УДК 51.681.3

С.Л. КРЫВЫЙ, А.В. ЧУГАЕНКО, Л.Е. МАТВЕЕВА

ИССЛЕДОВАНИЕ СВОЙСТВ ДОКУМЕНТОВ MSC С ПОМОЩЬЮ ПРЕОБРАЗОВАНИЯ ИХ В СЕТИ ПЕТРИ[1]

Ключевые слова: *сети Петри, MSC, верификация.*

ВВЕДЕНИЕ

В настоящее время формальные методы широко используются для формальной спецификации, анализа и верификации программных и аппаратных систем. Суть формальных методов состоит в автоматизации верификации систем. В результате дополняются другие методы контроля качества, такие как тестирование, проверка кода и статический анализ кода, и появляется возможность повышения качества продукции до уровня, обычно недостижимого одним лишь тестированием.

Исследования в этой области привели к тому, что в настоящее время верификация программных систем выделилась в отдельную проблему. В частности, активно развивается метод проверки на модели (model checking), сделавший возможной верификацию реактивных систем. При этом спецификации записываются в определенном формальном логическом языке, в то время как сама система выступает в качестве модели, на которой проверяется выполнимость ее спецификаций.

1. ПОДХОДЫ К ВЕРИФИКАЦИИ ПРОГРАММНЫХ СИСТЕМ

Процесс развития современного программного обеспечения ввиду своей сложности (наряду с необходимостью быстрого реагирования на требования рынка, жесткими ограничениями времени и ресурсов, а также текучестью кадров в организациях) нуждается в различных методов проверки для удовлетворения конкретных запросов. В этом случае представляется целесообразным направить усилия на обеспечение качества.

Процесс развития технологий разработки программного обеспечения прошел долгий путь, начиная с 1970 г., когда была изобретена водопадная модель [1]. С появлением CMM/CMMI [2] системы обеспечения качества разработки программного обеспечения процесса получили широкое распространение, при этом продолжалась работа над созданием новых программных моделей. В последнее время популярными становятся гибкие методы (Agile Programming) [3, 4]. Эти методы обычно используются при инкрементальной и быстрой разработке приложений в ситуациях, когда определяющей является скорость выпуска продукта.

Современные гибкие подходы требуют разработки соответствующих новых методов и улучшенных процессов, в частности для интеграции формальных методов. Сочетание гибких подходов и средств автоматизированного контроля дает возможность быстрой адаптации контроля качества к изменению бизнес-среды. Итерационная разработка, быстрое прототипирование, метод частых релизов могут быть эффективно дополнены применением автоматизированной системы контроля. Такая система контроля позволит находить и определять изначально нестабильные и/или неполные требования и в то же время не приводит к увеличению продолжительности итерации.

[1]Работа выполнена при финансовой поддержке МОН Украины в рамках совместного Украинско-Болгарского проекта №145/23.02.2009 «Разработка распределенных лабораторий на основе прогрессивных методов доступа для поддержки проектирования сенсорных систем» и Болгарского национального научного фонда в рамках совместного Болгарско-Украинского проекта Д002-331/19.12.2008 с тем же названием.

