СБОРНИК **НАУЧНЫХ ТРУДОВ НГТУ. – 2014. – № 1(75).** – 105–123

УДК 004.4'22

АНАЛИЗ МЕТОДИК ИССЛЕДОВАНИЯ ПРОСТРАНСТВА СОСТОЯНИЙ СЕТЕЙ ПЕТРИ*

А.В. МАРКОВ

Описан и представлен в унифицированной форме математический аппарат сетей Петри и системы меченых переходов как один из вариантов пространства состояний. Проанализированы методики исследования дерева достижимости: хеширование и метод «плавающей линии».

Метод, использующий хеширование состояний, заключается в применении хеш-функций к состояниям с целью уменьшения требуемой памяти для хранения о них информации. Но при использовании данного способа появляется вероятность частичного исследования пространства достижимостей. В связи с чем, проанализирована примерная оценка возможности совпадения хеш значений. Другой метод, о котором упоминалось, – это метод плавающей линии, который заключается в удалении состояний, не связанных с оставшейся частью пространства состояний. То есть происходит удаление тех состояний, потомки которых вычислены при условии, что они существуют. Также проанализирован модифицированный способ, названный как ComBack-метод, при котором происходит хеширование информации о состоянии, а также сохранение данных о его родителях и связях, из которых получена достижимость. В заключение работы представлены выводы о возможности применения данных способов, а также об их положительных свойствах и недостатках.

Ключевые слова: сети Петри, *UML*, анализ пространства состояний, дерево достижимости, алгоритм анализа, sweep-line-метод, bit state hashing, прогрессивная оценка, хеширование, коллизия.

ВВЕДЕНИЕ

Одной из основных стадий при разработке программных продуктов является тестирование, а также проверка состояний системы, в которых она может находиться при функционировании. С постоянным ростом сложности разрабатываемого программного обеспечения увеличивается и количество состояний системы, т. е. происходит расширение алгоритмов и логики их работы. Для анализа спроектированных систем необходимо представить её в виде одной из парадигм, позволяющих в дальнейшем анализировать все состояния

* Статья получена 30 октября 2013 г.

системы. Моделирование разрабатываемого продукта можно осуществить при помощи математического аппарата сетей Петри.

Сеть Петри – упорядоченное множество $N = (P, T, F, m_I)$, где P – множество мест, T – множество переходов, таких что $P \cap T = \emptyset$, $F \subseteq P \times T \cup T \times P$ – отношение, а $m_I : P \to N$ – начальная маркировка [1, 2].

Как говорилось ранее, математический аппарат сетей Петри разрабатывался с целью моделирования систем с последующим их анализом. Существующие виды анализа спроектированных сетей заключаются в построении дерева достижимости, в матричном представлении сети, а также в генерации отчета о пространстве состояний при помощи среды моделирования CPN Tools (version 3.4.0) [3–27]. Граф достижимости – это упорядоченное множество (V, E, src, trg), где V – множество вершин, E – множество рёбер между вершинами, а $src, trg : E \to V$ отображает в каждом ребре вершину, из которого оно получено, и вершину, к которому приводит выполнение срабатывания перехода, соответственно. Корневой граф – это множество (V, E, src, trg, r), такое что (V, E, src, trg) – граф, а $r \in V$ – корень[1]. Граф можно описать следующим образом:

- $V = \left[m_I \right\rangle$ – множество узлов – это множество достижимых маркировок;
- $E = \left\{ (m, t, m') \in V \times T \times V \mid m \left[t \right\rangle m' \right\}$ – множество узлов – это множество переходов из одной достижимой маркировке к другим;
- src представлено для $src(m, t, m') = m$;
- trg представлено для $trg(m, t, m') = m'$;
- $r = m_I$ – корень представлен, как начальная маркировка.

Одним из вариантов представления дерева достижимости является система меченых переходов. Система меченых переходов[2] (Labelled Transition System, LTS) – это упорядоченное множество $LTS = (S, T, \Delta, s_I)$, где

- $S \neq \emptyset$ – множество состояний;
- T – множество переходов;
- $\Delta \subseteq S \times T \times S$ – отношение перехода, определяющее потомков состояний;

[1] Корень – это часть ребра такая, что $r(m, t, m') = (t, m')$

[2] Labelled Transition System, LTS – (англ.) система меченных переходов

- $s_I \in S$ – начальное состояние.

Пусть $s, s' \in S$ есть два состояния, а $t \in T$ – переход. Если $(s, t, s') \in \Delta$, тогда выполнение t в s приведёт к s'. Это можно записать $s \xrightarrow{t} s'$. Текущая последовательность является последовательностью состояний s_i и переходов t_i, которая может быть записана как последовательность вида $s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 \ldots s_n \xrightarrow{t_n} s_{n+1}$ и удовлетворяющая $s_i \xrightarrow{t_i} s_{i+1}$ для $1 \le i \le n$. Также существует отношение $s \to *s'$, если и только если текущая последовательность $s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 \ldots s_n \xrightarrow{t_n} s_{n+1}$, $n \ge 1$, с $s = s_1$ и $s' = s_{n+1}$. Состояние s' достижимо из s, если и только если $s \to *s'$ и $reach(s) = \{s' \in S \mid s \to *s'\}$ указывает на множество состояний достижимых из s.

