УДК 004.75

**Д.К. ГЛОМОЗДА, Н.Н. ГЛИБОВЕЦ, А.Н. МАКСИМЕЦ**

# АВТОМАТИЗАЦИЯ ПРЕОБРАЗОВАНИЯ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ С КАЧЕСТВЕННЫМИ ФИШКАМИ В РАСКРАШЕННЫЕ СЕТИ ПЕТРИ С КОЛИЧЕСТВЕННЫМИ ФИШКАМИ

**Аннотация.** Описан алгоритм преобразования цветной сети Петри с качественными фишками в раскрашенную сеть Петри с количественными фишками с сохранением ограниченности, взаимоисключаемости и живучести. Такое преобразование позволяет применить к раскрашенной сети Петри метод инвариантов, использующий алгоритм поиска усеченного множества решений уравнения состояния сети Петри, которое записывается в виде систем линейных однородных диофантовых уравнений. Работоспособность алгоритма продемонстрирована на примере цветной сети Петри, моделирующей работу грид-системы. Эквивалентность сетевых моделей проверена путем построения и анализа эквивалентных им конечных автоматов.

**Ключевые слова:** раскрашенные сети Петри, диофантовые уравнения, конечные автоматы, грид-структура.

## ВВЕДЕНИЕ

Современные информационные системы становятся все более сложными и разноплановыми, вследствие чего требуется согласованность в работе большого количества подсистем, нередко создаваемых независимо одна от другой различными коллективами разработчиков. В связи с этим возникает необходимость в создании таких моделей информационных систем, которые являлись бы ориентиром для разработчиков при планировании их построения и позволяли исследовать и верифицировать свойства уже имеющихся программных комплексов. Поскольку нужно моделировать не только структуру системы, а и ее в общем случае недетерминированное поведение, а также учитывать многопоточность и обеспечивать масштабируемость модели, популярным средством моделирования таких систем являются раскрашенные сети Петри (РСП) [1]. Последние широко применяются для моделирования бизнес-процессов в производственных системах и логистических сетях [2], для верификации распределенных систем [3], разработки аппаратных компонентов и систем научного, гражданского и военного назначения [4]. Интерактивные автоматизированные системы построения и анализа РСП, например CPN Tools [5, 6], ProM [7] и LoLA [8], широко известны и способствуют повышению спроса на эффективные алгоритмы верификации построенных с их помощью сетевых моделей.

Основными подходами к верификации свойств РСП являются анализ графов достижимости и покрытия, а также структурный анализ [1]. Анализ графов достижимости и покрытия дает возможность выявлять и верифицировать широкий спектр свойств РСП, однако при этом возникает проблема взрывоподобного разрастания множества возможных состояний. В случае структурного анализа такой проблемы не существует, но с его помощью можно изучать лишь ограниченный набор свойств РСП, используя при этом не исчерпывающие данные, как при исследовании графов достижимости и покрытия, а выполняя необходимые и/или достаточные условия, сформулированные на основании инвариантов мест и переходов. Однако в сочетании с эффективными алгебраическими алгоритмами, например алгоритмом нахождения усеченного множества решений (TSS-алгоритм) системы линейных однородных диофантовых уравнений (СЛОДУ), приме-

ненным к записанному в ее виде уравнению состояний сети Петри (СП), этот подход позволяет вычислить S- и T-инварианты заданной СП и верифицировать такие ее свойства, как достижимость, ограниченность, повторяемость, консервативность и непротиворечивость [9, 10].

Об эффективности данного подхода свидетельствуют примеры его использования для исследования СП, моделирующих взаимодействие абонентов в телефонной сети с применением разнообразных потенциально конфликтующих сервисов [11, 12], организацию движения железнодорожного транспорта на узле взаимосвязанных специализированных станций [13], функционирование вычислительного узла грид-системы [14]. Однако использованные в этих работах чистые и компонентные СП уступают РСП мощностью и выразительностью. В то же время автоматизированный анализ РСП с помощью метода инвариантов затруднен ввиду его функциональной, а не целочисленной, как у классических СП, природы [15]. Поэтому актуальна задача нахождения способа применения к РСП метода определения S- и T-инвариантов.

В работах [16, 17] подход на основе S- и T-инвариантов применен к РСП-моделям взаимодействия потоков в многопоточных Java-приложениях по шаблону «производитель–потребитель» с использованием комбинаций методов wait()/notifyAll() и wait()/notify(). Также в них описан способ преобразования РСП-модели из базовой формы с качественными, т.е. семантически нагруженными фишками, в форму с количественными, т.е. семантически ненагруженными фишками, характерными для классических СП. Анализ двух форм моделей показал сохранение их основных свойств, в том числе отсутствие дедлоков в варианте wait()/notifyAll() и их наличие в варианте wait()/notify(). Полученные результаты легли в основу созданной авторами утилиты CpnToUnit для автоматического преобразования РСП с качественными фишками в аналогичную РСП с количественными фишками.

В настоящей статье описывается работа утилиты CpnToUnit на примере РСП-модели вычислительной среды грид-системы, построенной на основе СП-модели, предложенной в [18]. В разд. 1 описывается построенная авторами с помощью пакета CPN Tools РСП-модель вычислительной среды грид-системы с качественными фишками. В разд. 2 представлена архитектура и основные функции утилиты CpnToUnit. В разд. 3 рассматривается РСП-модель с количественными фишками, сгенерированная с помощью утилиты CpnToUnit.

## 1. РСП-МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СРЕДЫ ГРИД-СИСТЕМЫ С КАЧЕСТВЕННЫМИ ФИШКАМИ

СП-модель вычислительной среды грид-системы [18, рис. 4], далее $G_0$, состоит из $10+2m+s$ мест и $8+m$ переходов, где $m$ — количество вычислителей, $s$ — количество носителей данных. Построенная авторами РСП-модель $G_1$ вычислительной среды грид-системы с качественными фишками (рис. 1) состоит из восьми мест и восьми переходов с начальной маркировкой $M_0 = (-, 1'(15, [1, 2, 3, 4, 5]) + 1'(30, [1]), -, 1'10 + 1'20 + 1'30 + 1'40, -, -, -, 1'0)$. Ввиду выразительной мощности РСП, фишки которой могут являться списками задач, вычислителей и носителей данных, ее размер не зависит от их количества.

