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

А. А. Стененко, В. А. Непомнящий

ВЕРИФИКАЦИЯ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ МЕТОДОМ ПРОВЕРКИ МОДЕЛЕЙ

Препринт
178

Новосибирск 2015

Раскрашенные сети Петри (РСП) применяются для моделирования и анализа распределённых систем, например, таких, как коммуникационные протоколы.

В работе представлены метод верификации РСП с временными конструкциями и система верификации РСП CPNVer (Coloured Petri Nets Verifier), включающая транслятор из РСП в язык Promela и известную систему верификации SPIN, входным языком которой является язык Promela. Применимость нашего подхода продемонстрирована на примерах верификации коммуникационных протоколов PAR и ATMR.

Работа частично поддержана грантом РФФИ № 14-07-00401

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

A. A. Stenenko, V, A, Nepomniaschy

MODEL CHECKING APPROACH TO VERIFICATION OF COLOURED PETRI NETS

Preprint
178

Novosibirsk 2015

Colored Petri nets (CPN) are used for modeling and analysis of distributed systems, such as communication protocols. Naturally the problem of CPN formal verification appears. Model checking method was used for verification of CPN not involving the time concept. However, the problem of timed CPN verification remains open and relevant.

This paper presents a CPN verification system called CPNVer. The system components are a translator from CPN to Promela language and the SPIN verifier that uses Promela as a model language. CPNVer system allows automatic verification of timed CPN models. The verified properties can be presented with linear temporal logic (LTL) or specified in the form of some post-conditions. The applicability of our approach is demonstrated with examples.

1. ВВЕДЕНИЕ

Развитие методов и средств моделирования, анализа и верификации распределённых систем и, в частности, коммуникационных протоколов, является актуальной проблемой современного программирования. В качестве моделей распределённых систем используются конечные автоматы, сети Петри и их обобщения. Среди этих моделей можно выделить раскрашенные сети Петри (РСП), так как они имеют значительную выразительную силу, большой опыт применения и для них реализованы мощные средства анализа [4, 9]. Естественно возникает задача разработки средств формальной верификации РСП.

Верификация РСП рассматривалась в работах [5, 6, 7, 11], в которых в качестве метода верификации применялся метод проверки моделей. Однако в этих работах рассматривались РСП без временных конструкций. Таким образом, задача верификации РСП с временными конструкциями является актуальной открытой проблемой.

В данной работе представлены метод верификации РСП с временными конструкциями и система верификации РСП CPNVer (Coloured Petri Nets Verifier), включающая транслятор из РСП в язык Promela, который является входным языком известной системы верификации SPIN [8, 12]. Система CPNVer позволяет проводить автоматическую верификацию свойств РСП, представленных в линейной временной логике LTL, а также свойств, заданных в виде некоторых постусловий. Применимость нашего подхода продемонстрирована на примерах верификации коммуникационных протоколов PAR и ATMR.

Данная работа состоит из семи разделов. В разделе 2 описывается язык Promela, даётся определение графа состояний программы на этом языке, а также приводится обзор системы верификации SPIN. В разделе 3 даётся краткий обзор РСП. В разделе 4 описан алгоритм трансляции из РСП с временными конструкциями в язык Promela. В разделе 5 приводится оценка размера результирующей программы относительно размера исходной РСП. В разделе 6 описана верификация моделей коммуникационных протоколов PAR и ATMR. Заключение приводится в разделе 7.

2. ЯЗЫК PROMELA. ОБЗОР

Входной язык Promela системы верификации SPIN имеет синтаксис, похожий на синтаксис современных языков программирования высокого уровня, таких как язык C. В данном разделе приводится описание конструкций языка Promela, использующихся в выходных моделях транслятора из РСП. Так как модель на языке Promela является исполняемой программой, то далее модели на этом языке будем называть программами.

Основным понятием языка Promela является процесс, представляющий поток управления в программе. Код процесса является последовательностью исполняемых инструкций. Примерами инструкций языка Promela являются присваивания значений выражений переменным, операторы условного и недетерменированного выбора, атомарно исполняющиеся блоки (atomic), операторы перехода на метки (goto). Также в программе на языке Promela могут использоваться утверждения (assert) и вставки кода на языке C, которые могут содержать функции. В программе могут использоваться переменные следующих типов: целочисленные, перечисления (т.е. наборы именованных констант), массивы, а также переменные языка C, в том числе структуры. Динамическая память не может быть использована для представления и сохранения состояния программы на языке Promela. Состояние программы состоит из значений переменных и указателей на текущую исполняемую инструкцию каждого из процессов программы. Состояния можно рассматривать как последовательности байт. Промежуточные состояния могут быть исключены из рассмотрения посредством введения атомарно исполняющегося блока, который содержит данную последовательность инструкций. Блок кода на языке C не порождает промежуточных состояний.

Графом состояний программы назовём граф, вершинами которого являются состояния данной программы, а дуга из состояния S_1 в состояние S_2 присутствует в графе тогда и только тогда, когда программа может перейти из состояния S_1 в состояние S_2, минуя прочие состояния (не учитывая состояния, исключённые из рассмотрения). В данном графе могут присутствовать дуги из состояния S_1 в S_1. Достижимыми состояниями программы будем называть её начальное состояние и все вершины графа состояний, в которые существует путь из начального состояния.

Система SPIN позволяет проверять свойства, специфицированные формулами линейной темпоральной логики LTL. При верификации с использованием системы SPIN выполняется обход графа состояний программы на языке Promela. В процессе такого обхода верификатор запоминает посе-

щённые состояния во избежание зацикливания. При ветвлении графа состояний верификатор выполняет обход каждой из возможных ветвей исполнения. SPIN завершает работу, после того как будет осуществлён полный обход графа состояний, либо если в процессе верификации будет обнаружена ошибка. В последнем случае верификатор сообщит, какая последовательность исполнения привела к этой ошибке.

3. РАСКРАШЕННЫЕ СЕТИ ПЕТРИ

Класс рассматриваемых РСП является подмножеством сетей, с которыми работает система CPN Tools [4, 9], позволяющая просматривать, создавать и редактировать модели на языке РСП, а также предоставляющая возможность проводить их симуляцию. Модель на языке РСП представляет собой двудольный ориентированный граф, вершины одной из долей которого, называются *местами*, а вершины другой – *переходами*. Также для модели определено множество типов данных (так называемых «*наборов цветов*»), и переменных, имеющих типы, принадлежащие этому множеству. Для каждого из мест определён набор цветов. *Разметкой* места является некоторое мультимножество фишек, имеющих цвета, принадлежащие набору цветов этого места.