© С.Л. Крывый, А.В. Чугаенко, Л.Е. Матвеева, 2009

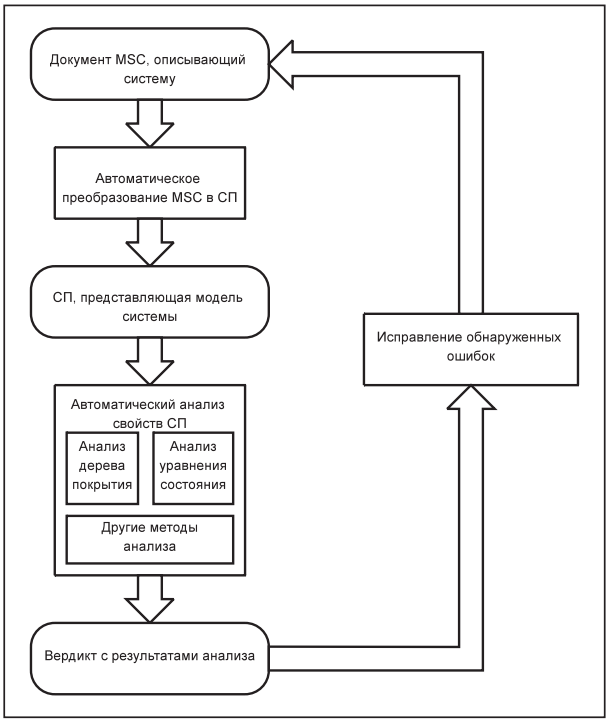

Рис. 1. Схема технологического процесса проверки системы

Настоящая работа завершает ряд статей, представленных ранее [5–8]. В сферу исследования входила разработка системы (рис. 1) для анализа и верификации программного обеспечения и аппаратных систем, заданных с помощью языка MSC-2000. В статье описан алгоритм автоматической трансляции такой системы в сеть Петри. Достоинством такого подхода является то, что от разработчиков не требуется изучения нового языка спецификаций и дополнительного математического образования.

Система позволяет применять разнообразные формальные методики моделирования к сетям Петри для уточнения и проверки требований и архитектуры исходной аппаратной или программной системы. Для этого исходная система, заданная на языке MSC-2000, переводится в ординарную сеть Петри.

В данной статье входной язык алгоритма расширен до всего языка MSC-2000. При этом последовательная композиция диаграмм интерпретируется как строгая и значения элементов типа *условие* игнорируется.

2. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Определение 1. Пусть $P = [S, \leq]$ — частично упорядоченное множество, a и b — его элементы. Будем говорить, что a доминирует над b, если $a > b$ (т.е. $a \geq b$ и $a \neq b$) и $\exists x \in S: a > x > b$. Будем говорить также, что a и b находятся в отношении доминирования, если a доминирует над b или b доминирует над a.

Отношение доминирования связано с частичным порядком, а именно частичный порядок однозначно восстанавливается из отношения доминирования в любом конечном частично упорядоченном множестве.

Язык MSC. MSC — язык моделирования динамических систем, который использует как графическое, так и текстовое представления и стандартизирован ITU (International Telecommunication Union) в [9] (синтаксис) и в [10] (семантика). Область приложения MSC включает в себя телекоммуникации, а также другие распределенные реактивные системы реального времени. MSC широко используется на ранних этапах разработки для сбора системных требований. Наличие стандартизированной семантики является одной из отличительных черт языка MSC по сравнению с другими языками моделирования.

Сети Петри. В статье используются определения сетей Петри, приведенные в [11]. В качестве формальной модели для определения семантики и анализа документов MSC используются ординарные сети Петри [6].

3. АЛГОРИТМ ПРЕОБРАЗОВАНИЯ ДОКУМЕНТОВ MSC В СЕТИ ПЕТРИ

3.1. Общие допущения и обозначения. Предполагается, что исходный документ MSC синтаксически корректен и соответствует статическим требованиям, изложенным в [9].

Тела встроенных выражений рассматриваются как анонимные диаграммы MSC (в соответствии с [10]). При этом согласно [9] принимается, что все события в документе MSC уникальны (при необходимости они переименовываются). В отличие от [9] здесь используется строгая последовательная композиция диаграмм MSC.

Полученная в результате применения данного алгоритма сеть Петри отражает только структуру исходного документа MSC, т.е. статическую часть поведения рассматриваемой системы; информация, находящаяся внутри элементов типа *условие*, не используется.

