только тогда, когда существует мост $\mathbb{P}$-открытых морфизмов $X \xleftarrow{m} Z \xrightarrow{m'} Y$.

## 5. КАТЕГОРИЯ $\mathcal{CTS}_{ACT}$

В данном разделе вводится категория временных стабильных структур событий $\mathcal{CTS}_{Act}$, морфизмами которой являются функции моделирования, определенные в соответствии с подходом Винскеля, Джояля и Нильсена [8].

Для начала напомним определение морфизма.

**Определение 10.**

Пусть $TS = (E, Con, \vdash, l, D)$ и $TS' = (E', Con', \vdash', l', D')$ – две времен-

ные стабильные структуры событий. Отображение $\mu\colon TS \to TS'$ называется *морфизмом*, если $\mu\colon E \to E'$ – функция, $l = l' \circ \mu$ и для всех $TC = (C, T) \in \mathcal{TC}(TS)$ верно:

- $\mu TC \in \mathcal{TC}(TS')$, где $\mu TC = (\mu C, T')$ и $T' \circ \mu = T$;
- $\forall\, e, e' \in C \diamond \mu(e) = \mu(e') \Rightarrow e = e'$.

**Утверждение 1.** Пусть $TS, TS' \in \mathcal{TS}$, $\mu\colon TS \to TS'$ – морфизм и $TC \in \mathcal{TC}(TS)$. Тогда $TS\lceil TC$ является приращением $TS'\lceil\mu(TC)$.

Теперь приведем понятие категории $\mathcal{CTS}_{Act}$. Временные стабильные структуры событий (помеченные над $Act$) с морфизмами между ними составляют категорию временных структур событий, $\mathcal{CTS}_{Act}$, в которой композиция двух морфизмов $\mu_1\colon TS_1 \to TS_2$ и $\mu_2\colon TS_2 \to TS_3$ определяется как $\mu_2 \circ \mu_1\colon TS_1 \to TS_3$, и единичный морфизм является тождественной функцией.

Пусть $\mathcal{TP}_{Act}$ обозначает полную подкатегорию категории $\mathcal{CTS}_{Act}$, объектами которой являются временные частично упорядоченные множества из $\mathcal{TP}$.

Приведем основной результат работы [1], доказывающий, что $thhp$-бисимуляция совпадает с $\mathcal{TP}_{Act}$-бисимуляцией.

**Теорема 1** ([1]). *Пусть $TS_1$ и $TS_2$ – две временные стабильные структуры событий. Тогда $TS_1$ и $TS_2$ thhp-бисимуляционны, если и только если они $\mathcal{TP}_{Act}$-бисимуляционны.*

В следующем разделе выясним, можно ли сохраняющую историю бисимуляцию и наследственную сохраняющую историю бисимуляцию охарактеризовать с помощью языка модальной логики, как была охарактеризована (сильная) сохраняющая историю бисимуляция в контексте временных первичных структур событий в работе [8].

## 6. БИСИМУЛЯЦИЯ ВЫЧИСЛЕНИЙ

Пусть $\mathbb{M}$ – категория моделей, а $\mathbb{P}$ – малая категория вычислений, являющаяся подкатегорией $\mathbb{M}$ и имеющая с этой категорией общий начальный объект $I$. В случае, когда $\mathbb{M}$ является категорией временных структур событий $\mathcal{CTS}_{Act}$, а $\mathbb{P}$ – ее подкатегорией, то начальный объект $I$ представляет собой пустую временную структуру событий $\mathcal{O}$.

**Определение 11.** *Бисимуляцией $\mathbb{P}$-вычислений* между объектами $X_1$ и $X_2$ в $\mathbb{M}$ называется множество $\mathcal{R}$ пар морфизмов $\mathbb{P}$-вычислений $(p_1, p_2)$ с общей областью определения $P$, где $p_1 : P \to X_1$ является морфизмом $\mathbb{P}$-вычислений в $X_1$, а $p_2 : P \to X_2$ – морфизмом $\mathbb{P}$-вычислений в $X_2$

такое, что:

(o) $(i_1, i_2) \in \mathcal{R}$, где $i_1 \colon I \to X_1$ и $i_2 \colon I \to X_2$ – уникальные морфизмы $\mathbb{P}$-вычислений, и для всех пар $(p_1, p_2) \in \mathcal{R}$ и всех $m \colon P \to Q$, где $m$ – морфизм в $\mathbb{P}$, верно:

(i) если существует $q_1 \colon Q \to X_1$ и $q_1 \circ m = p_1$, то существует $q_2 \colon Q \to X_2$ такой, что $q_2 \circ m = p_2$ и $(q_1, q_2) \in \mathcal{R}$;

(ii) если существует $q_2 \colon Q \to X_2$ и $q_2 \circ m = p_2$, то существует $q_1 \colon Q \to X_1$ такой, что $q_1 \circ m = p_1$ и $(q_1, q_2) \in \mathcal{R}$.

Бисимуляция $\mathbb{P}$-вычислений называется *сильной* бисимуляцией $\mathbb{P}$-вычислений, если выполнено следующее условие:

(iii) если $(q_1, q_2) \in \mathcal{R}$, $q_1 \colon Q \to X_1$, $q_2 \colon Q \to X_2$ и $m \colon P \to Q$ – морфизм в $\mathbb{P}$, то $(q_1 \circ m, q_2 \circ m) \in \mathcal{R}$.

$X_1$ и $X_2$ называются *(сильно) бисимуляционными относительно* $\mathbb{P}$*-вычислений*, если и только если между ними существует (сильная) бисимуляция $\mathbb{P}$-вычислений.

*Утверждением о морфизмах* $\mathbb{P}$-вычислений называется составное логическое высказывание $A$, построенное с помощью четырех логических связок (или модальностей):

$$A := \bigwedge_{j \in J} A_j \mid \neg A \mid \langle m \rangle A \mid \overline{\langle m \rangle} A,$$

где $J$ – индексируемое множество, возможно пустое или бесконечное, а $m$ – морфизм подкатегории $\mathbb{P}$. Модальность $\langle m \rangle$ называется «прямой» модальностью, а модальность $\overline{\langle m \rangle}$ – «обратной». Пустая конъюнкция (т. е. конъюнкция с пустым индексируемым множеством $J$) обозначает тождественно истинное высказывание.

Для фиксации семантики утверждений необходимо задать, когда морфизм $\mathbb{P}$-вычислений $p \colon P \to X$ удовлетворяет утверждению $A$. Определим отношение $\vDash$ по структурной индукции:

- $p \vDash \bigwedge\limits_{j \in J} A_j \iff p \vDash A_j$ для всех $j \in J$;
- $p \vDash \neg A \iff p \nvDash A$;
- $p \vDash \langle m \rangle A$ для $m \colon P \to P' \iff$ существует морфизм $\mathbb{P}$-вычислений $p' \colon P' \to X$, для которого $p' \vDash A$ и $p = p' \circ m$;
- $p \vDash \overline{\langle m \rangle} A$ для $m \colon P' \to P \iff$ существует морфизм $\mathbb{P}$-вычислений $p' \colon P' \to X$, для которого $p' \vDash A$ и $p' = p \circ m$.

Будем называть *прямыми утверждениями о морфизмах* $\mathbb{P}$*-вычислений* те утверждения, которые построены без использования обратной

модальности.

Приведем логические характеризации для (сильной) бисимуляции $\mathcal{TP}_{Act}$-вычислений.

**Теорема 2.** Пусть $TS_1$ и $TS_2$ – две временные стабильные структуры событий. Тогда

1. $TS_1$ и $TS_2$ бисимуляционны относительно $\mathcal{TP}_{Act}$-вычислений $\Leftrightarrow$ их начальные морфизмы $\mathcal{TP}_{Act}$-вычислений $p_1^0\colon \mathcal{O} \to TS_1$ и $p_2^0\colon \mathcal{O} \to TS_2$ удовлетворяют одному и тому же набору прямых утверждений о морфизмах $\mathcal{TP}_{Act}$-вычислений;
2. $TS_1$ и $TS_2$ сильно бисимуляционны относительно $\mathcal{TP}_{Act}$-вычислений $\Leftrightarrow$ их начальные морфизмы $\mathcal{TP}_{Act}$-вычислений $p_1^0\colon \mathcal{O} \to TS_1$ и $p_2^0\colon \mathcal{O} \to TS_2$ удовлетворяют одному и тому же набору утверждений о морфизмах $\mathcal{TP}_{Act}$-вычислений.

*Доказательство.* Следует из теоремы 15, приведенной в работе [8]. □

**Утверждение 2.** Для всех $(p_1, p_2) \in \mathcal{R}$ $p_1$ является изоморфизмом между $TP$ и $TS_1 \lceil TC_{p_1}$ тогда и только тогда, когда $p_2$ является изоморфизмом между $TP$ и $TS_2 \lceil TC_{p_2}$.

*Доказательство.* Следует из утверждения 1 и того, что $\mathcal{R}$ – бисимуляция $\mathcal{TP}_{Act}$-вычислений. □

**Теорема 3.** Пусть $TS_1$ и $TS_2$ – две временные стабильные структуры событий. Тогда

1. $TS_1$ и $TS_2$ *thp*-бисимуляционны $\Leftrightarrow$ они бисимуляционны относительно $\mathcal{TP}_{Act}$-вычислений;
2. $TS_1$ и $TS_2$ *thhp*-бисимуляционны $\Leftrightarrow$ они сильно бисимуляционны относительно $\mathcal{TP}_{Act}$-вычислений.

*Доказательство.*

1. Проверим, что *thp*-бисимуляция влечет бисимуляцию $\mathcal{TP}_{Act}$-вычислений. Пусть $\mathcal{B}$ – *thp*-бисимуляция между $TS_1$ и $TS_2$. Для тройки $(TC_1, f, TC_2) \in \mathcal{B}$ введем морфизмы $p_1^f : TS_1 \lceil TC_1 \to TS_1$ и $p_2^f : TS_2 \lceil TC_2 \to TS_2$ как обычные вложения. Определим отношение $\mathcal{R} = \{(p_1^f \circ m,\, p_2^f \circ f \circ m) \mid m\colon TP \to TS_1 \lceil TC_1$ – морфизм в $\mathcal{TP}_{Act}$ и $(TC_1, f, TC_2) \in \mathcal{B}\}$. Используя свойства *thp*-бисимуляции $\mathcal{B}$, нетрудно показать, что отношение $\mathcal{R}$ удовлетворяет свойствам, предъявляемым к бисимуляции $\mathcal{TP}_{Act}$-вычислений.

Теперь докажем, что существование бисимуляции $\mathcal{TP}_{Act}$-вычислений между $TS_1$ и $TS_2$ является достаточным условием для наличия *thp*-бисимуляции между ними. Пусть $\mathcal{R}$ – бисимуляция $\mathcal{TP}_{Act}$-вычислений между $TS_1$ и $TS_2$. Рассмотрим произвольное временное частично упорядоченное множество

$$TP = (E, Con, \vdash, l, D).$$

Положим $TC = (E, D)$. По определению 6 $TC$ является временной конфигурацией в $TP$. Для морфизма $\mathcal{TP}_{Act}$-вычислений $p_i \colon TP \to TS_i (i = 1, 2)$ пусть $TC_{p_i}$ обозначает временную конфигурацию $p_i(TC)$ в $TS_i$. Определим отношение $\mathcal{B}$ как множество $\{(TC_{p_1}, p_2 \circ p_1^{-1}, TC_{p_2}) \mid (p_1, p_2) \in \mathcal{R}$ и для любого $i = 1, 2$ $p_i \colon TP \to TS_i$ – изоморфизм между $TP$ и $TS_i\lceil TC_{p_i}\}$.

Из свойств бисимуляции $\mathcal{TP}_{Act}$-вычислений следует, что отношение $\mathcal{B}$ обладает всеми свойствами, описанными в определении *thp*-бисимуляции.

2. По теореме 1, *thhp*-бисимуляция совпадает с $\mathcal{TP}_{Act}$-бисимуляцией. Согласно лемме 16 [8], существование $\mathcal{TP}_{Act}$-бисимуляции влечет существование сильной бисимуляции $\mathcal{TP}_{Act}$-вычислений. Покажем, что сильная бисимуляция $\mathcal{TP}_{Act}$-вычислений влечет *thhp*-бисимуляцию. Пусть $\mathcal{R}$ – сильная бисимуляция $\mathcal{TP}_{Act}$-вычислений между $TS_1$ и $TS_2$. Рассмотрим произвольное временное частично упорядоченное множество $TP = (E, Con, \vdash, l, D)$. Положим $TC = (E, D)$. По определению 6 $TC$ является временной конфигурацией в $TP$. Для морфизма $\mathcal{TP}_{Act}$-вычислений $p_i \colon TP \to TS_i$ $(i = 1, 2)$ пусть $TC_{p_i}$ обозначает временную конфигурацию $p_i(TC)$ в $TS_i$. Определим отношение $\mathcal{B}$ как множество

$$\{(TC_{p_1}, p_2 \circ p_1^{-1}, TC_{p_2}) \mid (p_1, p_2) \in \mathcal{R} \text{ и для любого } i = 1, 2$$
$$p_i \colon TP \to TS_i - \text{ изоморфизм между } TP \text{ и } TS_i\lceil TC_{p_i}\}.$$

По аналогии с пунктом 1 получаем, что $\mathcal{B}$ является *thp*-бисимуляцией. Проверим, что $\mathcal{B}$ – это *thhp*-бисимуляция. Пусть $TC' \to TC_{p_1}$ в $TS_1$ (случай, когда $TC'' \to TC_{p_2}$ в $TS_2$ доказывается аналогично). Это означает, что $C' \subseteq C_{p_1}$ и $T' = T_{p_1} \mid_{C'}$. Пусть $TP' = TS_1\lceil TC'$. Определим $p_1'$ как тождественное отображение $p_1' \colon TP' \to TS_1\lceil TC'$. Ясно, что $TS_1\lceil TC_{p_1'} = TS_1\lceil TC' \subseteq TS_1\lceil TC_{p_1}$. Положим $m = (p_1)^{-1} \circ p_1'$. Очевидно, что $m \colon TP' \to TP$ – мор-

физм в $\mathbb{P}$. Так как $(p_1, p_2) \in \mathcal{R}$ и $\mathcal{R}$ – сильная бисимуляция $\mathcal{TP}_{Act}$-вычислений, то $(p_1 \circ m, p_2 \circ m) \in \mathcal{R}$. Более того, $p_1 \circ m = p_1 \circ (p_1)^{-1} \circ p'_1 = p'_1$. Выберем в качестве $p'_2$ отображение $p_2 \circ m$. Так как $(p'_1, p'_2) \in \mathcal{R}$ и $p'_1$ – изоморфизм, по Утверждению 2, $p'_2$ – изоморфизм. Следовательно, $(TC_{p'_1}, p_2 \circ (p'_1)^{-1}, TC_{p'_2}) \in \mathcal{B}$.

□

**Следствие 1.** *Пусть $TS_1$ и $TS_2$ – BCCC. Тогда*

1. *$TS_1$ и $TS_2$ thp-бисимуляционны $\Longleftrightarrow$ их начальные морфизмы $p_1^0 \colon \mathcal{O} \to TS_1$ и $p_2^0 \colon \mathcal{O} \to TS_2$ удовлетворяют одному и тому же набору прямых утверждений о морфизмах $\mathcal{TP}_{Act}$-вычислений;*
2. *$TS_1$ и $TS_2$ thhp-бисимуляционны $\Longleftrightarrow$ их начальные морфизмы $p_1^0 \colon \mathcal{O} \to TS_1$ и $p_2^0 \colon \mathcal{O} \to TS_2$ удовлетворяют одному и тому же набору утверждений о морфизмах $\mathcal{TP}_{Act}$-вычислений.*

Таким образом, логика утверждений о морфизмах $\mathcal{TP}_{Act}$-вычислений является характеристической логикой для *thhp*-бисимуляции, а логика прямых утверждений о морфизмах $\mathcal{TP}_{Act}$-вычислений – это характеристическая логика для *thp*-бисимуляции.

## 7. ЗАКЛЮЧЕНИЕ

В данной работе представлена логическая унифицирующая характеризация сохраняющей историю бисимуляции и наследственной сохраняющей историю бисимуляции в контексте временных стабильных структур событий. В частности, в работе была построена бисимуляция вычислений и ее сильный вариант, показано совпадение построенных бисимуляций с исследуемыми поведенческими эквивалентностями и разработаны характеристические модальные логики для них.

## СПИСОК ЛИТЕРАТУРЫ

1. Андреева М. В. Открытые отображения и поведенческие эквивалентности временных стабильных структур событий. Вестник НГУ. Серия: Математика, механика, информатика. 2008. Т. 8, вып. 2. С. 14–29.
2. Грибовская Н. С. Логическая унификация поведенческих эквивалентностей временных структур событий. Вестник НГУ. Серия: Математика, механика, информатика. 2013. Т. 13, вып. 1. С. 32–46.
3. Andreeva M. V., Virbitskaite I. B. Observational Equivalences for Timed Stable Event Structures // Fundamenta Informaticae. 2006. Vol. 72. No. 1–3. P. 1–19.

4. Boudol G., Castellani I. Concurrency and Atomicity // Theor. Comput. Sci. 1989. Vol. 59. P. 25–84.
5. Cattani G . L., Sassone V. Higher Dementional Transition System // $11^{th}$ Annual IEEE Symp. on Logic in Computer Scince. Washington: IEEE Comp. Soc. Press, 1996. P. 55–62.
6. Cleaveland R., Hennessy M. Testing Equivalence as a Bisimulation Equivalence // Lect. Notes in Comput. Sci. 1989. Vol. 407. P. 11–23.
7. Hoare C. A. R. Communicating Sequential Processes. L.: Prentice-Hall, 1985.
8. Joyal A., Nielsen M., Winskel G. Bisimulation from Open Maps // Information and Computation. 1996. Vol. 127. No. 2. P. 164–185.
9. Langerak R. Bundle Event Structures: a Non-interleaving Semantics for LOTOS. Formal Description Techniques V. // IFIP Transactions. 1993. Vol. C–10. P. 331–346.
10. De Nicola R., Hennessy M. Testing Equivalence for Processes // Theor. Comput. Sci. 1984. Vol. 34. P. 83–133.
11. Nielsen M., Cheng A. Observing Behavior Categorically // LNCS. 1995. Vol. 1026. P. 263–278.
12. Oshevskaya E. S. Open Maps Bisimulations for Higher Dimensional Automata Models // LNCS. 2009. Vol. 5699. P. 274–286.
13. Park D. Concurrency and Automata on Infinite Sequences // Lect. Notes in Comput. Sci. 1981. Vol. 104. P. 167–183.
14. Winskel G. Event in Computation. PhD Thesis. University of Edinburgh, 1980.
15. Winskel G. Event Structures // In Advances in Petri Nets: Part II. Lect. Notes in Comput. Sci. 1987. Vol. 255. P. 325–392.

**В. А. Боровлёв**

# РЕДУКЦИЯ РАЗВЁРТОК БЕЗОПАСНЫХ ВРЕМЕННЫХ СЕТЕЙ ПЕТРИ*

## 1. ВВЕДЕНИЕ

Безопасные сети Петри (БСП) – это формальная модель, используемая для анализа распределённых и параллельных систем [1]. Традиционно поведение БСП представляется в терминах последовательностей срабатываний переходов. Однако при этом теряется информация об отношениях причинной зависимости и параллелизма между событиями системы. Поэтому была предложена техника процессов [7], в которых эти отношения представляются явно. Следующим шагом развития техники процессов стало понятие *ветвистых процессов* [10], в которых стало возможно представление конфликтов между событиями. Максимальный ветвистый процесс был назван *развёрткой* и объединял все процессы БСП. Но использование развёртки для автоматической верификации затруднено из-за её подверженности проблеме *быстрого роста количества состояний* [11, 3].

В литературе было предложено построение *слитного процесса* развёртки БСП [5], который соответствовал редукции развёртки. Таким образом, слитный процесс развёртки БСП стал эффективным средством для автоматической верификации поведения БСП за счёт преемственности свойств развёртки и частично решил проблему быстрого роста количества состояний. Дальнейшим развитием понятия слитных процессов стала работа [6], в которой было предложено построение слитного процесса развёртки для контекстных БСП [9], представляющих собой расширение БСП за счёт добавления *контекстных* дуг.

Для безопасных временных сетей Петри (БВСП), представляющих собой расширение БСП посредством добавления понятия времени и широко используемых для моделирования систем реального времени, например, протоколов коммуникаций [2], были разработаны понятия *временных расширенных процессов*, представляющих поведение БВСП, и развёртки [4]. Установленная между ними связь дала возможность ве-

*Данная работа частично финансируется DFG и РФФИ (проект CAVER, грант N BE 1267/14-1, грант No 14-01-91334).

рификации свойств системы реального времени по развёртке – структуре, не содержащей временной составляющей, в отличие от временных расширенных процессов.

В данной работе отмечено, что структура развёртки БВСП схожа с развёрткой контекстных БСП, и предложено использование техники построения слитного процесса развёртки контекстных БСП для редуцирования развёртки БВСП. Также установлена связь полученной редуцированной развёртки и *временных слитных процессов* БВСП, разработанного аналога временных расширенных процессов.

## 2. БЕЗОПАСНЫЕ ВРЕМЕННЫЕ СЕТИ ПЕТРИ

В этом разделе будут даны определения структуры и последовательного поведения безопасных временных сетей Петри.

*Безопасная временная сеть Петри (БВСП)* – это кортеж $\langle P, T, pre, post, efd, lfd\rangle$, где $P$ и $T$ – конечные множества *мест* и *переходов*, $pre$ и $post$ – отображения из $T$ в $P$, сопоставляющие каждому $t \in T$, соответственно, множество его *входных и выходных* мест (как правило, используют обозначения ${}^\bullet t \stackrel{\text{def}}{=} pre(t)$ и $t^\bullet \stackrel{\text{def}}{=} post(t)$); $efd\colon T \longrightarrow \mathbb{Q}_{\geq 0}$ и $lfd\colon T \longrightarrow \mathbb{Q}_{\geq 0} \cup \infty$ – отображения, сопоставляющие каждому $t \in T$, соответственно, его *минимальное* и *максимальное время ожидания срабатывания.*

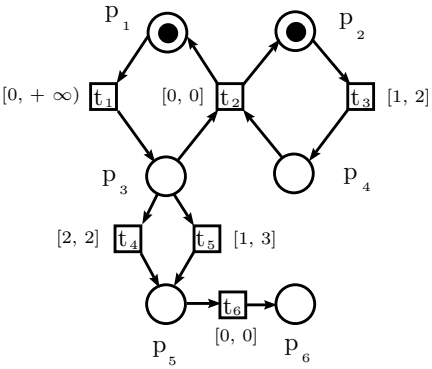



*Рис. 1.* Безопасная временная сеть Петри

**Пример 1.** *На рис. 1 представлена БВСП. Рядом с каждым переходом* $t \in T$ *подписаны его границы ожидания срабатывания* $[efd(t), lfd(t)]$.

*Маркировкой БВСП* называют множество мест $M \subseteq P$, содержащих фишки. Говорят, что переход $t \in T$ *допустим в маркировке* $M$, если ${}^\bullet t \subseteq M$. Множество допустимых в маркировке $M$ переходов обозначают через $Ena(M)$.

*Состоянием* БВСП называют тройку $s \stackrel{\text{def}}{=} \langle M, dob, \theta\rangle$, где $M$ – маркировка сети, $dob : M \longrightarrow \mathbb{Q}_{\geq 0}$ – функция, ставящая в соответствие каждому месту $p \in M$ момент времени появления фишки в этом месте $dob(p) \leq \theta$, а $\theta \in \mathbb{Q}_{\geq 0}$ – глобальное время, связанное с данным состоянием. *Начальное состояние* обозначают как $s_0 \stackrel{\text{def}}{=} \langle M_0, dob_0, \theta_0\rangle$, где $M_0$ называют *начальной маркировкой*, $\theta_0$ – *начальным моментом времени* и $\forall p \in M_0\ dob_0(p) \stackrel{\text{def}}{=} \theta_0$.