Переходы вместе с дугами определяют поведение модели. Каждой дуге приписано выражение. Для перехода может быть определено *охранное условие*. Переход является *допустимым* в некотором состоянии, если существует такой набор значений переменных модели, при которых его охранное условие истинно, а значения выражений на всех входящих в переход дугах являются элементами (либо подмножествами) разметки инцидентных им мест. Если переход является допустимым, то он может *сработать*. При срабатывании перехода из его входных мест извлекаются, а в его выходные места помещаются фишки, значения которых определяются выражениями на дугах. Для записи выражений используется язык CPN ML.

Разметка всех мест сети определяет состояние модели. Для задания начального состояния определяется *начальная разметка* каждого из мест. Ёмкостью места в некотором состоянии будем называть количество фишек в его разметке. Рассматриваются сети с ограниченной ёмкостью мест. Требуется указать верхнюю границу ёмкости мест.

Для модели, представленной на языке РСП, можно определить граф состояний, вершины которого являются состояниями модели, а дуга из состояния T_1 в состояние T_2 присутствует в графе тогда и только тогда, когда

модель может перейти из состояния T_1 в состояние T_2, минуя прочие состояния (т. е. в результате срабатывания одного перехода). Достижимыми состояниями РСП будем называть её начальное состояние и все вершины графа состояний, в которые существует путь из начального состояния.

РСП могут быть иерархическими. Иерархическая сеть состоит их нескольких сетей, находящихся на разных уровнях иерархии. Сеть может включать в себя подсеть с помощью так называемого *перехода-подстановки*, который при исполнении модели будет работать так, как будто на его месте находится экземпляр указанной подсети. Иерархия РСП должна быть ациклической. На корневом уровне иерархии находится одна корневая сеть. Места РСП, имеющие одинаковые тип и начальную разметку, могут быть объединены в так называемое *место-слияние*. Во время исполнения модели места, входящие в место-слияние, ведут себя как одно место, т.е. разметка таких мест меняется одновременно.

Рассматриваются РСП, использующие в качестве наборов цветов следующие типы данных: целые числа, булевские, перечисления, кортежи, записи, списки (требуется указать верхнюю границу длины списков), а также тип данных UNIT, содержащий одно значение.

Рассматриваемое множество языка выражений CPN ML состоит из переменных, целочисленных констант, элементов перечислений, логических констант, константы типа UNIT, констант пустого мультимножества и пустого списка, арифметических операторов и операторов сравнения для целых чисел, булевских операторов, функций для работы со списками, конструкторов кортежей и записей, операторов доступа к элементам кортежей и записей, операторов конструирования и объединения мультимножеств, условного оператора, а также временных конструкций. В рассматриваемое множество входят следующие функции для работы со списками:

- hd – доступ к значению первого элемента списка;
- tl – суффикс списка без начального элемента;
- cons – конструирование списка, полученного в результате добавления указанного элемента в начало указанного списка;
- nth – доступ к значению элемента списка по его индексу;
- nthtail – суффикс списка, начинающийся с указанного индекса;
- nthreplace – список, полученный в результате замены элемента с указанным индексом на указанное значение;
- concat – конкатенация двух списков;
- rev – указанный список в обратном порядке;

- rmall – список, полученный в результате удаления всех вхождений указанного значения в указанный список.

Кроме того, в выражениях могут использоваться дополнительные функции, для реализации которых используется язык C.

Переменными перехода T назовём переменные, которые входят в выражение, задающее охранное условие перехода T, либо в выражения на дугах, инцидентных переходу T. Возможные значения переменных каждого из переходов модели определяются значениями разметки его входных мест. Это означает, что каждая переменная перехода либо встречается в выражении на входящей дуге перехода, либо имеет булевский или перечислимый тип и при этом не имеет временной конструкции. Переменные, значения которых невозможно определить, основываясь на разметке входных мест, при срабатывании переходов принимают значения в результате недетерминированного выбора.

Понятие времени в РСП определяется через фишки, для которых типом данных является *тип со временной конструкцией*. Значения, которые могут принимать фишки, имеющие такой тип данных, отличаются от значений аналогичного типа данных без временных конструкций тем, что для них определена целая неотрицательная величина – *временная задержка*. Значение задержки определяет момент времени, начиная с которого данная фишка может быть использована при срабатывании перехода.

В состояние моделей с временными конструкциями, помимо разметки мест, входит также текущий момент времени, значение которого вместе с задержкой фишки определяет её доступность в текущем состоянии.

Значение счётчика времени в состоянии модели изменяется только в том случае, когда в текущем состоянии нет допустимых переходов [9]. При каждом изменении счётчика времени он увеличивается на минимальную величину, необходимую для того, чтобы сделать хотя бы один переход допустимым. Если ни один переход не может стать допустимым, то счётчик времени не изменяется, а такое состояние модели называется конечным.

В графе состояний РСП с временными конструкциями также могут присутствовать дуги между состояниями, которые отличаются только значением счётчика времени. Используемое понятие времени основано на понятии времени, используемом в системе CPN Tools, где для каждого из состояний определёно значение счётчика времени, которое показывает, сколько времени прошло с начала исполнения модели. Однако в отличие от моделей CPN Tools, величины меток времени фишек рассматриваются относительно текущего момента времени, а сама величина текущего момента не входит в состояние сетевой модели. Это позволяет считать состояния,

отличающиеся лишь сдвигом во времени меток всех фишек, одинаковыми. Когда в состоянии модели изменяется момент времени, вместо увеличения значения текущего момента времени уменьшаются все ненулевые задержки фишек.

Для целого неотрицательного числа C и фишки F со значением val и временной задержкой t определим функцию $j_{time}(F, C)$ как фишку со значением val и временной задержкой $max\{t - C, 0\}$. Для целого неотрицательного числа C и конечного мультимножества $A = \{a_1, a_2 \ldots a_n\}$ определим функцию $h_{time}(A, C)$ следующим образом: если мультимножество содержит фишки с временной задержкой, то $h_{time}(A, C)$ – это мультимножество $\{j_{time}(a_1, C), j_{time}(a_2, C) \ldots j_{time}(a_n, C)\}$; иначе $h_{time}(A, C) = A$.

