УДК 681.31

Программный комплекс верификации алгоритмов программного обеспечения с помощью иерархических сетей Петри

И.В. Рудаков [1], А.В. Пащенкова [1]

[1] МГТУ им. Н.Э. Баумана, Москва, 105005, Россия

Рассмотрен метод формализации вычислительных алгоритмов с помощью иерархических сетей Петри. Разработан программный комплекс, реализующий работу излагаемого метода. Данный программный комплекс позволяет проверять модели алгоритмов программного обеспечения на наличие взаимоблокировок, невыполнимых операций, циклов и зацикливаний. В основе проверки моделей алгоритмов лежит такой метод анализа сетей Петри, как дерево достижимости.

E-mail: anna.pashchenkova@gmail.com

Ключевые слова: *сложные системы, формализация, верификация, сети Петри, иерархические сети Петри, дерево достижимости.*

Сложные программные системы характеризуются большим разнообразием взаимосвязей элементов, обработкой крупных массивов информации, элементов конкуренции при использовании ресурсов ЭВМ. Проектирование систем обработки данных связано с синтезом оптимального состава модулей программного обеспечения (ПО) на этапе технического проектирования программной системы. Структура программных модулей определяется обычно без учета альтернативных вариантов обработки, возможности параллельной реализации отдельных процедур, ветвей алгоритма и программных моделей. Возникает необходимость в разработке моделей системы для изучения взаимодействия ее элементов.

Одним из известных методов исследования процесса функционирования сложных систем является их формализация в виде сетей Петри. Данный математический аппарат позволяет формировать адекватные модели сложных систем и разрабатывать оптимальные алгоритмы решения задач.

Разработка ПО — сложный многоэтапный процесс, включающий в себя этапы анализа, непосредственного написания, тестирования и внедрения. Проектирование программного обеспечения, как и любых других сложных систем, выполняется поэтапно с использованием блочно-иерархического подхода, который основан на разбиении сложной задачи большой размерности на последовательно и/или параллельно решаемые группы задач малой размерности. Такой подход позволяет разбивать исследуемый объект на компоненты требуемой степени детализации и проверять работу каждой из компонент посредством моделирования.

Моделирование работы отдельных частей модели вычислительного алгоритма позволяет учитывать особенности процесса функционирования разрабатываемого программного обеспечения. Модель при этом анализируется с поведенческой точки зрения, рассматривается в виде последовательности дискретных событий.

Сети Петри предназначены для моделирования упорядочения инструкций и потока информации, но не для действительного вычисления самих значений. Модель системы по своей природе является абстракцией моделируемой системы, поэтому она игнорирует все возможные специфические детали. Если бы моделировались все детали, то модель была бы дубликатом моделируемой системы, а не абстракцией.

Смоделировав алгоритм работы ПО с помощью сетей Петри, можно получить информацию

– о количестве процессов в системе;

– наличии взаимоблокировок;

– наличии невыполнимых операций;

– количестве циклов, которые при определенных ситуациях могут стать причиной зацикливания.

С учетом предметной области разрабатываемый программный продукт должен выполнять ряд функций, в частности анализировать алгоритм на наличие взаимоблокировок, невыполнимых операций, циклов и зацикливаний; входные данные; типовые алгоритмические структуры; алгоритм, формализованный в виде сети Петри; сохранять модели в формате, обеспечивающем последующее редактирование ее параметров; предоставлять пользователю отчет с описанием программного комплекса; обеспечивать возможность интерактивного запуска работы сети.

Модуль, рассматриваемый как единое целое на определенных стадиях разработки или в процессе эксплуатации, является структурной составляющей ПО. Принципы модульности и иерархичности дают возможность организовывать коллективную параллельную разработку различных частей ПО, создавать открытые программные системы, облегчают их комплексную отладку и информационное согласование.

Выделяют следующие иерархические уровни представления и соответственно нисходящего проектирования ПО: системный, прикладных программ, подпрограмм.

На *системном уровене* конкретизируют функции программного комплекса, планируют его структуру и состав, выбирают или разрабатывают языки проектирования, устанавливают степень использования доступного для приобретения готового общесистемного и базового ПО, разрабатывают спецификации на отдельные программы пакета.

На *уровне прикладных программ* выбирают математическое обеспечение, разрабатывают специфические алгоритмы, устанавли-

вают модульную структуру программ, выбирают структуры данных, способы информационного интерфейса и язык программирования, разрабатывают спецификации на отдельные программные модули.