Момент времени $doe(t)$, в который во всех входных местах перехода $t$ появились фишки, вычисляется так: $doe(t) \stackrel{\text{def}}{=} max_{p \in {}^\bullet t} dob(p)$.

Переход $t$ *готов сработать в момент времени* $\theta' \geq \theta$ из состояния $s = \langle M, dob, \theta\rangle$ (иногда говорят в маркировке $M$), если:

1. $t \in Ena(M)$;
2. $(M \backslash {}^\bullet t) \cap t^\bullet = \emptyset$;
3. $\theta' \geq doe(t) + efd(t)$;
4. $\forall t' \in T\ t' \in Ena(M) \Rightarrow \theta' \leq doe(t') + lfd(t')$.

Если выполнены все условия, кроме условия 2, то говорят, что переход $t$ *находится в контакте*, и множество таких переходов обозначают $Contact(s)$.

Срабатывание готового перехода $t$ в момент времени $\theta'$ из состояния $s = \langle M, dob, \theta\rangle$ приводит к новому состоянию $s' = \langle (M \backslash {}^\bullet t) \cup t^\bullet, dob', \theta'\rangle$, где $dob'(p) \stackrel{\text{def}}{=} dob(p)$, если $p \in M \backslash {}^\bullet t$, и $dob'(p) \stackrel{\text{def}}{=} \theta'$, если $p \in t^\bullet$. В этом случае пишут $s \stackrel{(t,\theta')}{\longrightarrow} s'$. Последовательность вида $s_0 \stackrel{(t_1,\theta_1)}{\longrightarrow} s_1, \ldots, s_{n-1} \stackrel{(t_n,\theta_n)}{\longrightarrow} s_n$ $(n \geq 0)$ называют *последовательностью срабатываний* и обозначают $((t_1, \theta_1), \ldots, (t_n, \theta_n))$. Пустую последовательность срабатываний обозначают $\varepsilon$.

Состояние $\langle M, dob, \theta\rangle$ *достижимо*, если существует последовательность срабатываний, приводящая в это состояние. При этом маркировку $M$ называют *достижимой*. Если для любого достижимого состояния $s$ БВСП $Contact(s) = \emptyset$, то БВСП называют *свободной от контактов*. БВСП называют *прогрессирующей во времени*, если для любого мно-

жества переходов $\{t_1, \ldots, t_n\}$, таких что $t_i^\bullet \cap {}^\bullet t_{i+1} \neq \emptyset$ и $t_n^\bullet \cap {}^\bullet t_1 \neq \emptyset$ для каждого $1 \leq i \leq n-1$, верно, что $\sum_{1 \leq i \leq n} efd(t_i) > 0$. Далее будем рассматривать свободные от контактов и прогрессирующие во времени БВСП.

**Пример 2.** *Легко убедиться, что БВСП, представленная на рис. 1, – свободная от контактов и прогрессирующая во времени. В качестве примера рассмотрим её последовательность срабатываний:* $((t_1, 0), (t_5, 1.3), (t_6, 1.3))$. *Можно заметить, что после срабатывания перехода* $t_5$*, глобальное время не сможет идти дальше, пока переход* $t_6$ *не сработает, так как переход* $t_6$ *станет готовым сработать в момент времени* 1.3 *и он не сможет больше ждать из-за своих границ времени ожидания срабатывания.*

## 3. РАЗВЁРТКА БВСП

В этом разделе будут представлены ключевые определения и понятия работы [4], а именно: семантика последовательностей локальных срабатываний, временные расширенные процессы, отражающие представленную семантику, а также развёртка БВСП.

### 3.1. Семантика последовательностей локальных срабатываний БВСП

Будем считать, что известно разбиение множества мест $P$ на *множества взаимно исключающих мест* $P_i \subseteq P$ $(i \in \mathbb{N})$, т.е. для любой достижимой маркировки $M$ верно, что пересечение множеств $P_i \cap M$ – множество, содержащее единственное место. Возможность вычисления такого разбиения доказывается в работе [4]. Для всех мест $p \in P$ определим $\bar{p} \stackrel{\text{def}}{=} P_i \backslash \{p\}$. Для БВСП, представленной на рис. 1, разбиением будет $\{p_1, p_3, p_5, p_6\}, \{p_2, p_4\}$. Далее разбиение будет использоваться для выявления мест, отсутствующих в маркировке, если имеется информация не обо всей маркировке. Например, если мы хотим, чтобы сработал переход $t_4$ или $t_5$, мы должны убедиться, что ранее не сработал переход $t_2$, удалив фишку в месте $p_3$. Если же мы знаем, что место $p_2$ входит в маркировку, то мы можем заключить, что место $p_4$ не входит в маркировку и переход $t_2$ не допустим. В этом случае говорят, что переходы $t_4$ и $t_5$ *контекстно* зависят от фишки в месте $p_2$.

*Частичным состоянием* БВСП называют тройку $\langle L, dob, lrd \rangle$, где $L \subseteq M$ – *частичная маркировка*, а функции $dob$, $lrd\colon L \to \mathbb{Q}_{\geq 0}$ соответ-

ственно связывают с каждым местом $p \in L$ время появления в нём фишки и время последнего срабатывания перехода контекстно-зависимого от $p$.

Частичное состояние $\langle L, dob, lrd\rangle$ называют *максимальным частичным состоянием*, если $L$ содержит по одному месту из множеств взаимно исключающих мест. Если $L = M$, то $\langle M, dob, lrd\rangle$ называют *максимальным состоянием. Начальным максимальным состоянием* называют максимальное состояние $\langle M_0, dob_0, lrd_0\rangle$, в котором $M_0$ – начальная маркировка, и для любого $p \in M_0$ верно, что $dob_0(p) \stackrel{\text{def}}{=} lrd_0(p) \stackrel{\text{def}}{=} \theta_0$, где $\theta_0$ – начальный момент времени. Заметим, что максимальное частичное состояние даёт достаточно информации, о возможности срабатывания переходов в нём.

Введём предикат $LFC$, гарантирующий возможность срабатывания перехода $t$ в момент времени $\theta$ из частичного состояния $\langle L, dob, lrd\rangle$ при $lrd(p) \le \theta$ для любого $p \in L$.

**Определение 1.** Пусть предикат $LFC'(L, dob, t, \theta)$ выполнен, тогда и только тогда, когда:

– $t$ допустим при частичной маркировке $L$: ${}^\bullet t \subseteq L$;

– достигнуто минимальное ожидание срабатывания $t$: $\theta \ge doe(t) + efd(t)$;

– переходы, за исключением $t$, которые могут *использовать* фишки из $L$, не допустимы при полной маркировке, или не превышено их максимальное время ожидания срабатывания: $\forall t' \in T \setminus \{t\}\ {}^\bullet t' \cap L \ne \emptyset \Rightarrow \exists p \in {}^\bullet t'\ \bar{p} \cap L \ne \emptyset \vee \theta \le \max_{p \in {}^\bullet t' \cap L} dob(p) + lfd(t')$.

Предикат $LFC(L, dob, t, \theta)$ выполнен тогда и только тогда, когда

$$\begin{cases} \text{предикат } LFC'(L, dob, t, \theta) \text{ выполнен;} \\ \nexists L' \subset L : \text{предикат } LFC'(L', dob|_{L'}, t, \theta) \text{ выполнен.} \end{cases}$$

Переход $t$ *готов сработать в момент времени* $\theta$ *при частичной маркировке* $L$ *из максимального состояния* $\langle M, dob, lrd\rangle$, если для $L \subseteq M$ предикат $LFC(L, dob|_L, t, \theta)$ выполнен, и для любого $p \in L$ $\theta \ge lrd(p)$.

Срабатывание готового перехода $t$ в момент времени $\theta'$ из максимального состояния $s = \langle M,\ dob, lrd\rangle$ приводит к новому максимальному состоянию $s' = \langle (M \setminus {}^\bullet t) \cup t^\bullet, dob', lrd'\rangle$, где

$$dob'(p) \stackrel{\text{def}}{=} \begin{cases} dob(p), \text{ если } p \in M \setminus {}^\bullet t; \\ \theta, \text{ если } p \in t^\bullet, \end{cases} \quad lrd'(p) \stackrel{\text{def}}{=} \begin{cases} lrd(p), \text{ если } p \in M \setminus L; \\ \theta, \text{ если } p \in (L \setminus {}^\bullet t) \cup t^\bullet. \end{cases}$$

Последовательность вида $s_0 \stackrel{(t_1, L_1, \theta_1)}{\longrightarrow} s_1, \ldots, s_{n-1} \stackrel{(t_n, L_n, \theta_n)}{\longrightarrow} s_n$ на-

зывают *последовательностью локальных срабатываний* и обозначают $((t_1, L_1, \theta_1), \ldots, (t_n, L_n, \theta_n))$. Пустую последовательность срабатываний обозначают $\varepsilon$.

**Пример 3.** *В качестве примера рассмотрим БВСП, изображённую на рис. 1, и её последовательность локальных срабатываний* $((t_1, \{p_1\}, 0),\ (t_5, \{p_2, p_3\}, 1.3),\ (t_6, \{p_5\}, 1.3))$. *Очевидно, что для срабатывания переходов* $t_1$ *и* $t_6$ *достаточно информации о наличии фишек только в их входных местах, а для возможности говорить о срабатывании перехода* $t_5$ *недостаточно информации лишь о наличии фишки в месте* $p_3$.

### 3.2. Временные расширенные процессы и развёртка БВСП

Для представления в виде временных процессов последовательностей локальных срабатываний БВСП в [4] было предложено понятие *временных расширенных процессов*, которые обобщают *временные процессы* [8] за счёт введения *расширенных событий*.

*Временной процесс* БВСП – пара $\langle E, \Theta \rangle$, где $E$ – множество *событий* и $\Theta : E \to \mathbb{Q}_{\geq 0}$ – отображение, связывающее с каждым событием $e \in E$ момент времени срабатывания соответствующего перехода в БВСП $tran(e)$. Каждое событие $e \in E$ – это пара $(IN(e), tran(e))$, где $IN(e)$ – это множество пар вида $(in(b), place(b)) \in E \times P$. Здесь $in(b)$ – событие временного процесса, которое соответствует входному переходу места $place(b)$, а $b$ соответствует входному месту $place(b)$ перехода $tran(e)$ в БВСП. Для любого подмножества условий $B$ определено множество $Place(B) \stackrel{\text{def}}{=} \{place(b) \mid b \in B\}$.

**Определение 2.** *Временной расширенный процесс БВСП* – это временной процесс $\langle \dot{E}, \Theta \rangle$, где *расширенное событие* $\dot{e} \in \dot{E}$ – это пара $(\dot{D}, t)$, где $\dot{D} \subseteq B$ состоит из множества *входных условий* ${}^\bullet\dot{e} \stackrel{\text{def}}{=} Place|_{\dot{D}}^{-1}({}^\bullet t)$, а также из *контекстных условий* $\underline{\dot{e}} \stackrel{\text{def}}{=} \dot{D} \setminus {}^\bullet\dot{e}$.

Пусть $\uparrow(\dot{E}) \stackrel{\text{def}}{=} \bigcup_{\dot{e} \in \dot{E}} \dot{e}^\bullet \setminus \bigcup_{\dot{e} \in \dot{E}} {}^\bullet\dot{e}$. Функцию $\dot{\Pi}$, отображающую последовательность локальных срабатываний $((t_1, L_1, \theta_1), \ldots, (t_n, L_n, \theta_n))$ во временной расширенный процесс, определяют так:

– $\dot{\Pi}(\varepsilon) \stackrel{\text{def}}{=} \langle \{\perp\}, \{(\perp, \theta_0)\} \rangle$, где $\perp \stackrel{\text{def}}{=} (\emptyset, -)$ представляет *начальное событие*, а $\theta_0$ – начальный момент времени. Заметим, что начальное событие не является представлением срабатывания реального перехода.

– $\dot{\Pi}((t_1, L_1, \theta_1), \ldots, (t_{n+1}, L_{n+1}, \theta_{n+1})) \stackrel{\text{def}}{=} \langle \dot{E} \cup \{\dot{e}\}, \Theta \cup \{(\dot{e}, \theta_{n+1})\} \rangle$,

где $\langle \dot{E}, \Theta \rangle \stackrel{\text{def}}{=} \dot{\Pi}((t_1,\ L_1,\ \theta_1),\ \ldots,\ (t_n,\ L_n,\ \theta_n))$ и расширенное событие $\dot{e} \stackrel{\text{def}}{=} (Place|^{-1}_{\uparrow(\dot{E})}(L_{n+1}),\ t_{n+1})$ представляет последнее срабатывание в локальной последовательности срабатываний.

Множество всех временных расширенных процессов, содержащих образ $\dot{\Pi}$ из последовательностей локальных срабатываний, обозначают через $\dot{X}$.

Введём отношение на расширенных событиях временного расширенного процесса:

– $\dot{e} \to \dot{f}$, если $\dot{e}^\bullet \cap ({}^\bullet\dot{f} \cup \underline{\dot{f}}) \neq \emptyset$ (*строгая зависимость*);

– $\dot{e} \nearrow \dot{f}$, если $(\dot{e} \to \dot{f}) \vee (\underline{\dot{e}} \cap {}^\bullet\dot{f} \neq \emptyset)$ (*слабая зависимость*).

Через $\to^*$ обозначают рефлексивное и транзитивное замыкание отношения $\to$, которое называют *причинной зависимостью*. Для всех расширенных событий $\dot{e}$ и множеств расширенных событий $\dot{E}$ определяют, соответственно, $\lceil \dot{e} \rceil \stackrel{\text{def}}{=} \{\dot{f} \in \dot{E} \mid \dot{f} \to^* \dot{e}\}$ и $\lceil \dot{E} \rceil \stackrel{\text{def}}{=} \bigcup_{\dot{e} \in \dot{E}} \lceil \dot{e} \rceil$, при этом множество $\lceil \dot{E} \rceil$ называют *причинно-замкнутым*.

**Определение 3.** Развёртка БВСП получается объединением всех расширенных событий из временных расширенных процессов сети: $U \stackrel{\text{def}}{=} \bigcup_{\langle \dot{E}, \Theta \rangle \in \dot{X}} \dot{E}$.

Развёртка БВСП даёт информацию о причинной зависимости, *параллелизме* и *конфликтах* между расширенными событиями. Говорят, что два события $\dot{e}, \dot{f} \in \dot{E}$ находятся в *конфликте*, если существуют различные расширенные события $\dot{g}_1, \dot{g}_2 \in \dot{E}$ такие, что ${}^\bullet\dot{g}_1 \cap {}^\bullet\ \dot{g}_2 \neq \emptyset$ и $\dot{g}_1 \to^* \dot{e},\ \dot{g}_2 \to^* \dot{f}$. Если $\nexists \dot{e}, \dot{f} \in \dot{E} \mid \dot{e} \# \dot{f}$, то множество $\dot{E}$ называют *свободным от конфликтов*. Если расширенные события не находятся в конфликте или причинной зависимости, то они *параллельны*. Также с появлением конфликтов понятие *слабой зависимости* становится шире:

– $\dot{e} \nearrow \dot{f}$, если $(\dot{e} \to \dot{f}) \vee (\underline{\dot{e}} \cap {}^\bullet\dot{f} \neq \emptyset) \vee (\dot{e} \# \dot{f})$.

**Пример 4.** *На Рис. 2 представлен срез развёртки БВСП, изображённой на рис. 1. Дуга из условия $b$ в расширенное событие $\dot{e}$ означает, что $b \in {}^\bullet\dot{e}$. Дуга из расширенного события $\dot{e}$ в условие $b$ обозначает, что $b \in \dot{e}^\bullet$. Если $b \in \underline{\dot{e}}$, то связь между ними обозначает контекстная дуга, представляемая ребром между $b$ и $\dot{e}$.*

Для любого множества условий $B$ таких, что $place|_B$ – инъективно, и для любого отображения $\Theta : \bigcup_{b \in B} \lceil {}^\bullet b \rceil \to \mathbb{Q}_{\geq 0}$, обозначим через $dob_{B,\Theta}$ отображение, определённое следующим образом: $dob_{B,\Theta}(p) \stackrel{\text{def}}{=} \Theta({}^\bullet(place|^{-1}_B(p)))$ для всех $p \in Place(B)$. Также понадобится понятие

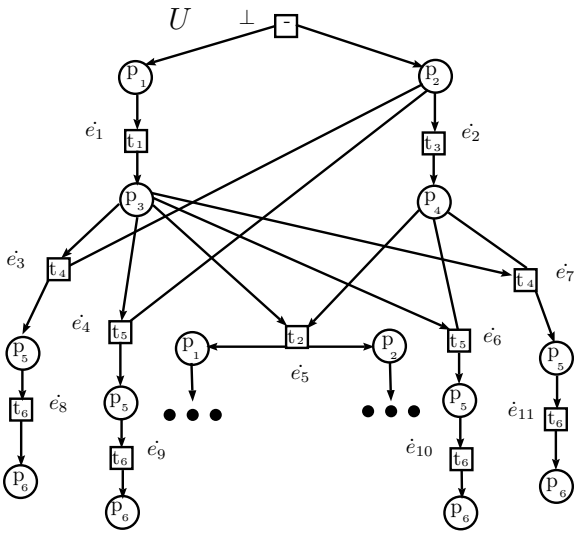



*Рис. 2.* Развёртка БВСП, изображённой на Рис. 1

*конфигурации* расширенных событий $U$. *Конфигурацией* $U$ называется непустое конечное и причинно-замкнутое множество расширенных событий $C$, ацикличное по $\nearrow$ ($\nexists \dot{e}_0, \dot{e}_1, \ldots, \dot{e}_n \in C \ : \ \dot{e}_0 \nearrow \dot{e}_1 \nearrow \ldots \nearrow \dot{e}_n \nearrow \dot{e}_0$).

Следующая теорема показывает связь развёртки и временного расширенного процесса БВСП.

**Теорема 1.** ([4]) *Пусть $\dot{E} \subseteq U$ – подмножество расширенных событий и $\Theta : \dot{E} \to \mathbb{Q}_{\geq 0}$ – отображение, связывающее каждое расширенное событие $\dot{E}$ с некоторым моментом времени. $\langle \dot{E}, \Theta \rangle$ расширенный временной процесс тогда и только тогда, когда выполнены следующие условия:*

$$\begin{cases} \dot{E} \text{ – конфигурация } U; \\ \forall \dot{e}, \dot{e}' \in \dot{E} \dot{e} \nearrow \dot{e}' \Rightarrow \Theta(\dot{e}) \leq \Theta(\dot{e}'); \\ \forall \dot{e} = (B, t) \in \dot{E} \backslash \{\bot\} \text{ предикат } LFC(Place(B), dob_{B,\Theta}, t, \Theta(\dot{e})) \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{выполнен.} \end{cases}$$

## 4. ОБЪЕДИНЕНИЕ РАСШИРЕННЫХ СОБЫТИЙ РАЗВЁРТКИ БВСП

В этом разделе будет показано, как техника слитного процесса развёртки позволяет проводить редукцию развёртки БВСП и как редуцированная таким образом развёртка связана с расширенными временными процессами БВСП.

Пусть $\dot{E} \subseteq U$ – подмножество расширенных событий развёртки БВСП и $\dot{E} = \lceil \dot{E} \rceil$. Для любого $b \in B$ *глубиной условия* $b$ назовём максимальное число $place(b)$-помеченных условий в ориентированном графе представления $\dot{E}$ на различных путях, начинающихся с $\bot$ и заканчивающихся в $b$ (считается, что контекстная дуга всегда направлена от условия к событию). Глубину условия $b$ обозначим $od(b)$. Заметим, что это определение корректно, так как существует по крайней мере один путь в ориентированном графе представления $\dot{E}$, начинающийся с $\bot$ и заканчивающийся в $b$, и число всех таких возможных путей конечно.

**Определение 4.** Пусть $\dot{E} \subseteq U$ – подмножество расширенных событий развёртки БВСП и $\dot{E} = \lceil \dot{E} \rceil$. Кортеж $\mu_{\dot{E}} \stackrel{\text{def}}{=} \langle \hat{B}, \hat{E}, \bar{h} \rangle$ назовём *$\mu$-сетью* $\dot{E}$, где $\hat{B} \subseteq P \times \mathbb{N}, \hat{E} \subseteq T \times 2^{\hat{B}} \times 2^{\hat{B}} \times 2^{\hat{B}}$, $\bar{h}$ – гомоморфизм из $\dot{E}$ в $\mu_{\dot{E}}$ и выполнены следующие условия:

– $\forall b \in B$, $\bar{h}(b) \stackrel{\text{def}}{=} \langle place(b), od(b) \rangle$; $\hat{B} \stackrel{\text{def}}{=} \bar{h}(B)$ назовём множеством $\mu$-условий;

– $\forall \dot{e} \in \dot{E}$, $\bar{h}(e) \stackrel{\text{def}}{=} \langle tran(\dot{e}), \bar{h}({}^\bullet\dot{e}), \bar{h}(\underline{\dot{e}}), \bar{h}(\dot{e}^\bullet) \rangle$; $\hat{E} \stackrel{\text{def}}{=} \bar{h}(\dot{E})$ назовём множеством $\mu$-событий;

– $\forall \hat{e} = \langle t, Pre, Cont, Post \rangle \in \hat{E}$ выполнено ${}^\bullet\hat{e} = Pre$, $\underline{\hat{e}} = Cont$, $\hat{e}^\bullet = Post$;

Неформально, $\mu_{\dot{E}}$ получается из $\dot{E}$ за два шага. На первом шаге объединяются условия, одинаково $place$-обозначенные и имеющие одинаковую глубину в $\dot{E}$. На втором шаге объединяются расширенные события, одинаково $tran$-обозначенные и имеющие одинаковые отношения с объединёнными условиями. Заметим, что при построении $\mu_{\dot{E}}$ объединяются расширенные события $\dot{E}$, находящиеся в $\dot{E}$ в конфликте.

**Пример 5.** *На рис. 3 изображён срез $\mu$-сети развёртки на рис. 2. Видно, что четыре расширенных события $\dot{e_8}, \dot{e_9}, \dot{e_{10}}, \dot{e_{11}}$, находящиеся в конфликте в развёртке, объединены в $\mu$-сети развёртки в одно $\mu$-событие $\hat{e_8}$.*

**Определение 5.** Пусть $\langle \dot{E}, \Theta \rangle$ – временной расширенный процесс БВСП. Тогда пару $\langle \mu_{\dot{E}}, \hat{\Theta} \rangle$, где отображение $\hat{\Theta} : \hat{E} \to \mathbb{Q}_{\geq 0}$ такое, что $\hat{\Theta}(\bar{h}(\dot{e})) =$

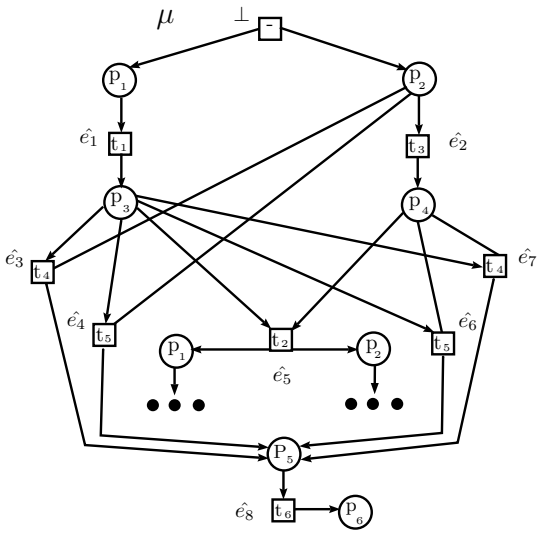



*Рис. 3.* $\mu$-сеть развёртки БВСП, изображённой на Рис. 2

$\Theta(\dot{e})$, назовём *временным слитным процессом БВСП.*

**Определение 6.** Множество $\mu$-событий $\hat{C}$ назовём $\mu$-конфигурацией $\mu$-сети развёртки $U$, если существует конфигурация $C$ в $U$, для которой выполнено равенство $\bar{h}(C) = \hat{C}$.