В модели $G_1$ используются следующие множества цветов:

— colset ID = INT;
— colset DATA = list INT;
— colset DATA_LIST = list DATA;
— colset LEVEL = INT;
— colset NODE = LEVEL;
— colset TASK = product LEVEL * DATA;
— colset TASK_NODE_MAPPING = product LEVEL * NODE.

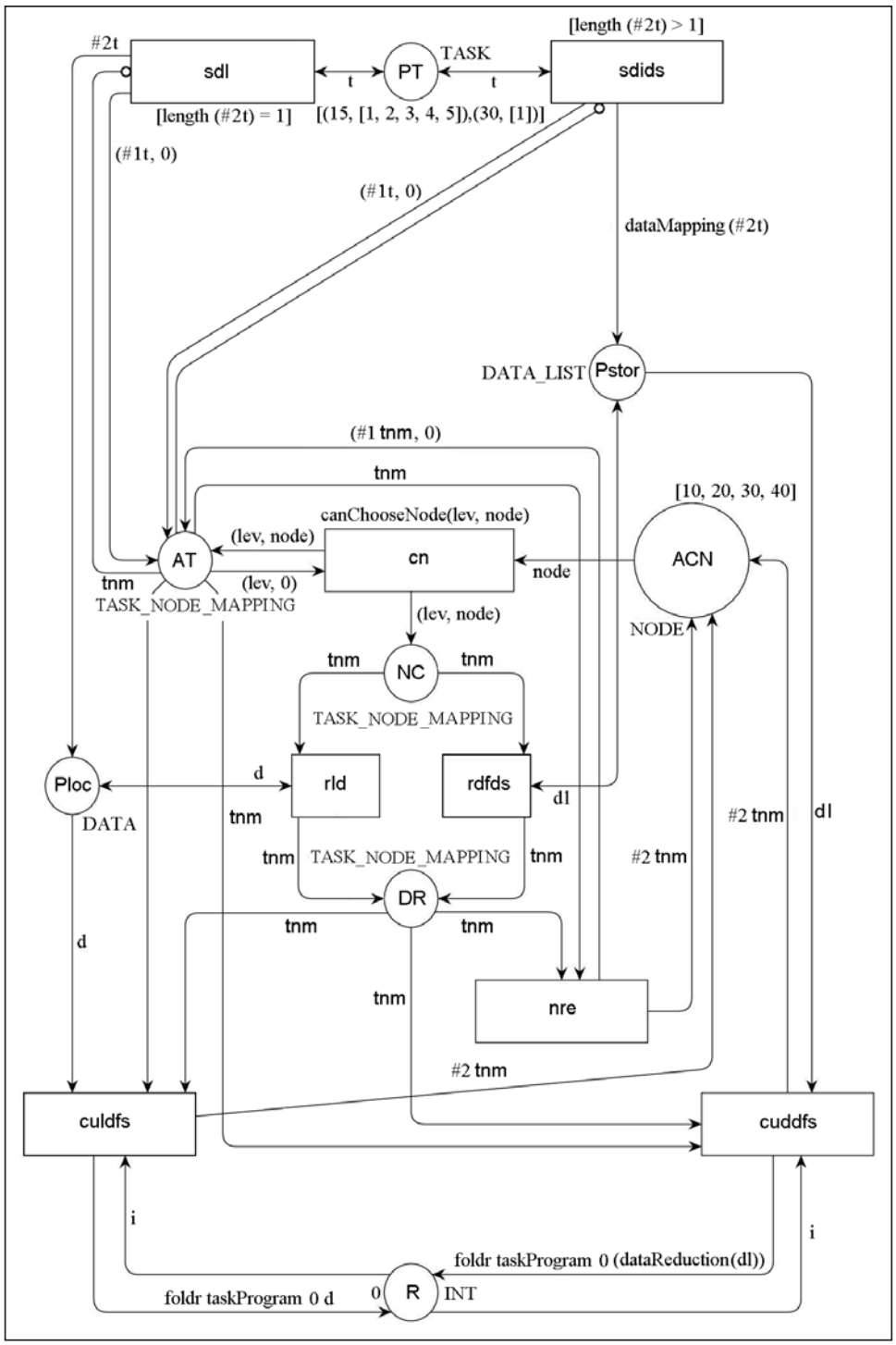


*Рис. 1.* РСП-модель $G_1$

Множества цветов ID, LEVEL и NODE эквивалентны множеству целых чисел и введены для бóльшей выразительности. Цвет ID обозначает идентификатор задачи либо вычислительного узла, LEVEL — уровень сложности задачи, которому должна соответствовать мощность вычислителя, NODE — вычислитель заданной мощности. Множество DATA — списки целых чисел для представления

наборов данных, поступающих с задачами. Множество TASK образовано в результате декартового произведения множеств LEVEL и DATA, и его элементы являются задачами определенного уровня сложности с определенным набором входных данных. Множество TASK_ NODE_ MAPPING — декартово произведение множеств LEVEL и NODE, необходимое для определения, какому вычислительному узлу поручена поступившая задача.

В табл. 1 приведено взаимосоответствие мест и переходов моделей $G_0$ и $G_1$. Вследствие способности фишек РСП хранить и передавать данные и состояния, а также использования сторожевых условий, в частности функции *canChooseNode*(*lev* : *LEVEL*, *node* : *NODE*) : *BOOL* = ((*lev* > 0) *andalso* (*node* >= *lev*)), определяющей наличие свободного вычислительного узла, пригодного для решения поступившей задачи, модель $G_1$ гораздо компактнее $G_0$. В результате усовершенствования модели поведения системы при отказе вычислителя задачу можно поручить другому пригодному вычислителю еще до восстановления работоспособности предыдущего вычислителя. Дополнительно смоделирована иная обработка задач с входными данными, превышающими объем локального хранилища: если в задающем входные данные списке больше одного элемента, данные отправляются в распределенное хранилище. С помощью подписей на дугах, входящих в место Result, смоделировано выполнение вычислений на основе входных данных, а именно суммирование элементов входного списка, умноженных на 10. Для этого использована функция языка ML *foldr*, определяемая следующим образом: *foldr f z l* = *f* (*e*l, *f* (*e*2, …, *f* (*en*, *z*)…)), где *l* = [*e*l, *e*2, …, *en*].