При описании алгоритма используются следующие обозначения:

- события MSC (включая элемент типа условие), как и соответствующие им фрагменты СП, обозначаем ev_i;
- последовательности и неупорядоченные группы событий MSC, как и соответствующие им фрагменты СП, обозначаем sev_i;
- фрагменты СП, используемые для представления управляющих конструкций MSC, называем конструкциями; в полученной СП метки переходов размещаются рядом со знаком перехода, в то время как дополнительная служебная информация (например, для обратной трассировки в MSC) помещается внутри значков перехода или места;
- $L^p(a)$ — префиксный язык размеченной сети a, который назовем языком СП.

3.2. Описание алгоритма. Представленный алгоритм основывается на том, что согласно семантике MSC ([10] с расширением, описанным в [6]) документ MSC может рассматриваться как набор событий, связанных последовательной, параллельной и альтернативной композициями. Тип используемой композиции выбирается в зависимости от управляющих конструкций языка MSC. Таким образом, алгоритм симулирует эти управляющие структуры с помощью фрагментов сети Петри, получая в результате сеть Петри, префиксный язык которой событийно эквивалентен набору трасс исходного документа MSC. Доказательство корректности алгоритма приведено в [8].

3.2.1. Шаги алгоритма. Рассмотрим построение сети Петри по документу MSC.

Шаг 1. Каждая диаграмма MSC (включая анонимные диаграммы, находящиеся в телах встроенных выражений) переводится в конструкцию, показанную на рис. 2,*а*, где $sev_1, \ldots, sev_n$ обозначают последовательности событий и конструкций, принадлежащих этой диаграмме. Если документ MSC содержит несколько ссылок на одну и ту же диаграмму, то они рассматриваются как несколько независимых экземпляров диаграммы и копируются в нужном количестве.

Шаг 2. Для каждого события MSC ev_i строится $Prev(ev_i)$ — множество событий и конструкций, непосредственно предшествующих этому событию. Затем ev_i переводится в конструкцию, показанную на рис. 2,*б*, где $n = |Prev(ev_i)|$, после чего входные места $p_1, \ldots, p_n$ соединяются с соответствующими событиями и конструкциями из $Prev(ev_i)$.

Шаг 3. Параллельная композиция диаграмм MSC (представлена конструкцией *par* в ссылочном либо встроенном выражении MSC или конструкцией *parallel frame* в HMSC) переводится в конструкцию, показанную на рис. 2,*в*, где n соответствует количеству аргументов этой композиции.

Шаг 4. Альтернативная композиция диаграмм MSC (представлена конструкциями *alt*, *opt*, *exc* в ссылочном либо встроенном выражении MSC или разделяющейся линией в HMSC) переводится в конструкцию, показанную на рис. 2,*г*, где n соответствует количеству аргументов этой композиции.

Шаг 5. Последовательная композиция диаграмм MSC (представлена конструкцией *seq* в ссылочном выражении MSC или линией в HMSC) переводится в конструкцию, показанную на рис. 2,*д*.

