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

**В.Е. Козюра, В.А. Непомнящий, Р.М. Новиков**

# ВЕРИФИКАЦИЯ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ МЕТОДОМ ПРОВЕРКИ МОДЕЛЕЙ

**Препринт**
**89**

**Novosibirsk 2001**

Для раскрашенных сетей Петри, ограниченных системами с конечным числом состояний, свойства которых представлены в мю–исчислении, разработана и реализована система верификации PNV (Petri net verifier), базирующаяся на методе проверки моделей. Наряду с описанием системы PNV в работе представлены эксперименты по верификации раскрашенных сетей Петри, моделирующих битовый и кольцевой коммуникационные протоколы, а также обедающих философов.

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

**Kozura V.E., Nepomniaschy V.A., Novikov R.M.**

# VERIFICATION COLOURED PETRI NETS BY MODEL CHECKING METHOD

**Preprint**
**89**

**Novosibirsk 2001**

A verification tool PNV (Petri net verifier) based on model checking method has been developed and implemented. The tool PNV is applied to coloured Petri nets limited by finite state systems when properties are presented in mu–calculus. The paper describes PNV and verification experiments with coloured Petri nets which model alternating bit and ring communication protocols as well as dining philosophers.

## 1.ВВЕДЕНИЕ

Верификация распределенных систем и, в частности, коммуникационных протоколов является быстро развивающейся областью современного программирования, практическое значение которой трудно переоценить. Естественный подход к проблеме верификации состоит в моделировании распределенных систем конечными автоматами или сетями Петри и в верификации полученных моделей.

Среди перечисленных моделей выделяются сети Петри высокого уровня — раскрашенные сети Петри [10], для которых развит теоретический аппарат, накоплен значительный опыт использования и реализована система симуляции и анализа Design/CPN[12]. Поэтому проблема верификации раскрашенных сетей Петри (РСП) методом проверки моделей актуальна. Отметим, что рассматривались две постановки этой проблемы для сетей Петри: относительно систем с конечным и бесконечным числом состояний. В [8] предложен метод проверки моделей относительно мю-исчисления линейного времени для стандартных сетей Петри, которые рассматриваются как системы с бесконечным числом состояний без каких-либо ограничений. Высокая оценка сложности этого метода существенно понижена в [9]. Для РСП, ограниченных системами с конечным числом состояний, разработан метод проверки моделей относительно новой логики, близкой к логике ветвящегося времени CTL, который позволяет в ряде случаев уменьшать пространство поиска [5].

Цель настоящей работы — описать систему верификации PNV (Petri net verifier), которая реализует метод проверки моделей для РСП, ограниченных системами с конечным числом состояний, относительно мю-исчисления. В разд. 2 кратко излагается стандартный метод проверки моделей для мю-исчисления. Разд. 3 посвящен определению РСП. В разд. 4 описана система верификации PNV. Разд. 5 посвящен описанию экспериментов с системой PNV, для которых были выбраны следующие примеры: задача об обедающих философах, битовый и кольцевой протоколы. В разд. 6 обсуждаются перспективы применения и развития системы PNV.

## 2. МЕТОД ПРОВЕРКИ МОДЕЛЕЙ ДЛЯ ФОРМУЛ МЮ-ИСЧИСЛЕНИЯ

Кратко опишем стандартный метод проверки моделей для мю-исчисления.

*Системой переходов* (моделью Крипке) T назовем тройку (S,Act,→), где S — множество состояний, Act — множество действий, а → — подмножество множества S × Act × S. Если <s1,a,s2> принадлежит →, то будем писать s1-a→s2. Системы переходов являются моделями для мю-исчисления.

Пусть мы имеем счетные непересекающиеся алфавиты пропозициональных констант C={P,Q...}, пропозициональных переменных V={X,Y...} и символов действий Act={a,b...}.

*Формулами мю-исчисления* назовем следующие выражения:

1. P, где P — пропозициональная константа.
2. X, где X — пропозициональная переменная.

Вхождение X в Φ позитивное, если X находится под действием четного числа отрицаний. Если Φ,Φ1,Φ2 — уже построенные формулы мю-исчисления, то формулами мю-исчисления также являются:

3. ¬Φ, Φ1 & Φ2, Φ1 ∨ Φ2.
4. <a>Φ, [a]Φ , где a — символ действия.
5. μX.Φ, νX.Φ , где X входит позитивно.

Других формул нет.

Ниже приведен пример формулы мю-исчисления.

Φ = νX1.μX2.(X1 ∨X2 ∨ νY1.μY2.νY3(Y1 & Y2 & Y3))

Пусть дана некоторая система переходов (S,Act,→). Определим семантику замкнутой формулы мю-исчисления. Для этого построим функцию I(Φ)→S. Пусть I определена на множестве C. Тогда

1. I(¬Φ)=S \ I(Φ). I(Φ1 ∨ Φ2)=I(Φ1)+I(Φ2).
   I(Φ1 & Φ2)=I(Φ1)-I(Φ2).
2. I(<a>Φ)={s ∈S | ∃ s' ∈ I(Φ):s-a→s'}.
   I([a]Φ)={s ∈ S | ∀s' : (s-a→s') => (s' ∈I(Φ) }.
3. I(μX.Φ(X))=пересечение множеств вида
   { L подмножество S | I(Φ(X))[X←L]=L }.
   I(νX.Φ(X))=обьединение множеств вида
   { L подмножество S | I(Φ(X))[X←L]=L }.
   Причем I(X)[X←L]=L , а I(P)[X←L]=I(P).