Для удобства представим структуру анализа графа достижимости в виде алгоритма 1 [2], к которому добавлены комментарии для лучшего понимания его работы.

Алгоритм 1. **Алгоритм исследования графа достижимости**

Require: $LTS = (S, T, \Delta, s_I)$ // система меченых переходов

Ensure: (V, E) // соответствующий граф достижимости

1: $V := \{s_I\}$ // переменной V (множество посещённых состояний) присваивается множество с единственной переменной s_I

2: $W := \{s_I\}$ // переменной W (множество не посещённых состояний) присваивается множество с единственной переменной s_I

3: $E := \emptyset$ // переменной E (множество рёбер) присваивается значение $\emptyset$

4:

5: *while* $W \neq \emptyset$ *do* // пока множество W не станет пустым, выполняем …

6: *Select an* $s \in W$ // выборку состояния s из множества W

7: $W := W \setminus \{s\}$ // убираем из множества W состояние s

8: *for all* t, s' *such that* $s \xrightarrow{t} s'$ *do* // для всех t, s', удовлетворяющих $s \xrightarrow{t} s'$ выполняем …

9 : $E := E \cup \{(s, t, s')\}$ // добавление нового ребра во множество

10 : *if* $s' \notin V$ *then* // если найдено новое состояние, не входящее во множество V, тогда

11 : $V := V \cup \{s'\}$ // добавляем его к этому множеству

12 : $W := W \cup \{s'\}$ // и к множеству W

13 :

14 : *return*(V, E) // возвращаем обновленные значения переменных V и E

Одной из основной проблемой при анализt крупных систем является возможность взрыва пространства состояний[3], которая заключается в экспоненциальном росте количества состояний, что приводит к досрочному завершению анализа из-за недостатка нужных объёмов оперативной памяти. Следовательно, возможно без оценки останется значительная часть дерева достижимости, в которой могут присутствовать зацикливания и тупиковые маркировки, что свидетельствует о некорректном построении системы. Конечно, для хранения высчитанных состояний системы можно использовать постоянную память машины, но это приведет к значительному росту времени, требуемого на анализ [27].

Также существуют нестандартные методики анализа пространства состояний, т. е. методики, использующие иные алгоритмы по отношению к традиционному – хранение всех посещенных состояний в оперативной памяти. Это *sweep-line method*[4] и *bit-state hashing*[5] и их возможные модификации [2].

1. SWEEP-LINE-МЕТОД

На рис. 1 [23] показано представление sweep-line-метода, s_0 обозначает начальное состояние системы, а две серые площади показывают состояние, удерживаемые в памяти. Некоторые из них были вычислены (светло-серый цвет), некоторые только будут вычислены (тёмно-серый). Пунктиром изображена линия, которая соответствует текущей прогрессивной оценки[6]. Существует большая вероятность, что состояния строго левее плавающей более не будут учувст-

[3] Взрыв пространства состояний – state explosion problem (англ.) [24].

[4] Метод плавающей линии – sweep-line-method (англ.) [24]. Дословный перевод – метод линии-диапазона. Но более точно излагает суть перевод, предложенный авторами.

[5] Хеширование битового состояния – bit-state hashing (англ.) [25].

[6] Прогрессивная оценка – мера для определения глубины исследования дерева достижимости.

вовать при анализе системы и поэтому могут безопасно быть удалены. Передвигая пунктирную линию через пространство состояний, происходит вычисление достижимых состояний впереди неё и удаление состояний позади. Каждому состоянию для данного способа исследования графа достижимости присваивается соответствующая прогрессивная оценка (алгоритм 2) [23], к данному алгоритму добавлены комментарии для лучшего понимания его работы.

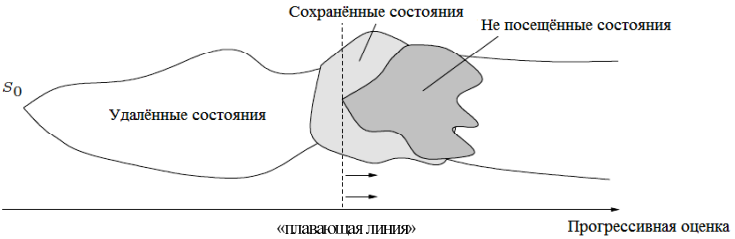

Рис. 1. Представление sweep-line-метода

Для применения данного подхода необходимо представить систему, как граф достижимости или систему меченых переходов $LTS = (S, T, \Delta, s_1)$. Большинство моделей распределённых систем включают модели цветных сетей Петри, попадающих в эту категорию систем.

Sweep-line-метод использует понятие прогрессивной оценки. Прогрессивная оценка определяет порядок $(V, \sqsubseteq)$ состояний системы и прогрессивных значений φ, назначая прогрессивное значение $\varphi(s) \in V$ каждому состоянию s.