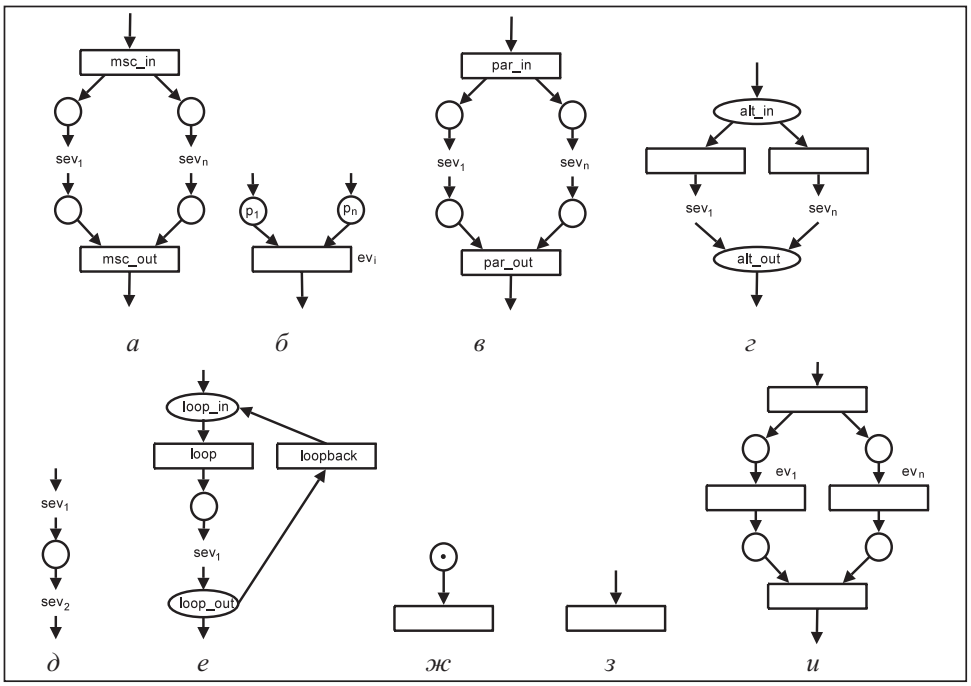

Рис. 2. Фрагменты СП, соответствующие конструкциям MSC

Шаг 6. Итерация диаграммы MSC (представлена конструкцией *loop* в ссылочном или встроенном выражении MSC) рассматривается как вариант альтернативной композиции и переводится в конструкцию, показанную на рис. 2,*е*.

Шаг 7. Последние элементы и конструкции (кроме элемента *stop*) композиции или диаграммы MSC соединяются с соответствующей конструкцией *_out*.

Шаг 8. Элемент HMSC start переводится в конструкцию, показанную на рис. 2,*ж*.

Шаг 9. Элемент HMSC stop переводится в конструкцию, показанную на рис. 2,*з*.

Шаг 10. Область параллельного выполнения рассматривается как параллельная композиция (согласно [10]) и переводится в конструкцию, показанную на рис. 2,*и*.

3.2.2. Построение множества $Prev(ev_i)$**.** Для каждого события ev_i множество $Prev(ev_i)$ включает следующие элементы и конструкции.

1. События, имеющие общую с ev_i сущность и расположенные непосредственно выше ev_i на линии этой сущности.

2. Конструкция *msc_in* диаграммы MSC, если ev_i расположено в начале диаграммы и не является событием типа *create in*.

3. События *message out*, *create out*, *call out* или *reply out* , если ev_i — соответствующее событие типа *message in*, *create in*, *call in* или *reply in*.

4. События, непосредственно предшествующие ev_i согласно явному упорядочению (выражены конструкциями *before* и *after*).

Чтобы уменьшить размер результирующей сети Петри, из множества $Prev(ev_i)$ можно исключить элементы, которые не удовлетворяют доминированию относительно порядка элементов. Данное сокращение $Prev(ev_i)$ не влияет на язык полученной сети Петри.

3.2.3. Свойства алгоритма. Определение 2. Документ MSC будем называть ациклическим, если в нем отсутствуют конструкции *loop* и его HMSC представляют ациклические графы.

Теорема 1. Для ациклического документа MSC сеть Петри, построенная приведенным выше алгоритмом, генерирует префиксный язык, включающий множество статических трасс, которые строятся на основе исходного документа MSC.

Теорема 2. Добавление конструкций итерации (конструкций *loop* или циклических ребер в HMSC) в документ MSC согласно вышеприведенному алгоритму не нарушает включения множества статических трасс, генерируемых исходным документом MSC, в язык построенной по нему СП.

Теорема 3. Для документа MSC и построенной по нему с помощью вышеописанного алгоритма сети Петри префиксный язык, генерируемый этой сетью, включается в множество статических трасс, генерируемых исходным документом MSC.

Доказательства этих теорем приведены в [13]

Из теорем 1–3 следует событийная эквивалентность статических трасс документа MSC и префиксного языка построенной по нему сети Петри.