Теперь становится ясной задача проверки модели с использованием логики мю-исчисления: по входной формуле Φ и системе переходов T найти те s из S, на которых Φ истинна. В общем случае задача проверки моделей состоит в проверке истинности формулы в данном состоянии (так называемая локальная проверка модели) и выяснении тех состояний, в которых она истинна (глобальная проверка модели).

Выбранный алгоритм работает с представлением мю-формулы в виде таблиц равенств. Таблица равенств имеет вид <X1-Ф1...XN-ФN>, где X — это переменная, '-' — это либо '→' , либо '←', Ф — формула. Выражение Xi→Фi представляет наибольшую, а Xi←Фi — наименьшую неподвижную точки. Алгоритм перехода от мю-формулы к таблице имеет линейную сложность. Например для формулы из вышеуказанного примера таблица C будет следующей:

$$C = \langle X1 \rightarrow X2,\ X2 \leftarrow X1 \ \&\ X3,\ X3 \leftarrow X2 \vee X4,\ X4 \rightarrow X5,\ X5 \leftarrow X6,$$
$$X6 \rightarrow X4 \ \&\ X7,\ X7 \rightarrow X5 \ \&\ X6 \rangle.$$

Ниже будет неформально описан алгоритм вычисления значения мю-формулы по ее таблице равенств E в данной системе переходов $T=(S,Act,\rightarrow)$.

Алгоритм разбивается на этапы, состоящие из двух шагов: инициирование левых переменных E и их вычисление. При инициировании значения переменных берутся с предыдущего этапа. При первоначальном инициировании переменным, соответствующим наибольшей неподвижной точке, присваиваем значение true, а переменным, соответствующим наименьшей неподвижной точке — значение false. На шаге вычисления новые значения левых переменных вычисляются по правилам темпоральной логики без неподвижных точек при использовании текущих значений переменных и данных системы переходов. Вычисления идут по одной формуле таблицы E снизу вверх. Вычисления считаются законченными, когда значения вычисленных переменных полностью совпадут с их значениями на предыдущем шаге.

Алгоритм имеет экспоненциальную сложность, которая была улучшена, например в работе [6], для некоторых частных случаев, что, однако, является несущественным в случаях, рассматриваемых в работе формул.

## 3. РАСКРАШЕННЫЕ СЕТИ ПЕТРИ

Коротко напомним определения раскрашенных сетей Петри [10]. Базовым понятием при определении раскрашенных сетей Петри является мультимножество, которое отличается от известного понятия множества тем, что в мультимножестве каждый элемент может иметь несколько вхождений. Над мультимножествами естественным образом определяются операции сложения, умножения на скаляр, сравнения , вычитания и модуль (размер) мультимножества. Будем считать синтаксически и семантически понятными следующие конструкции: Type(v), Type(expr), Var(expr).

Раскрашенной сетью Петри (CP-сетью, CPN ) назовем следующую конструкцию:

CPN = (S, P, T, A, N, C, G, E, I) , где

S — конечное множество цветов;

P — конечное множество мест;

T — конечное множество переходов;

A — конечное множество дуг, такое что $P \cap T = P \cap A = T \cap A = \varnothing$;

N — отображающая функция из A в $P*T \cup T*P$;

C — функция раскраски , определенная из P в S;

G — функция логического условия на переход, определенная на T и удовлетворяющая условию:

$$\forall\, t \in T\ [Type(G(t))=Bool \wedge Type(Var(G(t))) \subseteq S];$$

E — функция на дугах, определенная на A и удовлетворяющая условию:

$\forall a \in A : (Type(E(a))=C(p) \wedge Type(Var(E(a))) \subseteq S )$, где p — место в N(a);

I — инициализирующая функция, определенная на P и удовлетворяющая условию: $\forall\, p \in P: (Type(I(p)) = C(p))$.

Естественным образом вводятся следующие множества:

$A(x) = \{a \in A | \exists x1 \in X: [N(a)=(x,x1) \vee N(a)=(x1,x)]\}$;

$\forall t \in T: Var(t) = \{v \in Var(G(t)) \vee \exists a \in A(t): v \in Var(E(a))\}$;

$A(x1,x2) = \{a \in A | N(a)= (x1,x2)\}$;

$E(x1,x2) = \sum E(a)$, где суммирование проводится по всем $a \in A(x1,x2)$.

*Означивание* перехода t есть функция b, определенная на Var(t) следующим образом:

$\forall v \in Var(t) : b(v) \in Type(v)$, при условии G(t)<b>, где G(t)<b> означает результат вычисления G(t) при означивании b.

*Знаковым элементом (token element)* назовем пару (p,c) , где $p \in P$, а $c \in C(p)$, а *означивающим элементом (binding element)* назовем пару (t,b) , где $t \in T$, а $b \in B(t)$ (множества всех означиваний t). Множество всех знаковых элементов будем обозначать TE, а множество всех означивающих элементов — BE.

*Разметкой* назовем мультимножество над множеством всех знаковых элементов, а *шагом* — непустое и конечное мультимножество над множеством всех означивающих элементов. *Начальной разметкой* $M_0$ назовем разметку, удовлетворяющую следующей формуле: $\forall (p,c) \in TE : M_0(p,c) = (I(p))(c)$.