Для состояния S, в котором значения счётчика времени равно C_s, и целого неотрицательного числа C определим функцию $g_{time}(S, C)$, значением которой является состояние РСП, в котором счётчик времени равен $C_s - C$, а для каждого места P разметкой является $h_{time}(M_p, C)$, где M_p – разметка места P в состоянии S.

Рассмотрим функцию $f_{time}(S)$, определённую на множестве состояний сетевой модели следующим образом:

1. Если в S есть допустимые переходы, то $f_{time}(S) = S$.

2. Иначе $f_{time}(S) = g_{time}(S, C)$, где C – это минимальное число, для которого верно, что в $g_{time}(S, C)$ есть допустимые переходы (если такой величины не существует, то C – это максимальная временная задержка по всем фишкам, которые присутствуют в состоянии S).

При исполнении сетевой модели в состоянии S, в котором отсутствуют допустимые переходы, система CPN Tools увеличивает текущий момент времени, переходя в состояние S'. При условии, что сетевая модель удовлетворяет указанному выше ограничению на временные конструкции, в состоянии $f_{time}(S)$ набор допустимых переходов не отличается от набора допустимых переходов в состоянии S'.

Рассматриваются временные конструкции, в которых в выражениях на дугах запрещено использование оператора «@», указывающего момент времени относительно начала исполнения модели, хотя данный оператор может использоваться для задания начальной разметки сети. В выражениях на дугах разрешается использовать оператор «@+», указывающий величину времени, на которую будет задержана фишка после срабатывания перехода.

4. ТРАНСЛЯТОР ИЗ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ В ЯЗЫК PROMELA

4.1. Обзор алгоритма трансляции

В данном разделе приводится описание транслятора из языка РСП в язык Promela. В системе CPNVer данный транслятор используется вместе с системой SPIN для верификации РСП, как показано на рис. 1.

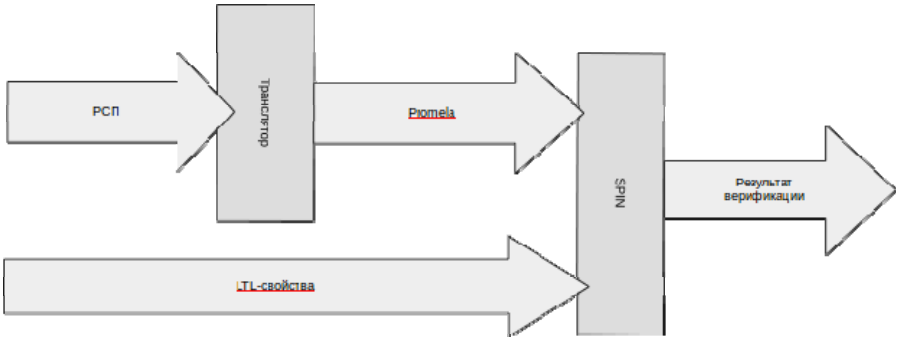

Рис. 1. Система CPNVer верификации РСП

Для программы на языке Promela, являющейся результатом работы транслятора, могут быть верифицированы следующие свойства:

1. Свойства, представленные в виде формул логики линейного времени LTL. Формулами LTL являются формулы вида p, $\neg f_1$, $f_1 \vee f_2$, $f_1 \& f_2$, $\Box f_1$, $\Diamond f_1$, где p – это предикат, определённый на множестве состояний программы, а f1 и f2 являются формулами LTL.

2. Свойство, заключающееся в выполнении постусловия, в качестве которого используется предикат по имени valid_end_state. Данный предикат входит в определяемую пользователем спецификацию модели. В состоянии, в котором нет допустимых переходов, истинность этого предиката проверяется конструкцией assert языка Promela. Если в состоянии есть допустимые переходы (либо возможно увеличение времени), то истинность данного предиката не проверяется. Таким образом, множество состояний, в которых нет допустимых переходов, разбивается на два непересекающихся подмножества: множество допустимых конечных состояний модели и множество «тупиков». Постусловие в таком случае обозначает, что модель

не оказывается в тупике, а завершает исполнение в допустимом конечном состоянии.

3. Свойство, заключающееся в отсутствии ошибок при вычислении выражений исходной модели, таких, например, как деление на ноль. Это свойство проверяется верификатором. Подробнее об ошибках, которые могут возникнуть при вычислении выражений, говорится в разделе 4.2.

Входная РСП должна быть представлена в формате, использующемся системой CPN Tools. Алгоритм трансляции работает поэтапно. Сначала происходит построение внутреннего представления транслируемой модели. Иерархические РСП разворачиваются в одноуровневые так, что каждый переход-подстановка корневой сети заменяется на экземпляр соответствующей ему подсети. Данный процесс продолжается, пока все переходы-подстановки в модели не будут заменены. Места, объединённые в место-слияние, заменяются единым местом. Такому месту будут инцидентны все дуги, инцидентные объединённым в место-слияние местам. Если для какого-либо места, входящего в место-слияние, определена начальная разметка, то эта разметка будет начальной для получившегося единого места. Наличие различной начальной разметки у объединённых в одно место-слияние мест является синтаксической ошибкой.

Внутреннее представление модели состоит из следующих множеств: типов данных, переменных, мест и переходов, а также дуг. В исходной модели для каждого места определёны имя, тип и выражение начальной разметки. В случае отсутствия выражения, задающего начальную разметку, таковым считается константа «Ø». Для переходов определены имя и охранное условие. В случае отсутствия охранного условия, таковым считается тождественно истинное условие. Для дуг определены выражения, а также пары соединяемых дугой вершин сети. В случае отсутствия выражения на дуге, выражением считается константа типа UNIT. На этапе построения внутреннего представления производится лексический и синтаксический анализ и проверка типов выражений языка CPN ML, используемых для задания начальной разметки мест, охранных условий переходов и выражений на дугах. При проверке типов проверяются следующие условия:

- аргументы и результат вычисления арифметических операторов являются целочисленными выражениями;
- аргументы и результат вычисления логических операторов являются булевскими выражениями;
- аргументы операторов-неравенств являются целочисленными выражениями, а результат вычисления этих операторов – булевским выражением;

- аргументы операторов-равенств имеют совместимый друг с другом тип, а результат вычисления этих опраторов является булевским выражением;
- элементы списка имеют совместимый друг с другом тип;
- в условном операторе два альтернативных варианта выражений имеют совместимый друг с другом тип.