В модели $G_0$ результат вычислений поступает на вход системы и открывает переход $t_1$ [18, рис. 7], обеспечивая ее живучесть. В модели $G_1$ это привело бы к неограниченности. Поэтому при срабатывании переходов sdl (локальное сохранение данных) и sdids (сохранение данных в распределенном хранилище) фишка TASK, соответствующая выбранной для обработки задаче, возвращается в исходное место PT (задачи, ожидающие выполнения). Поскольку по условию не могут одновременно выполняться несколько задач, в модель включены исходящие из места AT (активные задачи) ингибиторные дуги, не позволяющие переходам sdl и sdids сработать, если задача для выполнения уже выбрана.

**Таблица 1**

| Модель $G_0$ | Модель $G_1$ |
|---|---|
| Места | |
| 1 | R |
| 10, 14 | PT, AT |
| 11 | Ploc |
| 12, 15, 16 ... ($Д_1$, $Д_2$ … $Д_s$) | Pstor |
| 17 | NC |
| 18, 19, 26, 27 | DR |
| 20–25 | ACN |
| Переходы | |
| $t_8$ [16, рис. 7] | sdl |
| $t_{10}$ | sdids |
| $t_{11}$, $t'_{11}$, $t''_{11}$ | cn |
| $t_{12}$ | rld |
| $t_{13}$, $t_{14}$ | rdfds |
| $t_{15}$ | culdfs, cuddfs |
| $t_{16}$, $t_{17}$, $t_{18}$ | nre |

Убедимся, что модель $G_1$ имеет такие же свойства, что и модель $G_0$, а именно живучесть, ограниченность и справедливость [18, теоремы 1 и 2]. Воспользуемся методом анализа графов достижимости и рассмотрим отчет, сгенерированный инструментарием CPN Tools для исследования пространства состояний (State Space Tool) [19]. Согласно отчету граф достижимости модели $G_1$ имеет 39 вершин, следовательно, она ограниченная. Поскольку терминальных вершин и мертвых переходов не существует, модель живая. Что касается справедливости, то переход cn (выбор вычислителя) срабатывает бесконечно часто во всякой бесконечной последовательности срабатываний (БПС), переходы rdfds и rld (чтение данных из распределенного и локального хранилищ соответственно) — во всех БПС, в которых они бесконечно часто от-

крываются, а переходы cuddfs (успешное завершение вычислений на основе данных из распределенного хранилища), culdfs (успешное завершение вычислений на основе локальных данных), nre (ошибка вычислителя), sdids и sdl — во всех БПС, где они с какого-то момента становятся открытыми. Ущемленных переходов, т.е. таких, для которых существует БПС, где они с какого-то момента становятся открытыми, но при этом никогда больше не срабатывают, не имеется. Таким образом, как и в модели $G_0$, если задание находится в грид-системе, то оно рано или поздно будет выполняться.

Полученные результаты позволяют сформулировать такое утверждение.

**Теорема 1.** Модель $G_1$ является ограниченной, живой и в ней выполняется свойство справедливости.

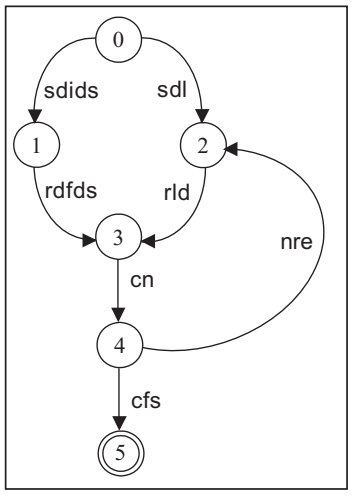


*Рис. 2.* Конечный автомат $A_0$

Рассмотрим эквивалентность моделей $G_0$ и $G_1$ с точки зрения распознаваемых ими формальных языков. Для этого представим модели в виде конечных автоматов. Конечный автомат $A_0$, соответствующий модели $G_0$ (рис. 2), имеет шесть состояний и семь переходов. Переход sdl обозначает срабатывание перехода $t_{11}$ при отсутствии фишки в позиции Pstor, sdids — срабатывание переходов $t_{10}$ и $t_{11}$, rld и rdfds — срабатывание переходов $t_{12}$ и $t_{13}$ соответственно, cn — срабатывание перехода $t_{14}$, cfs (успешное завершение вычислений) — перехода $t_{15}$. Наконец, переход nre соответствует последовательному срабатыванию переходов $t_{16}$, $t_{17}$ и $t_{18}$, моделирующих реакцию системы на аварийное состояние вычислителя и восстановление его работоспособности. Отметим, что после ликвидации последствий аварийной ситуации вычисления можно проводить только с использованием локальных данных, поскольку переход $t_{16}$ не связан с позицией Pstor.