Определение $\mu$-конфигурации не даёт прямого способа её нахождения в $\mu$-сети. Следующее утверждение позволяет выделять $\mu$-конфигурации в $\mu$-сети.

**Утверждение 1.** *Пусть $\mu_U$ – $\mu$-сеть развёртки БВСП. Тогда непустое конечное множество $\mu$-событий $\hat{C}$ будет $\mu$-конфигурацией $\mu_U$ тогда и только тогда, когда выполнены следующие условия:*

$$\begin{cases} \hat{C} \text{ ациклично по } \nearrow; \\ \forall \hat{e} \in \hat{C} : \forall \hat{c} \in {}^\bullet\hat{e} \cup \underline{\hat{e}} : (\hat{c} \in \hat{\perp}^\bullet \vee \exists \hat{e}' \in {}^\bullet\hat{c} : \hat{e}' \in \hat{C}); \\ \forall k \geq 1, \text{ если } \langle p, k+1\rangle \in \hat{C}^\bullet, \text{ то } \langle p, k\rangle \in \hat{\perp}^\bullet \cup \hat{C}^\bullet \text{ и существует путь} \\ \quad \text{между } \langle p, k\rangle \text{ и } \langle p, k+1\rangle \text{ в направленном графе представления } \hat{C}. \end{cases}$$

*Доказательство.* Утверждение является следствием определения конфигурации развёртки. □

Пусть $\mu_U = \langle \hat{B}, \hat{E}, \bar{h} \rangle$ – $\mu$-сеть развёртки БВСП. Если $\hat{B}' \subseteq \hat{B}, \hat{E}' \subseteq \hat{E}$ такие, что $place|_{\hat{B}'}$ – инъективно и $\forall p \in Place(\hat{B}')\ |^\bullet(place|_{\hat{B}'}^{-1}(p)) \cap \hat{E}'| = 1$, тогда для произвольного отображения $\hat{\Theta} : \hat{E}' \to \mathbb{Q}_{\geq 0}$ обозначим через $dob_{\hat{B}', \hat{E}', \hat{\Theta}}$ отображение, определённое так:

$$\forall p \in Place(\hat{B}'), dob_{\hat{B}', \hat{E}', \hat{\Theta}}(p) \stackrel{\text{def}}{=} \hat{\Theta}(^\bullet(place|_{\hat{B}'}^{-1}(p)) \cap \hat{E}').$$

**Теорема 2.** *Пусть $\mu_U = \langle \hat{B}, \hat{E}, \bar{h} \rangle$ – $\mu$-сеть развёртки БВСП и $\hat{E}' \subseteq \hat{E}$. Отображение $\hat{\Theta} : \hat{E}' \to \mathbb{Q}_{\geq 0}$ связывает каждое $\mu$-событие $\hat{E}'$ с некоторым моментом времени. $\langle \hat{B}|_{\hat{E}'}, \hat{E}', \bar{h}|_{\hat{E}'}, \hat{\Theta} \rangle$ – временной слитный процесс тогда и только тогда, когда выполнены следующие условия:*

$$\begin{cases} \hat{E}' - \mu\text{-конфигурация } \mu_U; \\ \forall \hat{e}, \hat{e}' \in \hat{E}'\ \hat{e} \nearrow \hat{e}' \Rightarrow \hat{\Theta}(\hat{e}) \leq \hat{\Theta}(\hat{e}'); \\ \forall \hat{e}' = \langle t, Pre, Cont, Post \rangle \in \hat{E}' \backslash \{\hat{\perp}\} \\ \quad \text{предикат } LFC(Place(Pre \cup Cont), dob_{Pre \cup Cont, \hat{E}', \hat{\Theta}}, t, \hat{\Theta}(\hat{e}')) \\ \quad \text{выполнен.} \end{cases}$$

*Доказательство.* Пусть $\langle \hat{B}', \hat{E}', \bar{h}, \hat{\Theta} \rangle$ – временной слитный процесс. Следовательно, существует временной расширенный процесс $\langle \dot{E}', \Theta \rangle$, где отображение $\Theta$ такое, что $\hat{\Theta}(\bar{h}(\dot{e})) = \Theta(\dot{e})$, а $\bar{h}(\dot{E}') = \hat{E}$. Из теоремы 1 следует, что $\dot{E}'$ – конфигурация $U$ и выполнены два последних условия теоремы для $\dot{E}'$ и $\Theta$. По определению 6, получаем, что $\hat{E}'$ $\mu$-конфигурация $\mu_U$, и, в силу связи $\hat{\Theta}$ и $\Theta$, следует выполнение двух последних условий теоремы для $\hat{E}'$ и $\hat{\Theta}$.

Пусть выполнены условия теоремы. Покажем, что $\langle \hat{B}|_{\hat{E}'}, \hat{E}', \bar{h}|_{\hat{E}'}, \hat{\Theta} \rangle$ – временной слитный процесс. Для этого покажем, что существует временной расширенный процесс $\langle \dot{E}', \Theta \rangle$ и $\langle \hat{B}|_{\hat{E}'}, \hat{E}', \bar{h}|_{\hat{E}'}, \hat{\Theta} \rangle = \langle \mu_{\dot{E}'}, \hat{\Theta} \rangle$, $\hat{\Theta}(\bar{h}(\dot{e})) = \Theta(\dot{e})$. Раз $\hat{E}'$ – $\mu$-конфигурация $\mu_U$, то по определению 6 существует $\dot{E}'$ – конфигурация $U$ такая, что $\bar{h}(\dot{E}') = \hat{E}'$. При этом, если $\Theta$ определить на $\dot{E}'$ так, что $\forall\ \dot{e} \in \dot{E}'\ \Theta(\dot{e}) = \Theta(\bar{h}(\dot{e}))$, то, в силу выполнения последних двух условий теоремы для $\hat{\Theta}$, по теореме 1 $\langle \dot{E}', \Theta \rangle$ будет искомым временным расширенным процессом. □

**Пример 6.** *Рассмотрим $\mu$-сеть развёртки на рис. 3. Возьмём подмножество $\mu$-событий $\hat{E}' = \{\hat{\perp}, \hat{e}_1, \hat{e}_4, \hat{e}_8\}$ и функцию $\hat{\Theta} : \hat{E}' \to \mathbb{Q}_{\geq 0}$ такую, что $\hat{\Theta}(\hat{\perp}) = 0, \hat{\Theta}(\hat{e}_1) = 0, \hat{\Theta}(\hat{e}_4) = 1.3, \hat{\Theta}(\hat{e}_8) = 1.3$. Очевидно, кортеж*

*$\langle \hat{B}|_{\hat{E}}, \hat{E}', \bar{h}|_{\hat{E}}, \hat{\Theta}\rangle$ удовлетворяет условиям теоремы 2, поэтому является временным слитным процессом, и существует локальная последовательность срабатываний исходной БВСП, порождающая временной расширенный процесс, $\bar{h}$-образ которого даёт построенный временной слитный процесс.*

*Нетрудно заметить, что множество $\mu$-событий $\hat{E}'$ будет $\mu$-конфигурацией по утверждению 1, а значение функции $\hat{\Theta}$ удовлетворяет условиям теоремы 2.*

*В результате, соответствующая последовательность локальных срабатываний является корректной и выглядит так: $\sigma = ((t_1, \{p_1\}, 0), (t_5, \{p_2, p_3\}, 1.3), (t_6, \{p_5\}, 1.3))$.*

*Напротив, временной слитный процесс $\langle \hat{E}', \hat{\Theta}\rangle$, соответствующий $\sigma$, удовлетворяет условиям теоремы 2.*

## 5. ЗАКЛЮЧЕНИЕ

В работе было проведено исследование особенностей развёртки БВСП перед развёрткой БСП с целью расширения определения слитного процесса развёртки. Такие особенности были выявлены в виде контекстных дуг и временной компоненты. Было сформулировано конструктивное определение слитного процесса развёртки БВСП, где была учтена особенность контекстных дуг, а также сформулирована и доказана связь слитного процесса развёртки и временного слитного процесса БВСП, что даёт понятие временной компоненты для слитного процесса развёртки.

Дальнейшим развитием работы может стать расширение для случая БВСП алгоритма построения слитного процесса развёртки напрямую по БСП [12], называемого *распутыванием*, что позволит полностью использовать преимущество распутывания перед развёртыванием.

## СПИСОК ЛИТЕРАТУРЫ

1. T. Murata: Petri nets, properties, analysis and applications. Proceedings of the IEEE 77(4) (1989) 541–580.
2. B. Berthomieu, M. Diaz: Modeling and verification of time dependent systems using time Petri nets. IEEE Transactions on Software Engineering 17(3) (1991) 259–273.
3. A. Valmari: The state explosion problem. In: W. Reisig, G. Rozenberg (Eds.), Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science 1491 (1998) 429–528.
4. T. Chatain, C. Jard: Complete finite prefixes of symbolic unfoldings of safe time Petri nets. In: ICATPN'06 Lecture Notes in Computer Science 4024 (2006) 125–145.

5. V. Khomenko, A. Kondratyev, M. Koutny, W. Vogler: Merged processes – a new condensed representation of Petri net behaviour. In: CONCUR'05 Lecture Notes in Computer Science 3653 (2005) 338–352.
6. C. Rodriguez, S. Schwoon, V. Khomenko: Contextual merged processes. Lecture notes in computer science 7927 (2013) 29–48.
7. M. Nielsen, G. Plotkin, G. Winskel: Petri nets, event structures and domains, Part I. Theoretical Computer Science 13(1) (1981) 85–108.
8. T. Aura, J. Lilius: Time processes for time Petri nets. In: ICATPN'97 Lecture Notes in Computer Science 1248 (1997) 136–155.
9. P. Baldan, A. Corradini, U. Montanari: Contextual Petri nets, asymmetric event structures, and processes. Information and Computation 171(1) (2001) 1–49.
10. J. Engelfriet: Branching processes of Petri nets. Acta Informatica 28 (1991) 575–591.
11. K. L. McMillan: A technique of state space search based on unfolding. Formal Methods in System Design 6(1) (1995) 45–65.
12. V. Khomenko, A. Mokhov: An algorithm for direct construction of complete merged processes. Lecture Notes in Computer Science 6709 (2011) 89–108.

**Е.К. Ерофеев**

# АЛГЕБРАИЧЕСКИЕ РЕШЁТКИ ПЕРВИЧНЫХ СТРУКТУР СОБЫТИЙ*

## ВВЕДЕНИЕ

В теории параллельных систем и процессов для представления и изучения поведения параллельных и распределенных систем применяется ряд абстрактных моделей (например, частично-упорядоченные множества, сети-процессы, структуры событий и т. д.). При помощи этих моделей был установлен ряд фундаментальных фактов, которые позволили лучше понять природу и закономерности параллельных вычислений. Например, К. Петри в своих классических работах [1, 2] ввел ряд «аксиом параллельности» (в частности, свойства $K$-плотности, $N$-плотности, $D$-непрерывности и т. д.), которые были подробно изучены в контексте сетей-процессов [3, 4, 5, 6] и частично-упорядоченных множеств, наделенных двумя базовыми отношениями между событиями моделируемой системы: причинной зависимостью и параллелизмом. В работе [7] эти результаты были обобщены для класса сетей-процессов с причинной зависимостью и недетерминированным выбором. Этот подход получил дальнейшее развитие в работах [8, 9], в которых смысл «аксиом параллельности» был определен в контексте более общих моделей первичных и локальных структур событий.

Актуальным подходом к изучению взаимосвязей базовых отношений событий параллельных систем является представление семантики моделей систем в терминах решеток. В работах [10, 11, 12] исследованы решетки частично-упорядоченных множеств, ассоциированных с сетями-процессами. В работе [13] были предложены методы построения ортомодулярных решеток – комбинаторного представления пространства-времени – конфигураций первичных структур событий, которые являются обобщением моделей частично-упорядоченных множеств и сетей-процессов.

В данной работе предлагаются способы построения алгебраических решеток конфигураций первичных структур событий посредством вве-

*Данная работа работа частично финансируется DFG и РФФИ (проект CAVER, грант N BE 1267/14-1, грант No 14-01-91334).

дения понятия причинно-замкнутых множеств, представляющих собой множества событий, замкнутые относительно правила срабатывания. Показана взаимосвязь причинно-замкнутых множеств событий с замкнутыми множествами, которые рассматривались в работе [13], а также продолжено изучение «аксиом параллельности», а именно $K$-плотности и ее характеризации в терминах теории решеток. Важность аксиомы $K$-плотности заключается в том, что она позволяет различать семантически верные модели параллельных систем и теоретически допустимые.

Статья организована следующим образом. В разделе 1 приведены необходимые понятия и используемые факты из теории решеток. Синтаксис и семантика первичных структур событий описаны в разделе 2. В разделе 3 строятся алгебраические решетки конфигураций первичных структур событий.

## 1. ЭЛЕМЕНТЫ ТЕОРИИ РЕШЕТОК

В данном разделе приведены используемые в работе понятия и факты из теории решеток.

**Определение 1.** *Если $X$ – множество, а $\mathbb{P}(X)$ – множество всех подмножеств $X$, то отображение $\mathcal{C} : \mathbb{P}(X) \to \mathbb{P}(X)$ называется оператором замыкания на $X$, если для всех $A \subseteq X, B \subseteq X$ выполнены следующие условия:*

*1.* $A \subseteq \mathcal{C}(A)$,

*2.* $A \subseteq B \Rightarrow \mathcal{C}(A) \subseteq \mathcal{C}(B)$,

*3.* $\mathcal{C}(\mathcal{C}(A)) = \mathcal{C}(A)$.

*Решеткой* $\mathcal{L}$ называется частично-упорядоченное множество $(L, \leqslant)$, в котором любая пара элементов имеет наибольшую нижнюю грань (обозначается $\bigwedge$) и наименьшую верхнюю грань (обозначается $\bigvee$). Решетка $\mathcal{L}$ является *полной*, если любое подмножество множества $L$ имеет наибольшую нижнюю и наименьшую верхнюю грани.

Пусть $\mathcal{L} = (L, \leqslant)$ является полной решеткой и $k \in L$. Элемент $k$ называется *компактным*, если для любого множества $S \subseteq L$ такого, что $k \leqslant \bigvee S$, выполнено $k \leqslant \bigvee T$ для некоторого конечного $T \subseteq S$. Множество компактных элементов решетки $\mathcal{L}$ обозначается $K(\mathcal{L})$.

Полная решетка $\mathcal{L}$ называется *алгебраической*, если $x = \bigvee\{k \in K(\mathcal{L}) \mid k \leqslant x\}$ для всех $x \in L$.

Пусть $X$ является множеством, и $\mathcal{X}$ – семейством подмножеств множества $X$. Семейство $\mathcal{X}$ называется *замкнутым по пересечению*, ес-

ли для любого семейства $(A_i)_{i\in I}$, принадлежащего $\mathcal{X}$, его пересечение $\bigcap_{i\in I} A_i$ также принадлежит $\mathcal{X}$. Назовем частично-упорядоченное множество $(\mathcal{X}, \subseteq)$ *системой замыкания*, если $\mathcal{X}$ замкнуто по пересечению.

В работе [14] установлен следующий факт.

**Утверждение 1.** *Каждая система замыкания является полной решеткой.*

Непустое подмножество $\mathcal{A} \subseteq \mathcal{X}$ называется *направленным*, если для любого конечного подмножества $\mathcal{B} \subseteq \mathcal{A}$ существует $A \in \mathcal{A}$ такой, что $B \subseteq A$ для всех $B \in \mathcal{B}$. *Направленным объединением* будем называть объединение элементов направленного подмножества $\mathcal{A} \subseteq \mathcal{X}$. Непустое семейство $\mathcal{X}$ называется *замкнутым относительно направленного объединения*, если для любого направленного семейства $\mathcal{A} = \{A_i\}_{i\in I} \subseteq \mathcal{X}$ выполнено $\bigcup_{i\in I} A_i \in \mathcal{X}$.

В работе [14] показана справедливость следующего утверждения.

**Утверждение 2.** *Каждая система замыкания $(\mathcal{X}, \subseteq)$, в которой $\mathcal{X}$ замкнуто относительно направленного объединения, является алгебраической решеткой.*

## 2. СТРУКТУРЫ СОБЫТИЙ И ИХ СВОЙСТВА

В данном пункте приводятся определения базовых понятий теории (первичных) структур событий, а также устанавливаются необходимые в дальнейшем факты. Структуры событий описывают параллельную систему в виде множества событий, которые представляют собой выполнение некоторых действий. Поведение структур событий описывается в терминах конфигураций, т. е. множеств событий, произошедших в системе. Конфигурацию можно также понимать как состояние, достигнутое системой после того, как произошли все события из этой конфигурации.

**Определение 2.** *Первичной структурой событий называется тройка $ES = (E, \preceq, \#)$, в которой*

1. *$E$ – это множество событий,*
2. *$\preceq \subseteq E \times E$ – отношение причинной зависисмости (частичный порядок), удовлетворяющее «принципу конечности причин»: $\forall e \in E : |\{d \in E \mid d \preceq e\}| < \infty$,*
3. *$\# \subseteq E \times E$ – симметричное иррефлексивное отношение, удовлетворяющее «принципу наследования конфликта»: $\forall e_1, e_2, e_3 \in E : (e_1 \# e_2 \preceq e_3) \Rightarrow (e_1 \# e_3)$.*

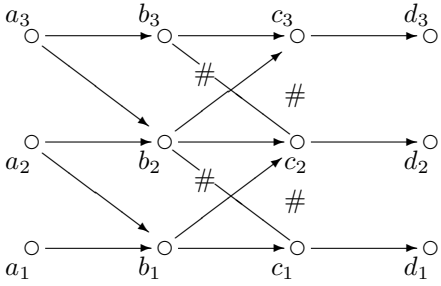



*Рис. 1.* Пример структуры событий

Отношением *параллелизма* между событиями из множества $E$ структуры событий $ES = (E, \preceq, \#)$ называется бинарное отношение $\smile = (E \times E) \backslash (\preceq \cup \succeq \cup \#)$. Если два события *причинно связаны*, то они находятся в отношении $li = \preceq \cup \succeq$ между собой. Отметим, что отношения $\#, \smile, li$ симметричные, а $\#$ и $\smile$ являются также иррефлексивными, в то время как $= \subseteq (\preceq \cup \succeq)$, что означает рефлексивность отношения $li$.

В качестве примера рассмотрим первичную структуру событий, представленную на рис. 1. Множество событий структуры включает двенадцать событий: $a_i, b_i, c_i, d_i$, где $i \in \{1, 2, 3\}$. Нетрудно заметить, что $(a_i \preceq b_i \preceq c_i \preceq d_i)$ для $i \in \{1, 2, 3\}$, $(a_3 \preceq b_2 \preceq c_3)$ и $(a_2 \preceq b_1 \preceq c_2)$. На рис. 1 видно, что $(b_3 \# c_2 \# c_3)$ и $(b_2 \# c_1 \# c_2)$. Из принципа наследования конфликта следует, что $(b_3 \# d_2)$, $(c_3 \# d_2)$, $(d_3 \# c_2)$ и т. д. В свою очередь, например, события $a_1, a_2, a_3$ не связаны между собой причинно и попарно не являются конфликтными, а значит $(a_1 \smile a_2 \smile a_3 \smile a_1)$.

Структура событий функционирует, переходя из одного состояния в другое. Состояния структуры событий называются конфигурациями. В этой работе будут рассмотрены COF-конфигурации, включающие события, находящиеся в отношении причинной зависимости или недетерминированного выбора (конфликта), и CFF-конфигурации, содержащие события, связанные причинно или параллельные.

**Определение 3.** *Множество событий $P \subseteq E$ первичной структуры событий $ES = (E, \preceq, \#)$, называется*

- *COF-конфигурацией, если $P$ – левозамкнутое (т. е. $(e \in P \wedge d \preceq e) \Rightarrow d \in P$) и свободное от параллелизма множество (т. е. для любых $e, d \in P$ выполнено $\neg(e \smile d)$),*
- *CFF-конфигурацией, если $P$ – левозамкнутое и свободное от*

*конфликта множество (т. е. для любых* $e, d \in P$ *выполнено* $\neg(e \# d)$*).*

Из определения отношения $\smile$ следует, что если множество $P \subseteq E$ свободно от параллелизма, то для любых $c, d \in P$ выполнено $(e\ li\ d) \vee (e \# d)$.

В структуре событий на рис. 1 COF-конфигурацией является, например, множество событий $\{a_2, b_2, c_i, d_i \mid i = 1, 2, 3\}$, в то время как множество событий $\{a_i, b_i, c_3 \mid i = 1, 2, 3\}$ является CFF-конфигурацией.

Здесь и далее (за исключением случаев, оговоренных особо) будем полагать, что $ES = (E, \preceq, \#)$ является первичной структурой событий, а $P \subseteq E$ – это $*$-конфигурация структуры $ES$, где $* \in \{COF, CFF\}$. Под $V_* \in \{\#, \smile\}$ для $* \in \{COF, CFF\}$ будем понимать отношение первичной структуры событий, т. ч. $V_{COF} = \#$ и $V_{CFF} = \smile$.

Пусть $\mathcal{P}(P)$ – это множество всех подмножеств $P$. Для $* \in \{COF, CFF\}$ определим оператор $(\cdot)^{\perp *} : \mathcal{P}(P) \to \mathcal{P}(P)$ по следующему правилу: для каждого $A \subseteq P$ будем полагать $A^{\perp *} = \{a' \in P \mid \forall a \in A\ (a\ V_*\ a')\}$. В работе [13] было показано, что оператор $(\cdot)'_* = (\cdot)^{\perp * \perp *} : \mathcal{P}(P) \to \mathcal{P}(P)$ является оператором замыкания на множестве $P$. Будем называть множество событий $A \subseteq P$ *замкнутым в* $*$*-конфигурации* $P$, если оно совпадает со своим замыканием: $A = A'_*$.

**Определение 4.** *Пусть* $A \subseteq E' \subseteq E$ *и отношение* $V \in \{li, \#, \smile\}$. *Множество событий* $A \subseteq E' \subseteq E$ *называется*

- $V$*-множеством в* $E'$*, если* $(a_1\ V\ a_2)$ *для всех* $a_1, a_2 \in A$,
- $V$*-сечением в* $E'$*, если* $A$ *– максимальное* $V$*-множество в* $E'$.

Для $*$-конфигурации $P$ справедлива

**Лемма 1.** *Если* $C$ *является* $V_*$*-множеством в* $*$*-конфигурации* $P$*, и* $A \subseteq C$*, то* $C \subseteq A \cup A^{\perp *}$.

*Доказательство.* Ясно, что $C = A \cup (C \backslash A)$. В силу того, что $C$ является $V_*$-множеством, для произвольного $c \in C \backslash A$ справедливо $(c\ V_*\ a)$ для всех $a \in A$. Это означает, что $c \in A^{\perp *}$, то есть $(C \backslash A) \subseteq A^{\perp *}$. Следовательно, $C \subseteq A \cup A^{\perp *}$. □

В используемых обозначениях верна

**Лемма 2.** *Пусть* $P$ *является* $*$*-конфигурацией. Если* $A$ *является* $V_*$*-множеством в* $P$*, то* $A$ *–* $V_*$*-сечение в* $A'_*$.