Совместимость типов определяется следующим образом:

- каждый тип совместим с собой;
- пустой список совместим со списком, содержащим элементы любого типа;
- пустое мультимножество совместимо с мультимножеством, содержащим элементы любого типа;
- кортежи (или записи) с одинаковым набором полей совместимы, если совместимы типы соответствующих полей;
- списки совместимы, если совместимы типы их элементов;
- добавление временной конструкции не влияет на совместимость типов.

Когда внутреннее представление построено, происходит генерация выходной программы на языке Promela. При генерации транслируются типы данных, места (в том числе их начальная разметка) и переходы.

Используемые типы данных языка CPN ML транслируются в типы данных языка C следующим образом:

- целые, булевские, перечисления и тип UNIT переводятся в целые числа так, что значение «ложь» представляется числом 0, значение «истина» – числом 1, значение типа UNIT – числом 0, элементы перечисления представляются числами от нуля до N–1, где N является количеством элементов данного перечисления;
- кортежи и записи переводятся в структуры с полями соответствующих типов;
- списки переводятся в структуры с двумя полями, одно из которых является массивом для хранения элементов (неиспользуемая часть массива хранит начальные значения соответствующего типа данных), а другое представляет собой целочисленную переменную, значение которой представляет длину списка;
- временные типы данных транслируются в структуры с двумя полями, хранящими значение и временную задержку.

Заметим, что для каждого допустимого значения исходного типа данных существует единственное значение транслированного типа.

Для каждого типа данных генерируются функции cmp и null, предназначенные для работы с переменными данного типа. Функция cmp сравнивает два аргумента данного типа. Она возвращает ноль, если сравниваемые значения равны, или положительное значение, если первый аргумент больше второго, либо отрицательное в противном случае. Конкретное значение, возвращаемое данной функцией, может зависеть от значений аргументов. Списки, кортежи и записи эта функция сравнивает лексикографически. Типы с временной конструкцией сравниваются лексикографически как кортежи из двух полей: временной задержки и значения.

Функция null имеет один аргумент – указатель на переменную данного типа, которой эта функция присваивает начальное значение. Таким значением является 0 для целых чисел, false – для булевских, первое значение – для перечислений. Для кортежей и записей начальным значением является значение, при котором все поля кортежа или записи имеют свои начальные значения. Пустой список является начальным значением для списков. Для типов с временной конструкцией начальным значением является значение, при котором значение без времени является начальным и временная задержка равна нулю.

В коде результирующей программы макросы языка C используются для определения типов, в которые транслируются типы-списки, а также функций, работающих с переменными таких типов. Также макросы используются для определения типов с временными конструкциями.

Чтобы исключить служебные переменные из состояний программы, в результирующей программе используются вставки на языке C. Переменные, определённые внутри такого кода, используются во время поиска допустимых переходов и моделирования их срабатывания, а значения таких переменных не входят в состояние программы.

4.2. Трансляция выражений

Для синтаксического анализа выражений, записанных на языке CPN ML, используется алгоритм, основанный на преобразовании в обратную польскую нотацию (с учётом приоритетов операторов) и последующим построением дерева разбора. В процессе построения дерева разбора осуществляется вывод и проверка типов выражений.

Выражения на дугах используются двумя способами:

1. При моделировании срабатывания переходов результирующая программа вычисляет значение выражения, используя значения входящих в него переменных.

2. При поиске допустимых переходов программа находит значения переменных, входящих в выражение, основываясь на известном значении выражения.

В первом случае при трансляции выражения по его дереву разбора генерируется код на языке C, вычисляющий значение данного выражения. В результирующей программе выражения помещаются во вставки на языке C. Для трансляции выражений производится обход дерева разбора, во время которого генерируется код, вычисляющий значения подвыражений и сохраняющий их в служебных переменных.

Сгенерированный код содержит проверки, определяющие ошибочные ситуации, которые могут возникнуть во время исполнения программы на языке Promela. К ошибкам во время исполнения программы приводит выполнение следующих операций: деление на ноль, взятие элементов пустого списка, обращение к элементу списка по индексу, превышающему длину списка, конструирование мультимножества с отрицательным количеством элементов, превышение верхней границы длины списков и ёмкости мест.

Если вычисление выражения происходит без ошибок, то программа получает вычисленное значение и продолжает исполнение. Если же во время вычисления выражения возникает ошибочная ситуация, то вместо дальнейшего вычисления выполняется безусловный переход в конец вычисляющего выражение кода. Там обнаруживается наличие ошибки конструкцией assert языка Promela, при этом верификатор сообщает об ошибке и прекращает работу.

Во втором случае в качестве возможного значения выражения на дуге рассматриваются все элементы разметки инцидентного ей места. При генерации кода, определяющего значения переменных по значению элемента мультимножества, дерево синтаксического разбора выражения рассматривается от корня к листьям.

Процедура, генерирующая код, предназначенный для определения значений переменных, входящих в выражение на дуге, работает следующим образом:

- если выражение представляет собой переменную, то этой переменной присваивается значение выражения;
- если выражение представляет собой запись, кортеж или список, то определение значений переменных выполняется для его элементов, а для подвыражений, представляющих элементы записи, кортежа или списка, процедура генерации кода вызывается рекурсивно.

В иных случаях определение значений переменных, входящих в подвыражение, не производится.

Во время исполнения программы при определении значений переменных по значению выражения может возникнуть следующая ситуация: переменным невозможно сопоставить значения, так как выражение, являющееся списком, имеет длину, отличную от длины списка, являющегося требуемым значением выражения. Это означает, что для данного значения выражения переход не может сработать. В таком случае описанный процесс продолжается для других элементов.

4.3. Трансляция мест

Места РСП транслируются в переменные языка С. Значения этих переменных являются мультимножествами над соответствующими типами данных и представляют разметку мест сети. Способ представления мультимножества зависит от типа его элементов. Малыми будем называть типы данных, число возможных значений которых невелико. Такими считаются следующие типы без временных конструкций: UNIT, булевские и перечисления. Большими будем называть прочие типы, а именно целые числа, кортежи, записи и списки, а также все типы с временными конструкциями.