Шаг Y возможен при разметке M тогда и только тогда, когда

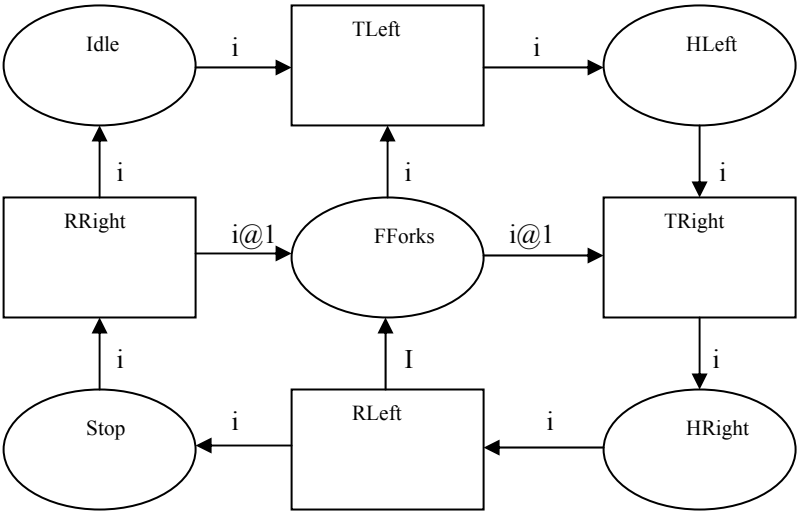
$$\forall p \in P : \ M(p) \ge \sum_{(t,b) \in Y} E(p,t)<b>.$$

Если шаг Y возможен при разметке M1, он может сработать, изменяя $M_1$ на $M_2$ по правилу:

$$\forall p \in P : M_2(p) = (M_1(p) - \sum_{(t,b)\in Y} E(p,t)<b>) + \sum_{(t,b)\in Y} E(t,p)<b> .$$

В этом случае разметку непосредственно достижимую из $M_1$ назовём $M_2$ и будем писать $M_1[Y\rangle M_2$. Множество достижимых разметок обозначается $[M_0\rangle$.

В качестве примера раскрашенной сети Петри рассмотрим сеть, представляющую известную задачу об обедающих философах (рис. 1), формулирующуюся следующим образом: за круглым столом обедают философы, у которых с каждой стороны лежит по одной вилке, тогда как для принятия пищи необходимо наличие двух вилок. На рис. 1 i@1 означает сложение i с 1 по модулю числа обедающих философов.




color PHIL = index ph with 1..3; place PHIL Idle=Fforks={1'1,1'2,1'3};
place PHIL Hleft=Hright=Stop=0;

*Рис.1.* Обедающие философы

Для проведения проверки моделей на раскрашенных сетях Петри в качестве системы переходов берется граф достижимости. Графом достижимости называется граф (V,A,N), где $V=[M_0\rangle$ — множество вершин , A —

множества троек $(M_1,b,M_2)\in V\times BE\times V$ таких, что $M_1[b\rangle M_2$ — множество дуг и N — функция смежности такая, что $a=(M_1,b,M_2)\in A \Rightarrow N(a)=(M_1,M_2)$.

Следующий алгоритм строит граф достижимости по данной РСП [10]:

```
Waiting:=∅
Node(M0)
repeat
    select a node M1∈Waiting;
    forall (b,M2)∈Next(M1) do
    begin
        Node(M2);
        Arc(M1,b,M2);
    end;
    Waiting:=Waiting-{M1};
until Waiting ≠ ∅;
```

Где $Next(M_1) = \{(b,M_2)\in BE\times M \mid M_1[b\rangle M_2\}$, а Node добавляет $M_2$ в Waiting.

Используя Граф достижимости в качестве системы переходов и формулы мю-исчисления, написанные для такой модели, мы можем решать задачи проверки моделей для формул мю-исчисления на раскрашенных сетях Петри.

## 4. СИСТЕМА ВЕРИФИКАЦИИ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ PNV

### 4.1. Общая структура системы PNV

В данной части описана система верификации моделей, построенных по раскрашенной сети Петри. Она состоит из двух частей: построение моделей по заданной спецификации раскрашенной сети Петри и проверка моделей, анализирующая результат, полученный в процессе работы первой части системы. Структура системы PNV представлена на рис. 2.

В системе кроме частей, необходимых для верификации раскрашенных сетей Петри, также имеются переводы сети как в программу на C++, так и в программу на языке СПЕКТР.


*Рис. 2.* Система PNV

Входной язык системы СПЕКТР-2 разработан для нужд верификации, базируется на логике Хоара и расширяет Паскаль посредством конструкции недетерминарованного выбора oneof. Действие этого оператора заключается в недетерминированном выборе элемента из некоторого домена. Например в коде:

type RangeInt := {1, 3, 5, 7, 9}; x := oneof ( RangeInt ) ;

значением x может быть любое нечетное число в диапазоне от 1 до 9.

Подобные конструкции в СПЕКТР-программе используются для моделирования недетерминизма поведения РСП, как например, выбор срабатывающего перехода из нескольких возможных, выбор означивания и т.п.

### 4.2. Подсистема построения моделей

Подсистема построения моделей состоит из транслятора, переводящего заданную спецификацию раскрашенной сети Петри в объектный код, и построителя моделей, который работает с этим кодом. Промежуточное построение объектного кода необходимо в связи с тем, что кроме модели, предназначенной для анализа, этот модуль может генерировать еще несколько выходов, которые далее не описываются.

