МИНИСТЕРСТВО ТРАНСПОРТА И СВЯЗИ УКРАИНЫ
ОДЕССКАЯ НАЦИОНАЛЬНАЯ АКАДЕМИЯ СВЯЗИ им. А.С.ПОПОВА

На правах рукописи

ЗАЙЦЕВ ДМИТРИЙ АНАТОЛЬЕВИЧ

УДК 621.39, 004.7

МЕТОДЫ АНАЛИЗА И СИНТЕЗА МОДЕЛЕЙ ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМ НА ОСНОВЕ ФУНКЦИОНАЛЬНЫХ СЕТЕЙ ПЕТРИ

05.12.02 – телекоммуникационные системы и сети

Диссертация на соискание учёной степени
доктора технических наук

Научный консультант
Воробиенко Пётр Петрович,
д.т.н, профессор

Одесса – 2006

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

IETF – Internet Engineering Task Force – Оперативная группа разработчиков Интернет

IEEE – Institute of Electrical and Electronics Engineers – Институт инженеров электротехники и электроники

ITU – International Telecommunication Union – Международный союз телекоммуникаций

ECMA – European Computer Manufacturer Association – Европейская ассоциация производителей компьютеров

IP – Internet Protocol – протокол Интернет

TCP – Transmission Control Protocol – протокол управления передачей

BGP – Border Gateway Protocol – протокол граничных шлюзов

IOTP – Internet Open Trading Protocol – открытый протокол торговли (электронной коммерции) Интернет

MPLS – Multiprotocol Label Switching – многопротокольная коммутация меток

WLAN – Wireless Local Area Network – беспроводные локальные вычислительные сети

WPAN – Wireless Personal Area Networks – беспроводные персональные вычислительные сети

ЛВС – локальная вычислительная сеть

СЛАУ – система линейных алгебраических уравнений

СЛДУ – система линейных диофантовых уравнений

ВВЕДЕНИЕ

Телекоммуникационные системы представляют собой разновидность параллельных систем (синхронных и асинхронных), в которых отдельные телекоммуникационные устройства взаимодействуют посредством специальной среды для передачи информации. Среда может быть представлена проводниками электрических либо оптических сигналов в кабельной связи, эфиром для распространения электромагнитных полей в беспроводной связи. Теория электросвязи исследует принципы передачи сигналов в среде, а также принципы построения передающих и принимающих устройств. Однако в настоящее время всё более возрастает сложность процессов взаимодействия систем, представленная множеством протоколов, а также устройств и программного обеспечения, реализующих протоколы.

На абстрактном уровне при исследовании телекоммуникационных систем целесообразно рассматривать лишь порядок их взаимодействия и архитектуру, обеспечивающую правильное взаимодействие. Известная эталонная архитектура взаимодействия открытых систем является большей частью декларативной, определяющей лишь принципы организации уровней, перечни реализуемых функций. Конкретное семейство протоколов требует формального представления порядка взаимодействия, а также формальных языков описания архитектур систем, а в большинстве случаев и архитектуры среды, которая может содержать вероятных злоумышленников и источники помех.

Как правило, большинство современных стандартов используют подмножество естественного языка для построения спецификаций, а также содержат схемы и диаграммы, предназначенные для дополнительных пояснений. При этом ошибки могут возникать на каждом из этапов создания телекоммуникационных систем от стандартных спецификаций протоколов до процессов проектирования устройств и программного обеспечения. В диссертационной работе предложен универсальный язык функциональных сетей Петри, который позво-

ляет представлять как сами протоколы, так и архитектуру телекоммуникационных систем, включающую физические устройства и программное обеспечение. Композиция функциональных подсетей обеспечивает синтез иерархических многоуровневых систем, а также существенное ускорение процессов их анализа.

Применение сетей Петри для исследования протоколов впервые было осуществлено М.Диазом и Ж.Берселотом в 1982-м году. В их работах были сформулированы свойства моделей идеальных протоколов. В Советском Союзе основателем научного направления, разрабатывавшего специальные классы сетей Петри для верификации протоколов, выступил В.Е.Котов, возглавивший затем лабораторию параллельных вычислений фирмы HP. Методы проектирования систем управления, в том числе для компьютерных сетей и телекоммуникационных устройств на основе нагруженных сетей Петри разработаны А.И.Слепцовым и представлены в его монографии в 1986 году. Развитием теории регулярных сетей В.Е.Котова выступили работы Н.А.Анисимова, посвящённые композиции моделей Петри протоколов с помощью специального множества алгебраических операций. Эти работы по существу завершают множество публикаций посвященных формальным методам верификации протоколов, поскольку исследователи столкнулись с проблемой размерности сетей Петри, сформулированной в работах А.Марсана при исследовании протоколов Ethernet. Детализированная модель реального протокола имеет, как правило, размерность, не позволяющую выполнить её анализ за приемлемое время. Не смотря на многочисленные эвристики, предложенные для решения линейных диофантовых систем в целых неотрицательных числах М.Силвой и Дж.Коломом, для повышения эффективности метода Тудика, существенные ускорения вычислений не были получены.