*Доказательство.* Предположим, что утверждение не выполнено, то есть существует $b \in A'_* \backslash A$ такое, что $(b\ V_*\ a)$ для всех $a \in A$. Это означает,

что $b \in A^{\perp}*$, что невозможно в силу иррефлексивности отношения $V_*$ и того факта, что $b \in A'_*$. □

Говорят, что события $e, f \in P$ *непосредственно причинно связаны* ($e \prec\!\!\cdot f$), если выполняется $(e \preceq f) \wedge (\forall g \in P : \ (e \preceq g \prec f) \Rightarrow (e = g))$. Для события $e \in P$ множества *непосредственных предшественников* и *непосредственных последователей* обозначаются как ${}^\bullet e = \{e' \in P \mid e' \prec\!\!\cdot\ e\}$ и $e^\bullet = \{e' \in P \mid e \prec\!\!\cdot\ e'\}$, соответственно. Для пары событий $e, f \in P$ множество событий, находящихся причинно между ними, обозначается $[e, f] = \{g \in P \mid e \preceq g \preceq f\}$. Заметим, что если структура событий $ES = (E, \preceq, \#)$ удовлетворяет принципу конечности причин, то для любых $e, f \in P$ выполнено $|[e, f]| < \infty$.

**Определение 5.** *Множество событий $A \subseteq P$ называется* выпуклым в $P$, *если для любых $e, f \in A$ справедливо $[e, f] \subseteq A$.*

Далее будут подробно рассмотрены причинно-замкнутые множества событий. Эти множества можно понимать как множества событий, замкнутые относительно правила срабатывания вперед и назад.

**Определение 6.** *Множество $A \subseteq P$ является* причинно-замкнутым в $P$, *если*

1. *$A$ – выпуклое,*
2. *для любого $e \in P$ справедливо: ${}^\bullet e \neq \varnothing \wedge \ {}^\bullet e \subseteq A \Rightarrow e \in A$ и $e^\bullet \neq \varnothing \wedge e^\bullet \subseteq A \Rightarrow e \in A$.*

Множество всех причинно-замкнутых множеств событий в $P$ будем обозначать $CC(P)$.

В работе [13] были изучены замкнутые множества событий в конфигурациях. Следующая лемма устанавливает их взаимосвязь с причинно-замкнутыми множествами событий.

**Лемма 3.** *Если множество событий $A \subseteq P$ является замкнутым в $*$-конфигурации $P$, то оно – причинно-замкнутое в $P$.*

*Доказательство.* Проверим выполнение условий причинно-замкнутого множества для $A$.

1. Пусть $a_1, a_2 \in A$ и при этом $[a_1, a_2] \nsubseteq A$. Значит, существует $a_3 \in [a_1, a_2]$ такое, что $a_3 \notin A$. Предположим, что $(a_3 \ V_* \ a')$ для всех $a' \in A^{\perp}*$. Тогда, в силу замкнутости $A$ в $P$, получаем $a_3 \in A$, что противоречит выбору $a_3$. Следовательно, существует $a_4 \in A^{\perp}*$ такое, что $\neg(a_4 \ V_* \ a_3)$, то есть $(a_4 \ li \ a_3)$. Если $(a_3 \preceq a_4)$, то по транзитивности $\preceq$ из $(a_1 \preceq a_3 \preceq a_4)$ имеем $(a_1 \preceq a_4)$, а это противоречит иррефлексивности $V_*$ и тому факту, что $a_1 \in$

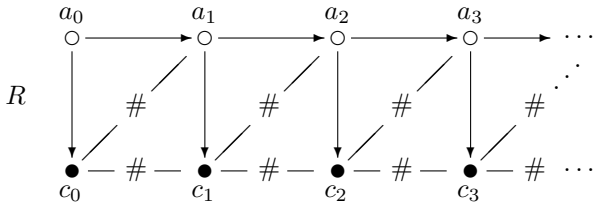



*Рис. 2.* Пример причинно-замкнутого множества, не являющегося замкнутым

$A$, $a_4 \in A^{\perp}*$. Если же $(a_4 \preceq a_3)$, то по транзитивности $\preceq$ получаем $(a_4 \preceq a_2)$ – снова противоречие. Значит, множество $A$ является выпуклым.

2. Предположим, что второе условие причинно-замкнутого множества не выполнено для $A$, то есть существует $e \in P$ такое, что $e^{\bullet} \neq \varnothing \wedge e^{\bullet} \subseteq A$ и при этом $e \notin A$ (случай ${}^{\bullet}e \neq \varnothing \wedge {}^{\bullet}e \subseteq A \wedge e \notin A$ симметричен). Так как $e^{\bullet} \subseteq A$, то $e \notin A^{\perp}*$. Если $(e\ V_*\ f)$ для всех $f \in A^{\perp}*$, то $e \in A'_*$, но по условию $A$ – замкнутое множество, то есть получили противоречие выбору $e \notin A$. Следовательно, существует $f \in A^{\perp}*$ такое, что $\neg(e\ V_*\ f)$. Если $(f \preceq e)$, то в силу транзитивности $\preceq$ имеем $(f \preceq g)$ для каждого $g \in e^{\bullet}$. Это противоречит $f \in A^{\perp}* \wedge g \in A$. Пусть $(e \preceq f)$. Так как $f \in A^{\perp}*$ и $e^{\bullet} \subseteq A$, то $[e, f]\backslash\{e, f\} \neq \varnothing$. Это означает, что для некоторого $g \in e^{\bullet}$ выполнено $g \in [e, f]$. Следовательно, $(g \preceq f)$, что противоречит $g \in A \wedge f \in A^{\perp}*$.

□

Заметим, что обратное утверждение в общем случае не является верным. Так, например, в COF-конфигурации $R$ на рис. 2 множество событий $C = \{c_i \mid i \in \mathbb{N}\}$ является причинно-замкнутым, но $C^{\perp_{COF}} = \varnothing$ и $C'_{COF} = R \neq C$, что означает незамкнутость множества $C$ в $R$.

**Определение 7.** *$*$-конфигурация $P$ называется $K$-плотной, если $|A \cap B| = 1$ для любых li-сечения $A$ и $V_*$-сечения $B$ в $P$.*

В дальнейшем нам понадобится следующая техническая

**Лемма 4.** *Пусть $P$ является $*$-конфигурацией и $a, b \in P$. Если $(a \preceq b) \wedge (a \neq b)$, то существует непустая конечная последовательность $\{e_1, e_2, \ldots, e_k\}$ $(k \geqslant 0)$ такая, что $a \prec e_1 \prec e_2 \prec \ldots \prec e_k \prec b$. Если при этом $a, b \in A$, где $A$ – причинно-замкнутое множество в $P$, то $\{e_1, e_2, \ldots e_k\} \subseteq A$.*

*Доказательство.* В силу принципа конечности причин имеем $|F = \{e \mid a \preceq e \preceq b\}| \leqslant |\{e \mid e \preceq b\}| < \infty$. Так как $a \neq b$, то $|F| > 1$. Значит, существует конечная непустая последовательность $a \prec e_1 \prec e_2 \prec \ldots \prec e_k \prec b$.

Если $a, b \in A$ и $A$ –причинно-замкнутое множество, то из свойства выпуклости получаем $\{a, e_1, \ldots e_k, b\} \subseteq A$. □

**Определение 8.** *$*$-конфигурация $P$ называется $V_*V_*$-свободной, если* ${}^\bullet e_1 \cap {}^\bullet e_2 \neq \varnothing \;\Rightarrow\; |{}^\bullet e_1| = |{}^\bullet e_2| = 1$ *для всех* $e_1, e_2 \in P$.

В используемых обозначениях справедлива

**Лемма 5.** *Каждая COF-конфигурация $P$ первичной структуры событий $ES = (E, \preceq, \#)$ является $V_{COF}V_{COF}$-свободной.*

*Доказательство.* Предположим, что для COF-конфигурации $P$ утверждение не выполнено. Это означает, что существуют события $e_1, e_2 \in P$ такие, что ${}^\bullet e_1 \cap {}^\bullet e_2 \neq \varnothing$ и при этом $|{}^\bullet e_1| \geqslant 2$. Тогда найдутся события $f_1, f_2 \in {}^\bullet e_1$ такие, что $f_1 \neq f_2$. В силу того, что $f_1 \prec e_1$, верно отрицание $\neg(f_1 \preceq f_2)$. Аналогично получаем $\neg(f_2 \preceq f_1)$. Отсюда следует, что $(f_1 \# f_2)$, так как $P$ является COF-конфигурацией. Тогда из принципа наследования конфликта получаем $(e_1 \# e_1)$, что противоречит иррефлексивности отношения $\#$. □

## 3. РЕШЕТКИ, ПОРОЖДАЕМЫЕ КОНФИГУРАЦИЯМИ СТРУКТУР СОБЫТИЙ

В данном разделе рассматривается методика построения алгебраических решеток конфигураций структур событий, а также взаимосвязь этих решеток с аксиомой $K$-плотности.

**Определение 9.** *Пусть $\mathcal{P}(P)$ – множество всех подмножеств $*$-конфигурации $P$. Определим оператор $\phi : \mathcal{P}(P) \to \mathcal{P}(P)$ по правилу $\phi(A) = \bigcap\{D_i \mid D_i$ – причинно-замкнутое в $P$ и $A \subseteq D_i\}$ для всех $A \subseteq P$.*

Очевидно, что $\phi(A)$ является причинно-замкнутым множеством в $*$-конфигурации $P$.

**Лемма 6.** *Пусть $P$ является $K$-плотной и $V_*V_*$-свободной $*$-конфигурацией. Если $Y$ – причинно-замкнутое множество в $P$ и $A$ является $V_*$-сечением в $Y$, то $\phi(A) = Y$.*

*Доказательство.* Согласно определению, $\phi(A)$ – наименьшее причинно-замкнутое множество, содержащее $A$. Следовательно, $\phi(A) \subseteq Y$.

Покажем теперь, что $Y \subseteq \phi(A)$. Предположим, что это неверно, то есть существует $z_0 \in Y \backslash \phi(A)$. Так как по условию $A$ является $V_*$-сечением в $Y$, то для некоторого $a_0 \in A$ выполнено $\neg(z_0\ V_*\ a_0)$. Следовательно, $(z_0\ li\ a_0)$. Рассмотрим случай, когда $(z_0 \preceq a_0)$ (случай $(a_0 \preceq z_0)$ симметричен). По лемме 4 существует конечная непустая последовательность $z_0 \prec\cdot\ e_1 \prec\cdot\ \ldots \prec\cdot\ e_{m_0} \prec\cdot\ a_0$ $(m_0 \geqslant 0)$ такая, что $\mu_0 = \{z_0, e_1, e_2, \ldots e_{m_0}, a_0\} \subseteq Y$. Так как $z_0 \in \mu_0 \backslash \phi(A)$ и $a_0 \in \phi(A) \cap \mu_0$, то существуют $y_0 \in \mu_0 \backslash \phi(A)$ и $x_0 \in \mu_0 \cap \phi(A)$ такие, что $x_0 \in y_0^\bullet$. В силу того, что $\phi(A)$ – причинно-замкнутое множество в $P$, $x_0 \in \phi(A)$ и $y_0 \notin \phi(A)$, существует $z_1 \in y_0^\bullet \backslash \phi(A)$. Тогда $z_1 \notin A$, так как $A \subseteq \phi(A)$. Предположим, что $z_1 \notin Y$. Так как $Y$ является причинно-замкнутым в $P$ и $y_0 \in Y$, то существует $t_0 \in {}^\bullet z_1 \backslash Y$. Очевидно, что $y_0 \neq t_0$. Получаем ${}^\bullet z_1 \cap {}^\bullet x_0 \neq \varnothing \wedge |{}^\bullet z_1| \geqslant 2$, что противоречит свойству $V_*V_*$-свободы $*$-конфигурации $P$. Следовательно, $z_1 \in Y$ (см. рис. 3). Тогда для некоторого $a_1 \in A$ имеем $\neg(z_1\ V_*\ a_1)$, то есть $(z_1\ li\ a_1)$. Пусть $(a_1 \preceq z_1)$. По лемме 4 существует непустая последовательность $a_1 \prec\cdot\ e_1 \prec\cdot\ e_2 \ldots \prec\cdot\ e_{k_0} \prec\cdot\ z_1$ $(k_0 \geqslant 0)$. Если $e_{k_0} = y_0$, то $(a_1 \preceq a_0)$ в силу транзитивности $\preceq$. Это противоречит тому факту, что $A$ является $V_*$-сечением в $Y$. Значит, $e_{k_0} \neq y_0$. Следовательно $|{}^\bullet z_1| \geqslant 2$, что невозможно в силу свойства $V_*V_*$-свободы $P$ и того факта, что ${}^\bullet z_1 \cap {}^\bullet x_0 \neq \varnothing$. Следовательно, $(z_1 \preceq a_1)$.

Снова, по лемме 4 существует непустая конечная последовательность $z_1 \prec\cdot\ e_1 \prec\cdot\ \ldots e_{m_1} \prec\cdot\ a_1$ $(m_1 \geqslant 0)$ такая, что

$$\mu_1 = \{z_1, e_1, \ldots e_{m_1}, a_1\} \subseteq Y.$$

Так как $z_1\ \in \mu_1 \backslash \phi(A)$ и $a_1 \in \mu_1 \cap \phi(A)$, то существуют $y_1 \in \mu_1 \backslash \phi(A)$ и $x_1 \in \mu_1 \cap \phi(A)$, удовлетворяющие $x_1 \in y_1^\bullet$. В силу того, что $\phi(A)$ является причинно-замкнутым в $P$, а также $x_1 \in \phi(A)$ и $y_1 \notin \phi(A)$, существует $z_2 \in y_1^\bullet \backslash \phi(A)$. Ясно, что $z_2 \notin A$. Предположим, что $z_2 \notin Y$. Так как $Y$ является причинно-замкнутым множеством в $P$ и $y_1 \in Y$, то существует $t_1 \in {}^\bullet z_2 \backslash Y$. Отсюда $t_1 \neq y_1$. Имеем ${}^\bullet z_2 \cap {}^\bullet x_1 \neq \varnothing \wedge |{}^\bullet z_2| \geqslant 2$, что противоречит свойству $V_*V_*$-свободы $*$-конфигурации $P$. Следовательно, $z_2 \in Y$. По условию множество $A$ является $V_*$-сечением в $Y$. Значит, существует $a_2 \in A$ такое, что $\neg(z_2\ V_*\ a_2)$, то есть $(z_2\ li\ a_2)$. Если $(a_2 \preceq z_2)$, то по лемме 4 существует конечная непустая последовательность $a_2 \prec\cdot\ e_1 \prec\cdot\ \ldots \prec\cdot\ e_{k_1} \prec\cdot\ z_2$ $(k_1 \geqslant 0)$. Предположим, что $e_{k_1} = y_1$. Тогда $(a_2 \preceq a_1)$ в силу транзитивности $\preceq$. Это противоречит тому факту, что $A$ – $V_*$-сечение в $Y$. Следовательно, $e_{k_1} \neq y_1$. Тогда

$^\bullet z_2 \cap \ ^\bullet x_1 \neq \varnothing \wedge |^\bullet z_2| \geqslant 2$, что противоречит свойству $V_*V_*$-свободы $*$-конфигурации $P$. Таким образом, $\neg(a_2 \preceq z_2)$, то есть $(z_2 \preceq a_2)$.

Повторяя рассуждения, мы получаем бесконечную последовательность $z_0 \preceq y_0 \prec z_1 \preceq \ldots \preceq y_i \prec z_{i+1} \preceq \ldots$ такую, что $L = \{z_0, y_0, z_1, \ldots\}$ является $li$-множеством в $P$, в силу транзитивности $\preceq$. Заметим, что по построению $L \subseteq Y \backslash \phi(A)$. Так как $A \subseteq \phi(A)$, то $L \cap A = \varnothing$. В силу иррефлексивности отношения $V_*$, имеем $Y \cap A^{\perp *} = \varnothing$, так как $A$ является $V_*$-сечением в $Y$. Поэтому $L \cap A^{\perp *} = \varnothing$. Ясно, что $A$ является $V_*$-множеством в $P$. Возьмем $V_*$-сечение $C$ в $P$ такое, что $A \subseteq C$. По лемме 1, $C \subseteq A \cup A^{\perp *}$. Следовательно, $L \cap C = \varnothing$. Пусть $T$ – это $li$-сечение в $P$ такое, что $L \subseteq T$. Выберем произвольное событие $x \in T \backslash L$ и покажем, что $x \notin A \cup A^{\perp *}$. Если $(z_i \preceq x)$ для всех $i \geqslant 0$, то в силу транзитивности $\preceq$ получаем $z_i \in [z_0, x]$ для всех $i \geqslant 0$, что противоречит принципу конечности причин. Значит, $(x \preceq z_j)$ для некоторого $j \geqslant 0$. По построению множества $L$, существует $a_j \in A$ такое, что $(z_j \preceq a_j)$. Поэтому, в силу транзитивности $\preceq$, $x \notin A^{\perp *}$. Предположим, что $x \in A \subseteq \phi(A)$. Так как $\phi(A)$ – причинно-замкнутое множество в $P$ и $a_j \in A \subseteq \phi(A)$, получаем $z_j \in [x, a_j] \subseteq \phi(A)$, что противоречит $z_j \in L \subseteq Y \backslash \phi(A)$. Значит, $x \notin A$. Следовательно, $T \cap (A \cup A^{\perp *}) = \varnothing$. Это противоречит свойству $K$-плотности $P$. □

Следующая теорема устанавливает взаимосвязь множества замкнутых подмножеств событий с множеством причинно-замкнутых.

**Теорема 1.** *Пусть $P$ – это $K$-плотная, $V_*V_*$-свободная $*$-конфигурация. Тогда множество $A$ является причинно-замкнутым в $P$ в том и только том случае, если оно является замкнутым в $P$.*

*Доказательство.* Согласно лемме 3 любое замкнутое множество в $*$-конфигурации является причинно-замкнутым. Предположим, что $A$ – причинно-замкнутое в $P$. Возьмем произвольное $V_*$-сечение $C$ в $A$. По лемме 6, справедливо $\phi(C) = A$. С другой стороны, $C$ – это $V_*$-множество в $P$. Поэтому $C$ является $V_*$-сечением в $C'_*$ в силу леммы 2. Более того, так как множество $C'_*$ замкнутое, то по лемме 3 оно является причинно-замкнутым. Следовательно, $\phi(C) = C'_*$ в силу леммы 6. Таким образом, $A$ является замкнутым множеством в $*$-конфигурации $P$.

□

Для получения следующего результата нам понадобится понятие конечной по степени конфигурации структуры событий.

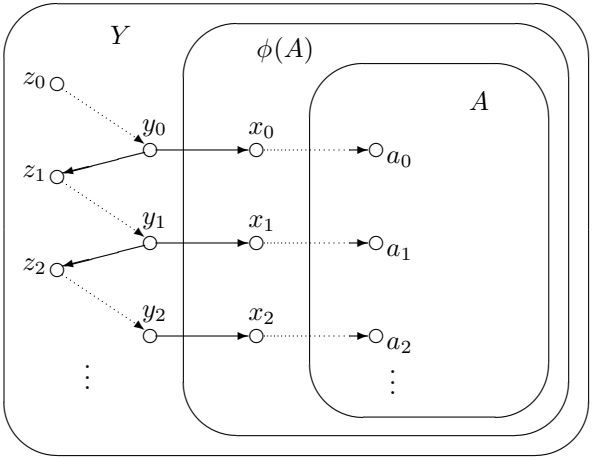



*Рис. 3.* Построение бесконечного $li$-множества

**Определение 10.** *$*$-конфигурация $P$ структуры событий $ES = (E, \leq, \#)$ называется* конечной по степени*, если для любого $e \in P$ выполнено $|\,{}^\bullet e| < \infty \wedge |e^\bullet| < \infty$.*

В используемых обозначениях выполнена

**Теорема 2.** *Пусть $P$ является конечной по степени $*$-конфигурацией структуры событий $ES$. Тогда $CC(P)$ замкнуто относительно направленного объединения.*

*Доказательство.* Возьмем произвольное направленное подмножество $\{C_i \mid i \in I\} \subseteq CC(P)$ и покажем, что $C = \bigcup_{i\in I} C_i$ является причинно-замкнутым множеством.

Проверим первое условие причинно-замкнутого множества. Пусть $b, e \in C$. По построению $C$, существуют $j, k \in I$ такие, что $b \in C_j$ и $e \in C_k$. Так как $\{C_i \mid i \in I\}$ – направленное подмножество, существует $m \in I$ такое, что $C_j \subseteq C_m \wedge C_k \subseteq C_m$. Значит, $b, e \in C_m$. Следовательно, в силу причинной замкнутости множества $C_m$, получаем $[b, e] \subseteq C_m \subseteq C$.

Теперь покажем, что для $C$ выполнено второе условие причинно-замкнутого множества. Пусть для некоторого $e \in P$ справедливо $e^\bullet \neq \varnothing \wedge e^\bullet \subseteq C$ (случай, когда ${}^\bullet e \neq \varnothing \wedge {}^\bullet e \subseteq C$ аналогичен). Покажем, что $e \in C$. Из того факта, что $C = \bigcup_{i\in I} C_i$ следует, что для каждого $b_j \in e^\bullet$ найдется $i_j \in I$ такой, что $b_j \in C_{i_j}$. Пусть $J = \{j \mid b_j \in e^\bullet\}$. Из свойства конечности $P$ по степени следует, что $|J| < \infty$. Значит, в силу

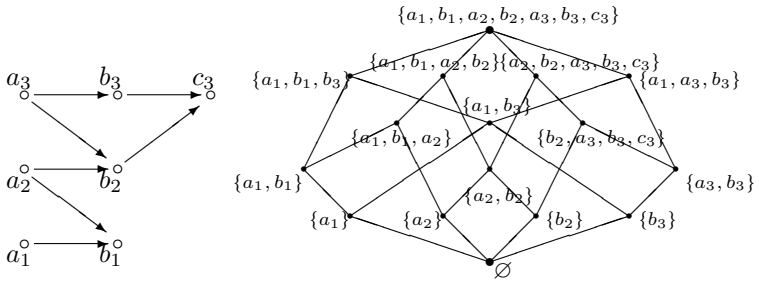



*Рис. 4.* Пример CFF-конфигурации и решетки ее причинно-замкнутых подмножеств

направленности множества $\{C_i \mid i \in I\}$, существует $m \in I$ такое, что $C_{i_j} \subseteq C_m$ для всех $j \in J$. Следовательно, $e^\bullet \subseteq C_m$, и значит, благодаря причинной замкнутости множества $C_m$, выполнено $e \in C_m$, то есть $e \in C$. □

Сформулируем теперь основной результат данной работы.

**Теорема 3.** *Для конечной по степени $V_*V_*$-свободной $K$-плотной $*$-конфигурации $P$ первичной структуры событий $ES$ решетка замкнутых в $P$ подмножеств событий является алгебраической.*

*Доказательство.* Ясно, что $(CC(P), \subseteq)$ – это система замыкания. Более того, из теоремы 2 следует, что множество всех причинно-замкнутых множеств в $P$ замкнуто относительно направленного объединения. Следовательно, в силу факта 2, оно образует алгебраическую решетку. Так как по теореме 1 решетка замкнутых в $P$ множеств совпадает с решеткой причинно-замкнутых, то необходимое получено. □

На рис. 4 приведен пример CFF-конфигурации $\{a_1, b_1, a_2, b_2, a_3, b_3, c_3\}$ первичной структуры событий, изображенной на рис. 1, и решетки причинно-замкнутых в ней множеств.