На *уровне подпрограмм* (*модулей*) проводится конкретизация типов и структур данных, осуществляется кодировка алгоритмов — их запись на выбранном языке программирования [1].

Процесс проектирования ПО состоит из нескольких этапов (рис. 1). Этапы 1—4 относят к синтезу ПО, их выполняют в нисходящей последовательности, на этапах 5—7 проводят отладку ПО, их выполняют в восходящей последовательности.

Рис. 1. Этапы проектирования программного обеспечения

На этапе 2 разрабатывают спецификации на отдельные программы программного комплекса. На этапах 3 и 4 решают охарактеризованные выше задачи уровней прикладных программ и подпрограмм.

На этапах 5—7 осуществляется отладка, цель которой — обнаружение и устранение ошибок, допущенных на этапах синтеза ПО. Отладка выполняется с помощью процедур выбора тестов и верификации.

Частичная верификация разрабатываемых алгоритмов и структур ПО возможна в рамках этапа 3 [1].

Сети Петри используют для формального моделирования ПО [2]. В каждой программе выделяют два различных аспекта процесса: вычисление и управление. Вычисление связано с текущими арифметическими и логическими операциями, вводом и выводом данных, обычными манипуляциями над содержимым памяти. Управление связано только с порядком выполнения вычислений.

Сети Петри удачно отображают структуру управления программ. Стандартным способом такого представления является блок-схема, которая представляет поток управления в программе и во многом по-

добна сети Петри: блок-схему изображают в виде узлов двух типов (принятия решения — графически показано ромбами — и вычисления — показывают прямоугольниками) и дуг между ними. Удобный способ выполнения блок-схемы — введение фишки, которая обозначает текущую инструкцию. По мере выполнения инструкций фишка передвигается по блок-схеме.

Перевод блок-схемы в эквивалентную сеть Петри заменяет узлы блок-схемы на переходы сети Петри, а дуги блок-схемы — на позиции сети Петри. Каждая дуга блок-схемы соответствует точно одной позиции в сети Петри. Узлы блок-схемы представляют поразному в зависимости от их типа: узел вычисления или узел принятия решения.

Фишка, находящаяся в позиции, означает, что счетчик команд установлен на готовность выполнения следующей инструкции. Каждая позиция имеет единственный выходной переход, за исключением позиции, которая имеет по два выходных перехода, соответствующих истинному и ложному значению предиката [2].

Для интерпретации сети Петри необходимо отображать каждый переход. Следует также отметить, что переходы для вычислений имеют по одному входу и выходу.

Иерархические сети Петри позволяют анализировать разные участки алгоритма с различной степенью детализации. Это дает возможность на этапе формализации использовать уже существующие модели, интегрируя их с разрабатываемой сетью, а также локализовать обнаруженные при анализе ошибки путем уточнения проблемных участков.

Дерево достижимости является одним из методов анализа сети Петри и представляет множество достижимости сети. Первоначальная маркировка находится в корневой вершине. Из каждой вершины исходят дуги, соответствующие разрешенным переходам. Всякий путь в дереве, начинающийся в корне, соответствует допустимой последовательности переходов. Сеть Петри может иметь бесконечное дерево достижимости. Для получения дерева, которое можно считать полезным инструментом анализа, необходимо найти средства ограничения его до конечного размера.

Особенностью алгоритма построения конечного дерева достижимости является специальная классификация маркировок, согласно которой каждую вершину дерева рассматривают как граничную, терминальную, дублирующую или внутреннюю вершину.

Граничными являются вершины, которые еще не обработаны алгоритмом, после чего эти вершины становятся либо терминальными, либо дублирующими, либо внутренними.

Маркировки, в которых отсутствуют разрешенные переходы, являются *терминальными* вершинами дерева достижимости. Другой класс маркировок — это маркировки, ранее встречавшиеся в дереве: такие маркировки называют *дублирующими* вершинами. Никакие по-

следующие маркировки рассматривать не требуется, так как все они будут порождены из места первого появления дублирующей маркировки в дереве.

Для сведе́ния дерева достижимости к конечному представлению используется еще одно средство. Для позиций, которые увеличивают число фишек некоторой последовательностью запусков переходов, можно создать произвольно большое число фишек, просто повторяя данную последовательность столько раз, сколько это нужно. Бесконечное число маркировок, получающихся из циклов такого типа, обозначают с помощью специального символа w, который означает «бесконечность». Таким образом, в маркировке число фишек может быть либо неотрицательным целым, либо w.