4. ПРИМЕР

Рассмотрим применение представленного алгоритма для исследования свойств системы, описанной на языке MSC. Пусть дан документ, состоящий из трех диаграмм MSC и одной HMSC (рис. 3). Этот документ описывает простую систему с взаимным исключением, в которой два потребителя (`cons1` и `cons2`) конкурируют за общий ресурс (`res`). Получив доступ к ресурсу, каждый потребитель захватывает его, выполняет критическую секцию (`Cr1` или `Cr2`) и затем возвращает.

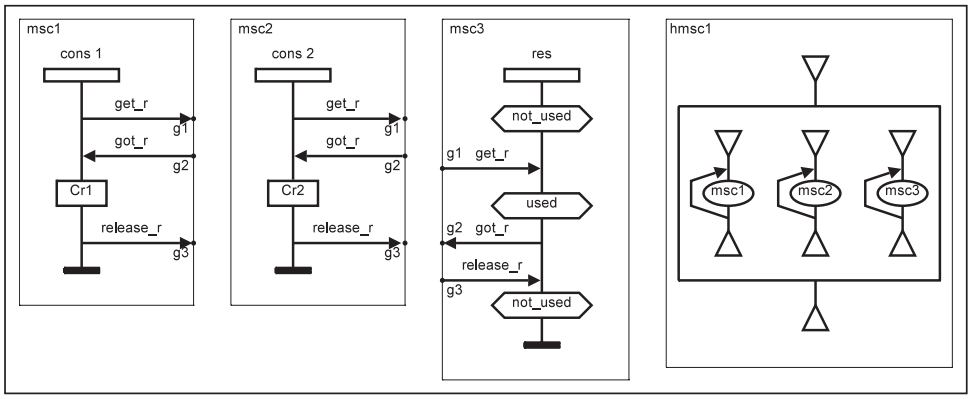

Рис. 3. Документ MSC для взаимного исключения

Вначале построим сеть Петри по приведенному документу MSC. После применения вышеописанного алгоритма получим сеть, показанную на рис. 4. Внутри элементов сети помещены названия соответствующих им элементов и конструкций исходного документа MSC. Для упрощения анализа редуцируем построенную сеть Петри согласно правилам, приведенным в [11]. Полученная сеть показана на рис. 5.

Далее применим метод матрицы инцидентности для проверки живости переходов сети [11]. Для этого необходимо решить диофантово уравнение $Ax = 0$, где A — матрица инцидентности сети Петри. Решив это уравнение методом, описанным в [12], получим следующий урезанный набор решений (они же T-инварианты сети Петри):

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Как видно из набора решений, все переходы сети, за исключением начального (t_0) и конечного (t_9), покрыты единицами и, следовательно, живы. Это означает, что в исходной системе нет дедлоков и ловушек.

Далее проверим ограниченность полученной сети Петри с помощью ее дерева покрытия. Полное дерево покрытия слишком громоздко для включения в статью (см. http://avch.org.ua/downloads/mutex_PN_tree.dot.gz), но его анализ показывает, что в нем отсутствуют ω-листы (т.е. в исследуемой сети нет неограниченных мест). Это значит, что данная сеть ограничена и в исследуемой системе не может быть неограниченного накопления запросов ресурса.