Алгоритм 2. **Исследование пространства состояний, используя sweep-line-метод**

1: $Unprocessed \leftarrow \{s_1\}$ // выбираем все не посещённые состояния

2: $Nodes.Add(s_1)$ // добавляем выбранные состояния к вершинам

3: *while*¬*Unprocessed.Empty*() *do* // пока существуют не посещённые состояния, выполняем …

4: $s \leftarrow Unprocessed.GetMinElement()$ // выборку состояний с минимальным индексом

5 : $for\ all\left(t,s'\right)such\ that\ s\xrightarrow{t}s'\ doa$ // для всех корней, полученных из рёбер, выполняем …

6 : $if\ \varphi\left(s\right)\not\sqsubseteq\varphi\left(s'\right)then$ // если прогрессивная оценка s не входит в прогрессивную оценку s', тогда

7 : $Stop\left("Progress\ measure\ rejected:",\left(s,t,s'\right)\right)$ // останавливаем выполнение и не сохраняем данное ребро

8 : $end\ if$ // конец, если

9 : $if\neg\left(Nodes.Contains\left(s'\right)\right)then$ // если вершины не содержат s', тогда

10 : $Nodes.Add\left(s'\right)$ // добавляем к вершинам состояние s'

11 : $Unprocessed.Add\left(s'\right)$ // добавляем к не посещённым состояниям состояние s'

12 : $end\ if$ // конец если

13 : $end\ for$ // конец цикла *for*

14 : $Nodes.GarbageCollect\left(\min\left\{\varphi(s)\mid s\in Unprocessed\right\}\right)$ // Собираем состояния с наименьшей прогрессивной оценкой

15 : $end\ while$ // конец цикла *while*

К достоинствам данного способа определим тот факт, что для анализа даже весьма крупных систем используется определённое количество ресурсов анализатора, а именно оперативной памяти. Но отметим, что sweep-line-метод отлично подходит только к монотонным системам, сужая его применение к различным системам.

2. ВОЗВРАТНЫЕ РЁБРА И ОБОБЩЁННАЯ ПРОГРЕССИВНАЯ ОЦЕНКА

В работе [24] решение данной проблемы предложено реализовать при помощи возвратных рёбер.

При обнаружении состояний, потомками которых могут являться уже ранее проанализированные состояния, «плавающая линия» отодвигается до соответствующего потомка, но состояния с такой же прогрессивной оценкой не восстанавливаются, а дальнейшее исследование продолжается с найденного потомка, который сохраняется в памяти до окончания исследования (рис. 2).

Для наглядности представим sweep-line-метод с использованием возвратных рёбер (алгоритм 3) для системы меченых переходов [24], к данному алгоритму добавлены комментарии для лучшего понимания его работы.. Пусть $S = (S, T, \Delta, s_I)$ – система. Упорядоченное множество $(s, t, s') \in \Delta$ – есть возвратное ребро для прогрессивной оценке $\varphi : S \to V$, если и только если $\varphi(s) \supset \varphi(s')$.

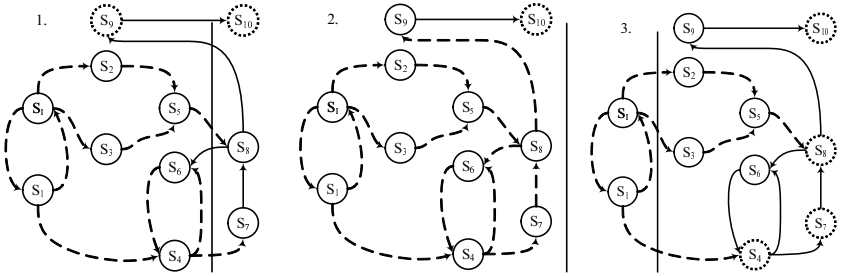

Рис. 2. Обобщённый sweep-line-метод

Алгоритм 3. **Обобщённый sweep-line метод**

1: $Roots \leftarrow \{s_I\}$ // Из корней получаем множество состояний

2: $Nodes.Add(s_I)$ // Добавляем состояния к узлам

3: $while \neg(Roots.Empty())\, do$ // пока корни непустые, выполняем …

4: $Unprocessed \leftarrow Roots$ // выборку не посещённых состояний из корней

5: $Roots \leftarrow \emptyset$

6: $while \neg(Unprocessed.Empty())\, do$ // пока список непосещённых состояний не станет пустым

7: $s \leftarrow Unprocessed.GetMinElement()$ // выбираем из непосещённых состояний состояния с минимальным индексом

8: $for\ all\ (t, s')\ such\ that\ s \xrightarrow{t} s'\ do$ // для всех корней, входящих в существующие рёбра выполняем условие

9: $if \neg(Nodes.Contains(s'))\, then$ // если узлы содержат состояние s', тогда

10: $Nodes.Add(s')$ // добавляем состояние s' в узлы

11: *if* $\varphi(s) \supset \varphi(s')$ *then* // если прогрессивная оценка s входит в прогрессивную оценку s', тогда

12: $Nodes.MarkPersistent(s')$ // добавляем состояние в список постоянных состояний

13: $Roots.Add(s')$ // добавляем состояние s' в корни

14: *else* // иначе

15: $Unprocessed.Add(s')$ // добавляем состояние s' в не посещённые состояния

16: *end if* // конец если

17: *end if* // конец если

18: *end for* // конец цикла *for*

19: $Nodes.GarbageCollect\left(\min\left\{\varphi(s) \mid s \in Unprocessed\right\}\right)$ // Собираем состояния с наименьшей прогрессивной оценкой

20: *end while* // конец цикла *while*

21: *end while* // конец цикла *while*

Предлагаемый способ лучше справляется с анализом систем, чем sweep-line-метод. Область его применения значительно расширяется за границы монотонных систем. Также можно быть уверенным, что более двух раз состояние не будет посещено и в анализе всех состояний системы.