Входная спецификация является описанием раскрашенной сети Петри и состоит из описания типов (цветов), используемых в сети, констант, переходов сети и её мест. Константы могут быть строками, заключенными в кавычки, вещественными или целыми числами и могут быть использованы в любом месте спецификации. Как и в случае описания констант, типы перечисляются через запятую, имеют имя и конструируются из нескольких базовых типов. Описание мест вводится декларацией *place* с указанием типа места, его имени и инициализирующего значения. Совокупность описанных в данном блоке мест соответствует начальной разметке сети. Процесс трансляции происходит методом рекурсивного спуска в один проход. После синтаксического и семантического анализа строится модель по алгоритму для построения графа достижимости из разд. 3.

Работа системы верификации проходит по следующему сценарию: системе подаётся на вход РСП во входном формате, транслятор порождает три выхода:

1) СПЕКТР-программу, моделирующую предоставленную на вход РСП,

2) интерпретируемую форму РСП,

3) программу на языке C++, моделирующую исполнение предоставленной РСП.

Получающаяся интерпретируемая форма РСП используется построителем моделей для генерации модели, и пользователь ни на какой стадии разработки или отладки сети не работает с ней напрямую. Моделирующая программа на языке C++ предназначена для пошаговой отладки РСП, в ходе которой пользователь сам выбирает переходы для срабатывания и их означивание. Система реализована на языке C++ для платформы Windows NT.

### 4.3. Подсистема проверки моделей

Подсистема проверки моделей состоит из синтаксического анализатора формул мю-исчисления, блока вычисления предикатов и собственно алгоритмической части проверки формулы на модели. Синтаксический анализатор дает на выходе таблицу разбора, необходимую для проведения прямого алгоритма проверки формулы мю-исчисления на данной модели. Входная модель, получаемая из блока генерации модели, используется без изменений. Для вычисления предикатов используется описание мест, полученное на этапе генерации модели.

Подсистема реализована в системе Delphi и кроме тестовых испытаний также прошла проверку при верификации выполнимых спецификаций языка Basic-Real [3].

Ниже приведены фрагменты модели и файла с описанием мест. Модель представлена списком, состоящим из троек: номер исходного состояния, имя перехода, номер полученного состояния. Например:

16 to 24, 24 to 26, 24 to 14, ...

Каждый номер однозначно идентифицируется состоянием в модели и представляется в файле описания мест. Например:

```
State number        : 5        places :
from state number   : 4              Active2   =1
with binding        :                    Data1     =0
  transition: m1                     Data2     =1
  binding   : f=[1 2 1 1 ]           Listen1   =1
                                     Out1m     =[1 2 1 1 ]
```

В описании представлены только непустые места исходной раскрашенной сети Петри, т.е. места, содержащие, по крайней мере, одну фишку.

## 5. ОПИСАНИЕ ЭКСПЕРИМЕНТОВ С СИСТЕМОЙ PNV

### 5.1. Обедающие философы

В качестве первого примера для верификации рассмотрим РСП (рис.1), представляющую задачу об обедающих философах. Сеть была промоделирована для n=2..5.

Проверялись следующие свойства:

***1. Свойства прогресса.***

а) μ x. ( [to](P_begin ∨ x) & <to>true ), где P_begin — предикат начальной разметки. Формула верна в моделях, являющихся единой компонентой связности. В случае обедающих философов был получен отрицательный ответ, и формула была ослаблена до следующей.

б) μ x. ( <to>(P_begin ∨ x) ). Формула верна в состояниях, представляющих компоненту связности, в которой верен предикат P_begin, т.е. проверяется возможность возврата в исходное состояние. Данная компонента связности была получена.

***2. Свойства тупиков.*** Ввиду отрицательного ответа в 1(а) встает вопрос о возможности тупиков в системе, которые обнаруживаются формулой.

а) μ x. ( ¬<to>true ). Для каждого n было найдено одно тупиковое состояние в котором каждый философ взял левую палочку.

б) μ x. ( <to>(deadlock ∨ x) ). Формула показывает путь достижения тупика в модели.

Далее в сеть были внесены следующие изменения.

1. Был добавлен цвет CS = index cs with 1..n+1 для описания места Free Chopsticks. Философ p брал для еды палочки p и p+1. Моделируется ситуация: философы за прямоугольным столом. Формула 1(а) становится истинной во всех состояниях. Тупиков в системе нет.

2. Аналогичные предыдущим получаются результаты, если философы берут сразу по две палочки.

3. Ситуация, когда философы не возвращают одну палочку на место, делает ложной как формулу 1(а), так и 1(б). В системе обнаруживается несколько тупиков.

## 5.2. Битовый протокол

Мы рассматриваем битовый протокол (ABP), описанный в [10]. Сеть Петри, соответствующая этому протоколу, представлена на рис. 3. Протокол моделирует в простейшем виде передачу и прием сообщений через нестабильную среду. Пользователь передатчик передает набор сообщений в заданном порядке пользователю-приемнику. Пользователь-приемник, получив сообщение, отправляет обратно подтверждение приема данного сообщения. Получив подтверждение, пользователь-приемник начинает отправку следующего сообщения. Сообщение может быть отправлено несколько раз, так как среда может терять отправленные сообщения (как и

отправленные подтверждения приема). Вносить какие-либо искажения в сообщения среда не может.