Для построения конечного автомата, соответствующего модели $G_1$, воспользуемся библиотекой FSM от AT&T [20]. Для перевода РСП-модели, созданной с помощью CPN Tools, в формат, воспринимаемый библиотекой FSM, используем функции *og2fsmtrans* и *og2fsmhalts* [21]. Дополним модель $G_1$ следующим блоком кода на языке ML:

```
fun be2str (Bind.Net_Model'sdl (_,_)) = "1"
  | be2str (Bind.Net_Model'sdids (_,_)) = "2"
  | be2str (Bind.Net_Model'cn (_,_)) = "3"
  | be2str (Bind.Net_Model'rld (_,_)) = "4"
  | be2str (Bind.Net_Model'rdfds (_,_)) = "5"
  | be2str (Bind.Net_Model'culdfs (_,_)) = "6"
  | be2str (Bind.Net_Model'cuddfs (_,_)) = "7"
  | be2str (Bind.Net_Model'nre (_,_)) = "8"
  | be2str (_) = "0"
fun isHaltMarking(n : Node) : BOOL = ((n = 14) orelse (n = 15));
fun ArcToFSM a = be2str(ArcToBE(a));
fun FindHalts n = isHaltMarking(n);
use "og2fsm.sml";
og2fsmtrans ArcToFSM "trans-G1.txt";
og2fsmhalts FindHalts "halts-G1.txt";
```

Здесь функция *be2str* присваивает переходам исходной РСП-модели цифровые индексы, используемые средствами библиотеки FSM для обозначения переходов

автомата, функция *isHaltMarking* определяет конечные состояния автомата, каковыми считаются вершины графа достижимости с номерами 14 и 15, соответствующие маркировкам

$$M_{13} = (-, 1'(15, [1, 2, 3, 4, 5]) + 1'(30, [1]), -, 1'10 + 1'20 + 1'30 + 1'40, -, -, -, 1'150),$$

$$M_{14} = (-, 1'(15, [1, 2, 3, 4, 5]) + 1'(30, [1]), -, 1'10 + 1'20 + 1'30 + 1'40, -, -, -, 1'10),$$

где 1'150 и 1'10 — фишки в месте R (результат вычислений), обозначающие результат вычислений на основе наборов входных данных [1, 2, 3, 4, 5] и [1] соответственно (нумерация вершин графа достижимости начинается с единицы, а маркировок — с нуля). Далее функция *og2fsmtrans* создает текстовый файл, содержащий информацию о переходах модели $G_1$ в формате «индекс входного места – индекс выходного места – индекс перехода автомата», а функция *og2fsmhalts* — текстовый файл, содержащий перечень вершин графа достижимости РСП, соответствующих конечным состояниям автомата. После этого средствами библиотеки FSM с помощью последовательности команд, приведенной в [21], создаются файлы, описывающие конечный автомат.

Конечный автомат $A_1$, соответствующий модели $G_1$ (рис. 3), имеет восемь состояний и 12 переходов. В отличие от автомата $A_0$ он различает успешное завершение вычислений и восстановление после аварийной ситуации в случаях использования локальных данных и данных из распределенного хранилища. Кроме того, в силу живучести модели $G_1$ из конечного состояния 7 можно перейти к выполнению следующей задачи. Таким образом, формальный язык, распознаваемый автоматом $A_1$, мощнее языка, распознаваемого автоматом $A_0$, однако не за счет слов, соответствующих недопустимым состояниям моделируемой грид-системы, а за счет корректировок, внесенных в модель $G_1$ в целях увеличения степени детализации моделирования исходной грид-системы.

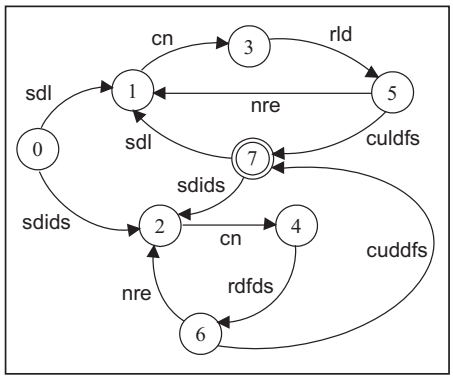


*Рис. 3*. Конечный автомат $A_1$

## 2. УТИЛИТА CPNTOUNIT

Разработанная авторами утилита CpnToUnit представляет собой консольное Java-приложение, входными аргументами которого являются имя файла исходной РСП-модели, созданной с помощью пакета CPN Tools, и опционально имя файла, в который будет записана РСП-модель с количественными фишками. Если второй параметр не указан, файл получает имя вида <имя_исходного_файла>_CpnToUnit.cpn.

Работа программы состоит из таких основных этапов:

— входной .cpn-файл переводится из формата XML в объектное представление;

— на основе объектного представления РСП с качественными фишками создается объектное представление РСП с количественными фишками;

— для полученного объектного представления строится матрица инцидентности, которая используется для нахождения S- и T-инвариантов РСП с помощью TSS-алгоритма;

— на основе полученного объектного представления генерируется XML-представление, которое записывается в выходной .cpn-файл для дальнейшей обработки средствами CPN Tools.

Объектное представление РСП с количественными фишками создается выполнением следующих шагов:

— пары дуг $A_{pxtytypx} = (a_{pxty}, a_{typx})$ такие, что $a_{pxty} = (p_x, t_y)$, $a_{typx} = (t_y, p_x)$, заменяются двунаправленными дугами $a_{pxtytypx}$. Поскольку в РСП с количественными фишками функциональные подписи дуг не используются, наличие отдельных входных и выходных дуг, ранее примененных для определения входных и выходных данных, поступающих из места в переход и обратно, более не требуется и их можно слить воедино;

— ингибиторные дуги $(p_i, t_j)_{inh}$ заменяются комбинацией из сигнального места $p_s \in \bullet t_j$ и дуги $(p_i, t_j)$, при этом если в месте $p_i$ имелись фишки, то место $p_s$ фишек не имеет, а если в месте $p_i$ фишек не имелось, то в место $p_s$ ставится одна фишка типа *UNIT*, что соответствует условиям открытости/закрытости перехода $t_j$ при наличии входящих в него ингибиторных дуг;

— места $p_i$, маркированные списками фишек из $N_i$ элементов, разворачиваются, т.е. заменяются множеством мест $\{p_{in}, 0 \le n \le N_i\}$, маркированными фишками типа *UNIT* (если список состоял из целых чисел, созданные места маркируются соответствующим количеством фишек типа *UNIT*). Еще неразвернутые переходы $\{t_j : p_i \in \bullet t_j\}$ и $\{t_k : p_i \in t_k \bullet\}$ разворачиваются во множества переходов $\{t_{jn} : p_{in} \in \bullet t_{jn}\}$ и $\{t_{kn} : p_{in} \in t_{kn} \bullet\}$ соответственно. Если $\{t_k\} \ne \varnothing$, то каждый переход из множества $\{t_{jn}\}$ соединяется двунаправленной дугой с каждым местом из множества $\{p_{im} : p_{im} \in \{p_{in}\}, m \ne n\}$, за счет чего исключается возможность срабатывания переходов из множества $\{t_{jn}\}$;