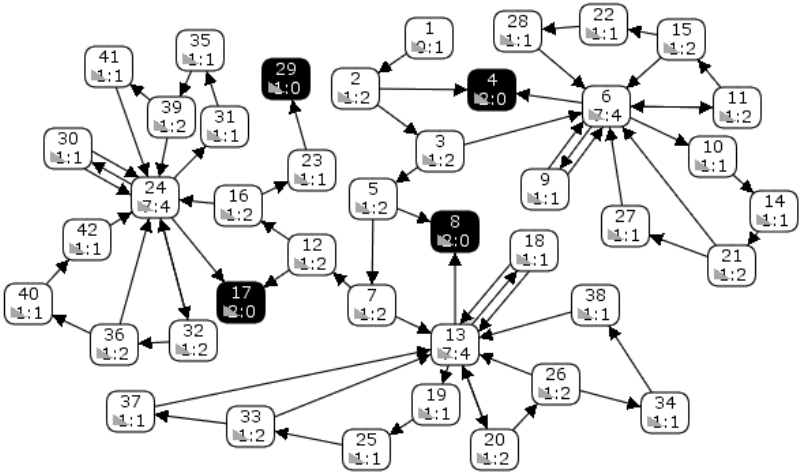

Рис. 3. Дерево достижимости сети взаимодействия банкомата и пользователя

К примеру, воспользуемся данным способом для анализа дерева достижимости (рис. 3) системы «Взаимодействие пользователя и банкомата», описанного в работе [14], количество состояний, которых потребуется запомнить до окончания исследования графа достижимости не велико. Так, в первой части дерева достижимости (рис. 4) состояния № 4, № 6 необходимо запомнить, поскольку к ним обращаются состояния с большей прогрессивной оценкой.

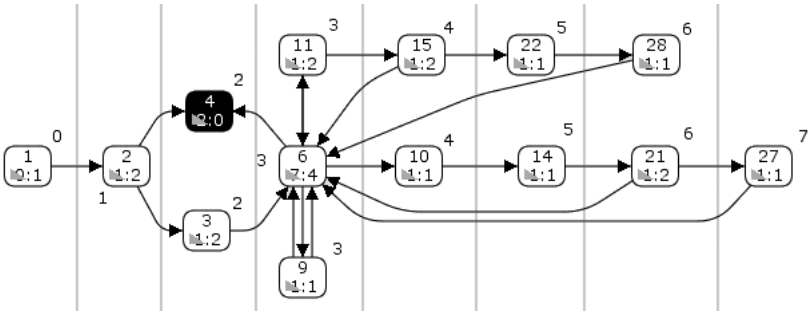

Рис. 4. Первая часть дерева достижимости взаимодействия пользователя и банкомата

Но нужно помнить, что существуют системы, у которых возвратные рёбра присущи, практически, каждой маркировке пространства состояний, что

может привести к постоянному хранению большинства достижимостей, т. е. обобщённый sweep-line-метод будет хранить состояний, как и при традиционном анализе графа достижимости.

3. ХЕШИРОВАНИЕ

Существует другой вариант анализа пространства состояний, который заключается в сжатии информации о каждом отдельном состоянии через использование хеш-функций [25].

Данный способ заключается в получении вектора состояний от каждой достижимости системы (рис. 5) [25]. То есть информация преобразуется в вектор состояния State Vector, вектор действия Action Vector, вектор предиката Predicate Vector, после чего вектор состояний хешируется с получением хеш-суммы[7]. Полученное значение преобразуется в адрес в битовом пространстве состояний фиксированного размера. Каждый вектор состояния использован для вычисления хеш-ключа, который напрямую используется для индексации больших размеров массива состояний.

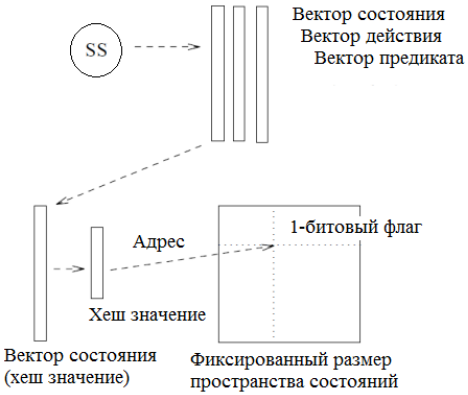

Рис. 5. Преобразование вектора состояния в одно битовый флаг

Отметим, что при появлении хеш конфликтов[8], метод продолжает работать без ошибок, но нужно понимать, что полной покрываемости пространства состояний получить не удаться. Коллизия у анализатора воспринимается, как уже обработанное ранее состояние и при исследовании игнорируется.

Для решения данной проблемы можно использовать повторение проверки с альтернативными хеш-функциями или альтернативной стратегией поиска. Это приведёт к восстановлению пропущенных состояний,

[7] Контрольная сумма (хеш-сумма) – некоторое значение, рассчитанное по набору данных путём применения определенного алгоритма и используемое для проверки целостности данных при их передаче или хранении.

[8] Хеш-конфликт (коллизия) – наличие состояний в виде сжатой информацией, имеющих одинаковую хеш-сумму.

что позволит приблизиться к покрываемости в 100 %.

Большим преимуществом данного способа является возможность хранения информации о состоянии всего в одном бите, что может быть существенным при анализе систем на маломощных машинах. Но также существует вероятность получения коллизий, что плохо отразится на исследовании пространства состояний и тем самым снизит корректность анализа системы.