Алгоритм (рис. 2) начинается с определения корнем дерева начальной маркировки, т. е. граничной вершины. До тех пор, пока имеются граничные вершины, они обрабатываются алгоритмом.

Пусть x — граничная вершина, которую необходимо обработать. Если в дереве имеется другая вершина y, не являющаяся граничной, и с ней связана та же маркировка, т. е. $\mu[x] = \mu[y]$, то вершина x — дублирующая.

Если для маркировки $\mu[x]$ ни один из переходов не разрешен (т. е. значение $\delta(\mu[x], t_j])$ не определено для всех $t_j \in T$), то x — терминальная вершина.

Для всякого перехода $t_j \in T$, разрешенного в $\mu[x]$ (т. е. $\delta(\mu[x], t_j$ определено), необходимо создать новую вершину z дерева достижимости. Маркировка $\mu[z]$, связанная с этой вершиной, определяется для каждой позиции p_i следующим образом.

Если $\mu[x]_i = w$, то $\mu[z]_i = w$. Если на пути от корневой вершины к x существует вершина y с $\mu[y] < \delta(\mu[x], t_j)$ и $\mu[y]_i < \delta(\mu[x], t_j)_i$, то $\mu[z]_i = w$. В противном случае $\mu[z]_i = \delta(\mu[x], t_j)_i$.

Дуга, помеченная как t_j, направлена от вершины x к вершине z. Вершина x переопределяется как внутренняя, вершина z становится граничной. Когда все вершины дерева становятся терминальными, дублирующими или внутренними, алгоритм останавливается [3].

Тупиковым состоянием или взаимоблокировкой называется такая ситуация, когда каждый из множества процессов ожидает событие, которое может вызвать только другой процесс из этого множества. Условие взаимоблокировки может возникнуть в любой системе с несколькими потоками. Тупиковая ситуация в терминах сетей Петри подразумевает наличие тупиковой маркировки, при которой ни один из переходов не является разрешенным. Однако фактическое тупиковое состояние сети не всегда означает взаимоблокировку в программе.

По завершении работы алгоритма сеть, в которой формализован данный алгоритм, описывается так называемой тупиковой маркировкой. Необходимо различать ситуации корректного завершения работы сети от тупиковой ситуации. Для этого подразделяют позиции сети на два типа: простые и конечные.

Рис. 2. Алгоритм построения дерева достижимости

Согласно определению, терминальной вершиной дерева достижимости является маркировка, в которой отсутствуют разрешенные переходы, поэтому при поиске тупиков просматриваются все терминальные вершины дерева. Подразумевается, что тупиковое состояние сети не является таковым с точки зрения предметной области, если в каждой простой позиции сети находится ровно 0 фишек, и только конечные позиции сети могут иметь неотрицательное число фишек. Состояния, не удовлетворяющие этому условию, считаются тупиковыми. Таким образом, маркировка сети (0,0,1) будет считаться тупиковой только в том случае, если позиция $p2$ не будет помечена как конечная (рис. 3).

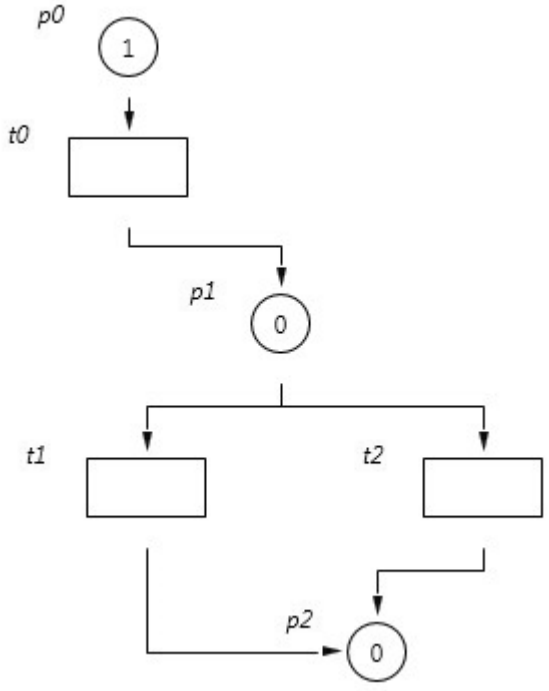

Рис. 3. Пример сети Петри

При анализе дерева достижимости целесообразно также осуществить поиск всех циклов. Это обусловлено тем, что циклы, в которых не предусмотрен или из которых никогда не выполняется выход, станут причиной зацикливания.