— если переход $t_z \in \{t_j : p_i \in \bullet t_j \cup t_k : p_i \in t_k \bullet\}$ уже был развернут в процессе развертывания другого места, то переходы из полученного в результате предыдущего разворачивания множества $\{t_{zm}\}$ соединяются такими же дугами с каждым местом из полученного в результате разворачивания множества $\{p_{in}\}$, за счет чего исключается возможность переходов из множества $\{t_{zm}\}$;

— со всех переходов убираются сторожевые условия, сегменты кода, часовые метки и приоритеты срабатывания;

— со всех дуг убираются подписи (пустая подпись аналогична передаче переменной типа *UNIT*), после чего все дуги, инцидентные местам, маркированным натуральными числами $n_i > 1$, подписываются равными натуральными числами (что аналогично добавлению $n_i$ кратных дуг);

— места и переходы сортируются в порядке возрастания имен.

Матрица инцидентности, построенная на основе полученного объектного представления РСП, используется модулем поиска S- и T-инвариантов с помощью TSS-алгоритма. Результатом работы является текстовый файл <имя_выходного_файла>_TSS_Report.txt. Он содержит отсортированные по именам нумерованные списки мест и переходов для облегчения их сопоставления со строками и столбцами матрицы инцидентности и координатами векторов-инвариантов, а также матрицу инцидентности и списки S- и T-инвариантов, информацию о наличии координат, принимающих во всех соответствующих S- и T-инвариантам векторах значение 0, что является признаком соответственно неограниченности и неповторяемости РСП.

Исходный код утилиты CpnToUnit представлен в [22].

### 3. РСП-МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СРЕДЫ ГРИД-СИСТЕМЫ С КОЛИЧЕСТВЕННЫМИ ФИШКАМИ

Полученная в результате работы утилиты CpnToUnit РСП-модель вычислительной среды грид-системы с количественными фишками состоит из $7 + t + m$ мест и $2 + 2t + 4m$ переходов, где $t$ — количество требующих решения задач, $m$ — количество вычислителей. В случае двух задач и четырех вычислителей модель (далее $G_2$) состоит из 13 мест и 22 переходов с начальной маркировкой $M_0 = (0, 1, 10, 20, 30, 40, 0, 0, 1, 1, 0, 0, 1)$ (рис. 4).

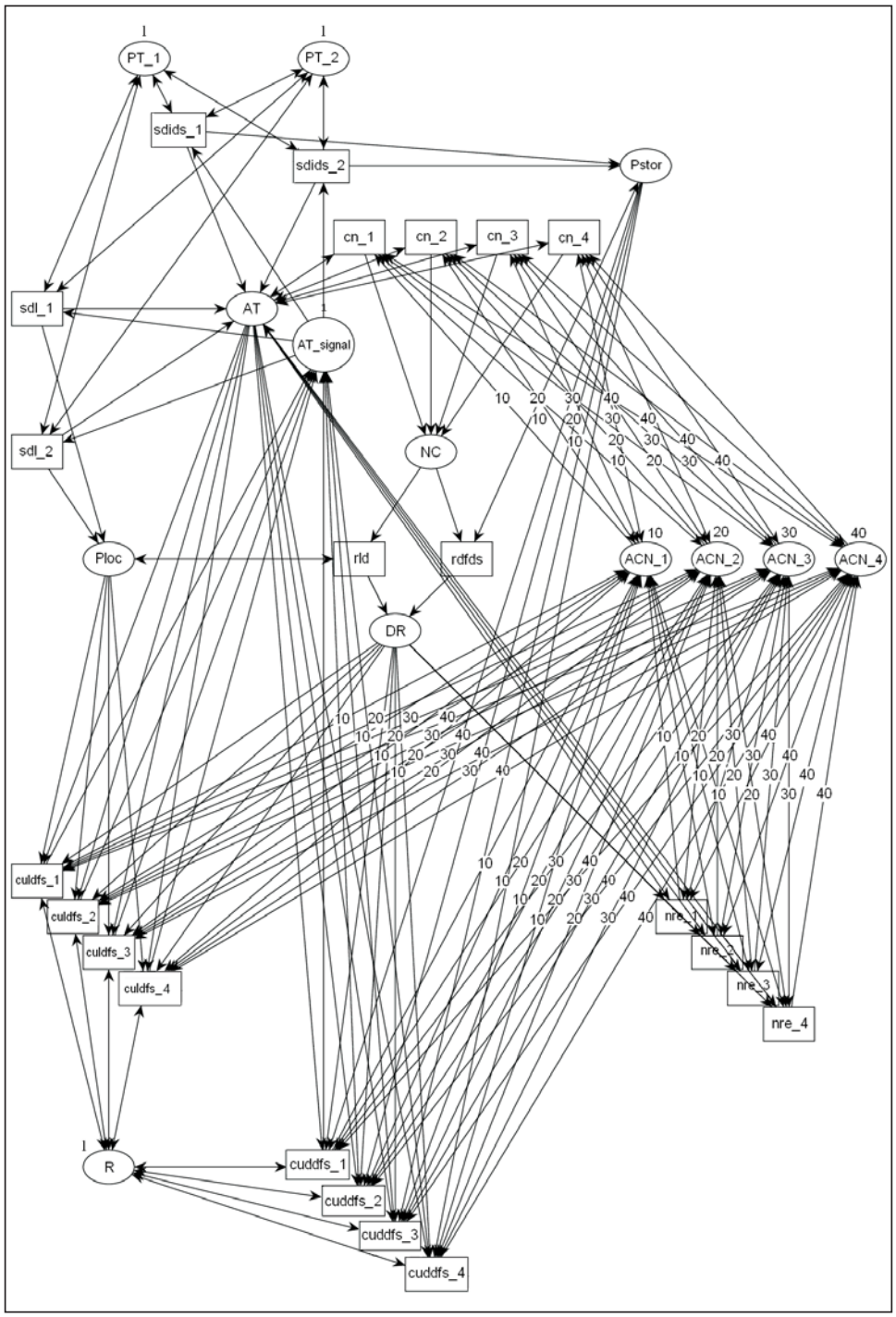


*Рис. 4*. РСП-модель $G_2$

Убедимся, что модель $G_2$, как и модель $G_1$, имеет свойства живучести, ограниченности и справедливости. Воспользуемся методом структурного анализа на основе уравнения состояний. Построенная средствами утилиты CpnToUnit матрица инцидентности $A$ модели $G_2$ имеет вид