Вероятность возникновения коллизий. Подсчитать вероятность коллизий при оценке систем с целью определения полноты последующего анализа весьма сложно, но иногда необходимо [25].

Пусть N – количество достижимых состояний, M – общее количество битов доступной памяти. Тогда, при $N < M$, первый элемент имеет нулевое значение возможности коллизии, а последний – максимальную $\frac{(N-1)}{M}$.

Средняя вероятность коллизии можно подсчитать по формуле

$$P_1 \le \frac{1}{N \cdot M} \sum_{i=0}^{N-1} i = \frac{N-1}{2M} \le 2^{n-m-1}.$$

Следовательно, ожидаемая покрываемость

$$(1 - P_1) \cdot 100\ \%\ .$$

При $N \ge M$, т. е. состояний больше, чем бит доступной памяти. Минимальное количество хеш-коллизий будет $(N - M)$.

Средняя вероятность коллизий тогда вычисляется по формуле

$$P_1 \ge 1 - \frac{M}{N} = 1 - 2^{m-n}.$$

Сжатие с использованием 20 различных хеш-функций[9].

Тогда ожидаемая вероятность коллизий при использовании h функций для двух случаев $h \cdot N < M, h \cdot N \ge M$.

При $h \cdot N < M$

$$P_h \le \frac{1}{N} \sum_{i=0}^{N-1} \left(\frac{h \cdot i}{M} \right)^h \le \frac{(2h)^h}{h+1} \left(P_1 \right)^h.$$

[9] Мульти-бит (20-ти) хеширование – хеширование, при котором используется 20 хеш функций.

Для $h = 2$ получаем

$$P_2 \le \frac{16}{3} 2^{2\cdot(n-m-1)}.$$

Стоит отметить, что приблизительный подсчёт вероятности коллизий условен и целиком опираться на него неразумно, но можно использовать данный подход расчёта для приблизительной оценки при выбора объема памяти и количества хеш-функций.

Таким образом, оценить количество состояния и необходимой памяти для анализа при использовании хеширования для сжатия информации о состояниях весьма тяжело, но получить приблизительную оценку и использовать её в дальнейшем можно.

4. COMBACK-МЕТОД

Так как при использовании функций хеширования существует высокая вероятность возникновения коллизий. Существуют методики с использованием различных хеш-функций к одному состоянию с получением дополнительных хеш-сумм. Отдельно стоит выделить ComBack-метод [2], который является хорошей альтернативой обычному способу хеширования состояния.

ComBack-метод – расширенный метод хеширования состояния с сохранением предыдущих рёбер. Для представления ComBack-метода, изначально предполагалось наличие детерминированных переходов, т. е. если $s \xrightarrow{t} s'$ и $s \xrightarrow{t} s''$, тогда $s' = s''$, что справедливо и для переходов различных сетей Петри. Также можно пользоваться обозначением $s \to *s'$, если и только если выполняется последовательность $s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 \ldots s_{n-1} \xrightarrow{t_{n-1}} s_n, n \ge 1$, с $s = s_1$ и $s' = s_n$. Состояние s' достижимо из s, если и только если $s \to *s'$, и $reach(s) = \{s' \in S \mid s \to *s'\}$ обозначает множество достижимых состояний из s. Пространство состояний системы может быть представлено, как направленный граф (V, E), где $V = reach(s_1)$ – это множество узлов и $E = \{(s,t,s') \in \Delta \mid s, s' \in V\}$ – множество рёбер.

В данном способе предлагается создавать две таблицы, в одной из которых будет содержаться информация о хеш-суммах и состояниях, соответствующих этим хеш-суммам, а в другой – информация о родители этого со-

стояния и переходе, при срабатывание которого было получено данное состояние.

Идея ComBack-метода заключается в следующем:

1) целочисленный номер $N(s)$ присваивается каждому посещенному состоянию s;

2) таблица состояний хранит информацию обо всех хеш-суммах и соответствующих номерах состояний;

3) таблица возвратных связей содержит информацию о номере посещённого состояния $N(s)$, а также информацию о родителях текущей достижимости, состоящей из перехода t, и номер состояния $N(s')$ посещенного состояния s', такого, что $s' \xrightarrow{t} s$.

Таким образом, можно представить ComBack-метод как алгоритм 4 [2], к которому добавлены комментарии для лучшего понимания его работы.

Алгоритм 4. **Алгоритм ComBack-метода**

1: $m \leftarrow 1$ // присваиваем номеру состояния 1

2: $StateTable.Init(); StateTable.Insert(H(s_1),1)$ // инициализируем глобальную структуру данных: таблица состояний (хеш-сумма для состояния и порядковый номер состояния)

3: $WaitingSet.Init(); WaitingSet.Insert(s_1,1)$ // инициализируем глобальную структуру данных: список ожидающих состояний (состояние и номер состояния)

4: $BackEdgeTable.Init(); BackEdgeTable.Insert(1,\perp)$ // инициализируем глобальную структуру данных: таблица возвратных рёбер (с номер состояния и возможных корней)

5:

6: $while \neg WaitingSet.Empty()\ do$ // пока список ожидающих состояний не пуст

7: $(s,n') \leftarrow WaitingSet.Select)()$ // из данного списка выбираем пару: состояние и соответствующий ей номер