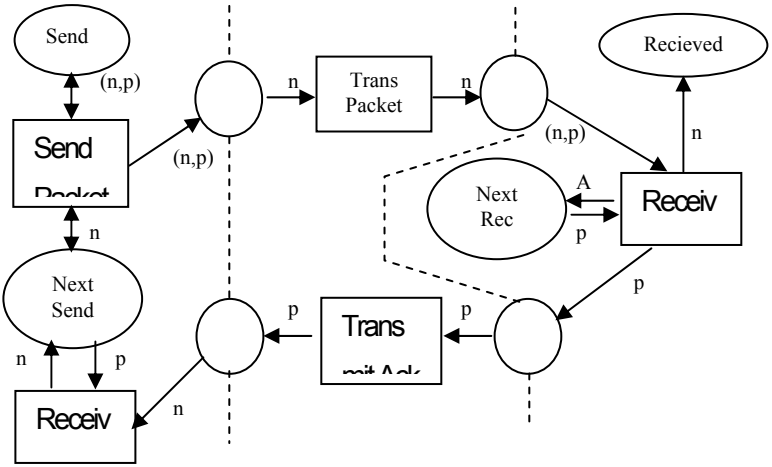



*Рис. 3.* Протокол ABP

Сеть была промоделирована для n=1..4.

На модели были проверены свойства прогресса, тупиков и безопасности.

***1. Свойства прогресса и тупиков.*** Начав функционировать, протокол может вернуться в начальное состояние, если на этапе передачи сообщения произойдет потеря данных. Поэтому формула 1(а) из раздела 5.1 оказывается ложной, а формула 1(б) из раздела 5.1 описывает приведенную выше ситуацию. Другие компоненты связности также были обнаружены в модели. С помощью формулы SCC → μ x. ( <to>(Final_states ∨ x) ), где SCC —

предикат, описывающий компоненту связности, можно проверить, что такие области модели не являются “ливлоками” протокола, т.к. из них существует путь к конечным состояниями системы (предикат Final_states). Более подробно изучение “ливлоков” системы описано ниже при верификации кольцевого протокола. Формулы проверки тупиков дают конечное состояние (все сообщения и подтверждения получены) и все состояния модели как решение второй формулы.

***2. Свойства безопасности.*** Важное место при проверке свойств коммуникационных протоколов занимают свойства безопасности. Частично безопасность была проверена выше — конечное состояние достижимо из любого места модели. Пусть предикат rec_all_ack_all означает — все сообщения и все подтверждения получены. Описанное выше свойство также выражается формулой μ x([to](rec_all_ack_all ∨ x) & <to>true)) ∨ rec_all_ack_all.

То, что тупики — это заключительные состояния, проверяется формулой ¬<to>true -> rec_all_ack_all, где формула ack_all -> rec_all означает, что, если получены все подтверждения, то получены и все сообщения.

Далее можно написать несколько формул, проверяющих порядок прихода сообщений и подтверждений, и проверить их конкатенацию. Примером является следующая формула:

μ x. ( <to>(ack_first ∨ x) ) -> μ x. ( <to>(ack_second ∨ x) ), которая означает невозможность получения второго подтверждения, пока не будет получено первое.

В протокол вносились изменения двух видов. Во-первых, среда в месте передачи сообщения или в месте передачи подтверждения делалась “плохой”, т.е. теряющей все сообщения. Во-вторых, вносились изменения во внутреннюю структуру протокола, связанные с неправильной работой счетчиков принятых сообщений и подтверждений. В первом случае нарушались свойства безопасности. Во втором случае нарушались как свойства безопасности, так и свойства живости. Были обнаружены дополнительные тупиковые состояния.

### 5.3. Кольцевой протокол

Исходным пунктом здесь является перевод кольцевого протокола, описанного в [7], в язык раскрашенных сетей Петри. Протокол описывает взаимодействие нескольких станций, соединенных в кольцо. По кольцу циркулирует фрейм фиксированной длины (рис. 4).

| .1.1.1.1 | .1.1.1.1. | .1.1.1. | .1.1.1.1. |
|---|---|---|---|
| RE | SRC | DEST | DATA |

*Рис.4.* Структура фрейма

Во фрейме четыре поля: RE — поле состояния фрейма, SCR — номер станции, пославшей сообщение, DEST — номер станции — адресата, DATA — передаваемые данные. Каждая станция, обработав пришедший к ней фрейм, отправляет его далее по кольцу. Выделенная станция выполняет функции монитора. Задача монитора — реинициализировать кольцо после обнаруженной ошибки передачи.

Описание работы станции и монитора в зависимости от полученного фрейма представлено на рис. 5 и 6 соответственно.