Для того чтобы найти цикл в дереве достижимости, просматривают все его дублирующие вершины. От каждой такой вершины проводится «подъем» по родителям этой вершины к корню дерева до тех пор, пока не найдется в дереве вершина такого же уровня вложенности с такой же маркировкой. Если такая вершина отсутствует, то цикл не найден. На рис. 4 показан пример сети с циклом. При срабатывании перехода $t0$ фишка попадет в цикл, из которого предусмотрен выход с помощью перехода $t3$.

Ситуация зацикливания программы возникает в случае, когда вычисления проходят по некоторому замкнутому циклу, не останавливаясь. Поиск зацикливаний подразумевает поиск циклов, из которых нет выхода. Для обнаружения ситуации зацикливания в дереве достижимости просматриваются все вершины. Если вершина является дублирующей и на данном уровне вложенности отсутствует терминальная вершина, выполнение сети на этом уровне не закончится никогда, а значит, имеет место зацикливание. В сети на рис. 5 при срабатывании перехода $t0$ фишка попадает в цикл, из которого никогда не выйдет. Таким образом, работа сети не будет завершена.

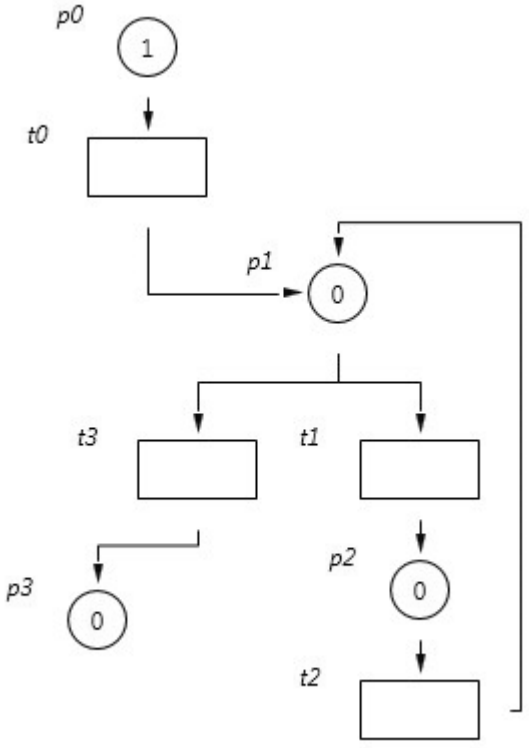

Рис. 4. Пример сети с циклом

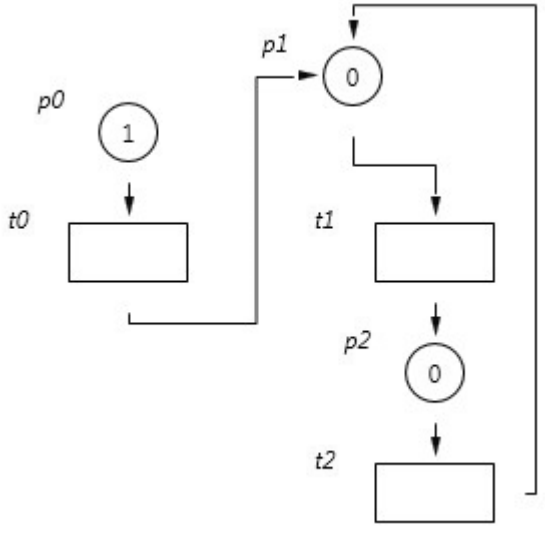

Рис. 5. Пример сети с зацикливанием

Помимо тупиковых ситуаций, циклов и зацикливаний следует отслеживать, все ли части алгоритма выполняются. Под невыполнимыми операциями подразумевают переходы, которые никогда не срабатывают.

Для нахождения таких переходов в процессе построения дерева достижимости необходимо отмечать переходы, которые могут сработать. Затем простым пересечением множества всех переходов и множества переходов, которые могут сработать, получается множество переходов, до которых выполнение никогда не дойдет.