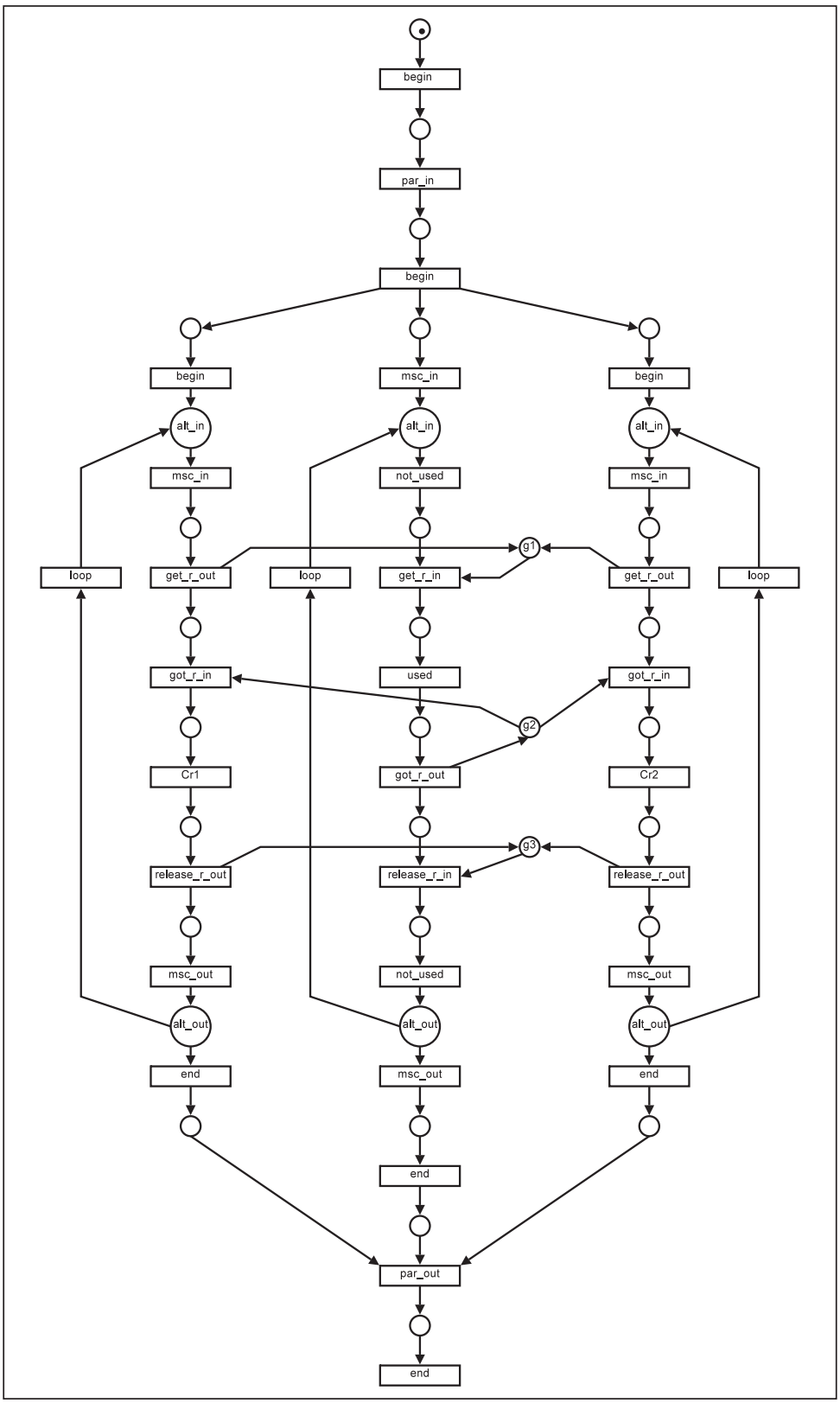

Рис. 4. Сеть Петри для примера с взаимным исключением

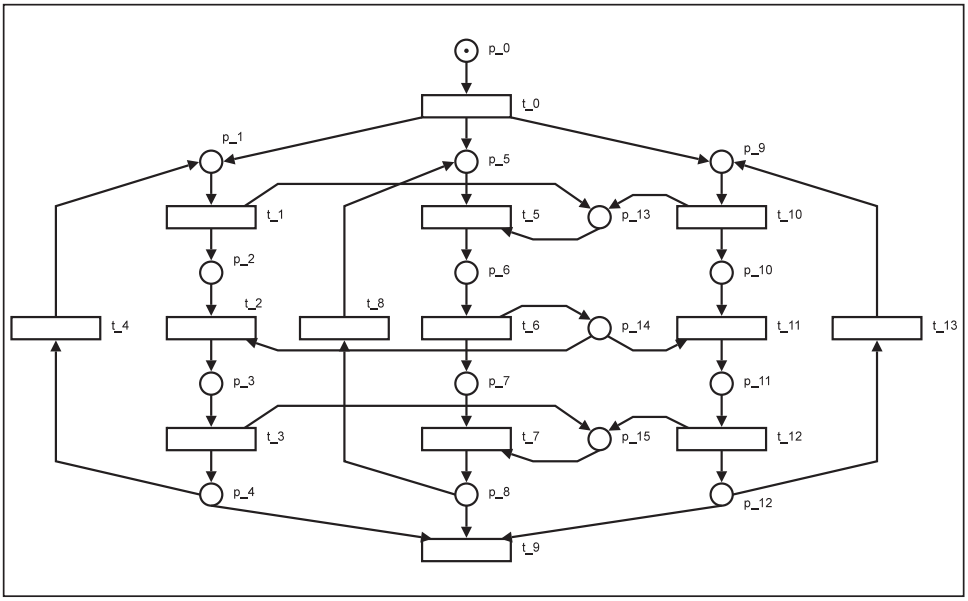

Рис. 5. Упрощенная сеть Петри для примера с взаимным исключением

ЗАКЛЮЧЕНИЕ

Настоящая работа завершает цикл статей [1–4]. В описанной версии алгоритма входной набор элементов расширен до полного языка MSC-2000, а также приведен пример, иллюстрирующий работу алгоритма. Представленный алгоритм реализован в виде полнофункционального прототипа и может использоваться для верификации программных систем.

СПИСОК ЛИТЕРАТУРЫ

1. Royce W. Managing the development of large software systems: Concepts and techniques / In Proceedings of IEEE WESCON, 1970. — P. 1–9.
2. CMMI product team. CMMI for system engineering/software engineering, Version 1.02 / Carnegie Mellon, Software Engineering Institute, 2000. — 606 p.
3. Beck K. Extreme programming explained: embrace change / Addison Wesley Longman, Inc., 2000. — 224 p.
4. Cockburn A. Agile software development. — Boston: Addison-Wesley, 2001. — 256 p.
5. Kryvyy S., Matvyeyeva L., Lopatina M. Automatic modeling and analysis of msc-specified systems // Fundamenta Informaticae. — 2005. — **67**, N 1–3. — P. 107–120.