8: $for\ all\ t,s'\ such\ that\ (s,t,s') \in \Delta\ do$ // для всех корней, входящих во множество рёбер, выполняем …

9 : *if* $\neg Contains(s')$ *then* // если состояние s' не было исследовано, то

10 : $m \leftarrow m+1$ // инкрементируем номер состояния

11 : $StateTable.Insert(H(s'), m)$ // добавляем в таблицу состояний хеш значение исследуемого состояния и его номер

12 : $WaitingSet.Insert(s', m)$ // в список ожидания добавляем состояние и его номер

13 : $BackEdgeTable.Insert(m, (n', t))$ // в таблицу возвратных рёбер добавляем номер состояния и возможные корни

14 :

15 : *proc* $Contains(s')$ *is* // процедура *Contains* выполняет цикл, при котором

16 : *for all* $n \in StateTable.Lookup(H(s'))$ *do* // для всех номеров состояний из таблицы состояний выполняем выборку

17 : *if* $Matches(n, s')$ *then* // если данный номер и состояние существует, тогда

18 : *return tt* // возвращаем *true*

19 : *return ff* // иначе возвращаем *false*

20 :

21 : *proc* $Matches(n, s')$ *is* // процедура *Matches* реализует

22 : *return* $s' = Reconstruct(n)$ // возвращение состояния s' равное номеру *Reconstruct*

23 :

24 : *proc* $Reconstruct(n)$ *is* // процедуре *Reconstruct* соответствует условие

25 : *if* $n = 1$ *then* // если номер состояния равен 1, тогда

26 : *return* s_1 // возвращаем состояние s_1

27 : *else* // иначе

28 : $(n', t) \leftarrow BackEdgeTable.Lookup(n)$ // выполняем поиск нужного номера состояния

29 : $s \leftarrow Reconstruct(n')$ // и присваиваем соответствующему состоянию

30 : *return Execute*(s,t) // возвращаем вычисленный дескриптор состояния

Предложенный способ значительно уменьшает возникновение коллизий при сжатии информации о состояниях и является отличной альтернативой классическому способу использования хеш-функций при исследовании пространства состояний. Из проанализированных выше методик данная является лучшей альтернативой между ограниченными ресурсами анализатора и полнотой исследования пространства состояний.

ЗАКЛЮЧЕНИЕ

При моделировании крупных систем с последующим исследованием их пространства состояний может оказаться недостаточным использовать стандартный способ анализа, который заключается в исследовании пространства достижимостей с сохранением всех вычисленных состояний. На помощь могут прийти альтернативные методики, которые заключаются в удалении уже посещённых ранее состояний – метод «плавающей линии», а также в сжатии информации, присущей каждому состоянию – так называемое хеширование.

К достоинствам этих методик можно отнести сокращение ресурсов требуемых на анализ систем со значительными размерами пространства состояний, но существенным недостатком будет являться либо частичная покрываемость всех исследуемых состояний, либо повторное посещение уже изученных ранее состояний. Данные методики имеют широкое применение: двойное хеширование реализовано в пакете SPIN [25], метод плавающей линии используется для генерации пространства состояний в пакете ASAP [2].

Из проанализированных выше методик можно выделить ComBack-метод, который является лучшей альтернативой между использованием ограниченных ресурсов анализатора и полнотой исследования пространства состояний.

[1] *Питерсон Дж.* Теория сетей Петри и моделирование: пер. с англ. / Дж. Питерсон. – М.: Мир, 1984.

[2] *Westergaard M.* Behavioral verification and visualization of formal models of concurrent systems: PhD dissertation / M. Westergaard. – Aarhus: University of Aarhus, 2007.

[3] *Воевода А.А.* О компактном представлении языков раскрашенных сетей Петри / А.А. Воевода, Д.О. Романников // Сб. науч. тр. НГТУ. – 2008. – № 3(53). – С. 105–108.

[4] *Воевода А.А.* О компактном представлении языков сетей Петри: сети с условиями и временные сети / А.А. Воевода, А.В. Марков // Сб. науч. тр. НГТУ. – 2010. – № 2(60). – С. 77–83.

[5] *Коротиков С.В.* Применение сетей Петри в разработке программного обеспечения центров дистанционного контроля и управления: дис. … канд. техн. наук / С.В. Коротиков. – Новосибирск: НГТУ, 2007.

[6] *Марков А.В.* Моделирование процесса поиска пути в лабиринте при помощи сетей Петри / А.В. Марков // Сб. науч. тр. НГТУ. – 2010. – № 4(62). – С. 133–141.

[7] *Романников Д.О.* Обзор работ посвященным разработке ПО с использованием UML и сетей Петри / Д.О. Романников, А.В. Марков, И.В. Зимаев // Сб. науч. тр. НГТУ. – 2011. – № 1(63). – С. 91–104.

[8] *Марков А.В.* Моделирование процесса поиска пути в лабиринте при помощи сетей Петри для системы из двух связных звеньев / А.В. Марков, А.А. Воевода // Сб. науч. тр. НГТУ. – 2011. – № 3(65). – С. 95–104.

[9] *Марков А.В.* Поиск манипулятором кратчайшего пути в лабиринте / А.В. Марков // Сб. науч. тр. НГТУ. – 2011. – № 4(66). – С. 75–91.