Для малых типов элементов мультимножество представляется в виде массива целых чисел. Элемент этого массива с индексом *i* равен количеству вхождений значения в мультимножество, которое представляется в программе целым числом i.

Для больших типов мультимножество представляется с помощью массива, содержащего его элементы, и целочисленной переменной, указывающей количество элементов данного мультимножества (эти элементы могут быть одинаковыми). Элементы мультимножества хранятся в таком массиве упорядоченно, а неиспользуемая часть массива содержит начальные значения данного типа. С учётом того, что представления элементов в программе, имеющих одинаковые значения, совпадают, это позволяет сравнивать значения мультимножеств как последовательности байтов. Таким образом, удовлетворяющие ограничению на ёмкость мест значения разметки однозначно отображаются на значения переменных, в которые места транслируются.

Для работы с переменными, представляющими мультимножества, генерируются функции empty, get и put. Функция empty имеет один аргумент, являющийся указателем на мультимножество. Эта функция присваивает переменной значение «Ø». Функция put имеет два аргумента: указатель на мультимножество и значение добавляемого элемента. Она изменяет значение мультимножества, добавляя в него данный элемент. Функция get имеет

два аргумента: указатель на мультимножество и дескриптор извлекаемого элемента. Для мультимножеств над большими типами дескриптором элемента является его индекс в массиве элементов мультимножества, а для мультимножеств над малыми типами – этот элемент. Функция get изменяет значение мультимножества, удаляя из него одно вхождение элемента, имеющего данный дескриптор, и возвращает в качестве результата значение этого элемента. Кроме того, для мультимножеств над большими типами генерируется функция copy, имеющая аналогичную функции get сигнатуру, и возвращающая элемент мультимножества по его индексу, но не изменяющая само мультимножество. Для определения типов, представляющих мультимножества, и соответствующих функций, в программе используются макросы языка C.

В РСП могут присутствовать места, разметка которых всегда содержит одну фишку. Признаком таких мест является следующее условие: смежные месту переходы имеют как входящую в него дугу, так и исходящую, а в начальной разметке такого места одна фишка. Такие места могут быть обнаружены на этапе трансляции с помощью анализа типов выражений на инцидентных дугах. С целью оптимизации в программе на языке Promela таким местам в соответствие ставится не переменная, представляющая мультимножество, а переменная, представляющая единственный элемент разметки.

Выражения начальной разметки мест транслируются в код на языке C, который будет выполнен в начале работы программы, и изменит её состояние с неинициализированного на начальное.

Значения переменных, представляющих разметку мест сети, входят в состояние программы. Таким образом, разным состояниям исходной РСП соответствуют разные состояния программы на языке Promela.

4.4. Трансляция переходов

4.4.1. Структура результирующей программы

Так как переходы РСП определяют поведение модели, конструкциями результирующей программы, которые соответствуют переходам РСП, являются исполняемые инструкции языка Promela. Эта программа содержит один процесс, включающий так называемый *главный цикл*, одно исполнение которого моделирует срабатывание одного перехода исходной модели.

Введём на множестве мест исходной модели функцию fp, отображающую место P в переменную $fp(P)$, являющуюся результатом трансляции

данного места. Определим на множестве состояний исходной модели функцию *ft*, которая сопоставляет состоянию *S* исходной модели состояние *ft*(*S*) программы на языке Promela. Функция *ft* определяется следующим образом: для каждого места *P* значение переменной *fp*(*P*) результирующей программы в состоянии *ft*(*S*) – это мультимножество, являющееся разметкой места *P* в состоянии *S*. Для любого состояния *S* значением указателя на инструкцию процесса результирующей программы в состоянии *ft*(*S*) является начальный оператор главного цикла. Состояние *Z* программы не соответствует никакому состоянию исходной модели, если не существует такого состояния *S*, что *ft*(*S*) = *Z*. Такими состояниями являются, например, состояния, в которых указатель на инструкцию процесса программы указывает не на начальный оператор главного цикла.

Тело главного цикла результирующей программы состоит из двух основных частей. Когда программа находится в состоянии *ft*(*S*), первая часть производит поиск переходов, допустимых в состоянии *S*. Вторая часть моделирует их срабатывание, то есть переводит программу из состояния *ft*(*S*) в состояние *ft*(*S′*), где *S′* – состояние, в которое переходит РСП в результате срабатывания любого перехода, найденного первой частью главного цикла.

Перед тем, как начать исполнение главного цикла, программа выполняет инициализацию переменных, в которые были транслированы места исходной РСП. После этого программа производит поиск допустимых переходов, а также значений их переменных и моделирует срабатывание недетерминированно выбранного перехода в главном цикле. Исполнение цикла происходит, пока программа не достигнет состояния *ft*(S_e), где S_e – состояние, в котором нет допустимых переходов.

В соответствии с определением времени в РСП момент времени изменяется только в том случае, если в текущем состоянии отсутствуют допустимые переходы. По этой причине изменять значения временных задержек требуется только в конце главного цикла программы и только в случае, если в его первой части не было найдено допустимых переходов. Если программа находится в состоянии, когда ни один переход не является допустимым, требуется проверить последующие моменты времени на наличие в них допустимых переходов. Для осуществления такой проверки требуется производить уменьшение значений временных задержек. Их уменьшение возможно, если в соответствующем состоянии РСП в разметке мест присутствуют фишки, помеченные положительными значениями времени. Все ненулевые временные задержки нужно уменьшить на единицу, после чего повторить поиск допустимых переходов.

Если все элементы мультимножеств с временной конструкцией имеют нулевую задержку, т.е. доступны в текущем состоянии, то при отсутствии допустимых переходов нужно перейти к завершению главного цикла. Прежде чем выйти из главного цикла и закончить исполнение, программа присваивает начальное значение всем входящим в состояние переменным. Это делается для того, чтобы избежать появления «лишних» состояний, то есть таких, которые не соответствуют никаким состояниям исходной модели.

Тело главного цикла заключено в атомарно исполняющийся блок языка Promela, поэтому все промежуточные состояния, в которых оказывается программа в процессе исполнения этого цикла, не порождают новых состояний, и такие промежуточные состояния не требуется сохранять при обходе графа состояний программы. Конечное состояние, в которое переходит программа после выхода из главного цикла, не соответствует какому-либо из состояний исходной РСП. Вне зависимости от того, в каком состоянии программа была до выхода из цикла, после выхода из него она оказывается в состоянии, в котором все входящие в состояние переменные имеют начальные значения.