$$A=\begin{pmatrix}
0&0&0&0&-1&-1&-1&-1&-1&-1&-1&-1&0&0&0&0&0&0&1&1&1&1\\
0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&-1&-1&-1&-1\\
-1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&-1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&-1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&-1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&-1&-1&-1&-1&-1&-1&-1&-1&-1&-1&-1&-1&1&1&0&0&0&0\\
1&1&1&1&0&0&0&0&0&0&0&1&0&0&1&0&-1&-1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&-1&-1&-1&-1&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&-1&-1&-1&-1&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}.$$

Полученное средствами утилиты CpnToUnit в результате решения СЛОДУ $Ay=0$ множество Т-инвариантов состоит из 26 линейно независимых векторов:

$$x_1=(1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0),$$
$$x_2=(1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0),$$
$$x_3=(1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0),$$
$$x_4=(1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0),$$
$$x_5=(1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0),$$
$$x_6=(1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0),$$
$$x_7=(0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0),$$
$$x_8=(0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0),$$
$$x_9=(0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0),$$
$$x_{10}=(0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0),$$
$$x_{11}=(0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0),$$
$$x_{12}=(0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0),$$
$$x_{13}=(0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0),$$
$$x_{14}=(0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0),$$
$$x_{15}=(0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0),$$
$$x_{16}=(0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0),$$
$$x_{17}=(0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,1,0,1,0,0,0),$$
$$x_{18}=(1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,2,0,1,0,1,0),$$
$$x_{19}=(1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,1,1,0,1,0),$$
$$x_{20}=(1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,2,0,1,0,0,1),$$
$$x_{21}=(1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,1,1,0,0,1),$$
$$x_{22}=(1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,1,1,0,1,0),$$
$$x_{23}=(1,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,2,1,0,1,0),$$
$$x_{24}=(0,2,0,0,0,1,0,0,0,1,0,0,0,0,0,0,2,0,1,0,1,0),$$
$$x_{25}=(0,1,1,0,0,1,0,0,0,0,1,0,0,0,0,0,2,0,1,0,1,0),$$
$$x_{26}=(0,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0,2,0,1,0,1,0).$$

Инварианты $x_1, x_2, x_7, x_8, x_{13}, x_{14}, x_{16}$ соответствуют аварийным ситуациям с вычислителями при обработке локальных либо распределенных данных. Инварианты $x_3, x_4, x_5, x_6, x_9, x_{10}, x_{11}, x_{12}, x_{15}, x_{17}$ соответствуют ситуациям успешного решения одной из задач на одном из вычислителей с использованием локального либо распределенного хранилища. Остальные инварианты соответствуют ситуациям успешного решения обеих задач на одном и том же либо на двух различных вычислителях с использованием локального или распределенного хранилища.

Из этого множества инвариантов следует, что все переходы в модели $G_2$ живые. Кроме того, все переходы покрываются ненулевыми координатами, следовательно, РСП повторяема.

Полученное средствами утилиты CpnToUnit в результате решения СЛОДУ $A^{\mathrm{T}} y = 0$ множество S-инвариантов состоит из восьми векторов:

(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0),

(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0),

(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1),

(1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),

(0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0),

(0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0),

(1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0),

(0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0).

Из этого множества инвариантов следует, что РСП ограниченная, поскольку все места покрываются ненулевыми координатами. Отсутствие единичного вектора свидетельствует об отсутствии дедлоков.

Сравним результаты структурного анализа с результатами анализа графа достижимости, проведенного средствами пакета CPN Tools. Согласно ему граф достижимости модели $G_2$ имеет 19 вершин, следовательно, она ограниченная. Для сравнения, у графа достижимости модели $G_1$ 39 вершин, т.е. переход от качественных фишек к количественным привел к сужению пространства достижимых состояний. Терминальных вершин и мертвых переходов не имеется, следовательно, модель живая. Что касается справедливости, то переходы rdfs и rld срабатывают во всех БПС, в которых они бесконечно часто открываются, а переходы из групп cn, cuddfs, culdfs, nre, sdids и rdl — во всех БПС, где они с какого-то момента становятся открытыми. Ущемленных переходов, т.е. таких, для которых существует БПС, где они с какого-то момента становятся открытыми, но при этом никогда больше не срабатывают, не имеется. Мертвых маркировок, т.е. дедлоков, так же не имеется. Таким образом, подтверждается выполнение свойства справедливости, поскольку решению поступившей в грид-систему задачи не препятствует ни возникновение дедлоков, ни зацикливание системы.

Полученные результаты позволяют сформулировать такое утверждение.

**Теорема 2.** Модель $G_2$ является ограниченной, живой и в ней выполняется свойство справедливости.

Рассмотрим эквивалентность моделей $G_1$ и $G_2$ с точки зрения распознаваемых ими формальных языков, для чего построим конечный автомат $A_2$, эквивалентный РСП-модели $G_2$.