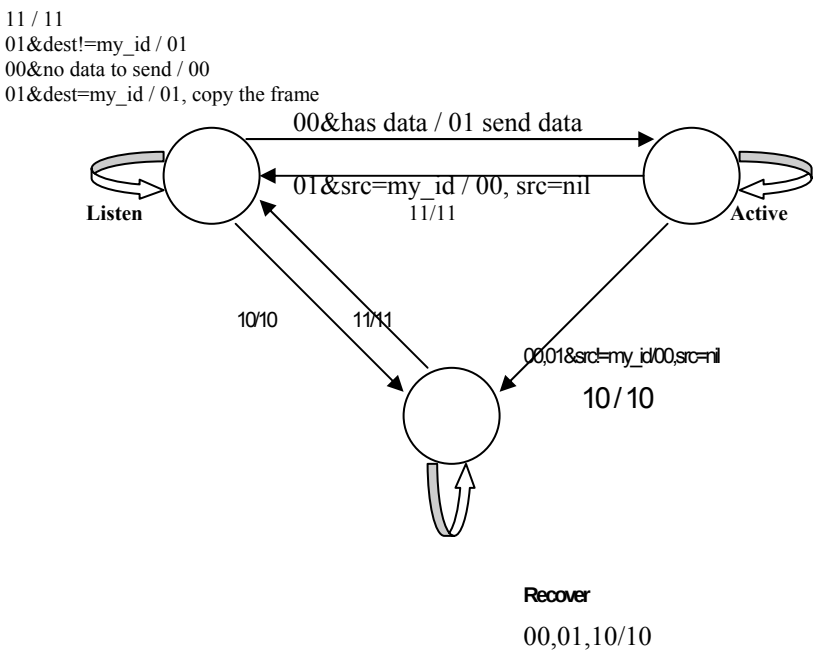



*Рис. 5.* Схема работы станции

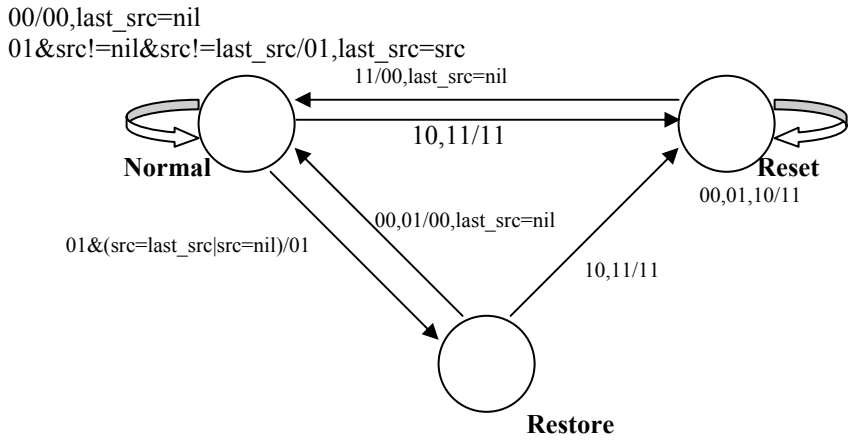



*Рис. 6.* Схема работы монитора

Протокол был промоделирован для нескольких станций и одного монитора. В силу ограниченных технических средств, эксперименты полной модели, т.е. модели, описывающей поведение системы при всевозможных сбоях в работе среды, не проводились, тем не менее был промоделирован ряд достаточно общих случаев. Таблица, показывающая параметры некоторых проведенных экспериментов, приведена в конце данной главы (табл. 1).

***1. Свойства прогресса и тупиков.*** В случае кольцевого протокола формулы 1 (а) и (б) из раздела 5.1 оказываются ложными, т.е. циклов, включающих начальное состояние в моделях кольцевого протокола, нет. Тем не менее в этих моделях существует множество циклов. Так как существует последовательная нумерация, "кандидатами" в циклические части модели являются возвраты на меньшие номера. Все такие возвраты можно проверить на цикличность формулой: $P \rightarrow \mu x (<a>(x \vee P))$.

Каждый цикл представляет потенциальную опасность бесконечного срабатывания в процессе функционирования системы. Однако в области коммуникационных протоколов предполагается некоторая справедливость (fairness) системы. Это свойство означает невозможность бесконечного (или сколь угодно частого) повторения "плохих" событий, приводящих к срабатыванию циклов. Кроме того, некоторые циклы являются естественными в работе системы (например циклическое срабатывание системы кольцевого протокола после передачи всех необходимых сообщений). Та-

ким образом, под термином “ливлок” (livelock), обозначающем “плохой” цикл, понимается циклическая работа системы с невозможностью выйти за пределы цикла. В общем случае такую ситуацию можно проверить формулой:

$$Loop \rightarrow \mu\, x\, (<a>(x \vee Out)).$$

В случае кольцевого протокола верна достаточно сильная формула безопасности (см. ниже), позволяющая говорить об отсутствии в системе ливлоков такого рода. Также модели кольцевого протокола проверялись формулой $\neg<to>true$ на наличие в них тупиковых состояний. Результаты таких проверок были отрицательные: тупиков обнаружено не было.

***2. Свойства безопасности.*** В качестве формулы безопасности разумно проверить формулу $\mu\, x.\ (<to>(P\_receive \vee x)\ )$, где предикат P_receive соответствует приему всех посылаемых сообщений. Данная формула была проверена, и был получен ответ “все состояния модели”. Это соответствует возможности попасть из любого состояния модели в состояние получения всех сообщений. Это, в частности, означает отсутствие в системе “плохих” тупиков и ливлоков.

Формулы безопасности, описывающие необходимый порядок прихода сообщений (наподобие формул, приведенных при описании верификации битового протокола), показали, что сообщения могут приходить (в случае ошибок в среде) в произвольном порядке.

***3. Неэффективность кольцевого протокола.*** В [1,2] была обнаружена следующая неэффективность данного протокола. В случае ошибок, возникающих в среде при передаче сообщений, возможен прием фиктивных сообщений. Также были предложены исправления, необходимые для устранения такого рода повторов. Эксперименты были проведены в режиме пошаговой симуляции и выявили несколько путей приема фиктивных сообщений. Одна из целей данной работы заключается в доказательстве в достаточно общем случае результативности исправлений, предложенных в [1,2], формальным методом проверки моделей.