Таким образом, граф состояний исходной модели является гомоморфным подграфу графа состояний результирующей программы. Такой подграф получается, если из графа состояний результирующей программы удалить неинициализированное (начальное) состояние и состояние, в котором оказывается программа после выхода из главного цикла. Начальной вершине S_0 графа состояний исходной модели соответствует вершина $ft(S_0)$, в которую ведёт дуга, исходящая из начальной вершины графа состояний результирующей программы, соответствующей состоянию <non-init>, в котором программа не инициализирована. Это состояние не соответствует какому-либо из состояний исходной модели. Для каждого состояния S_1 исходной модели в графе состояний результирующей программы из состояния $ft(S_1)$ выходит дуга, ведущая в конечное состояние <end>, кроме дуг, соответствующих дугам, присутствующим в графе исходной модели. В это состояние программа попадает, если во время перебора допустимых в состоянии S_1 переходов ни один из них не был выполнен. Это произойдёт, если недетерминированный выбор каждый раз завершался продолжением перебора. Наличие такого состояния следует учитывать при проверке свойств исходной модели.

4.4.2. Проверка допустимости переходов

Для проверки допустимости перехода необходимо найти все значения переменных, при которых он может сработать. Программа проверяет до-

пустимость перехода, производя перебор возможных значений этих переменных. Для нахождения возможных значений выражений на входных дугах перехода, производится перебор значений, являющихся элементами разметки его входных мест.

В соответствии с определением РСП для нахождения возможного значения переменной, которое осталось не определено после рассмотрения выражений на всех входных дугах перехода, производится полный перебор значений соответствующего типа данных. Если тип данных является большим, то алгоритм трансляции завершит работу в связи с ошибкой.

При рассмотрении возможного набора значений переменных перехода, требуется определить, является ли данный переход допустимым при данных значениях. Для этого вычисляется охранное условие перехода и проверяется истинность его значения. Если охранное условие ложно, то программа переходит к обработке следующего перехода. Если охранное условие истинно, то вычисляются значения выражений на всех дугах перехода для того, чтобы проверить, что вычисление значений этих выражений не приводит к ошибкам исполнения результирующей программы.

Затем для каждой входной дуги перехода проверяется, что значение выражения на ней является элементом или подмножеством разметки инцидентного ей места. Для каждой выходной дуги проверяется, что в результате добавления значения выражения на дуге в инцидентное ей место не будет нарушено ограничение на ёмкость мест. Во время выполнения такой проверки особым образом обрабатываются места, являющиеся для данного перехода как входными, так и выходными. Сначала происходит извлечение элементов разметки места, а затем – добавление новых элементов. Если ограничение на ёмкость будет превышено, то программа завершит исполнение с ошибкой.

Если найдены значения переменных, делающие переход допустимым, то программа выполняет недетерминированный выбор между тем, чтобы перейти к моделированию срабатывания данного перехода с данными значениями переменных, либо продолжить поиск допустимых переходов.

При проверке допустимости перехода, среди входных мест которого есть места, тип которых является типом с временной конструкцией, следует учитывать временные задержки элементов мультимножеств. Требуется, чтобы переход не был выполнен, если необходимые для осуществления перехода элементы недоступны в соответствующие моменты времени. Это значит, что переход будет считаться допустимым, только если в качестве значения выражения на каждой из его входящих дуг может быть выбрана

фишка, временная задержка которой не превышает значения временной задержки, вычисленной в выражении на данной дуге.

Если в выражении отсутствует оператор временной задержки, то задержка считается нулевой. При переборе элементов мультимножества, представляющего разметку входного места данного перехода, не рассматриваются элементы, временная задержка которых превосходит значение временной задержки в выражении на дуге, исходящей из данного места и ведущей в рассмативаемый переход.

4.3.3. Выполнение переходов

Фрагмент кода, моделирующий срабатывание перехода и таким образом изменяющий состояние программы, работает в соответствии с тем, как выполняются переходы в РСП. Если в первой части главного цикла было выбрано моделирование срабатывания перехода, то, используя найденные значения переменных перехода, производится вычисление выражений на дугах перехода. Согласно вычисленным значениям изменяются значения переменных программы, в которые были транслированы места исходной модели.

После того, как срабатывание данного перехода было смоделировано, результирующая программа возвращается к началу главного цикла, чтобы продолжить поиск и исполнение переходов в новом состоянии. Если переход не выбран в первой части главного цикла, то срабатывание перехода не моделируется.

5. ОЦЕНКА РАЗМЕРА ГРАФА СОСТОЯНИЙ ПРОГРАММЫ

Состояние программы на языке Promela определяется значениями указателя на исполняемую в данный момент инструкцию процесса, присутствующего в программе, и значениями переменных, в которые были транслированы места, а также служебных переменных, используемых во время поиска допустимых переходов.

Не все достижимые состояния программы являются вершинами графа её состояний, так как часть состояний исключается из рассмотрения с помощью атомарного блока. Все состояния, в которых оказывается программа, пока процесс исполняет атомарный блок, не входят в граф состояний программы.

Для нашей программы количество значений, которые может принимать указатель на инструкцию в различные моменты времени, не зависит от ис-

ходной модели, так как код, осуществляющий поиск допустимых переходов и моделирующий их срабатывание, заключён в атомарный блок.

Значения переменных, в которые были транслированы места исходной модели, взаимно однозначно отображаются на разметку сети. Значения этих переменных отличны от начальных в тех состояниях программы, в которых значением указателя инструкции является начало главного цикла. Когда программа начинает работу, эти переменные имеют начальные значения, а при выходе из цикла программа заново присваивает им начальные значения.

Служебные переменные принимают отличные от нуля значения только внутри атомарного блока, когда осуществляется поиск допустимых переходов и их срабатывание. Необходимость включения таких переменных в состояние программы обусловлена тем, как работает система SPIN. При осуществлении полного обхода графа состояний в глубину возникает необходимость возвращаться в предыдущее состояние. Это требуется для того, чтобы продолжить исполнение по другой ветви недетерминированного выбора, для чего нужно сохранять последовательность состояний, часть из которых не входит в граф.