## 4. ЗАКЛЮЧЕНИЕ

Для моделей первичных структур событий представлена методика построения алгебраических решеток конфигураций посредством причинно-замкнутых множеств, которые могут рассматриваться как замкнутые относительно правила срабатывания множества событий параллельной системы. Свойство алгебраичности построенной решетки обеспечено тем, что исходная модель удовлетворяет аксиоме $K$-плотности, которая дает возможность различать модели реальных параллельных вычислений и теоретически допустимые модели. Помимо этого показана взаимосвязь причинно-замкнутых множеств с изученными ранее замкнутыми множествами событий параллельной системы. Следующим шагом в этом направлении видится нахождение достаточного условия на решетку с тем, чтобы обеспечить $K$-плотность исходной структуры событий. Это позволит построить характеризацию «аксиомы параллелизма» для структур событий в терминах порождаемых решеток.

## СПИСОК ЛИТЕРАТУРЫ

1. Petri, C. Concurrency as a basis for system thinking. / St. Augustin: Gesellschaft für Mathematik und Detenverarbeitung. ISP-Rep. – 1978. – Vol. 78, No. 06.
2. Petri C. Concurrency / Lect. Notes Comput. Sci. – 1980. – Vol. 84. – P. 261–276.
3. Best E. The relative strength of K-density / Lect. Notes Comput. Sci. – 1980. – Vol. 84. – P. 261–276.
4. Best E. A theorem on the characteristics of non-sequential processes / Fundamenta Informaticae. – 1980. – Vol. 3. – P. 77–94.
5. Fernandez C., Thiagarajan P.S. D-continuous causal nets: A model of non-sequential processes / Theoret. Comput. Sci. – 1984. – Vol. 28. – P. 171–196.
6. Kummer O., Stehr M.-O. Petri's axioms of concurrency – A selection of recent results / Lect. Notes Comput. Sci. – 1997. – Vol. 1248. – P. 195–214.
7. Cherkasova L. A., Kotov V. E. Descriptive and analytical process algebras / Lect. Notes Comput. Sci. – 1989. – Vol. 424. – P. 77–104.
8. Virbitskaite I. Some characteristics of nondeterministic processes / Parallel Processing Letters. – 1993. – Vol. 3, No. 1. – P. 99–106.
9. Virbitskaite I., Bozhenkova E.. Unified characterization of some properties of event structures / Proc. Internat. Conf. CONPAR 94 – VAPP VI, Linz, Austria, September 1994. – RISC-Linz Report Series. – 1994. – Vol. 94-48. – P. 29–32
10. Bernardinello L., Pomello L., Rombola S. Orthomodular Lattices Induced by the Concurrency Relation. / EPTCS – 2009. – Vol. 9. – P. 12–21
11. Bernardinello L., Pomello L., Rombola S. Closure Operators and Lattices Derived from Concurrency in Posets and Occurrence Nets. / Fundamenta Informaticae. – 2010. – Vol. 105, No. 3. – P. 211–235.
12. Bernardinello L., Pomello L., Rombola S. On Orthomodular Posets Generated

by Transition Systems. / Electr. Notes Theor. Comput. Sci. – 2011. – Vol. 270, No. 1. – P. 147–154.

13. Вирбицкайте И.Б., Ерофеев Е.К. Построение ортомодулярных решеток первичных структур событий / Проблемы информатики. – 2012. – №2 (14). – С.12–21.
14. Abramsky S., Dov M. Gabbay, Maibaum T. S. E. Handbook of Logic in Computer Science, Vol. 3. / Clarendon Press, Oxford. – 1994.

**С.В. Лештаев**

# ОСНОВЫ SPARQL

## ВВЕДЕНИЕ

*SPARQL* [1] (SPARQL Protocol and RDF Query Language) – это язык запросов к данным, представленным в виде RDF [2]. Формат данных RDF и язык запросов SPARQL разработаны в контексте Semantic Web консорциумом W3C [3]. RDF данные – помеченный орграф представленный тройками субъект, предикат и объект:

$$data = \left\{ (s,p,o) \mid \begin{array}{c} s \in nodes - \text{идентификатор узла}, \\ p \in predicates - \text{идентификатор предиката}, \\ o \in nodes \cup literals - \text{идентификатор узла или литерал} \end{array} \right\},$$

здесь *nodes* и *predicates* – идентификаторы – IRI, например, <http://namespace/id>, *literals* – литералы. Литерал имеет значение и идентификатор типа. При типе «строка с указанием языка» у литерала есть язык (en, ru, т.п.). RDF обладает формализмом логики предикатов первого порядка, что позволяет с помощью SPARQL создавать семантические поисковые системы. SPARQL запрос описывает свойства и отношения сущностей, и в отличие от SQL не зависит от их размещения по таблицам.

Задача состоит в том, чтобы сформировать среду выполнения SPARQL запросов. SPARQL запрос дан в виде строки, и необходимо его распознать, т.е. транслировать SPARQL строку в объектное представление, которое может выполнять запрос на различных RDF данных. При выполнении используется данная RDF-СУБД, которая предоставляет все необходимые методы доступа. Необходимо сформировать объектное представление результатов выполнения с выводом в виде строки.

## 1. СТРУКТУРА SPARQL

SPARQL состоит из нескольких блоков. Основной блок WHERE – шаблон отбора подграфов.

### *1.1. Шаблонный орграф*

Блок отбора WHERE содержит описание орграфа, являющегося шаблоном подграфов RDF данных, подмножество его вершин и дуг–переменных $variables = \{variable_1, \ldots, variable_n\}$, изначально их значение не указано. При выполнении запроса RDF граф сопоставляется с этим шаблонным орграфом, и при этом переменные принимают конкретные значения. Пусть *node's variables* – это переменные которые могут принимать значениями только идентификаторы узлов. Шаблонный орграф представлен тройками:

$$G_{template} = \left\{ (s,p,o)_i \mid \begin{array}{c} s \in node's\ variables \cup nodes, \\ p \in node's\ variables \cup predicates, \\ o \in variables \cup nodes \cup literals \\ i = 1, \ldots, m \end{array} \right\},$$

где $s$ – либо идентификатор узла, либо переменная, принимающая значениями только идентификаторы узлов, $p$ – идентификатор предиката, либо переменная, принимающая значениями только идентификаторы предикатов, $o$ – либо литерал, либо идентификатор узла, либо переменная.

При выполнении запроса RDF граф сопоставляется с этим шаблонным орграфом, и при этом переменные принимают конкретные значения, образуя *множество значений переменных*

$$values = (value_1, \ldots, value_n) \subset nodes \cup predicates \cup literals.$$

Обозначим через $G_{template}\,|_{variables=values}$ орграф, в котором элементы множества *variables* (переменные) заменены соответствующими элементами *values* (значениями). При этом шаблонный орграф образует RDF орграф

$$G_{request\ data} = G_{template}\,|_{variables=values} =$$

$$= \left\{ (s,p,o)_i \mid \begin{array}{c} (s^t, p^t, o^t)_i \in G_{template}, \\ s = \begin{cases} s^t, & \text{если } s^t \in nodes \\ value_j \in values, & \text{если } \exists j (s^t = variable_j \in variables) \end{cases} \\ p = \begin{cases} p^t, & \text{если } p^t \in predicates \\ value_j \in values, & \text{если } \exists j (p^t = variable_j \in variables) \end{cases} \\ o = \begin{cases} o^t, & \text{если } o^t \in nodes \cup literals \\ value_j \in values, & \text{если } \exists j (o^t = variable_j \in variables) \end{cases} \\ i = 1, \ldots, m \end{array} \right\}$$

Если существует множество значений переменных такое, что получаемый орграф является подграфом орграфа RDF данных, то орграф RDF данных считается *сопоставимым* с этим шаблонным орграфом.

$$\exists values \mid G_{request\ data} = G_{template} \mid_{varables=values} \subset data.$$

RDF орграф данных может содержать несколько таких подграфов, каждый из которых образован своим множеством значений переменных, и среди них возможны повторения. Все возможные множества значений образуют результаты выборки данных шаблонным орграфом:

$$results_{all} = \{values \subset nodes \cup literals \cup predicates \mid \begin{matrix} G_{request\ data} = G_{template} \mid_{varables=values} \\ \& G_{request\ data} \subset data \end{matrix} \}$$

*Синтаксис.* В строке SPARQL запроса указаны дуги шаблонного орграфа тройками, разделёнными точками: $s_1\ p_1\ o_1\ .s_2\ p_2\ o_2\ \ldots$. Тройки могут быть сгруппированы в группы с одинаковым субъектом или с одинаковой парой субъект-предикат. Это не позволяет описывать висячие вершины. Переменные обозначены префиксным символом "?" или "@".

**Пример 1**

```
WHERE
{?x    <http://namespace1.net/name>        ?name      .
 ?x    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                                          < http://namespace1.net/person> .
 ?x    ?ageProperty  10    .}
```

### *1.2. Фильтры*

Блок отбора Where содержит также фильтры – условия на множества значений переменных,

$$filters = \{filter_i : results_{all} \to \{true, false\}\},\ i = 1,\ldots, f\ ,$$

преобразующие результаты выборки

$$results = \{\forall values \in results_{all} \mid \forall\ \ i = 1,\ldots, f$$
$$(filter_i \in filters\ \ \left(filter_i\left(values\right) = true\right))\}.$$

Определим шаблон выборки данных как:

$$template = \left(variables, G_{template}, filters\right).$$

Условия сопоставимости шаблону данных при множестве значений переменных *values*:

1. Шаблонный орграф при подстановке значений вместо соответствующих переменных становится подграфом данных
$$G_{request\ data} = G_{template}\,|_{varables=values} \subset data.$$
2. Выполняются условия всех фильтров
$$\forall\ \ i = 1,\ldots, f\ \left(filter_i \in filters\ \ \left(filter_i\left(values\right) = true\right)\right).$$

Множество таких множеств значений переменных составляет результат сопоставления шаблона.

$$results = mapping\left(data, template\right) =$$

$$= \left\{ values \subset nodes \cup literals \cup predicates\ \ \middle|\ \begin{array}{c} G_{request\ data} = G_{template}\,|_{varables=values} \subset data \\ \forall\ \ i = 1,\ldots, f \\ \left(filter_i \in filters\ \ \left(filter_i\left(values\right) = true\right)\right) \end{array} \right\}.$$

Шаблон является сопоставимым тогда и только тогда, когда множество результатов не пусто.

Фильтры могут содержать условия на значения переменных, не описанных в шаблонном орграфе. Их можно рассматривать как висячие вершины шаблонного орграфа. Тогда фильтры не расширяют результаты выборки.

**Пример 2.**

$$variables = \left\{x\right\},\ G_{template} = \varnothing,\ filters = \left\{x \neq c\right\},$$

для фиксированной константы $c \in nodes \cup literals$.

Если не рассматривать переменные, описанные только в фильтрах, как вершины шаблонного орграфа, то результат выборки пустым шаблонным орграфом пуст. Но результат выборки шаблонным орграфом из одной переменной – множество из одноэлементных множеств из всех идентификаторов, а фильтр его сужает:

$$results = \{\{value\} \mid \forall value \in nodes \cup literals \setminus \{c\}\ \}.$$

*Синтаксис.* Каждый фильтр начинается с префикса FILTER. В общем случае далее следует условие в скобках. Условие можно разделить на 3 уровня. Нижний уровень с константами и переменными, указывающими, что именно им присвоенные значения проверяются условием. Далее уровень арифметических операций и функций над константами и переменными. Типы операций, констант и значений переменных должны совпадать, выражение «строка» + 10 рассматривается как ошибка. Следующий уровень логических выражений – сравнение подвыражений низких уровней (<, =, и т.п.) или других функций, возвращающих логическое значение. Самый вы-

сокий уровень содержит комбинации подвыражений предыдущего уровня логическими операторами ИЛИ “||”, И “&&”, НЕ “!”.

**Пример 3**

```
WHERE
{ ?x  <http://namespace1.net/age>     ?age .
 FILTER (?age > 10  || !(?age > 4)) }
```

### *1.3. Подграфы*

Шаблонный орграф может содержать несколько выделенных шаблонных подграфов

$$G_{template}^{sub} \subset G_{template},$$

которые образуют свои шаблоны (являющиеся частью шаблона, «подшаблонами») выборки данных, со своими переменными, фильтрами. Но фильтры действуют на множество множеств значений всех переменных всего шаблона.

$$template^{sub} \subset template = (variables, G_{template}, filters)$$

$$template^{sub} = \begin{pmatrix} variables^{sub} \subset variables, G_{template}^{sub}, \\ filters^{sub} \subset filters \end{pmatrix}.$$

При сопоставлении образуются значения переменных $values^{sub} \subset values$.

Далее шаблон, являющийся «подшаблоном», назовём подграфом.

Если шаблон не имеет необязательных и альтернативных частей, то он не может быть частично сопоставим, все его подграфы должны быть сопоставимы. Подграфы обладают тем же свойством.

*Синтаксис.* Подграф описывается его орграфом и фильтрами. Они указываются в строке запроса группированием и выделением множества дуг, фильтров, подграфов фигурными скобками. Подграфом рассматривается только множество вершин дуг и переменных, которых нет в описании остальной части орграфа, других подграфах, которые принадлежат только подграфу.

### *1.4. Необязательные шаблонные подграфы*

Можно указать подграфу, что сопоставление с графом данных не является необходимым условием для сопоставимости основного шаблона. При выполнении запроса сопоставляется весь шаблон, включая необязательные подграфы. Если на множестве значений переменных *values* не выполняется

хотя бы одно условие сопоставимости $template^{sub}$ : а) необязательный подграф не образует подграф орграфа данных

$$values^{sub} \subset values\ G^{sub}_{template}\,|_{variables^{sub}=values^{sub}} \not\subset data,$$

или б) существует хотя бы один из его фильтров с невыполняемым условием

$$\exists filter \in filters^{sub}\left(filter\left(values\right) = false\right),$$

тогда подграф на этом множестве значений изымается из шаблона

$$template \setminus template^{sub} = \left(variables \setminus variables^{sub}, G_{template} \setminus \mathrm{G}^{sub}_{template}, filters \setminus fiters^{sub}\right).$$

Если условия сопоставимости шаблону на этом множестве значений становятся выполнимыми, то сопоставление засчитывается, но для подмножества значений переменных, оставшихся в шаблоне без изъятого подграфа. Смысл необязательного подграфа – по возможности формировать множество значений его переменных. Подразумевается, что оно не пустое. Все переменные необязательных подграфов назовём *необязательными переменными*.

При сопоставлении шаблона, который содержит несколько необязательных подграфов, изъятие возможно для каждого подграфа в отдельности. Если переменная участвует в нескольких необязательных подграфах (и только в них), то она тогда также необязательная (она принадлежит объединению этих подграфов)

$$excluded = \left\{template^{sub}_1, \ldots, template^{sub}_e\right\}$$

$$template \setminus excluded = template \setminus \bigcup_{i=1,\ldots,e} template^{sub}_i.$$

Все необязательные переменные изъятых подграфов (включая участвующие в нескольких подграфах, которые все изъяты) получают пустые значения. Т.е. в исходном множестве значений в их позициях значения заменяются на пустые.

$$values' = values\,|_{values^{sub}=empty} =$$

$$= \left\{\begin{cases} v_i, & variable_i \in variables \setminus variables^{sub} \\ empty, & variable_i \in variables^{sub} \end{cases}\right\} =$$

$$= \left\{\begin{cases} v_i, & v_i \in values \setminus values^{sub} \\ empty, & v_i \in values^{sub} \end{cases}\right\}.$$

Тогда множество изначально различных множеств сужается. Потому что подмножество множеств значений переменных, которые различались только в тех позициях, в которых теперь пустое значение, теперь состоит из одинаковых и заменяется одним представителем.

**Пример 4.** Для некоторого множества переменных, где переменные подграфа помечены *sub*

$$variables = \{variable_1, \ldots, variable_k, variable_{k+1}^{sub}, \ldots, variable_n^{sub}\}.$$

Рассмотрим несколько множеств значений,

$$\left\{values_i = \left\{v_1, \ldots, v_k, v_{i,k+1}^{sub}, \ldots, v_{i,n}^{sub}\right\}\right\},\ i = 1, \ldots, w,$$

где значения необязательных переменных, помеченные *sub*. И пусть для них не выполняются условия сопоставимости шаблона именно из-за значений необязательных переменных. Для множества значений $\{v_1, \ldots, v_k\}$ и шаблона, получаемого изъятием подграфа, выполняются условия сопоставления. Тогда результат сопоставления всего шаблона с необязательным подграфом будет содержать только одно множество значений переменных вместо рассматриваемых множеств, которое получается дополнением множества значений $\{v_1, \ldots, v_k\}$ до размера множества переменных исходного шаблона пустыми значениями.

$$\left\{\left\{v_1, \ldots, v_k, empty, \ldots, empty\right\}\right\} \subset results$$

Результат может содержать и другие множества значений переменных, условия сопоставимости для шаблона на которых выполняются.

При сопоставлении шаблона количество пустых значений необязательных переменных должно быть минимальным. Если два множества значений различной длины

$$values_i \in mapping\left(data, template \setminus template^{sub}\right), i = 1, 2,$$

и первое содержится в другом так, что при дополнении их пустыми значениями до полной длины

$$values_1 \subset values_2$$

$$v'_{ji} \in values'_i \quad j = 1, \ldots, n$$

все непустые значения первого множества в той же позиции находятся во втором

$$\left(\left(v'_{j1} = empty\right) \vee \left(v'_{j1} = v'_{j2}\right)\right),$$

то нет необходимости добавлять в результат первое множество. Но все найденные значения должны присутствовать в множествах результата.

Теперь определение шаблона таково:

$$template_{with\ optionals} = \left(variables, G_{template}, filters, optionals\right)$$

$$optionals = \left\{template_1^{sub}, \ldots, template_o^{sub}\right\}$$

$$All = \bigcup_{excluded \subset optionals} mapping\left(data, template \setminus excluded\right)$$

Объединение результатов всех сопоставлений со всеми возможными изъятиями, включая пустое множество изъятий $mapping\left(data, template\right)$, содержит множества различной мощности. Сопоставление нового шаблона

$$mapping\left(data, template_{with\ optionals}\right) = \left\{ \begin{array}{c} values' \text{ полученые} \quad \nexists values_2 \in All \\ \text{добавлением пустых к } | \left(values \subset values_2\right) \\ values \in All \end{array} \right\}.$$

*Синтаксис.* Если подграф необязателен, то он помечен префиксом «OPTIONAL».

**Пример 5.**

```
WHERE
{ ?x  <http://namespace1.net/name>  ?name      .
 ?x   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                                        < http://namespace1.net/person>   .
 OPTIONAL
  { ?x       <http://namespace1.net/age>     ?age .
  FILTER (?age > 10  || !(?age  > 4))}}
```

### *1.5. Альтернативные подграфы*

Шаблонный орграф может содержать набор взаимозаменяемых альтернативных подграфов

$$alternative = \left\{ template_1^{sub} \overset{alt}{\leftrightarrow} template_1^{sub} \overset{alt}{\leftrightarrow} \ldots \overset{alt}{\leftrightarrow} template_c^{sub} \right\}.$$

Этот набор образуют как минимум два подграфа, которые являются альтернативными друг другу, каждый сопоставляется отдельно от остальных. Триплеты, подграфы и фильтры, описанные в запросе, можно рассматривать как условия на переменные и поставить между ними логическое «и». Тогда набор альтернативных подграфов – это подграфы, между которыми стоит логическое «или».

Если набор альтернатив один в шаблоне

$$template = \left(variables, G_{template}, filters, optionals, \{altrnative\}\right),$$

то при сопоставлении поочерёдно выбирается каждый из альтернативных шаблонов и используется в качестве подграфа основному. Множество мно-

жеств значений переменных сопоставления основного шаблона является объединением множеств

$$result = mapping\left(data, template\right) = \bigcup_{\substack{\forall i=1,\ldots,c \\ templeate_i^{alt} \in alternative}} result_i,$$

получаемых при сопоставлении шаблонов

$$result_i = mapping\left(data, template_i\right),$$

содержащих только один подграф из набора альтернативных подграфов. При этом каждый подграф используется ровно один раз

$$template_i = template \cup \left\{template_i^{alt}\right\}.$$

Набор альтернативных подграфов имеет своё множество переменных, состоящее из переменных его подграфов. Когда сопоставляется один из них, его переменные и все участвующие в нём переменные получают значения. Остальные переменные в наборе, принадлежащие альтернативным подграфам, и только им, получают пустые значения, но входят в множество значений.

Чтобы шаблон являлся сопоставимым с данными, необходимо, чтобы хотя бы одно из объединяемых множеств было непусто, т.е. чтобы хотя бы с одним из альтернативных подграфов в качестве подграфа основной шаблон являлся сопоставимым.

В общем случае шаблон с множеством из $l$ наборов

$$template = \left(variables, G_{template}, filters, optionals, altrnatives\right)$$

$$alternatives = \{alternative_j\}, j = 1,\ldots,l,$$

где каждый набор $alternative_j$ состоит из $c_j$ альтернативных подграфов

$$alternative_j = \left\{template_{t_j}^{alt}\right\}, t_j = 1,\ldots,c_j,$$

при сопоставлении шаблона перебираются и добавляются все возможные комбинации из $l$ подграфов, где по одному подграфу $template_{t_j}^{alt}$ выбирается из каждого набора альтернатив $alternative_j$.

$$result = \bigcup_{\substack{\forall j=1,\ldots,l \\ \forall t_j=1,\ldots,c_j}} mapping\left(data, template \cup \bigcup_{\substack{\forall j=1,\ldots,l \\ template_{t_j}^{alt} \in alternative_j}} template_{t_j}^{alt}\right).$$

*Синтаксис.* Подграфы, составляющие набор альтернативных, располагаются в строке по порядку и разделены «UNION».

**Пример 6.**

```
WHERE
{ ?x  <http://namespace1.net/name>  ?name      .
 ?x   <http://namespace1.net/age>     ?age .
 {?x<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                                    < http://namespace1.net/person>}
UNION
 {?x<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
                                    < http://namespace1.net/city> }}
```

Кроме шаблона в запросе указан блок результата запроса. В статье рассматривается 4 типа запросов стандарта SPARQL 1.0, различающихся результатом:

ASK. Результат запроса – только логическое значение, указывающее найдено ли хотя бы одно множество значений переменных $result \neq \varnothing$.

SELECT с указанием подмножества переменных (возможно и всего множества переменных).

Результат порождает таблицу, столбцы – указанные переменные, строки – значения этих переменных, где на каждое множество значений переменных выделяется по одной строке.

DESCRIBE с указанием переменной или идентификатора узла орграфа. Результат порождает орграф, подграф данных. Если указан идентификатор узла, то орграф содержит его окрестность длины 1 – все триплеты, в которых он участвует. Если указана переменная, то для каждого найденного значения переменной орграф содержит его окрестность.

CONSTRUCT с указанием шаблонного орграфа, его переменные – это подмножество переменных шаблона в блоке отбора WHERE. Для каждого найденного множества значений переменных значения подставляются в советующие переменные в указанный шаблонный орграф, формируя орграф. Результат состоит из объединения формируемых орграфов.

## 2. ТРАНСЛЯЦИЯ СТРОКИ В ОБЪЕКТНОЕ ПРЕДСТАВЛЕНИЕ ЗАПРОСА

Полный синтаксис SPARQL 1.0 предоставлен в расширенной форме Бэкуса–Наура [1]. Трансляцию можно сделать, используя, например, ANTRL [2] генератор транслирующего кода. При трансляции шаблона шаблонные

триплеты различаются по количеству неизвестных переменных, по роли. Каждая новая переменная *s, p, o*, встретившаяся впервые в данном шаблоном триплете, рассматривается как *неизвестная*. *Известные* – неновые, уже встречавшиеся ранее хотя бы раз, переменные и константы.