В общем случае повторный прием сообщений может происходить произвольное число раз, что делает модель потенциально бесконечной. Чтобы избежать этого, мы вводим модифицированное условие равенства (эквивалентности) двух разметок. Согласно данному условию две разметки считаются эквивалентными (и соответственно, алгоритм построения графа достижимости, если они равны, останавливает данную ветку вычислений), если разметки отличаются только в одном знаковом элементе по цветовой

составляющей. Например, разметки M1 = {(p,2), (q,4)} и M2 = { (p,2), (q,2)} считаются эквивалентными.

При таком условии эквивалентности модель кольцевого протокола содержала максимум три повторных приема каждого сообщения. Формула проверки безопасности в этом случае была следующая:

$\mu$ x. (<to>(Received_one $\vee$ Received_two $\vee$ Received_three $\vee$ x) ).

Прием повторных сообщений легко обнаружить на этапе вычисления предиката P_receive. Соответствующая формула помогает достичь таких ситуаций. В проверенных случаях, после внесения предложенных исправлений, модель не содержала состояний с повторно принятыми сообщениями (предикаты Received_two и Received_three оказывались ложными). Результаты экспериментов с кольцевым протоколом приведены в таблице. Как можно заметить, после внесения предложенных в [1,2] исправлений размер моделей заметно уменьшался.

***4. Выход станции из кольца.*** В описанном в [7] кольцевом протоколе имеется возможность выхода станции из кольца в произвольный момент времени. Станция может выйти из кольца в произвольный момент и в любом состоянии и вернуться в кольцо также в произвольный момент в состояние Listen. Вышедшая из кольца станция пропускала все проходящие через нее фреймы без изменений. Ситуация с выходом станций также была промоделирована, и результаты некоторых экспериментов приведены в таблице 1. В этом случае были обнаружены нарушения свойств безопасности и наличия в модели ливлоков, не ведущих к получению посылаемых сообщений. Эти части модели соответствовали выходу станции адресата из кольца и пропуску приема сообщения. Станция “отправитель”, получив в состоянии Active обратно свой фрейм, считала данное сообщение полученным и, изменив соответственно поля фрейма, отправляла его далее по кольцу. Такая ситуация соответствует отсутствию в кольце адресата и должна обрабатываться протоколами более высокого уровня. Ливлоков другого типа, как и тупиковых состояний в моделях, обнаружено не было.

Таблица

**Эксперименты с кольцевым протоколом**

| N | ЭКСПЕРИМЕНТ | **Размер модели** | **Время (сек)** |
|---|---|---|---|
| 1 | 2 Станции. Монитор. Чистая среда | 14 | 10 |
| 2 | 2 Станции. Монитор. Среда — минимальная для обнаружения повторного приема (1-й случай) | 22 | 10 |
| 3 | 2 Станции. Монитор. Среда — минимальная для обнаружения повторного приема (оба случая) | 78 | 10 |
| 4 | Эксперимент N 2 с исправлениями | 22 | 10 |
| 5 | Эксперимент N 3 с исправлениями | 46 | 10 |
| 6 | 2 Станции. Монитор. Среда – плохая (по первому биту) в месте станция – монитор | 237 | 60 |
| 7 | 2 Станции. Монитор. Среда – плохая (по первому и второму битам) в месте станция – монитор | 1307 | 10 мин. |
| 8 | 2 Станции. Монитор. Среда – плохая (по трем битам) в месте станция – монитор | 8988 | 1.5-2 часа. |
| 9 | Эксперимент N 7. Внесены исправления | 747 | 10 мин. |
| 10 | 2 Станции. Монитор. Среда – плохая (по первому биту) в месте соединения станций, и (по первому биту) в месте монитор – станция | 5607 | 30 мин. |
| 11 | 2 Станции. Монитор. Среда – плохая (по первому и второму битам) в месте станция – станция | 1644 | 10-15 мин. |
| 12 | 2 Станции. Монитор. Среда – плохая (по первому биту) в месте станция – станция | 424 | 5-10 мин. |

| | | | |
|---|---|---|---|
| 13 | Эксперимент N 12. Внесены исправления | 352 | 5-10 мин |
| 14 | 2 Станции. Монитор. Среда – плохая (по второму биту) в месте монитор – станция | 113 | 60 |
| 15 | 2 Станции. Монитор. Среда – плохая (по третьему биту) в месте монитор – станция | 163 | 60 |
| 16 | 3 Станции. Монитор. Среда – хорошая | 33 | 10 |
| 17 | 3 Станции. Монитор. Среда – плохая (по первому биту) в месте станция – монитор | 288 | 60 |
| 18 | 3 Станции. Монитор. Среда – плохая (по первому и второму битам) в месте станция – монитор. | 1568 | 15 мин. |
| 19 | Эксперимент N 17. Внесены исправления | 252 | 60 |
| 20 | Эксперимент N 18. Внесены исправления | 937 | 10 мин. |
| 21 | 2 Станции. Монитор. Среда – хорошая. Одна станция может выходить из кольца | 91 | 10 |
| 22 | 2 Станции. Монитор. Среда – плохая ( по первому биту ) в месте станция – монитор. Одна станция может выходить из кольца | 1131 | 10 мин. |
| 23 | 2 Станции. Монитор. Среда – плохая ( по первому и второму битам ) в месте станция – монитор. Одна станция может выходить из кольца | 4086 | 25 мин. |
| 24 | 2 Станции. Монитор. Среда – плохая ( по первому биту ) в месте монитор – станция . Одна станция может выходить из кольца | 2040 | 20 мин. |