[10] *Романников Д.О.* Пример применения методики разработки ПО с использованием UML-диаграмм и сетей Петри / Д.О. Романников, А.В. Марков // Научный вестник НГТУ. – 2012. – № 1(67). – С. 175–181.

[11] *Романников Д.О.* Разработка программного обеспечения с применением UML диаграмм и сетей Петри для систем управления локальным оборудованием: дис. … канд. техн. наук / Д.О. Романников. – Новосибирск: Изд-во НГТУ, 2012.

[12] *Воевода А.А.* Рекурсия в сетях Петри / А.А Воевода, А.В. Марков // Сб. науч. тр. НГТУ. – 2012. – № 3(69). – С. 115–122.

[13] *Воевода А.А.* Понятие рекурсии в сетях Петри: факториал числа, числа Фибоначчи / А.А. Воевода, А.В. Марков // Сборник научных трудов НГТУ. – 2013. – № 1(71). – С. 72–77.

[14] *Марков А.В.* Анализ сетей Петри при помощи деревьев достижимости / А.В. Марков, А.А. Воевода // Сб. науч. тр. НГТУ. – 2013. – № 1(71). – С. 78–95.

[15] *Марков А.В.* Разработка программного обеспечения при совместном использовании UML-диаграмм и сетей Петри (обзор) / А.В. Марков // Сб. науч. тр. НГТУ. – 2013. – № 1(71). – С. 96–131.

[16] *Марков А.В.* Развитие системы «Перемещение манипулятора в пространстве с препятствиями» при помощи рекурсивных функций / А.В. Марков, А.А. Воевода // Автоматика и программная инженерия. – 2013. – № 2(4). – С. 35–41.

[17] *Марков А.В.* Матричное представление сетей Петри / А.В. Марков // Сб. науч. тр. НГТУ. – 2013. – № 2(71). – С. 61–67.

[18] *Марков А.*В. Анализ отдельных частей дерева достижимости сетей Петри / А.В. Марков // Сб. науч. тр. НГТУ. – 2013. – № 3(73). – С. 58–74.

[19] *Воевода А.А.* Применение UML диаграмм и сетей Петри при разработке встраиваемого программного обеспечения / А.А. Воевода, Д.О. Романников // Научный вестник НГТУ. – 2009. – № 4 (37). – С. 169–174.

[20] *Воевода А.А.* Редуцирование пространства состояний сети Петри для объектов из одного класса / А.А. Воевода, Д.О. Романников // Научный вестник НГТУ. – 2011. – № 4 (45). – С. 146–150.

[21] *Воевода А.А.* О модификации полного покрывающего дерева и графа разметок сети Петри / А.А. Воевода, С.В. Коротиков // Научный вестник НГТУ. – 2005. – № 1 (19). – С. 171–172.

[22] *Коротиков С.В.* Применение сетей Петри в разработке программного обеспечения центров дистанционного управления и контроля / С.В. Коротиков, А.А. Воевода // Научный вестник НГТУ. – 2007. – № 4 (29). – С. 16–30.

[23] *Christensen S.* A sweep-line method for state space exploration / S. Christensen, L.M. Kristensen, T. Mailund // Springer-Verlag Berlin Heidelberg. – 2001. – P. 450–464.

[24] *Kristensen L.M.* A generalized sweep-line method for safety properties / L.M. Kristensen, T. Mailund // FME. – 2002. – P. 549–567.

[25] *Holzmann G.J.* An analysis of bitstate hashing / G.J. Holzmann // Symposium on Protocol Specification. – 1995. – № 15. – P. 301–314.

[26] *Марков А.В.* Инверсия простой ординарной сети Петри / А.В. Марков, А.А. Воевода // Научный вестник НГТУ. – 2013. – № 4 (53). – С. 146–150.

[27] *Марков А.*В. Инверсия сетей Петри / А.В. Марков, А.А. Воевода // Сб. науч. тр. НГТУ. – 2013. – № 4 (74). – С. 97–121.

REFERENCES

[1] Peterson J. Petri nets theory and modeling. M.: Mir, 1984, 264 p.

[2] Westergaard M. Behavioral verification and visualization of formal models of concurrent systems: PhD dissertation. Aarhus: University of Aarhus, 2007.

[3] Voevoda A.A., Romannikov D.O. O kompaktnom predstavlenii yazykov raskrashennyx setej Petri. Sb. nauch. tr. NGTU, 2008, no. 3 (53), pp. 105–108.

[4] Voevoda A.A., Markov A.V. O kompaktnom predstavlenii yazykov setej Petri: seti s usloviyami i vremennye seti. Sb. nauch. tr. NGTU, 2010, no. 2 (60), pp. 77–83.

[5] Korotikov S.V. Primenenie setej Petri v razrabotke mnogopotochnogo programmnogo obespecheniya s ogranichennymi razdelyaemymi resursami na primere centrov distancionnogo upravleniya i kontrolya: dis. … kand. tehn. nauk Novosibirsk: NGTU, 2007.

[6] Markov A.V. Modelirovanie processa poiska puti v labirinte pri pomoshhi setej Petri. Sb. nauch. tr. NGTU, 2010, no. 4 (62), pp. 133–141.

[7] Romannikov D.O., Markov A.V., Zimaev I.V. Obzor rabot posvyashhennym razrabotke PO s ispolzovaniem UML i setej Petri. Sb. nauch. tr. NGTU, 2011, no. 1 (63), pp. 91–104.