Если субъект, предикат и объект в триплете известные, то на стадии выполнения эти переменные уже будут иметь значения, и роль такого шаблонного триплета – проверить значения. Если есть новые переменные в шаблонном триплете, то роль триплета – вычислить значения этих переменных. Так как значений несколько, то из одного множества значений триплет породит столько же новых множеств значений. Выполнение шаблонных триплетов:

1) перебор всевозможных идентификаторов и/или литералов и поочерёдное присваивание указанной переменной;
2) проверка триплета с известными *s, p, o* на наличие его в данных;
3) поочерёдное присваивание неизвестной переменной значения, советующего данным.

Таблица 1

**Определение выполнений и роли по известности *s, p, o* в триплете**

| количество известных | *(s, p, o)* | выполнение | роль |
|---|---|---|---|
| 3 | все известны | проверка | проверка |
| 2 | неизвестно либо *s*, либо *o*, либо *p* | присваивание либо *s*, либо *o*, либо *p* | порождение |
| 1 | известно *p* | перебор *s*, присваивание *o* | порождение |
| | известно *s* или *o* | присваивание *(p* и *o)* или *(p* и *s)* | порождение |
| 0 | все неизвестны | перебор *s*, присваивание *p* и *o* | порождение |

Трансляция фильтра – это трансляция функции с логическими и/или арифметическими подфункциями. Если в фильтре используется новая переменная, необходимо поочерёдно перебирать все возможные её значения из данных. Кроме редкого исключения. Рассмотрим фильтр, проверяющий выражения на равенство. Когда с одной стороны равенства расположено известное выражение *c*: константа, неновая переменная, функция от известных выражений, а с другой стороны равенства расположена только одна неизвестная переменная *x*, то выражение интерпретируется, как присваивание, а условие – как истинное.

Таблица 2

**Интерпретация фильтров**

| № | Левое выражение | Правое выражение | Интерпретация |
|---|---|---|---|
| 1 | $c_1$ | $c_2$ | Сравнение |
| 2 | $c$ | $x$ | Присваивание |
| 3 | $x_1$ | $x_2$ | Перебор $x_1$ и сведение к №2, т.е. присваивание $x_2$. |
| 4 | $c$ | $f(x_1,\ldots,x_k)$ | Перебор каждого из $x, x_1,\ldots,x_{k+t}$ и сведение к №1, т.е. сравнение. |
| 5 | $x \in \{x_1,\ldots,x_k\}$ | $f(x_1,\ldots,x_k)$ | |
| | $f(x_1,\ldots,x_k)$ | $f(x_{k+1},\ldots,x_{k+t})$ | |
| 6 | $x \notin \{x_1,\ldots,x_k\}$ | $f(x_1,\ldots,x_k)$ | Перебор каждого из $x_1,\ldots,x_k$ и сведение к №2, т.е. присваивание $x$. |

Транслировать и выполнять фильтр, содержащий логическое «или» между присваиваниями, нужно подобно трансляции и выполнению набора альтернативных подграфов. Тождественно истинные или ложные выражения в фильтрах не отслеживаются.

Трансляция необязательного подграфа отличается только тем, что необходимо запомнить множество неизвестных переменных, встретившихся впервые в нём. Для этого достаточно запомнить количество известных переменных до и после трансляции подграфа, т.е. подмножество значений переменных, образованное этими двумя индексами. Ещё необходимо запомнить, с какими необязательными подграфами есть общие необязательные переменные. Точно так же при трансляции набора альтернативных подграфов запоминается множество всех неизвестных переменных, встретившихся в подграфах набора. При трансляции каждого следующего подграфа переменные всех предыдущих в наборе альтернативных «забываются» – помечаются, что они тут ещё не получат значения, чтобы в следующем альтернативном подграфе они снова считались неизвестными и получали значения, а не проверялись. Т.к. переменные могут участвовать вне набора и не во всех альтернативных подграфах, то роли последующих триплетов могут различаться при выполнении различных альтернативных подграфов.

**Пример 7**

```
Where
{{?x  ?p   < http://namespace1.net/person>}
UNION
 {?y        ?p    < http://namespace1.net/city> }
?x          ?q   ?y    . }
```

последний триплет содержит либо неизвестную *?y,* либо неизвестную *?x.* Здесь для однозначности роли достаточно переставить набор альтернатив в конец запроса. Когда несколько наборов альтернативных подграфов имеют общие переменные не во всех своих подграфах, перестановка не поможет. Можно, например, перегруппировать наборы, или определять роль во время выполнения.

## 3. ВЫПОЛНЕНИЕ

Основная часть выполнения запроса – формирование множеств значений при последовательных повторных исполнениях ролей триплетов шаблонного орграфа и фильтров. От порядка триплетов, фильтров и подграфов зависит скорость выполнения. Например, необязательные подграфы лучше выполнять последними, а каждый фильтр – сразу после появления в шаблоне всех его переменных. Пока что оптимизирующие перестановки возложим на составителя запроса и выполнять будем в том порядке, в котором они описаны в строке.

В начале выполнения множество результатов содержит одно множество значений такой же мощности, что и множество переменных, состоящее из незаданных значений.

$$results_0 = \{\{unknown, unknown, \ldots, unknown\}\}.$$

Рассмотрим один шаг выполнения (шаблонного триплета, фильтра или подграфа). Накоплено *r* множеств значений, в каждом часть значений незаданные.

$$results_k = \{values_{i,k} = \{v_{i1}, v_{i2}, v_{i3}, \ldots, v_{ik}, unknown, unknown, \ldots, unknown\}, i = 1, \ldots, r\}.$$

**А.** Если следующий триплет при трансляции получил роль проверить значения, то у его переменных значения заданы. Такой шаблонный триплет подобен фильтру, где условие на множество значений переменных таково: при подстановке значений соответствующих переменных в триплет, получаемый триплет должен содержаться в данных, и не важно, каковы значения остальных переменных. Множество значений уменьшается, остаются те значения, что прошли проверку этим триплетом.

**Б.** Если роль триплета – вычислить значения, то он содержит *p* неизвестных переменных, *p=1, 2,3*.

1. В цикле последовательно перебираются все *r* накопленные множества значений

$$\forall i = 1, \ldots, r \quad values_{i,k} \in results_k \qquad \text{цикл (1)}$$

при этом для каждого выбранного множества.

2. Известные переменные триплета принимают значения, которые уже есть во множестве значений.
3. С помощью соответствующего метода доступа к данным, предоставляемого RDF-СУБД, формируется множество значений неизвестных переменных по шаблонному триплету. База данных содержит $t$ триплетов, подходящих под шаблонный триплет.
4. Формируется новое множество результатов, состоящее из копий выбранного множества значений, такой же мощности $t$, что и множество полученных значений неизвестных

$$values_{i,k} \to results_{k+1} = \{values_{j+1,k+p}, values_{j+2k+p}, \ldots, values_{j+t,k+p}\}.$$

Найденные значения $\{\{w_1^i, \ldots, w_p^i\},\ i = 1, \ldots, t\}$ подставляются в копии в соответствующие их переменным позиции, остальные остаются незаданными

$$values_{j+1j,k+p} = \{v_{i1,} v_{i2,} v_{i3,}, \ldots, v_{ik,} w_1^1, \ldots, w_p^1, unknown, \ldots, unknown\}$$

$$\ldots$$

$$values_{j+tj,k+p} = \{v_{i1,} v_{i2,} v_{i3,}, \ldots, v_{ik,} w_1^t, \ldots, w_p^t, unknown, \ldots, unknown\}.$$

5. При переходе к следующему шагу выполнения в качестве накопленного множества результатов используется порождённое множество

$$results_{k+p} = \{values_{j+1,k+p}, \ldots, values_{j+t,k+p}\}.$$

Все последующие шаги будут выполняться заново для каждого перебираемого множества значений в цикле (1).

**В.** В ситуации с фильтром всё аналогично.

**Г.** Необязательный подграф выполняется подобно основному шаблону, кроме внешних для подграфа действий. Подграф тоже выполняет роль – вычислить значения своих переменных, и всех участвующих неизвестных переменных. Перебираются все накопленные множества значений переменных (1), и каждое $values_{i,k}$ выбирается за начальное множество значений для выполнения подграфа $results_0^{sub} = \{values_{i,k}\}$. Результат выполнения подграфа используется для следующего шага выполнения основного шаблона. И выполнение подграфа и последующих шагов будет повторяться r раз для каждого $values_{i,k}$. При любом из следующих условий из подграфа начинают изыматься его внутренние необязательные подграфы, пока условия не нарушатся. Если их нет или они все изъяты, то изымается сам под-

граф: в множестве значений $values_{i,k}$ значения всех переменных подграфа, включая участвующие необязательные, устанавливаются пустыми, и повторяется попытка выполнения, пропустив подграф, на этом же множестве значений

$$results_{k+p} = \left\{\{v_{i1,} v_{i2,} v_{i3,}, \ldots, v_{ik,} empty, \ldots, empty, unknown, \ldots, unknown\}\right\}.$$

*Условие 1.* Выполнение подграфа и оставшейся части шаблона не дало результатов, не дошло до конца.

*Условие 2.* Существует изъятый необязательный подграф (невложенный в сам подграф), в котором участвует хотя бы одна общая с этим необязательная переменная, он или сам подграф содержит фильтр с отрицанием.

**Д.** При выполнении набора альтернативных подграфов, как и при трансляции, перед выполнением каждого подграфа все значения переменных набора устанавливаются пустыми. Затем последовательно выполняются: один из подграфов, оставшаяся часть запроса, затем опять все значения переменных набора устанавливаются пустыми, выполняется второй альтернативный подграф, оставшаяся часть запроса, и т.д.

Когда выполнен последний шаблонный триплет, фильтр и подграф, полученное множество результатов добавляется в общий конечный результат шаблона. И в последнем незаконченном цикле (1) выполняется переход к следующему.

## ЗАКЛЮЧЕНИЕ

В статье рассмотрена только основная часть SPARQL версии 1.0. Существует [4] множество реализаций различных версий. Есть отдельные трансляторы, SPARQL библиотеки, движки, сервисы и всё вместе, например, OpenLink Virtuoso. Решения реализованы в комплексе с RDF-СУБД (например, Open Anzo реализован используя реляционные решения для RDF) без возможности использовать сторонние RDF-СУБД. Для разрабатываемых в ИСИ СО РАН RDF-СУБД приведённый алгоритм был частично реализован на C# 4.0 с ограниченной функциональностью, но не опубликован. На данный момент реализация SPARQL 1.0 продолжается, разрабатываются алгоритмы для SPARQL версии 1.1. Простой SPARQL и алгоритм выполнения имеют неочевидные «подводные камни». Производительность алгоритма зависит от производительности RDF-СУБД. Но важно сократить количество переборов всех RDF данных до минимума.

## СПИСОК ЛИТЕРАТУРЫ

1. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/
2. Resource Description Framework (RDF). http://www.w3.org/RDF/
3. World Wide Web Consortium (W3C) http://www.w3.org/
4. SPARQL implementations http://www.w3.org/wiki/SparqlImplementations
5. ANother Tool for Language Recognition (ANTLR). http://www.antlr.org/

**С. Б. Панкратов**

# АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ ТЕСТОВ ДЛЯ ПРОВЕРКИ РАСПАРАЛЛЕЛИВАЮЩИХ И ВЕКТОРИЗУЮЩИХ ПРЕОБРАЗОВАНИЙ ЦИКЛОВ В КОМПИЛЯТОРЕ

## ВВЕДЕНИЕ

Компилятор – инструмент, требования к надежности которого чрезвычайно высоки. И это неудивительно, ведь от правильности работы компилятора зависит правильность работы всех скомпилированных им программ. Из-за сложности входных данных и производимых над ними преобразований, задача тестирования компиляторов является весьма трудоёмкой и непростой. С появлением многоядерных процессоров острее встала проблема перехода от однопоточных к многопоточным вычислениям. А с момента переноса графического ядра на один кристалл с вычислительным и увеличением его мощности, появилась идея по утилизации и его вычислительных мощностей в дополнение к процессорным. Появились открытые стандарты распараллеливания вычислений – OpenMP, OpenCL, Cilk Plus и другие, а также компиляторы реализующие их поддержку. Поэтому вопрос автоматизации всех фаз тестирования (создания тестов, их прогона, оценки полученных результатов) компилятора стоит особенно остро. Автоматическая генерация тестов – важная вспомогательная часть тестирования, ведь тесты написанные вручную не могут покрыть все возможные комбинации конструкций языка, а также все ситуации применения оптимизаций.

## 1. ПРОБЛЕМА ТЕСТИРОВАНИЯ КОМПИЛЯТОРОВ

Язык программирования определяется синтаксисом, статической семантикой и динамической семантикой. Основой для генерации тестов может стать любой из этих аспектов языка. Данные спецификации задают систему вложенных подмножеств всех возможных тестов.

Синтаксис языка задается грамматикой, содержащей терминальные символы, нетерминалы и правила вывода. Цепочки терминалов, выводимые из стартового символа грамматики, называются синтаксически корректными

программами. Увы, данные программы могут быть использованы только для проверки правильности работы синтаксического анализатора компилятора.

Статическая семантика, определяемая только для синтаксически корректных последовательностей, задает правила вычисления свойств программы, не требующих для этого ее выполнения. К таким свойствам относятся, например, типы переменных и выражений. Помимо правил вычисления задаются также правила проверки статической корректности программы – контекстные условия, накладывающие ограничения на возможные комбинации значений статических свойств программы. Программы, удовлетворяющие контекстным условиям, могут быть использованы для проверки работы статического анализатора компилятора.

Динамическая семантика определяет смысл выполнения статически корректных программ, то есть по сути, это контекстные ограничения, которые должны быть выполнены во время исполнения программы. Для ее тестирования используются статически корректные программы, которые компилируются тестовым компилятором, исполняются, а затем результат их выполнения сравнивается с эталонным, определяемым динамической семантикой языка. Логично, что если тестовая программа не является детерминированной, то такое сравнение становится очень сложной задачей. В общем случае, для используемых на практике языков программирования, в частности C/C++, автоматическое выяснение детерминированности произвольной программы является очень сложной и более того алгоритмически неразрешимой задачей при статическом анализе [12]. Поэтому, часто для целей тестирования достаточно использования некого подмножества языка, для программ из которого возможно автоматическое решение вопроса о строгой детерминированности. Нас, в первую очередь, интересуют динамически корректные программы – ведь именно они максимально приближены к пользовательским приложениям, для компиляции которых будет использован тестируемый компилятор. Именно генерации такого класса тестов будет посвящена данная работа.

## 2. СУЩЕСТВУЮЩИЕ ПОДХОДЫ К ЗАДАЧЕ ГЕНЕРАЦИИ ТЕСТОВ ДЛЯ КОМПИЛЯТОРОВ

На данный момент существует множество работ по автоматической генерации тестов за авторством К.В. Ханфорда [6] и П. Пардома, А. Чилентано [4], Дж.С. Хатчисона [5], А.С. Косачева и М.П. Посыпкина [7,9,10], Кнута [11], Базичи и Спадафора [3], а также Стасенко А.П. [1]. Сгенериро-

вать синтаксически верный тест не составляет особого труда и обычно для этого годится легко создаваемая на основе спецификаций целевого языка, контекстно-свободная грамматика. Однако генерация компилируемой (статически семантически правильной) программы значительно сложнее, ведь в ней нужно обеспечить соответствие всем контекстным ограничениям целевого языка. Добиться такого соответствия можно при использовании контекстно-зависимых грамматик, создание которых весьма нетривиально. Как уже говорилось выше, более сложной представляется задача генерации детерминированных программ (динамически семантически верных).

Так как сложно создать контекстно-зависимую грамматику, то генераторы детерминированных программ, как правило, представляют собой некую монолитную программу на языке высокого уровня. Такой генератор сложно модифицируем и может быть использован только для генерации определенного класса программ некоторого языка. Также существует подход к генерации исполняемых программ, удовлетворяющих некоторому набору заданных шаблонов, с заданным набором вариаций, но данный способ крайне не гибок и по трудоемкости почти равноценен ручному написанию тестов.

В виду недостатков указанных подходов, в отделе тестирования компилятора компании Intel, для генерации тестов был выбран подход, основанный на использовании параметрической контекстно-свободной грамматики, описываемой при помощи особого метаязыка [1]. Параметрические контекстно-свободные грамматики зарекомендовали себя в качестве хорошего формализма для построения генераторов семантически правильных и имеющих детерминированное поведение компиляторных тестов. Их отличала ясность описания генерируемых тестовых программ, а также гибкость и удобство в работе с их контекстом. Однако данный подход применялся только для генерации однопоточных тестов, исполняемых на центральном процессоре и оставался открытым вопрос, можно ли использовать его для эффективной генерации других классов тестов или есть более подходящие решения? В данной работе мы попытались дать ответ на этот вопрос.

## 3. КРАТКОЕ ФОРМАЛЬНОЕ ОПИСАНИЕ ПАРАМЕТРИЧЕСКОЙ КОНТЕКСТНО-СВОБОДНОЙ ГРАММАТИКИ И ГЕНЕРАТОРА

Для описания грамматики используется специализированный язык, напоминающий BNF-нотацию и функциональный язык. Грамматика задается в одном файле и является набором строк, с поддержкой комментариев.

Правило – это идентификатор, для которого задано, каким образом он может быть переписан. Правило может быть как генерирующим, так и ограничивающим. Любой идентификатор – это последовательность символов, не начинающаяся с цифры. Имя параметра шаблона контекста в левой части правила может быть любым именем распознанной группы символов регулярного выражения языка Python, стоящей в предыдущем шаблоне контекста в левой части правила. Правая часть правила может задаваться либо альтернативами, либо многострочным правилом.

Для более тонкой работы с правилами могут быть использованы специальные распознаватели контекста, при этом число таких распознавателей ограниченно только здравым смыслом. Основное их удобство состоит не только в возможности извлечения объектов из контекста и их переименования, но и в возможности изменять сам контекст для всех идентификаторов в правой части правила.

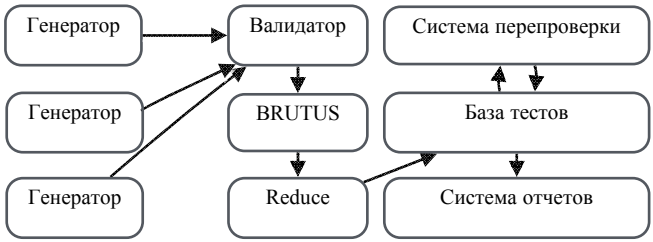
Помимо описанного механизма распознавателей, существует еще один способ описать и распознать контекст – условия. Условие – это выражение, вычисляющееся в булевский тип. А если условий несколько, то они объединяются по принципу логического "И". Также в языке грамматики предусмотрено несколько встроенных функций для удобной работы со списками, условиями и числами.

Данная параметрическая контекстно-свободная грамматика по наглядности сопоставима с обычной контекстно-свободной грамматикой, но позволяет генерировать более широкой класс контекстно-зависимых языков.

При генерации, выбор правил осуществляется случайно, что не гарантирует их достижимости, однако, как показывают другие исследования в этой области, такая проблема характерна и для других подходов к тестированию компиляторов [12].

Используемый в данной работе генератор подробно описан в статье Стасенко А.П. «Генерация исполняемых тестов для компилятора» [1].

Одного генератора и грамматики порождающей тестовые программы мало, для использования в процессе тестирования нужна система, которая будет валидировать сгенерированные тесты как в целях исследования их полезности для компилятора, так и в целях исключения тестовых проблем из-за потенциальной недетерменированности полученной программы.




И такая система была создана авторами генератора на языке Perl и названа Automatic Test Generator (в дальнейшем – ATG) и позволила в автоматическом режиме тестировать компилятор при помощи тестов, порождаемых тестовым генератором. Схема данной системы изображена на рисунке.

Давайте подробнее рассмотрим процесс работы всей системы. Каждый сгенерированный тест проходит 2 проверки – компиляция с оптимизациями и без. То есть, если тест успешно компилировался при помощи компилятора с отключенными оптимизациями и включенной проверкой указателей и запускался, то он является статически и динамически детерминированной программой, и результат ее исполнения является эталонным. Здесь стоит сделать уточнение насчет успешного запуска: успешный запуск – это отсутствие сообщений о проблемах с указателями и нулевой код завершения, а также отсутствие завершения программы по установленному тайм-ауту. Это первая ступень проверки. Следующая ступень, это проверка компиляции с оптимизациями. Если тест вызвал на данном этапе проблемы, то он уже полезен для тестирования и сохраняется для дальнейшей обработки. Если компиляция прошла успешно, то проверяется корректность исполнения программы с ее эталоном. И если программа проваливает эту проверку, то она также сохраняется. В случае, когда программа проходит проверку с эталоном, она просто удаляется и запускается процесс генерации нового теста.

Сохраненные тесты проходят дальнейшую обработку при помощи инструмента редукции программного кода Reduce для выделения той части теста, которая и приводит к проблемам в компиляторе. Также производится попытка найти компоненту компилятора, виновную в проблеме, при помощи функционала компилятора, который позволяет отключать применяемые оптимизации. Затем тесты сохраняются в базу системы с пометкой, на какой системе, когда и с какой проблемой они были получены.

С целью оптимизации места, занимаемого хранилищем тестов, сохраняются 10 минимальных по размеру тестов на каждую проблемную оптимизацию. Но если возможно найти дату проблемной сборки, то это позво-

ляет сохранять по 10 тестов на каждую дату. Система периодически производит перепроверку всех тестов из базы, помечая, на каких платформах тесты проходят, а на каких падают. Помимо проверки на последней версии компилятора, производится проверка и на продуктовых бранчах компилятора, чтобы не допустить попадания таких ошибок в конечный продукт, а также, чтобы выполнить требования процесса тестирования компилятора. Раз в сутки система высылает аналитику отчет о найденных проблемах. В дальнейшем аналитик создает дефект на компилятор и прикладывает к нему тест для воспроизведения проблемы, помечая в заголовке теста, что для него был создан дефект с определенным идентификатором. Состояние тестов, помеченных дефектом, также отслеживается в ежедневном отчете, и именно в случае, если дефект был закрыт, а тест продолжает падать, аналитику нужно обратить на него особое внимание.

## 4. ОБЩИЙ ПОДХОД К ГЕНЕРАЦИИ

В рамках данной работы были созданы грамматики для расширений языка C++, а именно Cilk Plus, OpenMP и гетерогенных вычислений на графическом процессоре – GFX-offload. Все эти расширения поддерживают параллелизацию циклов, что дало возможность сосредоточиться только на генерации циклов. Помимо упрощения задачи в упразднении необходимости следить за зависимостью между генерируемыми синтаксическими конструкциями расширений, это еще и усилит проверку векторизующих оптимизаций компилятора. Также все расширения имеют ограничения на код, который будет параллелизоваться или исполняться на графическом ускорителе. В общем виде они выглядят примерно так:

- никаких переходов из цикла, а значит внутри цикла не должно быть конструкций вроде return, break, goto;
- запрещено использование некоторых синтаксических конструкций разных расширений языка внутри специальных блоков, то есть, например, в offload блоке для GFX нельзя пользоваться _Cilk_spawn и _Cilk_sync;
- ограничения на тип переменных цикла.