## 6. ЗАКЛЮЧЕНИЕ

В данной работе описана система верификации раскрашенных сетей Петри PNV, которая была успешно применена для верификации кольцевого протокола [7]. В этом эксперименте была подтверждена обнаруженная ранее [1, 2] неэффективность данного протокола, а также проведена успешная верификация его модифицированной версии. Следует отметить, что ввиду потенциально бесконечной модели кольцевого протокола, проведенная верификация гарантирует корректность модифицированной версии только при указанных в табл. 1 ограничениях на среду и число станций, а проблема верификации кольцевого протокола в общем случае остается открытой.

Хотя в настоящее время система верификации PNV ориентирована на специальное подмножество РСП, планируется существенно расширить его посредством типов, таких как очередь и список, временных конструкций, а также более выразительных функций на дугах. Реализованный в системе PNV модуль перевода раскрашенных сетей Петри в эквивалентные C++ программы полезен для моделирования коммуникационных протоколов и является поэтому важным средством расширения системы Design/CPN [4]. Другой модуль PNV, который переводит РСП в эквивалентные программы на Паскале, расширенном недетерминированными конструкциями, позволит применить разрабатываемую систему СПЕКТР-2 для верификации РСП.

Предполагается также реализовать несколько форматов входного языка системы PNV с целью автоматической верификации сетевых моделей коммуникационных протоколов, полученных на выходе системы EPV [1, 2]. Кроме того, предполагается реализовать подсистему проверки моделей на языке C++.

Представляет значительный интерес применение системы PNV для верификации протокола скользящего окна, поскольку методом проверки моделей удалось обнаружить ошибку в опубликованной версии этого протокола [11].

## СПИСОК ЛИТЕРАТУРЫ

1. Алексеев Г.И. и др. Использование сетей Петри для верификации распределенных систем, представленных на языке Estelle // Известия РАН. Сер.: Теория и системы управления. 1999. — N5. — с.105-116.

2. Непомнящий В.А. и др. Верификация Estelle-спецификаций распределенных систем посредством раскрашенных сетей Петри. — Новосибирск: Институт систем информатики СО РАН, 1998.
3. Непомнящий В.А., Шилов Н.В., Бодин Е.В. REAL: язык для спецификации и верификации систем реального времени // Системная информатика. — т.7. — Новосибирск, Наука, — 2000. с.174 — 224.
4. Capellmann C., Dibold H., Herzog U. Using high-level Petri nets in the field of intelligent networks // Lecture Notes Comput. Sci., V. 1605, P. 1-36, 1999.
5. Cheng A., Christensen S., Mortensen K.H. Model cheking coloured Petri nets exploiting strongly connected components // Report DAIMI PB-519, Aarhus Univ., 1997.
6. Cleaveland R., Klein M., Steffen B. Faster model cheking for modal mu-calculus // Proc. Intern. Conf. On Computer-aided verification, 1992(CAV — 92). — (Lecture Notes Comput. Sci., V. 663, P. 410 — 422, 1992).
7. Cohen R., Segall A. An efficient reliable ring protocol // IEEE Transactions on Communications. — 1991. — V. 39, N 11. — P.1616 — 1624.
8. Esparza J. On the decidability of model cheking for several mu-calculi and Petri nets // Proc. Intern. Colloquium on Trees in Algebra and Programming, 1994 (CAAP-94). — (Lect. Notes Comput. Sci., V. 787, 1994).
9. Habermehl P. On the conplexity of the linear-time mu-calculus for Petri nets // Proc. Intern. Conf. on Application and Theory of Petri Nets, 1997 (ICATPN-97). — (Lect. Notes Comput. Sci., V. 1248, 1997, P.102 — 116).
10. Jensen K. Coloured Petri nets: basic concepts, analysis methods and practical use. — V. 1, 2, 3. — Springer, 1996.
11. Kaivola R. Using compositional preorders in the verification of sliding window protocol // Proc. Intern. Conf. Computer — aided verification, 1997 (CAV-97). — (Lect. Notes Comput. Sci., V. 1663, P. 48-59, 1997).
12. Kristensen L.M., Christensen S., Jensen K. The practitioner's guide to coloured Petri nets // Intern. Journal on Software Tools for Technology Transfer. — 1998. — V. 2, N 2. — P. 98 — 132.

**В.Е. Козюра, В.А. Непомнящий, Р.М. Новиков**

# ВЕРИФИКАЦИЯ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ МЕТОДОМ ПРОВЕРКИ МОДЕЛЕЙ

**Препринт**
**89**

Рукопись поступила в редакцию 23.06.01
Рецензент Ф. А. Мурзин
Редактор З. В. Скок

---

Подписано в печать 03.09.01
Формат бумаги 60 × 84 1/16 Объем 1.5 уч.-изд.л., 1.6 п.л.
Тираж 50 экз.

---

НФ ООО ИПО “Эмари” РИЦ, 630090, г. Новосибирск, пр. Акад. Лаврентьева, 6