Множество достижимых состояний программы включает состояния, соответствующие всем достижимым состояниям исходной модели. Также это множество включает состояния, которые не соответствуют состояниям исходной модели: начальное состояние программы и конечное состояние, в которое переходит программа, если не совершает допустимый переход. Таким образом, число достижимых состояний результирующей программы составляет $N + 2$, где N – количество достижимых состояний исходной модели.

Заметим, что при проверке свойств, заданных LTL-формулами, SPIN добавляет в программу вспомогательный процесс, называемый «*never claim*» [8], который исполняется вместе с другими процессами программы. В нашем случае этот процесс исполняется вместе с единственным процессом, а в состояние программы включается состояние вспомогательного процесса. Количество его состояний оценивается сверху как K^m, где K – некоторая константа, а m – длина заданной LTL-формулы [2]. Количество состояний программы, проверяющей свойство, специфицированное LTL-формулой, не превосходит произведения числа состояний результирующей программы и числа состояний процесса «never claim». Таким образом, число достижимых состояний программы, проверяющий истинность LTL-формулы длины m, не превосходит $(N + 2)\, K^{\boldsymbol{m}}$.

При верификации модели система SPIN производит обход графа достижимых состояний в глубину, в течение которого поддерживается стек

состояний программы. В этот стек помещаются все состояния, принадлежащие пути в графе достижимых состояний от начального состояния до текущего. В стеке хранятся не только те состояния, которые соответствуют вершинам графа достижимых состояний программы, но и промежуточные состояния, возникающие в процессе работы программы, поэтому использование памяти оказывается больше, чем необходимо для хранения вершин графа достижимости.

Оценим количество таких промежуточных состояний, возникающих в процессе верификации. Система SPIN сохраняет промежуточное состояние в случае, если в программе происходит недетерминированный выбор. Все операторы недетерминированного выбора содержатся в теле главного цикла программы, каждому из переходов исходной модели соответствует один такой оператор. Если из некоторого состояния S исходной модели непосредственно достижимо N состояний, то в результирующей программе состоянию S будет N промежуточных состояний, по одному промежуточному состоянию для каждой дуги графа достижимых состояний.

Таким образом, используемый размер памяти линейно зависит от количества дуг в графе достижимых состояний.

6. ПРИМЕРЫ ВЕРИФИКАЦИИ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ

6.1. Протокол PAR

В качестве примера рассмотрим РСП [14], моделирующую протокол PAR с подтверждением получения и повторной передачей [3]. Эта модель содержит процессы «отправитель» и «получатель», функционирующие в ненадёжной среде. «Отправитель» передаёт «получателю» последовательность пакетов. «Получатель» отправляет подтверждение о получении каждого пакета. «Отправитель» использует таймер для повторной отправки пакета, если не получает подтверждение в течение заданного интервала времени. Для моделирования таймера используется понятие времени в РСП. В протоколе используется бит чётности номера пакета для проверки того, что полученный пакет не является повторно принятым.

Была проведена верификация моделей, использующих этот протокол для передачи конечных последовательностей пакетов разной длины. Проверялись следующие свойства.

Свойство 1. Всегда выполняется предикат received_prefix «последовательность принятых пакетов является префиксом отправляемой последовательности».

Это свойство представлено в виде следующей LTL-формулы:

□ received_prefix

Свойство 2. Всегда, если «отправитель» посылает пакет c, и в дальнейшем какой-либо пакет будет получен и принят, то «получателем» будет получен и принят пакет c.

Представляющая это свойство LTL-формула записывается следующим образом:

□ (sent_c → ((◇ received) → (◇ received_c)))

При моделировании этого протокола посредством РСП предикат sent_c означает, что выполнен переход РСП, соответсвующий началу отправки пакета c. Предикат received означает, что был выполнен переход, соответствующий приёму некоторого пакета, а предикат received_c – что был выполнен приём пакета c.

Свойство 3. При завершении работы протокола длина последовательности принятых пакетов равна длине отправляемой последовательности пакетов.

Для проверки этого свойства соответствующим образом был определён предикат valid_end_state.

Верификация с помощью системы SPIN показала, что данные свойства для моделей выполняются. В таблице 1 приведены сведения о работе верификатора.

Таблица 1

Сведения о работе верификатора для моделей протокола PAR

Количество пакетов в последовательности	3	4	5
Число посещённых состояний модели	140	182	225
Число выполненных переходов	325	424	525
Объём использованной памяти	142 мегабайт	145 мегабайт	148 мегабайт
Время работы	0.06 секунд	0.07 секунд	0.09 секунд

Детали верификации протокола PAR даны в [14].

6.2. Протокол ATMR

Протокол ATMR является одним из стандартов ISO для кольцевых сетей с высокоскоростной передающей средой [1]. Согласно протоколу по кольцу перемещаются ячейки, которые могут иметь один из следующих типов: пустая, ячейка данных, и ячейка сброса. В пустую ячейку станция может поместить данные, сделав её ячейкой данных. Ячейка данных может быть сделана пустой только той станцией, которой адресованы содержащиеся в ней данные. Каждая станция может отправлять сообщения столько раз, сколько у неё имелось кредитов до того, как она начала отправлять сообщения. Получая ячейку сброса, станция возобновляет количество кредитов, устанавливая максимальное значение их числа. Ячейка также содержит поле, куда записывается номер станции, которая последней использовала её для передачи данных. Значение этого поля используется для того, чтобы определить, что станции нуждаются в возобновлении количества кредитов отправки сообщений.

Рассматривается модель данного протокола с тремя станциями [13] и с одной ячейкой, которая перемещается по кольцу. Максимальное количество кредитов для каждой станции составляет 2, а количество сообщений в последовательности, отправляемой каждой из станций, 3. РСП-модель включает место «cnt», содержащее единственную фишку целочисленного типа. Значение этой фишки показывает разность между количеством отправленных и количеством принятых сообщений. В начальном состоянии это значение равно 0. При отправке сообщения это значение увеличивается на единицу, а при приёме сообщения уменьшается на единицу.

При верификации проверялись следующие свойства:

Свойство 1. Отсутствие тупиков.

Для проверки этого свойства предикат valid_end_state определён как тождественно ложный.

Свойство 2. Отсутствие отправки сообщения до того, как был совершён приём последнего отправленного сообщения.

Свойство 3. Отсутствие повторного приёма принятого сообщения.

Свойство 4. Отсутствие приёма сообщения в случае, если никаких сообщений не было отправлено.