## 5. СОЗДАНИЕ ГРАММАТИКИ ДЛЯ OPENMP

В качестве основы для грамматики, была взята уже существующая грамматика для генерации тестовых программ на языке C++. Из нее были

взяты некоторые стандартные правила (генерация циклов, условий и т.д.). Как уже говорилось выше, для генерации была выбрана omp parallel for директива для циклов. Для описания правила генерации было необходимо, во-первых, изменить контекст, передаваемый между правилами грамматики так, чтобы можно было, используя механизм условий, узнать, находимся ли мы внутри блока кода с директивой omp parallel for или нет. Во-вторых, нужно было добавить необходимые распознаватели во все правила, которые генерируют выражения внутри блока, который окажется под директивой параллелизации. Это нужно, чтобы внутри таких блоков не генерировались конструкции, вызывающие состояние гонки [8]. В-третьих, написать собственно правило, порождающее циклы for с директивой omp parallel for. Это правило должно создавать новый контекст new-ctx для всех порождающих правил внутри parallel блока. Здесь мы столкнулись со следующей проблемой: использовать локальные или глобальные переменные в качестве переменных доступа к массивам проблематично, и это приводит к гонке данных, поэтому было решено использовать в качестве таких переменных только переменные параллельного цикла и переменные вложенных циклов, так как в таком случае мы в большинстве случаев не получим состояния гонки. Нам не нужно стопроцентной «правильной» генерации, так как даже если такая программа будет сгенерирована, она не попадет в базу тестов, а будет откинута в процессе валидации, о котором будет сказано позднее.

После описания новых правил было необходимо добавить поддержку parallel директив в проект Reduce. Для этого был изменен ретранслятор из абстрактного синтаксического дерева, используемого в качестве внутреннего представления Reduce, обратно в исходный код на языке C++.

**Пример** – Правила порождающие OpenMP циклы

```
omp-for-clause (():_)
::= "; /* omp-for-cycle skipped due to no free lvals variables */"

omp-for-clause ctx @
  (lv:rv:as:ivs:fs:ret:isl:_) = ctx,
  i       = "i",
  new-lv  = (),
  new-ivs = (i,),
  new-ctx = (new-lv:rv:as:new-ivs:fs:0:1:())
  ? and(le(len(new-ivs), 3), eq(isl, 0))
::= *10 { "_Pragma(\"omp parallel for" omp-private(new-lv)"\")"
"for (int " i "=" low-lim(ivs) "; " i " <= " big-lim(ivs) "; "
step-pos(i)  ") " block(new-ctx)}
```

```
omp-for-clause ctx @
  (lv:rv:as:ivs:fs:ret:isl:_) = ctx
  ? eq(isl, 1)
::= "; /* omp-for-cycle skipped due to no nesting special loops
*/"
```

После написания первой грамматики остро встал вопрос валидации получаемых с ее помощью тестов. Для этого предполагалось использовать стороннее программное обеспечение для статического или динамического анализа сгенерированной программы. К сожалению, подходящего решения для статического анализа найдено не было, поэтому в качестве возможных динамических анализаторов рассматривались DRD из состава Valgrind и Intel Inspector XE. Самой главной проблемой при использовании DRD стало наличие массы ложных сообщений. Даже простейшая программа, не содержащая ошибок, вызывала порядка сотни сообщений о конфликтах чтения/записи между потоками. Второй проблемой стала невозможность отсеять эти ложные диагностические сообщения от тех, которые действительно содержат информацию о проблемах, так как создание файла исключений практически невозможно из-за динамической природы всех проверок. В комплексе, данные проблемы показали, что использовать Valgrind для наших целей будет проблематично. Следующим в списке был Intel Inspector XE, позволяющий производить анализ многопоточных программ в динамическом режиме. Первое, с чем пришлось столкнуться, это интересная особенность работы функции проверки указателей во время исполнения. Как уже говорилось в описании ATG, компиляция с данной опцией позволяет убедиться в корректности динамической семантики сгенерированной программы, но использование данной опции для программ с parallel блоками приводило к появлению диагностических сообщений о гонке данных в Inspector'e даже на пустом блоке omp parallel for. После консультации с разработчиками данной опции компилятора стало понятно, что это происходит из-за особенностей проверки указателей во время исполнения. Проверка активно читает данные стека, что и вызывает появление ложной диагностики в Inspector'e. Чтобы избежать этого, в Automatic Test Generator был изменен процесс валидации для программ, полученных из генератора OpenMP. В него был добавлен этап проверки при помощи Inspector'a, в котором первый шаг – это компиляция новой эталонной версии теста без использования опции проверки указателей. Конечно, чтобы не потерять проверку указателей, этот этап был добавлен после запуска эталонной версии теста и получения результатов проверки указателей.

## 6. СОЗДАНИЕ ГРАММАТИКИ ДЛЯ GFX-OFFLOAD КОМПИЛЯТОРА

Основные правила грамматики были взяты из грамматики OpenMP, также в эти правила были внесены необходимые изменения под требования offload компилятора.

В первую очередь хотелось проверить качество кодогенератора компилятора для новой платформы. Поэтому было принято решение переложить исполнение большей части теста на плечи интегрированного графического процессора. В примере 8 показано, как это реализовано в грамматике для генератора. Однако исполнение offload программы имеет некоторые ограничения, например такие, как запрет на вызов не помеченных как offload функций, запрет на операции с стандартным вводом-выводом и несколько других. Поэтому все несовместимые синтаксические конструкции были вынесены за пределы offload части программы.

**Пример** – Правила добавляющие offload прагму в тест

```
offloadpragma ctx locals globals
::={"_Pragma (\"offload target(gfx) inout(" print-arrays(ctx)
print-gpointers(ctx) ")\")"
"_Pragma (\"parallel_loop\")"

"for (int i_i = 0; i_i < 1; i_i++){"
    declarations(locals, globals)
    statements(ctx)
    for-clause(ctx)
    statements(ctx)
"}"
}

print-arrays (lv:rv:as:_) ::= arr-list(as)
arr-list ()  ::= ""
arr-list qw:() ::= " " get(qw,0)
arr-list qw:qws ::= " " get(qw,0) "," arr-list(qws)
print-gpointers (lv:_) ::= glob-list(lv)
glob-list () ::= ""
glob-list qw:() ? is-prefix(qw, "g_")::= ", " qw
glob-list qw:qws ? is-prefix(qw, "g_")::= ", " qw glob-list(qws)
glob-list qw:() ::= ""
glob-list qw:qws::= glob-list(qws)
```

Как можно заметить, для корректности работы с глобальными и локальными для функции main переменными, они добавляются внутрь offload цикла при помощи опции offload блока inout.

Все глобальные переменные, а также вызываемые функции помечаются в соответствующих правилах при помощи добавления __declspec(target(gfx)), для возможности их вызова и использования внутри offload части теста. Самым простым offload блоком является цикл for, поэтому основной блок программы был помещен в цикл с одной итерацией, помеченный offload прагмой и указанием всех переменных, участвующих в вычислениях на графическом процессоре. После окончания этой offload части производится вывод всех глобальных переменных и массивов в стандартный поток вывода. Это необходимо для обнаружения ошибок времени исполнения.

Так как, помимо проверки работы оптимизирующих алгоритмов компилятора, мы хотели проверить работу кодогенератора, валидация тестов должна представлять собой следующую процедуру: компиляция тестовой программы с отключенной offload прагмой (чтобы тест исполнялся на центральном процессоре) и ее запуск, считались бы эталонным запуском, а компиляция offload-компилятором с максимальным уровнем оптимизаций и запуск программы с использованием графического ядра, считались бы тестовыми, и происходило бы сравнение результатов этих запусков. К сожалению, в первое время данный подход не срабатывал из-за найденной в компиляторе проблемы, и пришлось временно перейти на процедуру валидации, в которой эталонный результат получался компиляцией offload-компилятором с отключенными оптимизациями и запуском скомпилированной программы на графическом процессоре. Конечно, в этом случае мы потеряли возможность находить целый класс ошибок, но, в то же время, это позволило продолжать получать новые тесты и находить новые проблемы в компиляторе.

Особых проблем с динамической корректностью генерируемых тестов не было из-за выбранного подхода по генерации одного всеобъемлющего цикла с одной итерацией без параллелизма.

Возникли определенные трудности и при работе Reduce, который минимизирует исходный код тестов. Как можно заметить, вместо директив #pragma, используются _Pragma, это было сделано, чтобы отделить обычные директивы от добавленных в грамматику. Такое разделение необходимо, так как ретранслятор Reduce при восстановлении исходного кода из синтаксического дерева объединял лексемы директив и цикла, и в результате, при попытке удаления вершины с директивой offload, возникала синтак-

сическая ошибка. Поэтому все особые директивы добавляются при помощи _Pragma, удаление которых запрещено в правилах минимизации Reduce.

## 7. СОЗДАНИЕ ГРАММАТИКИ ДЛЯ CILK PLUS

Создание грамматики для Cilk Plus было во многом схоже с созданием грамматики для OpenMP. Для генерации была выбрана конструкция _Cilk_for. Для описания правила генерации было необходимо, во-первых, аналогично OpenMP, изменить контекст, передаваемый между правилами грамматики так, чтобы можно было, используя механизм условий, узнать, находимся ли мы внутри блока _Cilk_for, или нет. Во-вторых, добавить необходимые распознаватели контекста в те правила, которые не должны ничего продуцировать внутри этих блоков. Ну и наконец, создать правила, которые будут генерировать конструкции _Cilk_for. Эти правила должны создавать новый контекст new-ctx для всех порождающих правил внутри блока цикла. Здесь мы, как и в случае с OpenMP, столкнулись с такой проблемой: использовать локальные или глобальные переменные в качестве переменных доступа к массивам проблематично, и это приводит к гонке данных, поэтому было решено использовать в качестве таких переменных только переменную параллельного цикла и переменные вложенных циклов, так как тогда мы в большинстве случаев не получим состояния гонки. Нам также не нужно стопроцентной «правильной» генерации, так как, даже если такая программа будет сгенерирована, она не попадет в базу тестов, а будет откинута в процессе валидации, о котором будет сказано позднее.

**Пример** – Правила порождающие Cilk Plus циклы

```
cilk-for-clause (():_) ::= "; /* for-cycle skipped due to no free
lvals variables */"

cilk-for-clause ctx @
  (lv:rv:as:ivs:fs:ret:_) = ctx,
  i       = any(diff(var-set(150),lv)),
  new-lv  = diff(lv,(i,)),
  new-ivs = (i:ivs),
  new-ctx = (new-lv:rv:as:new-ivs:fs:0:1:())
  ? le(len(new-ivs), 3)

::= *10 {
      declarations((i,), rv)
      "_Cilk_for (" i "=" low-lim(ivs) "; " i " <= " big-lim(ivs)
```

```
"; " step-pos(i)  ") " block(new-ctx)}

cilk-for-clause ctx ::= "; /* for-cycle skipped due to nesting
limit */"
```

Выбор средств для валидации кода, написанного с использованием расширения языка Cilk Plus не так широк и, по сути, ограничен только cilkscreen и Intel Inspector XE. Inspector и cilkscreen используются попеременно, чтобы понять, какой продукт более надежен и дает меньше ложных срабатываний. В дальнейшем будет выбран какой-то один, и пока все выглядит так, что это будет cilkscreen из-за своей специализированности на Cilk Plus программах.

Некоторых изменений потребовал и инструмент по минимизации кода – Reduce. Во-первых, было необходимо добавить поддержку конструкции _Cilk_for в его парсер и транслятор, а также добавить новые правила редукции, работающие с данной конструкцией. После правок Reduce смог полноценно и без ошибок редуцировать сгенерированные тестовые программы.

## РЕЗУЛЬТАТЫ

Апробация и внедрение результатов исследования были произведены в компании Intel для тестирования их компилятора. После всех необходимых изменений в Automatic Test Generator, он был запущен с этими грамматиками и первые результаты дал генератор GFX-offload тестов. В бета версии компилятора было найдено 8 ошибок, по большей части на стадии компиляции, но две из них – во время исполнения. Из этих ошибок только одна находилась не в векторизаторе компилятора, а в части, отвечающей за внутреннее представление. На данный момент большинство из этих ошибок уже исправлено.

Следующие результаты были получены от генератора OpenMP программ. Первая найденная ошибка оказалась уже известной проблемой на этапе компиляции, но не имела небольшого теста для ее воспроизведения. Соответственно, этот тест был добавлен к существующему дефекту в компилятор. Также во время работы генератора были обнаружены и другие проблемы, но, к сожалению, возникли некоторые проблемы с Reduce – он не смог их корректно уменьшить до небольших тестов.

Генератор Cilk Plus помог найти проблемы во frontend части компилятора еще на стадии отладки, однако пока никаких проблем в оптимизациях он не выявил. О найденных проблемах было также сообщено разработчикам компилятора.

## ЗАКЛЮЧЕНИЕ

Таким образом, созданные генераторы позволили найти некоторое количество ошибок в компиляторах, особенно в том, который предназначался для новой платформы GFX-offload. Мы убедились, что формализм параметрических контекстно-свободных грамматик оказался удобным инструментом для генерации детерминированных тестовых программ. Хотя, конечно, еще необходимо внести некоторые изменения в соответствующие грамматики или поменять глубину рекурсии генератора, чтобы усложнить тесты, а значит сделать генераторы более продуктивными.

В дальнейшем планируется расширить набор генерируемых конструкций расширений языка C++, а также увеличить процент генерации «полезных» для компилятора тестов, проведя исследования по зависимости глубины рекурсии генератора и процента новых тестов, вызывающих ошибки. Также возможно удастся увеличить область оптимизаций компилятора, в которой генерируемые тесты смогут вызвать ошибки.

## СПИСОК ЛИТЕРАТУРЫ

1. Стасенко А.П. Генерация исполняемых тестов для компилятора // Конструирование и оптимизация параллельных программ. – Редактор: доктор физ.-мат. наук, профессор, чл.-корр. РАЕН В.Н. Касьянов. – Серия «Конструирование и оптимизация программ», Новосибирск, 2008. – с. 301-313.
2. Aho A.V., Sethi R., Ullman J.D. Compilers: principles, techniques, and tools. – Boston: Addison-Wesley Longman Publishing Co., Inc., 1986. – с. 796.
3. Bazzichi F, Spadafora I. An automatic generator for compiler testing // IEEE transactions on Software Engineering. – NY: IEEE, 1982. – Vol. SE-8. – с. 343-353.
4. Celentano A., Crespi Reghezzi S., Della Vigna P., Ghezzi C., Granata G., Savoretti F. Compiler Testing using a Sentence Generator. // Software – Practice and Experience, 1980. – Vol. 10. – с. 897-918.
5. Duncan A.G., Hutchison J.S. Using Attributed Grammars to Test Designs and Implementation. // In Proc. of the 5th international conference on Software engineering, 1981. – с. 170-178.
6. Hanford K.V. Automatic generation of test cases // IBM Systems Journal. – NY: IBM, Dec. 1970. – Vol. 9. – с. 242-257.
7. Kalinov A., Kossatchev A., Petrenko A., Posypkin M., Shishkov V. Coverage-driven automated compiler test suite generation // ENTSC, 2003. – Vol.82 No. 3.

8. Wikipedia. Race condition // Wikimedia Fondation, Inc.. – URL:http://en.wikipedia.org/wiki/Race_condition
9. Kalinov A., Kossatchev A., Petrenko A., Posypkin M., Shishkov V. Using ASM Specifications for automatic test suite generation for mpC parallel programming language compiler. // In Proc. of the Fourth International Workshop on Action Semantics, AS, 2002. – c. 96-106.
10. Kalinov A., Kossatchev A., Petrenko A., Posypkin M., Shishkov V. Using ASM Specifications for Compiler Testing. In Abstract State Machines // Advances in Theory and Applications 10th International Workshop, ASM 2003. – Vol. 2589.
11. Knuth D.E. Semantics of Context-Free Languages, Mathematical Systems Theory, 2, 2,1968. – c. 127-146.
12. Kossatchev A.S., Posypkin M.A. Survey of compiler testing methods // Programming and Computing Software. – NY: Plenum Press, 2005. – Vol. 31, No. 1. – c. 10-19.
13. Landi W. Undecidability of Static Analysis // ACM Letters on Programming Languages and Systems (LOPLAS). – Vol. 1, No. 4. – NY, USA,Dec. 1992 – c. 323-337.

**Ю. Е. Плотникова**

# АЛЬТЕРНАТИВНАЯ ХАРАКТЕРИЗАЦИЯ ПОНЯТИЯ ЗОНЫ ВРЕМЕННЫХ СЕТЕЙ ПЕТРИ С ДИНАМИЧЕСКИМИ ПРИОРИТЕТАМИ*

## ВВЕДЕНИЕ

За последнее десятилетие резко возрос интерес к разработке и исследованию параллельных систем реального времени (ПСРВ), поведение которых в значительной степени зависит от времени. Проектирование корректных ПСРВ – нетривиальная задача, успешное решение которой возможно только при привлечении современных формальных методов и программных средств, осуществляющих автоматизацию данного процесса. Поэтому автоматический анализ и верификация поведенческих свойств ПСРВ – одно из актуальных направлений исследований в области параллельной обработки информации.

В литературе ПСРВ часто представляются моделями временных автоматов [1] и временных сетей Петри [3]. Временные автоматы являются обобщением конечных автоматов за счет введения переменных времени, значения которых определяют поведение автоматов во времени. Посредством связывания переходов с временными интервалами можно расширить сети Петри (СП) до временных сетей Петри (ВСП). Интервалы определяют временные области срабатывания переходов.

Приоритеты, распространенные в некоторых классах ПСРВ, не поддерживаются в моделях временных сетей Петри и не могут быть в общем случае смоделированы с их помощью. Поэтому в литературе было предложено расширение временных сетей Петри со статическими приоритетами (ВСПП) [7], где переходу, готовому к срабатыванию при некоторой маркировке в некоторый момент времени, не разрешается сработать, если какой-либо другой переход с более высоким приоритетом может сработать при этой же маркировке и в этот же момент времени.

Известно, что поведение временных автоматов [1] и временных сетей Петри может быть представлено в виде графов достижимых состояний,

*Данная работа частично финансируется DFG и РФФИ (проект CAVER, грант N BE 1267/14-1, грант No 14-01-91334).

которые являются в общем случае бесконечными, т.е. не пригодными для анализа свойств моделей. С целью получения конечной абстракции пространства состояний временных автоматов была предложена техника регионов [1], которая затем была усовершенствована и развилась в технику зон [9]. В статье [6] на основе техники зон построена абстракция пространства состояний ВСП.

В данной статье введено понятие динамических приоритетов в контексте ВСП, которые имеют тот же принцип работы, что и ВСПП, но в этом случае приоритеты переходов могут меняться в зависимости от маркировки. Временные сети Петри с динамическими приоритетами (ВСПсДП) имеют некоторые преимущества перед временными сетями Петри со статическими приоритетами. Именно динамические приоритеты удобнее для моделирования систем, в которых работа некоторых компонентов приобретает значимость над работой других компонентов в зависимости от обстоятельств. Также в работе предложена альтернативная характеризация зон ВСПсДП, которая позволяет представить поведение ВСПсДП в виде конечного графа зон.

## 1. СТРУКТУРА ВСПСДП

Для начала введем некоторые условные обозначения:

- $\mathbb{N}_0$ – множество натуральных чисел с нулем;
- $\mathbb{Q}_{\geq 0}$ – множество положительных рациональных чисел с нулем;
- $A^n$ – вектор, состоящий из $n$ элементов множества $A$;
- $|A|$ – мощность множества $A$.

Теперь определим понятие динамических приоритетов в контексте временных сетей Петри.

**Определение 1.** *Временная сеть Петри с динамическими приоритетами* – это кортеж $\mathcal{N} = (P, T, Pre(\cdot), Post(\cdot), m_0, I_{st}, \rho)$, где

- $P = \{p_1, p_2, \ldots, p_m\}$ – конечное множество мест;
- $T = \{t_1, t_2, \ldots, t_n\}$ – конечное множество переходов ($P \cap T = \emptyset$);
- $Pre(\cdot) \in (\mathbb{N}_0^{|P|})^{|T|}$ – обратное отношение инцидентности;
- $Post(\cdot) \in (\mathbb{N}_0^{|P|})^{|T|}$ – прямое отношение инцидентности;
- $m_0 \in \mathbb{N}_0^{|P|}$ – начальная маркировка;
- $I_{st} : T \to \mathcal{I}$ – функция, сопоставляющая каждому переходу $t \in T$ статический временной интервал его срабатывания. Для каждого такого интервала через $\downarrow I_{st}(t)$ и $\uparrow I_{st}(t)$ обозначим соответственно верхнюю и нижнюю границу. *Раннее и позднее време-*

*на срабатывания перехода* $t$ будем обозначать $Eft(t) =\downarrow I_{st}(t)$ и $Lft(t) =\uparrow I_{st}(t)$, соответственно;

- $\rho : \mathbb{N}_0^{|P|} \to 2^{T\times T}$ – функция приоритетов.

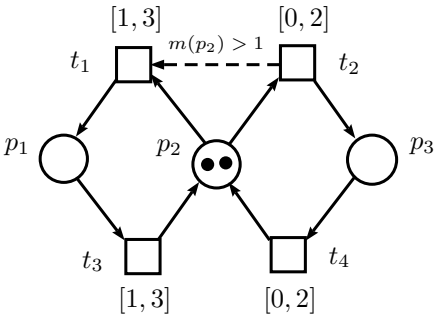



*Рис. 1.* ВСПсДП

*Маркировкой* в $\mathcal{N}$ называется отображение $m : P \to \mathbb{N}_0$, сопоставляющее месту количество фишек, находящихся в нем. Будем говорить, что переход $t \in T$ *допустим* при маркировке $m$, если для любого $p$ такого, что $Pre_p(t) > 0$ выполнено следующее условие: $m(p) \geqslant Pre_p(t)$, множество таких переходов обозначим через $En(m)$.

В ВСПсДП, как и во временной сети Петри, переход может сработать только в строго определенный промежуток времени. То есть, переход не может сработать раньше положенного срока, а также не может ждать дольше заданного времени. При описании поведения такой сети используют динамические временные интервалы срабатывания, которые меняются с течением времени.

*Состоянием* ВСПсДП называется пара $s = (m, I)$, где $m$ – маркировка, а $I : En(m) \to \mathcal{I}(\mathbb{Q}_{\geq 0})$ – интервальная функция, которая сопоставляет каждому допустимому при данной маркировке переходу динамический временной интервал его срабатывания. Значения интервальной функции могут быть представлены множеством векторов вида $\{\overline{\phi} \mid (\forall t \in En(m))\overline{\phi}_t \in I(t)\}$.

Пусть $\mathcal{N}$ – ВСПсДП. Будем говорить, что переход $t$ *может сработать*, если $t \in En(m)$, $0 \in I(t)$ и $\forall t' \in T \;\; (t, t')_m \in \rho(m) \Rightarrow t' \notin En(m)$. Здесь и далее запись $(t, t')_m \in \rho(m)$ означает, что переход $t'$ при маркировке $m$ имеет более высокий приоритет, чем переход $t$, для удобства этот факт будем записывать следующим образом: $t' \succ_m t$.

**Определение 2.** Семантика ВСПсДП

$$\mathcal{N} = (P, T, Pre(\cdot), Post(\cdot), m_0, I_{st}, \rho)$$

определяется временной системой переходов $TTS(\mathcal{N}) = (S, s_0, \rightsquigarrow)$, где

- $S$ – множество состояний ВСПсДП $\mathcal{N}$;
- $s_0 = (m_0, I_0)$ – начальное состояние;
- $\rightsquigarrow$ – отношение следования
  - по срабатыванию перехода $t$: $(m, I) \overset{t}{\rightsquigarrow} (m', I') \Leftrightarrow t \in En(m)$ и
    - $0 \in I(t)$,
    - $\forall t' \in En(m)\ ((t' \succ_m t) \Rightarrow 0 \notin I(t'))$,
    - $\forall t' \in En(m')$

$$I'(t') = \begin{cases} I(t'), \text{ если } t' \neq t \wedge m - Pre(t) \geq Pre(t'), \\ I_{st}(t'), \text{ иначе;} \end{cases}$$

  - по истечении времени $\theta$: $(m, I) \overset{\theta}{\rightsquigarrow} (m, I') \Leftrightarrow \theta \in \mathbb{Q}_{\geq 0}$ и
    - $\forall t' \in En(m)\ (\theta \leq \uparrow I(t'))$,
    - $\forall t' \in En(m)\ I'(t') = I(t') - \theta$, где $\uparrow I'(t')$ и $\downarrow I'(t')$ вычисляются следующим образом $\uparrow I'(t') = \uparrow I(t') - \theta$, $\downarrow I'(t') = max\{0, \downarrow I(t') - \theta\}$.