6. Kryvyi S., Matvyeyeva L. Algorithm of translation of msc-specified system into petri net // Ibid. — 2007. — **79**, N 3–4. — P. 1–15.
7. Kryvyi S., Matvyeyeva L., Chugaenko A. Extension of algorithm of translation of msc-specified system into petri net / Proceedings of the CS\&P'2007 Workshop, 2007. — P. 376–387.
8. Kryvyi S., Matvyeyeva L., Chugaenko A. Converting of MSC documents to Petri Nets / Proceedings of the CS\&P'2008 Workshop, 2008. — P. 83–93.
9. ITU-TS Recommendation Z.120: Message Sequence Chart (MSC) / Edited by ITU-TS. — ITU-TS, Geneva, 2000. — 128 p.
10. ITU-TS Recommendation Z.120 Annex B: Algebraic semantics of Message Sequence Charts / Edited by ITU-TS. — ITU-TS, Geneva, 1998. — 73 p.
11. Murata T. Petri nets: Properties, analysis and applications / Proceedings of the IEEE, 1989. — **77**. — P. 541–580.
12. Krivoi S. Criteria of satisfiability for homogeneous systems of linear diophantine constraints // Lecture Notes in Computer Science. Parallel Processing and Applied Mathematics. — 2002. — **2328**. — P. 264–271.
13. Крывый С.Л., Чугаенко А.В. Формальные методы анализа дискретных систем с использованием языка спецификаций // Кибернетика и системный анализ. — 2009. — № 4. — С. 31–48.

Поступила 07.07.2009