Функционирование разработанного программного комплекса можно описать с помощью схемы, приведенной на рис. 6.

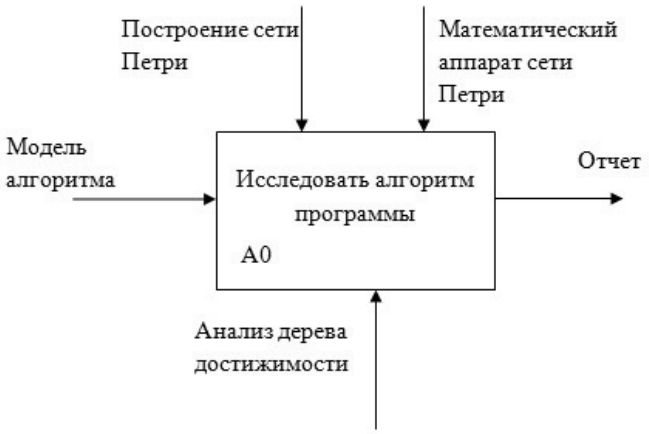

Рис. 6. IDEF0-диаграмма функционирования программы

Программный комплекс анализа алгоритмов, формализованных сетью Петри, включает следующие основные модули приложения:

– модуль редактора сети Петри — содержит как графическое представление сети, так и аналитическое;

– модуль графического редактора — описывает основные функции любого графического редактора, такие как добавление элементов, удаление, изменение размера и пр.;

– модуль сети Петри — содержит математическое описание сети, определение маркировки сети, допустимых переходов и прочие вспомогательные функции;

– модуль построения дерева достижимости — строит дерево достижимости данной сети Петри;

– модуль анализа дерева достижимости — анализирует построенное дерево применительно к данной предметной области;

– модуль матричного метода — решает задачу достижения некоторой маркировки.

Диаграмма модулей и взаимосвязей представлена на рис. 7.

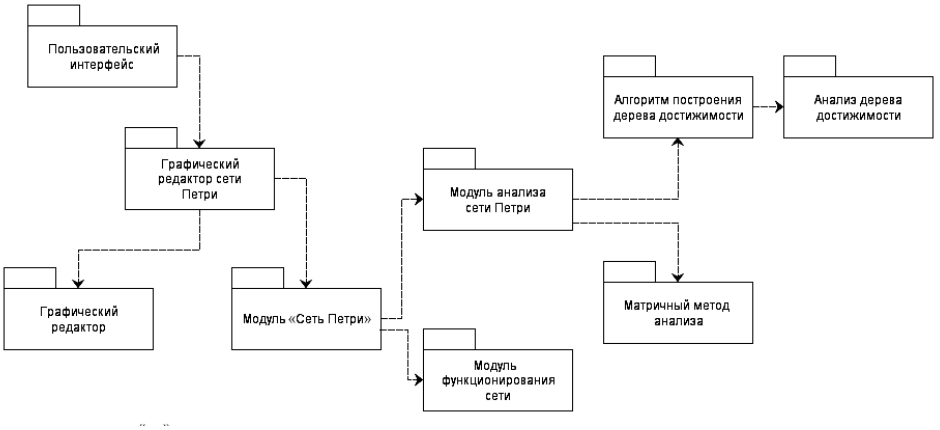

Рис. 7. Структура программного комплекса

Рассмотренный программный комплекс анализа алгоритмов работы ПО с помощью сетей Петри был протестирован на ряде известных алгоритмов, подтвердив свою работоспособность. Установлено, что время анализа модели алгоритма зависит непосредственно от размера построенного дерева достижимости сети: чем больше вершин содержит дерево достижимости, тем больше время анализа. Используя предложенный метод, можно получить информацию о наличии взаимоблокировок, невыполнимых операций, циклов и зацикливаний, что позволяет повысить надежность разрабатываемого программного обеспечения.

СПИСОК ЛИТЕРАТУРЫ

1. Норенков И.П. Основы автоматизированного проектирования: учеб. для вузов. М.: Изд-во МГТУ им. Н.Э. Баумана, 2002. 306 с.
2. Котов В.Е. Сети Петри. М.: Наука, Гл. ред. физ.-мат. лит., 1984. 160 с.
3. Питерсон Дж. Теория сетей Петри и моделирование систем: пер. с англ. М.: Мир. 1984. 264 с.

Статья поступила в редакцию 25.10.2012