Дополним модель $G_2$ следующим блоком кода на языке ML:

```
fun be2str (Bind.Net_Model'sdl_1 (_,_)) = "1"
  | be2str (Bind.Net_Model'sdl_2 (_,_)) = "1"
  | be2str (Bind.Net_Model'sdids_1 (_,_)) = "2"
  | be2str (Bind.Net_Model'sdids_2 (_,_)) = "2"
```

```
  | be2str (Bind.Net_Model'cn_1 (_,_)) = "3"
  | be2str (Bind.Net_Model'cn_2 (_,_)) = "3"
  | be2str (Bind.Net_Model'cn_3 (_,_)) = "3"
  | be2str (Bind.Net_Model'cn_4 (_,_)) = "3"
  | be2str (Bind.Net_Model'rld (_,_)) = "4"
  | be2str (Bind.Net_Model'rrdfds (_,_)) = "5"
  | be2str (Bind.Net_Model'culdfs_1 (_,_)) = "6"
  | be2str (Bind.Net_Model'culdfs_2 (_,_)) = "6"
  | be2str (Bind.Net_Model'culdfs_3 (_,_)) = "6"
  | be2str (Bind.Net_Model'culdfs_4 (_,_)) = "6"
  | be2str (Bind.Net_Model'cuddfs_1 (_,_)) = "7"
  | be2str (Bind.Net_Model'cuddfs_2 (_,_)) = "7"
  | be2str (Bind.Net_Model'cuddfs_3 (_,_)) = "7"
  | be2str (Bind.Net_Model'cuddfs_4 (_,_)) = "7"
  | be2str (Bind.Net_Model'nre_1 (_,_)) = "8"
  | be2str (Bind.Net_Model'nre_2 (_,_)) = "8"
  | be2str (Bind.Net_Model'nre_3 (_,_)) = "8"
  | be2str (Bind.Net_Model'nre_4 (_,_)) = "8"
  | be2str (_) = "0"
fun isHaltMarking(n : Node) : bool = (n = 1);
fun ArcToFSM a = be2str(ArcToBE(a));
fun FindHalts n = isHaltMarking(n);
use "og2fsm.sml";
og2fsmtrans ArcToFSM "trans-G2.txt";
og2fsmhalts FindHalts "halts-G2.txt";
```

Здесь функция *be2str* связывает несколько переходов с одним цифровым индексом, что позволяет представить их одним состоянием автомата. Относительно функции *isHaltMarking*, то поскольку отдельной маркировки, описывающей наличие результата вычислений, в модели $G_2$ не предусмотрено, конечным состоянием автомата будем считать вершину графа достижимости с индексом 1, соответствующую начальной маркировке $M_0$.

Конечный автомат $A_2$, соответствующий модели $G_2$ (рис. 5), имеет семь состояний и десять переходов. Его можно считать оптимизированным вариантом автомата $A_1$ , полученным за счет слияния состояний 0 и 7 и удаления дубликатов переходов sdl и sdids. Автомат $A_2$ распознает все слова, распознаваемые автоматом $A_1$. Кроме того, поскольку начальное состояние 0 одновременно является конечным, автомат $A_2$ распознает пустое слово. В терминах исходной грид-системы пустое слово можно интерпретировать как отсутствие требующих выполнения задач, следовательно, конечный автомат $A_2$ и РСП-модель $G_2$ корректно моделируют исходную систему.

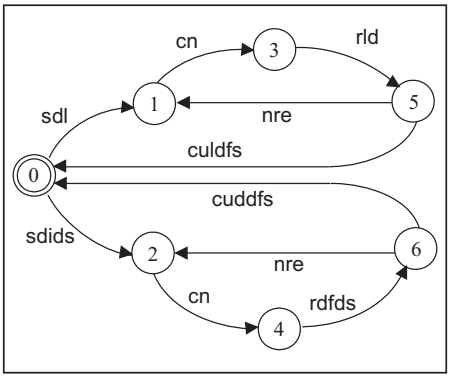


*Рис. 5*. Конечный автомат $A_2$

Файлы с моделями $G_1$ и $G_2$, сгенерированные пакетом CPN Tools и утилитой CpnToUnit отчеты с результатами их анализа, а также файлы, полученные в процессе построения автоматов $A_1$ и $A_2$ средствами библиотеки FSM, представлены в [22] в каталоге resources.

## ЗАКЛЮЧЕНИЕ

В статье предложено использование РСП для моделирования работы вычислительной среды грид-системы. Построена модель $G_1$ с качественными фишками, сохраняющая свойства ограниченности, живучести и справедливости исходной модели $G_0$ [18] с одновременным увеличением выразительности и степени детализации моделирования процесса ее работы. Построенные на основе РСП-моделей конечные автоматы $A_0$ и $A_1$ показали, что модель $G_1$ не только сохранила основные свойства модели $G_0$, но и более детально моделирует работу исходной грид-системы.

Кроме того, описана разработанная авторами утилита CpnToUnit для автоматического преобразования РСП с качественными фишками в РСП с количественными фишками. С ее помощью на основе модели $G_1$ построена и исследована модель $G_2$, сохранившая свойства ограниченности, живучести и справедливости, а также более высокую по сравнению с моделью $G_0$ степень детализации моделирования процесса работы вычислительной среды грид-системы. Созданные на основе РСП-моделей конечные автоматы $A_1$ и $A_2$ показали, что модель $G_2$ эквивалентна модели $G_1$ с точностью до распознавания пустого слова. Модель $G_2$ по сравнению с $G_1$ не только удобна для исследования методом структурного анализа, но и для анализа с помощью графа достижимости, поскольку имеет меньшее количество достижимых состояний: 19 против 39.