[8] Markov A.V., Voevoda A.A. Modelirovanie processa poiska puti v labirinte pri pomoshhi setej Petri dlya sistemy iz dvux svyaznyx zvenev. Sb. nauch. tr. NGTU, 2011, no. 3 (65), pp. 95–104.

[9] Markov A.V. Poisk manipulyatorom kratchajshego puti v labirinte. Sb. nauch. tr. NGTU, 2011, no. 4 (66), pp. 75–91.

[10] Romannikov D.O., Markov A.V. Example application of the methodology of software development using UML-diagrams and Petri nets. Nauch. vestnik NGTU, 2012, no. 1 (67), pp. 175–181.

[11] Romannikov D.O. Razrabotka programmnogo obespecheniya s primeneniem UML diagramm i setej Petri dlya sistem upravleniya lokalnym oborudovaniem: dis. … kand. tehn. nauk Novosibirsk: NGTU, 2012, 195 p.

[12] Voevoda A.A., Markov A.V. Rekursiya v setyax Petri. Sb. nauch. tr. NGTU, 2012, no. 3 (69), pp. 115–122.

[13] Voevoda A.A., Markov A.V. Ponyatie rekursii v setyax Petri: faktorial chisla, chisla Fibonachchi. Sb. nauch. tr. NGTU, 2013, no. 1 (71), pp. 72–77.

[14] Markov A.V., Voevoda A.A. Analiz setej Petri pri pomoshhi derevev dosti-zhimosti. Sb. nauch. tr. NGTU, 2013, no. 1 (71), pp. 78–95.

[15] Markov A.V. Razrabotka programmnogo obespecheniya pri sovmestnom ispolzovanii UML-diagramm i setej Petri (obzor). Sb. nauch. tr. NGTU, 2013, no. 1 (71), pp. 96–131.

[16] Markov A.V., Voevoda A.A. Development of the system «Moving the manipulator in space with obstacles» using recursive functions. Avtomatika i programmnaja inzhenerija, 2013 no. 2, pp. 35–41.

[17] Markov A.V. Matrichnoe predstavlenie setej Petri. Sb. nauch. tr. NGTU, 2013, no. 2 (72), pp. 61–67.

[18] Markov A.V. Analiz otdelnyx chastej dereva dostizhimosti setej Petri. Sb. nauch. tr. NGTU, 2013, no. 3 (73), pp. 58–74.

[19] Voevoda A.A., Romannikov D.O. Applying UML diagrams and Petri nets in developing embedded software. Nauch. vestnik NGTU, 2009, no. 4 (37), pp. 169–174.

[20] Voevoda A.A., Romannikov D.O. Reducirovanie prostranstva sostoyanij seti Petri dlya obektov iz odnogo klassa. Nauch. vestnik NGTU, 2011, no. 4 (45), pp. 146–150.

[21] Voevoda A.A., Korotikov S.V. O modifikacii polnogo pokryvayushhego dereva i grafa razmetok seti Petri. Nauch. vestnik NGTU, 2005, no. 1 (19), pp. 171–172.

[22] Korotikov S.V., Voevoda A.A. Primenenie setej Petri v razrabotke programmnogo obespecheniya centrov distancionnogo upravleniya i kontrolya. Nauch. vestnik NGTU, 2007, no. 4 (29), pp. 16–30.

[23] Christensen S., Kristensen L.M., Mailund T. A sweep-line method for state space exploration. Springer-Verlag Berlin Heidelberg, 2001, pp. 450–464.

[24] Kristensen L.M., Mailund T. A generalized sweep-line method for safety properties. FME, 2002, pp. 549–567.

[25] Holzmann G.J. An analysis of bitstate hashing. Symposium on Protocol Specification, 1995, no. 15, pp. 301–314.

[26] Markov A.V., Voevoda A.A. Inversion of ordinary Petri nets. Nauch. vestnik NGTU, 2013, no. 4 (53), pp. 146–150.

[27] Markov A.V., Voevoda A.A. Inversion of Petri nets. Sb. nauch. tr. NGTU, 2013, no. 4 (74), pp. 97–121.

Марков Александр Владимирович – аспирант кафедры автоматики Новосибирского государственного технического университета. Основное направление исследования – анализ UML-диаграмм и сетей Петри. Имеет более 20 публикаций. E-mail: muviton3@gmail.com

A.V. Markov

Analysis of various methods for investigating state space of Petri nets

Described and presented in the form of a unified mathematical formalism of Petri nets and labeled transition system as one of the options of the state space. Analyzed reachability tree research methods: hashing method «floating line. »

The method using hash states, lies in the use of hash functions for the states in order to reduce the required memory for storing information about them. But with this method becomes probable partial research space attainable. In this connection, analyzed rough estimate possible hash values match. Another method, which was mentioned – a method of floating line, which is to remove the conditions not related to the rest of the state space. I.e. removes those states whose descendants are calculated assuming that they exist. Also analyzed by a modified method, named as Comeback method by which the hashing state information, as well as saving data about his parents and relations, of which received reachability. In conclusion of the

findings on the possibility of applying these methods, as well as their positive features and shortcomings.

Key words: Petri nets, UML, analysis of the state space, tree reachability analysis, algorithm, sweep-line method, bit state hashing, progressive evaluation, hashing, collision.