Конъюнкция последних трёх свойств эквивалентна условию на значение фишки, содержащейся в месте «cnt»:

cnt == 0 || cnt == 1.

Если обозначить это условие как предикат «prop», то LTL-формула, представляющая данные свойства, имеет следующий вид: □ prop

Т.к. все станции в кольце равнозначны, и модель обладает симметрией, то приведённые ниже свойства можно проверять для любой из станций; для определённости выберем первую.

Свойство 5. После того, как количество кредитов у станции будет исчерпано, оно будет восстановлено.

□ (p_empty_c1 → (◊ p_full_c1))

Предикат «p_empty_c1» означает «у первой станции 0 кредитов», а «p_full_c1» имеет смысл «у первой станции максимум кредитов».

Свойство 6. Если станции поступил запрос на отправку сообщений, то она начнёт их отправку.

□ (p_req_st1 → (◊ p_ack_st1))

Предикат «p_req_st1» означает «первая станция получила запрос на отправку сообщений», а предикат «p_ack_st1» истинен тогда и только тогда, когда эта станция начала передачу сообщений.

Свойство 7. Из состояния reset станция перейдёт в состояние send.

□ (p_reset_st1 → (◊ p_send_st1))

Предикаты «p_reset_st1» и «p_send_st1» истинны в случае, если первая станция находится в состоянии reset и send соответственно.

Свойство 8. Из состояния, в котором количество кредитов у станции максимально, она перейдёт в состояние, где это количество исчерпано.

□ (p_empty_c1 → (◊ p_full_c1))

Предикаты «p_empty_c1» и «p_full_c1» здесь определены так же, как и предикаты, используемые в свойстве 5.

Верификация с помощью системы SPIN показала, что данные свойства для модели выполняются. В таблице 2 приведены сведения о работе верификатора для данной модели.

Таблица 2

Сведения о работе верификатора для модели протокола ATMR

	Модель с тремя станциями
Число посещённых состояний модели	34416
Число выполненных переходов	151768
Объём использованной памяти	1911 мегабайт
Время работы	403 секунды

Детали верификации протокола ATMR даны в [13].

7. ЗАКЛЮЧЕНИЕ

В данной работе описан алгоритм трансляции РСП в язык Promela, позволяющий проводить с помощью системы SPIN верификацию определённого класса РСП как без временных конструкций, так и с такими конструкциями. Также в работе приведены оценки размера результирующей программы и примеры применения нашего подхода для верификации коммуникационных протоколов.

С целью расширения области применения нашего подхода предполагается разработать новую версию алгоритма трансляции, для которой результирующая программа используют меньший объём памяти.

СПИСОК ЛИТЕРАТУРЫ

1. Белоглазов Д. М., Машуков М. Ю., Непомнящий В. А. Верификация телекоммуникационных систем, специфицированных взаимодействующими конечными автоматами, с помощью раскрашенных сетей Петри // Моделирование и анализ информационных систем. – 2011. – Т. 18, №4. – С. 144–156.
2. Карпов, Ю. Г. Model Checking. Верификация параллельных и распределенных программных систем. – СПб.: БХВ-Петербург, 2010.
3. Таненбаум Э. Компьютерные сети. 5-е издание. – СПб.: Питер, 2012, 552 с.
4. CPN Tools Homepage. – http://cpntools.org (01.05.2015).
5. Evangelista S. High Level Petri Nets Analysis with Helena // Proc. ICATPN 2005. – Lect. Notes Comput. Sci. – Springer, 2005. – Vol. 3536. – P. 455–464.
6. Fronc L., Duret-Lutz A. LTL Model-Checker with Neco // Proc. ATVA 2013. – Lect. Notes Comput. Sci. – Springer, 2013. – Vol. 8172. – P. 451–454.
7. Fronc L., Pommereau F. Towards a Certified Petri Nets Model-Checker // Proc. APLAS 2011. – Lect. Notes Comput. Sci. – Springer, 2011. – Vol. 7078. – P. 322–336.
8. Holzmann, G. J. The Spin model checker: primer and reference manual. – Addison Wesley, 2003, 596 p.
9. Jensen K., Kristensen, L. M. Coloured Petri nets: modelling and validation of concurrent systems. – Springer, 2009, 384 p.
10. Kristensen L.M., Simonsen K.I.F. Applications of Coloured Petri Nets for Functional Validation of Protocol Designs // Proc. ToPNoC VII. – Lect. Notes Comput. Sci. – Springer, 2013. – Vol. 7480. – P. 56–115.
11. Kozura V.E., Nepomniaschy V.A., Novikov R.M.: Verification of Distributed Systems Modelled by High-level Petri Nets // Proc. Intern. Conf. On Parallel

Computing in Electronical Engineering, Warsaw, Poland. – IEEE Comp. Society, Los Alamitos. – P. 61–66.
12. SPIN – Formal Verification. – http://spinroot.com (01.05.2015).
13. Stenenko A. A. Verification of Coloured Petri Net Model of ATMR Protocol. – https://bitbucket.org/Stirpar/atmr-protocol (20.05.2015).
14. Stenenko A. A. Verification of Coloured Petri Net Model of PAR Protocol. – https://bitbucket.org/Stirpar/par-protocol (20.05.2015).
15. Vizovitin N. V., Nepomniaschy V. A., Stenenko A. A. Verification of UCM-specifications of Distributed Systems Using Coloured Petri Nets // Proc. of the Fourth Workshop "Program Semantics, Specifications and Verification: Theory and Applications". – 2013. – P. 70–80.

А. А. Стененко, В. А. Непомнящий

ВЕРИФИКАЦИЯ РАСКРАШЕННЫХ СЕТЕЙ ПЕТРИ МЕТОДОМ ПРОВЕРКИ МОДЕЛЕЙ

Препринт
178

Рукопись поступила в редакцию 22.05.2015
Редактор Т. М. Бульонкова
Рецензент Т.Г. Чурина

Подписано в печать 08.06.2015
Формат бумаги 60 × 84 1/16 Объем 1.66 уч.-изд.л., 1.8 п.л.
Тираж 60 экз.

Типография Оригинал-2, г. Бердск, ул. Олега Кошевого, 6, оф. 2
тел./факс: 8 (383) 328-32-38, (38341) 2-12-42, сот.: 8 913 987 77 67