## СПИСОК ЛИТЕРАТУРЫ


1. Jensen K., Kristensen L.M. Coloured Petri nets. Modelling and validation of concurrent systems. Berlin: Springer, 2009. 384 p.
2. Van der Aalst W. M. P., Stahl C. Modeling business processes — a Petri net-oriented approach. Cambridge: The MIT Press, 2011. 400 p.
3. Chernenok S.A., Nepomniaschy V.A. The application of coloured Petri nets to verification of distributed systems specified by message sequence charts. *Proc. ISP RAS*. 2015. Vol. 27, N 3. P. 197–218.
4. Examples of industrial use of CP-nets. URL: http://www.daimi.au.dk/CPnets/intro/example_indu.html.
5. Jensen K., Kristensen L.M., Wells L. Coloured Petri nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer.* 2007. Vol. 9, N 3–4. P. 213–254.
6. Романников Д.О., Марков А.В. Об использовании программного пакета CPN Tools для анализа сетей Петри. *Сб. науч. трудов НГТУ.* 2012. № 2. С. 105–116.
7. ProM 6.6. URL: http://www.promtools.org/doku.php?id=prom66.
8. Schmidt K. LoLA: A low level analyser. *Proc. 21th International Conference on Application and Theory of Petri nets* (Aarhus, Denmark, June 26–30, 2000). Heidelberg: Springer-Verlag, 2000. P. 465–474.
9. Крывый С.Л. О вычислении минимального множества инвариантов сети Петри. *Искусственный интеллект*. 2001. № 3. С. 199–206.
10. Кривий С.Л. Лінійні діофантові обмеження та їх застосування. Чернівці; Київ: Букрек, 2015. 224 с.
11. Крывый С.Л., Матвеева Л.Е. О применении сетей Петри к решению некоторых задач телекоммуникации. *Искусственный интеллект.* 2002. № 3. С. 590–598.
12. Матвеева Л.Е. Автоматическая система анализа и верификации телекоммуникационной системы, описанной на языке MSC, с помощью формализма сетей Петри. *Проблеми програмування.* 2004. № 2–3. С. 108–117.
13. Лукьянова Е.А., Дереза А.В. Исследование однотипных структурных элементов CN-сети в процессе компонентного моделирования и анализа сложной системы с параллелизмом. *Кибернетика и системный анализ*. 2012. № 6. С. 20–29.
14. Шелестов А.Ю. Моделирование Grid-узла на основе сетей Петри. *Системні дослідження та інформаційні технології.* 2009. № 3. С. 52–65.
15. Jensen K. Coloured Petri nets and the invariant method. *Theoretical Computer Science*. 1981. Vol. 14. P. 317–336.

16. Гломозда Д.К. Застосування методу інваріантів до аналізу кольорових мереж Петрі. *Наукові записки НАУКМА. Комп'ютерні науки.* 2015. Т. 177. С. 44–52.
17. Гломозда Д.К. Застосування методу інваріантів до аналізу кольорових мереж Петрі із дедлоками. *Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка.* 2016. № 64. С. 38–46.
18. Крывый С.Л., Бойко Ю.В., Погорелый С.Д., Борецкий А.Ф., Глыбовец Н.Н. Проектирование грид-структур на основе транзиционных систем с обоснованием правильности их функционирования. *Кибернетика и системный анализ.* 2017. Т. 53, № 1. С. 122–133.
19. Jensen K. CPN tools state space manual. Aarhus: University of Aarhus, 2002. 49 p.
20. AT&T FSM Library 4.0 .URL: https://www3.cs.stonybrook.edu/~algorith/implement/fsm/ implement.shtml.
21. Gordon S. Converting coloured Petri net state space to finite state automata: CPNTools, FSM and Lextools. 2013. URL: https://sandilands.info/sgordon/cpntools-fsm-lextools.
22. Hlomozda D.K. CpnToUnit utility. URL: https://github.com/Gyrotank/CpnUnitTransformer.

*Надійшла до редакції 30.11.2017*

**Д.К. Гломозда, М.М. Глибовець, О.М. Максимець**

**АВТОМАТИЗАЦІЯ ПЕРЕТВОРЕННЯ КОЛЬОРОВИХ МЕРЕЖ ПЕТРІ З ЯКІСНИМИ ФІШКАМИ НА КОЛЬОРОВІ МЕРЕЖІ ПЕТРІ З КІЛЬКІСНИМИ ФІШКАМИ**

**Анотація.** Описано алгоритм перетворення кольорової мережі Петрі із якісними фішками на кольорову мережу Петрі із кількісними фішками зі збереженням обмеженості, взаємовиключності та живучості. Таке перетворення робить можливим застосування до кольорової мережі Петрі методу інваріантів, що використовує алгоритм пошуку зрізаної множини розв'язків рівняння стану мережі Петрі, яке записується у вигляді систем лінійних однорідних діофантових рівнянь. Працездатність алгоритму продемонстровано на прикладі кольорової мережі Петрі, яка моделює роботу грід-системи. Еквівалентність мережних моделей перевірено шляхом побудови та аналізу еквівалентних їм скінченних автоматів.

**Ключові слова:** кольорові мережі Петрі, діофантові рівняння, скінченні автомати, грід-структура.

**D.K. Hlomozda, M.M. Glybovets, O.M. Maksymets**

**AUTOMATING THE TRANSFORMATION OF COLORED PETRI NETS WITH QUALITATIVE TOKENS INTO COLORED PETRI NETS WITH QUANTITATIVE TOKENS**

**Abstract.** The authors describe an algorithm for transformation of colored Petri nets with qualitative tokens into colored Petri net with quantitative tokens preserving boundedness, mutual exclusion, and liveness properties. This transformation allows an invariance method to be applied to colored Petri nets, which uses Truncated Set of Solutions finding algorithm for Petri net state equations expressed through systems of linear homogenous Diophantine equations. To show the algorithm's efficiency, it is applied to the colored Petri net modeling the operation of a grid system. Equivalence of net models is tested by constructing and analyzing equal finite-state automata.

**Keywords:** colored Petri nets, Diophantine equations, finite-state machines, grid structure.

**Гломозда Дмитрий Константинович,**
кандидат техн. наук, старший преподаватель кафедры Национального университета «Киево-Могилянская академия», Киев, e-mail: glomozda@ukma.edu.ua.

**Глибовец Николай Николаевич,**
доктор физ.-мат. наук, профессор кафедры Национального университета «Киево-Могилянская академия», Киев, e-mail: glib@ukma.edu.ua.

**Максимец Александр Николаевич,**
инженер-программист компании «Facebook», Лос-Альтос (США), e-mail: maksymets@gmail.com.