Поведение ВСПсДП удобно описывать в терминах последовательностей срабатываний.

Будем говорить, что $\varsigma = \nu_1 ... \nu_n$ является *последовательностью срабатываний в ВСПсДП* $\mathcal{N}$, если в $TTS(\mathcal{N})$ существует путь вида $s_0 \overset{\nu_1}{\rightsquigarrow} s_1 \dots s_{n-1} \overset{\nu_n}{\rightsquigarrow} s_n$, где $\nu_i \in (T \cup \mathbb{Q}_{\geq 0})$ Множество всех таких последовательностей в ВСПсДП $\mathcal{N}$ обозначим через $FS(\mathcal{N})$.

Состояние s *достижимо* в $\mathcal{N}$, если существует путь в $TTS(\mathcal{N})$, ведущий из начального состояния $s_0$ в состояние $s$. Множество всех достижимых состояний обозначим $RS(\mathcal{N})$. Будем называть ВСПсДП *ограниченной*, если существует $n \in \mathbb{N}$ такое, что для любой маркировки $m$ любого достижимого состояния ВСПсДП и любого места $p$ из $P$ $m(p) < n$. Далее будем рассматривать только ограниченные ВСПсДП.

*Дискретная временная система переходов ВСПсДП* $\mathcal{N}$ – это тройка $DTTS(\mathcal{N}) = (RS(\mathcal{N}),\ s_0,\ \rightarrow)$, где отношение $\rightarrow$ определяется так: $s \overset{t}{\rightarrow} s' \Leftrightarrow (\exists\theta)(\exists s'')(s \overset{\theta}{\rightsquigarrow} s'' \wedge s'' \overset{t}{\rightsquigarrow} s')$. Будем говорить, что $\sigma = t_1 \dots t_n$ является *дискретной последовательностью срабатываний* в ВСПсДП $\mathcal{N}$, если в $DTTS(\mathcal{N})$ существует путь вида $s_0 \overset{t_1}{\rightarrow} s_1 \dots s_{n-1} \overset{t_n}{\rightarrow} s_n$.

**Пример 1.** *На рисунке 1 представлен пример ВСПсДП, где* $p_1, p_2, p_3$ *– места,* $t_1, t_2, t_3, t_4$ *– переходы, рядом с каждым переходом указан статический интервал времени его срабатывания. При начальной маркировке в месте* $p_2$ *находятся две фишки. Также на рисунке пунктирной направленной линией обозначено, что переход* $t_1$ *будет иметь приоритет выше, чем приоритет перехода* $t_2$*, если количество фишек в месте* $p_2$ *будет больше одной, данное условие написано над линией. При начальной маркировке допустимы переходы* $t_1$ *и* $t_2$*. Так как* $m_0(p_2) > 1$*, то переход* $t_2$ *имеет более низкий приоритет, поэтому данный переход может сработать, пока не настанет раннее время срабатывания для перехода* $t_1$*, то есть, в интервале* $[0, 1)$*. По истечении времени* $\theta = 1$*, в интервале* $[1, 2]$ *может сработать только переход* $t_1$*, так как он имеет приоритет выше, чем переход* $t_2$ *при данной маркировке.*

*Теперь приведем пример последовательности срабатываний для данной сети:* $s_0 \overset{1}{\rightsquigarrow} s_1 \overset{t_1}{\rightsquigarrow} s_2 \overset{1}{\rightsquigarrow} s_3 \overset{t_1}{\rightsquigarrow} s_4 \overset{2}{\rightsquigarrow} s_5 \overset{t_3}{\rightsquigarrow} s_6$*. Покажем, как выглядит эта последовательность в* $DTTS(\mathcal{N})$*:* $s_0 \xrightarrow{t_1} s_2 \xrightarrow{t_1} s_4 \xrightarrow{t_3} s_6$*. Для наглядности опишем некоторые состояния из этих последовательностей.*

- $s_0 = (m_0, I_0)$, $m_0 = (0, 2, 0)$, $I_0(t_1) = [1, 3]$, $I_0(t_2) = [0, 2]$;
- $s_1 = (m_1, I_1)$, $m_1 = (0, 2, 0)$, $I_1(t_1) = [0, 2]$, $I_1(t_2) = [0, 1]$;
- $s_2 = (m_2, I_2)$, $m_2 = (1, 1, 0)$, $I_2(t_1) = [1, 3]$, $I_2(t_2) = [0, 1]$, $I_2(t_3) = [1, 3]$;

## 2. ЗОНА ВСПСДП

В данном разделе будет введено понятие зоны ВСПсДП, а также доказано утверждение о представлении зоны посредством системы неравенств.

**Определение 3.** Пусть $\sigma$ – дискретная последовательность срабатываний в ВСПсДП $\mathcal{N}$. Тогда *зоной* $Z_\sigma$ ВСПсДП $\mathcal{N}$ будем называть множество состояний, определяемое по индукции: $Z_\varepsilon = \{s_0\}, Z_{\sigma t} = \{s \mid (\exists s' \in Z_\sigma)(s' \xrightarrow{t} s)$. Будем называть $Z_\varepsilon$ *начальной зоной.*

**Пример 2.** *Для ВСПсДП, изображенной на рисунке 1, начальная зона будет выглядеть так:* $Z_\varepsilon = \{(m_0 = (0, 2, 0), I_0 = ([1, 3], [0, 2]))\}$*. Теперь опишем зону, следующую по срабатыванию перехода* $t_2$*,* $Z_{t_2} = \{s_{t_2} \mid s_{t_2} = (m_{s_2} = (0, 1, 1), I_{s_2} = ([1 - a, 3 - a], [0, 2], [0, 2])$*, где* $a \in [0, 1)\}$.

Нетрудно увидеть, что число состояний в зоне, построенной в примере 2, бесконечно, что усложняет работу с такой зоной. Поэтому, необ-

ходимо ввести альтернативную характеризацию, которая позволит эффективно работать с зонами ВСПсДП.

**Определение 4.** С каждым состоянием $s = (m, I)$ в ВСПсДП $\mathcal{N}$ таким, что в $TTS(\mathcal{N})$ существует путь $r = s_0 \overset{\nu_1}{\rightsquigarrow} s_1 \dots\ s_{k-1} \overset{\nu_k}{\rightsquigarrow} s_k = s$, где $\nu_h \in T \cup \mathbb{Q}_{\geq 0}$ для всех $h = 1, \dots, k$, свяжем *временную функцию* $\gamma^s : En(m) \to \mathbb{N}_0$ следующим образом: для каждого $t \in En(m)$ пусть $i = max(0 \dots k)$ такой, что $I_i(t) = I_{st}(t)$, тогда

$$\gamma^s(t) = \begin{cases} \sum_{i<j\leq k, \nu_j \in \mathbb{Q}_{\geq 0}} \nu_j, \text{ если } i < k, \\ 0, \text{ если } i = k \end{cases}$$

Таким образом, для каждого состояния $s = (m, I)$ функция $\gamma^s$ сопоставляет каждому переходу $t \in En(m)$ время, которое прошло с того момента, как он стал допустимым. При помощи данной функции можно вычислить динамический интервал срабатывания каждого допустимого перехода заданного состояния.

**Лемма 1.** *Пусть $\mathcal{N}$ – ВСПсДП, $TTS(\mathcal{N})$ – ее временная система переходов и $r = s_0 \overset{\nu_1}{\rightsquigarrow} s_1 \dots\ s_{k-1} \overset{\nu_k}{\rightsquigarrow} s_k$ – путь в $TTS(\mathcal{N})$, причем $s_i = (m_i, I_i)$, $i = 0, \dots, k$. Тогда $I_k(t) = I_{st}(t) - \gamma^{s_k}(t), \forall t \in En(m_k)$.*

*Доказательство.* Возьмем произвольный переход $t \in T$. Будем доказывать индукцией по $k$, что $I_k(t) = I_{st}(t) - \gamma^{s_k}(t)$.

- $k = 0$.
  - Если $t \notin En(m_0)$, то $I_0(t)$ и $\gamma^{s_0}(t)$ не определены.
  - Если $t \in En(m_0)$, то $I_0(t) = I_{st}(t)$ и $\gamma^{s_0}(t) = 0$.
- $k > 0$. По индукционному предположению $I_{k-1}(t) = I_{st}(t) - \gamma^{s_{k-1}}(t)$.
  - Если $t \notin En(m_k)$, то $I_k(t)$ и $\gamma^{s_k}(t)$ не определены.
  - Если $t \in En(m_k)$, то возможны два случая:
    - $\nu_k = \theta \in \mathbb{Q}_{\geq 0}$. Тогда $\theta \leq\uparrow I_{k-1}(t)$ и $I_k(t) = I_{k-1}(t) - \theta = I_{st}(t) - \gamma^{s_{k-1}}(t) - \theta = I_{st}(t) - (\gamma^{s_{k-1}}(t) + \theta) = I_{st}(t) - \gamma^{s_k}(t)$.
    - $\nu_k = t' \in En(m_{k-1})$.

      Рассмотрим случай $t \neq t'$. По определению 2, если $m_{k-1} - Pre(t') \geq Pre(t)$, то $I_k(t) = I_{k-1}(t)$, иначе $I_k(t) = I_{st}(t)$, так как $t \in En(m_k)$. В случае $t = t'$, из определения 2 следует, что $I_k(t) = I_{st}(t)$.

      Теперь рассмотрим значения $\gamma^{s_k}(t)$. Согласно определению 4, $\gamma^{s_k}(t) = 0$, если $I_k(t) = I_{st}(t)$, иначе $\gamma^{s_k}(t) = \sum_{i<j\leq k, \nu_j \in \mathbb{Q}_{\geq 0}} \nu_j$, где $i = max(0, \dots, k)$ такое, что $I_i(t) = I_{st}(t)$. Так как $\nu_k \notin \mathbb{Q}_{\geq 0}$, то $\gamma^{s_k}(t) = \sum_{i<j\leq k-1, \nu_j \in \mathbb{Q}_{\geq 0}} \nu_j =$

$\gamma_t^{s_{k-1}}$, в случае $I_k(t) \neq I_{st}(t)$.
В итоге, если $I_k(t) = I_{st}(t)$, тогда $I_k(t) = I_{st}(t) - 0 = I_{st}(t) - \gamma^{s_k}(t)$, а если $I_k(t) = I_{k-1}(t)$, то $I_k(t) = I_{st}(t) - \gamma^{s_{k-1}}(t) = I_{st}(t) - \gamma^{s_k}(t)$.

$\diamondsuit$

В следующей лемме докажем, что значения функции $\gamma$ для переходов, допустимых при маркировке $m$ состояния $s$, не зависят от того, каким путем было достигнуто это состояние.

**Лемма 2.** *Пусть $\mathcal{N}$ – ВСПсДП, $TTS(\mathcal{N})$ – ее временная система переходов и $r = s_0 \overset{\nu_1}{\rightsquigarrow} s_1 \dots\ s_{k-1} \overset{\nu_k}{\rightsquigarrow} s_k = s$ и $r' = s_0 \overset{\nu'_1}{\rightsquigarrow} s'_1 \dots\ s'_{l-1} \overset{\nu'_l}{\rightsquigarrow} s_l = s$ – пути в $TTS(\mathcal{N})$, где $\nu_h, \nu'_g \in T \cup \mathbb{Q}_{\geq 0}$ для всех $h = 1, \dots, k$ и $g = 1, \dots, l$. Тогда $\gamma^{s_r} = \gamma^{s_{r'}}$, где записи $s_r$ и $s_{r'}$ обозначают, что состояние $s$ достигается путями $r$ и $r'$, соответственно.*

*Доказательство.* Пусть $s = (I, m)$. Если $En(m) = \emptyset$, то значения $\gamma^{s_r}$ и $\gamma^{s_{r'}}$ не определены.

Рассмотрим случай, когда $En(m) \neq \emptyset$. Тогда для всех $t \in En(m)$ определены значения $\gamma^{s_r}$:

$$\gamma^{s_r}(t) = \begin{cases} \sum_{i<j\leq k, \nu_j \in \mathbb{Q}_{\geq 0}} \nu_j, \text{ если } i < k, \\ 0, \text{ если } i = k, \end{cases}$$

где $i = max(0, \dots, k)$ такое, что $I_i(t) = I_{st}(t)$, $s_i = (m_i, I_i)$;
и $\gamma^{s_{r'}}$:

$$\gamma^{s_{r'}}(t) = \begin{cases} \sum_{i'<j\leq l, \nu'_j \in \mathbb{Q}_{\geq 0}} \nu'_j, \text{ если } i' < l, \\ 0, \text{ если } i' = l, \end{cases}$$

где $i' = max(0, \dots, l)$ такое, что $I'_{i'}(t) = I_{st}(t)$, $s'_{i'} = (m'_{i'}, I'_{i'})$;

По лемме 1, получаем, что для всех $t \in En(m)$ верно $I_k(t) = I_{st}(t) - \gamma^{s_r}(t)$ и $I'_l(t) = I_{st}(t) - \gamma^{s_{r'}}(t)$. Так как $I_k(t) = I'_l(t) = I(t)$, то, следовательно, $\gamma^{s_r} = \gamma^{s_{r'}}$.

$\diamondsuit$

Докажем корректность представления зоны ВСПсДП посредством маркировки и системы неравенств, связанных с функцией $\gamma$.

**Утверждение 1.** *Пусть $\mathcal{N}$ – ВСПсДП и $Z_\sigma = \{s \mid s_0 \overset{\sigma}{\rightarrow} s = (m, I)$ в $DTTS(\mathcal{N})\}$ – зона $\mathcal{N}$. Тогда $Z_\sigma = (m, Q)$, где $Q$ – множество решений системы неравенств вида: $Lft(t) - sup\{\uparrow I(t) \mid s = (m, I) \in Z_\sigma\} \leq \gamma^{Z_\sigma}(t) \leq Lft(t) - inf\{\uparrow I(t) \mid s = (m, I) \in Z_\sigma\},\ t \in En(m)$.*

*Доказательство.*

Рассмотрим зону $Z_\sigma = \{s \mid s_0 \xrightarrow{\sigma} s = (m, I)$ в $DTTS(\mathcal{N})\}$.

По Лемме 2, значения функции $\gamma^s$ для каждого $s \in Z_\sigma$ определено однозначно, хотя последовательности $\sigma$, приводящей в $s$ в $DTTS(\mathcal{N})$, могут соотвествовать различные пути, приводящие в $s$ в $TTS(\mathcal{N})$. Все состояния в зоне имеют одинаковые маркировки.

Представим зону в таком виде: $Z_\sigma = (m, \mathbb{I})$, где $\mathbb{I} = \{I_s(t) \mid s_0 \xrightarrow{\sigma} s = (m, I_s)$ *в* $DTTS(\mathcal{N})$, $t \in En(m)\}$. Возьмем произвольный переход $t \in En(m)$, по Лемме 1, $I_s(t) = I_{st}(t) - \gamma^s(t)$, значит, по $\gamma^s$ можно однозначно определить состояние. Из $\uparrow I_s(t) = Lft(t) - \gamma^s(t)$ выразим $\gamma^s(t) = Lft(t) - \uparrow I_s(t)$. Получим ограничения на $\gamma^{Z_\sigma}(t)$ в соответствии с состояними зоны $Z_\sigma$:

$$\begin{cases} Lft(t) - sup\{\uparrow I(t) \mid s = (m, I) \in Z_\sigma\} \le \gamma^{Z_\sigma}(t); \\ \gamma^{Z_\sigma}(t) \le Lft(t) - inf\{\uparrow I(t) \mid s = (m, I) \in Z_\sigma\}. \end{cases} \quad (1)$$

Осталось показать, что для каждого $\gamma^s \in Q$ состояние $s$, определяемое $\gamma^s$, принадлежит зоне $Z_\sigma$. Докажем индукцией по длине $\sigma$.

Далее обозначим через $Q_\sigma$ множество решений системы неравенств вида (1) для зоны $Z_\sigma$.

База индукции $\sigma = \varepsilon$, $Z_\varepsilon = \{s_0\}$. В этом случае $\gamma^{s_0}(t) = 0$, $\forall t \in En(m_0)$, что удовлетворяет неравенствам вида (1).

По индукционному предположению, для $\sigma = \sigma'$ любое состояние $s'$, определяемое $\gamma' \in Q_{\sigma'}$, принадлежит зоне $Z_{\sigma'}$. То есть, эту зону можно описать при помощи $Q_{\sigma'}$.

Шаг индукции $\sigma = \sigma' t', Z_\sigma = (m, \mathbb{I})$.

Будем рассматривать для произвольного перехода $t \in En(m)$. Пусть $s_1$ и $s_2$ – состояния, определяемые $\gamma_1(t) = Lft(t) - sup_{s \in Z_\sigma}(\uparrow I(t))$ и $\gamma_2(t) = Lft(t) - inf_{s \in Z_\sigma}(\uparrow I(t))$, $\forall t \in En(m)$, соответственно. Заметим, что они оба принадлежат зоне $Z_\sigma$. Необходимо доказать, что любое состояние $\tilde{s}$, определяемое $\tilde{\gamma} \in (\gamma_1, \gamma_2)$, принадлежит $Z_\sigma$. Возьмем состояние $s_1' \in Z_{\sigma'}$, соответствующее $\gamma_1'(t) = Lft(t) - sup_{s \in Z_{\sigma'}}(\uparrow I(t))$, из определения зоны следует, что существуют такие $\theta_1 = \gamma_1(t) - \gamma_1'(t)$ и $\theta_2 = \gamma_2(t) - \gamma_1'(t)$, что в $TTS(\mathcal{N})$ есть пути $s_1' \overset{\theta_1}{\rightsquigarrow} s' \overset{t'}{\rightsquigarrow} s_1$ и $s_1 \overset{\theta_2}{\rightsquigarrow} s'' \overset{t'}{\rightsquigarrow} s_2$. Следовательно, $\forall \gamma(t) \in (\gamma_1(t), \gamma_2(t))$ $\exists \theta \in (\theta_1, \theta_2)$ такое, что $s_1' \overset{\theta}{\rightsquigarrow} s''' \overset{t'}{\rightsquigarrow} \tilde{s}$ в $TTS(\mathcal{N})$. Что доказывает $\tilde{s} \in Z_\sigma$.

$\diamondsuit$

Теперь, зона $Z_\sigma$ может быть определена маркировкой $m$ и множеством $Q$, такое представление зоны даст возможность эффективного построения графа зон с последующим его применением для верификации ВСПсДП.

**Пример 3.** *В качестве примера, построим зоны $Z_\varepsilon, Z_{t_1}, Z_{t_2}$ для сети, приведенной на рисунке 1. Начальная зона $Z_\varepsilon = (m_0 = (0,2,0), \{0 \le \gamma^{Z_\varepsilon}(t_1) \le 0;\ 0 \le \gamma^{Z_\varepsilon}(t_2) \le 0\})$. Построим зону, получаемую из начальной по срабатыванию перехода $t_1$. Данный переход сможет сработать только в интервале $[1,2]$, соответственно для перехода $t_2$ значения функции $\gamma^{Z_{t_1}}$ будут лежать в интервале $[1,2]$. Также, после срабатывания перехода $t_1$ изменяется маркировка, и становятся допустимыми новые переходы $t_1$, $t_3$, для которых $\gamma^{Z_{t_1}}(t_1) = \gamma^{Z_{t_1}}(t_3) = 0$. Таким образом, получаем зону $Z_{t_1} = (m_{t_1} = (1,1,0), \{1 \le \gamma^{Z_{t_1}}(t_2) \le 2;\ 0 \le \gamma^{Z_{t_1}}(t_1) \le 0;\ 0 \le \gamma^{Z_{t_1}}(t_3) \le 0\})$. Теперь построим зону, получаемую из начальной по срабатыванию перехода $t_2$. После срабатывания данного перехода становятся допустимыми переходы $t_2, t_4$. Учитывая интервал времени, в который срабатывает $t_2$, получаем зону $Z_{t_2} = (m_{t_2} = (0,1,1), \{0 \le \gamma^{Z_{t_2}}(t_1) < 1;\ 0 \le \gamma^{Z_{t_2}}(t_2) \le 0;\ 0 \le \gamma^{Z_{t_2}}(t_4) \le 0\})$.*

## ЗАКЛЮЧЕНИЕ

В данной статье были введены динамические приоритеты в контексте ВСП. Посредством этого расширения ВСП удобно моделировать системы, в которых работа некоторых компонентов приобретает значимость над работой других компонентов в зависимости от обстоятельств, поэтому ВСПсДП более выразительны, по сравнению с ВСПП.

Также, в статье была предложена альтернативная характеризация зон ВСПсДП, позволяющая описать все состояния зоны посредством маркировки и системы неравенств. Такая характеризация способствует эффективному построению конечной абстракции поведения ВСПсДП – графа зон.

В дальнейшем планируется разработать алгоритм построения графа зон для ВСПсДП с последующим его применением для параметрической верификации ВСПсДП.

## СПИСОК ЛИТЕРАТУРЫ

1. R. Alur, D. Dill: The theory of timed automata. Theoretical Computer Science. 126 (1994) 183-235.

2. M. Hack: Petri net languages. CSG Memo 124, Proj. MAC, M.I.T. (1975).
3. P. Merlin, D.J. Faber: Recoverability of communication protocols. IEEE Trans. of Communication, 24-9 (1976).
4. B. Berthomieu, M. Diaz: Modeling and verification of time dependent system using time Petri nets. IEEE Trans. of Communication, 17-3 (1991) 259-273.
5. T. Yoneda, H. Ryuba: CTL model checking of Time Petri nets using geometric regions. IEEE Trans. of Information and System, E99-D(3) (1998) 1-10.
6. B. Berthomieu, F. Vernadat: State class constructions for branching analysis of time Petri nets. In TACAS'2003 Warsaw, Poland. Lecture Notes in Computer Science 2619 (2003) 442-457.
7. B. Berthomieu, F. Peres, F. Vernadat: Model checking bounded prioritized time Petri nets. Toulouse, France. LAAS-CNRS
8. G. Gardey, O. H. Roux, O. F. Roux: Using zone graph method for computing the state space of time Petri nets. FORMATS'03, volume 2791 (2003).
9. B. Berthomieu, M. Menasche: An enumerative approach for analyzing time Petri nets. IFIP Congress Series 9 (1983) 41-46.
10. M. Koutny: Modelling systems with dynamic priorities. Lecture Notes in Computer Science 609 (1992) 251-266.
11. E. Best, M. Koutny, Petri net semantics of priority systems. Theoretical Computer Science96(1) (1992) 175-215.

## СОДЕРЖАНИЕ

## CONTENTS

**МОЛОДАЯ ИНФОРМАТИКА**

**Вып. 4**

**СБОРНИК ТРУДОВ**
**АСПИРАНТОВ И МОЛОДЫХ УЧЕНЫХ**

**Под редакцией**
**к.ф.-м.н. Н.С. Грибовской**

Рукопись поступила в редакцию 07.11.2014

Подписано в печать 10.12. 2014
Формат бумаги 60 × 84 1/16 Объем 5.1 уч.-изд.л., 5.6 п.л.
Тираж 60 экз.

Центр оперативной печати «Оригинал 2»
г. Бердск, ул. Островского, 55, оф. 02, тел. (383) 214-45-35