Раскрашенные сети Петри, изученные в монографиях К.Йенсена в 1997 году, по существу представляют собой подмножество нагруженных сетей, исследованных А.И.Слепцовым. Практический успех применения раскрашенных сетей, прежде всего, обусловлен разработкой и свободным распространением

промышленных моделирующих систем, таких как Design/CPN и CPN Tools, выполненными при поддержке фирм Metasoft и Microsoft. К настоящему времени эти системы применены в десятках реальных проектов, более трети которых связаны с телекоммуникациями. Показательно, что в настоящее время система CPN Tools применяется как основная моделирующая система для управляемого моделью проектирования новых поколений мобильных телефонов фирмы Nokia. Научное направление, осуществляющее применение раскрашенных сетей Петри для моделирования телекоммуникационных протоколов основано Дж.Биллингтоном. Термином "раскрашенные сети Петри" в соответствии с трёхтомной монографией К.Йенсена в настоящее время характеризуют временные, нагруженные, иерархические сети Петри; для описания нагрузок элементов графа сети Петри применён язык функционального программирования CPN ML. Раскрашенные сети являются универсальной алгоритмической системой и могут быть использованы для спецификации произвольных систем и процессов.

Проблематика анализа и синтеза параллельных взаимодействующих систем, разновидностью которых являются телекоммуникационные системы, реализующие определённые протоколы, имеет два основных аспекта, кратко характеризуемые терминами: корректность и эффективность. Доказательство корректности осуществляют формальными методами и для этих целей предпочтительно применение классических сетей Петри. Для раскрашенных сетей единственным известным формальным методом исследования является тривиальная генерация полного пространства состояний, что практически осуществимо лишь для учебных моделей малой размерности.

Впервые введенный соискателем класс функциональных сетей Петри применён для композиции, как классических, так временных и раскрашенных сетей Петри. Причём в отличие от композиционных методов, изученных В.Е.Котовым и А.А.Анисимовым, использована лишь одна операция совмещения контактных позиций. Кроме того, класс функциональных сетей существенно отличается от S- и T-компонентов, изученных М.Хаком и известных других

способов декомпозиции. Основной результат, обусловленный осуществлённым впервые соискателем построением композиционного анализа на основе функциональных сетей, состоит в существенном ускорении исследования классических сетей Петри методами линейной алгебры. Ускорения вычислений в ряде случаев позволяют выполнить анализ моделей, ранее рассматривавший как практически неосуществимый. Таким образом, решена проблема доказательства корректности (верификации) детализированных моделей большой размерности для реальных протоколов.

Построение передаточной функции сети Петри позволило разработать теоретические основы для эквивалентных преобразований сетей, причём результаты сформулированы для класса временных сетей с многоканальными переходами. Разновидностью эквивалентных преобразований является редукция сетей, снижающая размерность модели, сохраняя её свойства. Правила редукции, сформулированные Ж.Берселотом, могут рассматриваться как частный случай редукции на основе алгебраических преобразований передаточной функции сети. Таким образом, эквивалентные преобразования являются методом, предназначенным для ускорения анализа временных моделей Петри большой размерности.

Соискателем предложено выполнять синтез моделей телекоммуникационных систем как композицию подсетей с контактными позициями (функциональных подсетей) для класса раскрашенных сетей Петри в целях построения эффективных телекоммуникационных систем. Для оценки функциональных характеристик моделей предложен метод измерительных фрагментов, позволяющий вычислять характеристики в процессе имитации динамики сети Петри. Такой подход не требует построения полного множества состояний модели, а состоит в сборе и статистической обработке информации в процессе имитации на достаточно продолжительных интервалах времени, обеспечивающих наблюдение стационарного режима.

Актуальность темы. Динамичное развитие телекоммуникаций приводит к удвоению количества используемых протоколов каждые пять лет, а также уд-

воению количества разновидностей выпускаемых устройств каждые четыре года. Кроме того, возрастает сложность самих протоколов, что может быть охарактеризовано, по крайней мере, объёмом исходных спецификаций и количеством рассматриваемых в них терминов. Возрастают объёмы передаваемой информации, так, например, объём ежедневного трафика всемирной сети Интернет оценивается сотнями петабит. Экспертные оценки позволяют оценить возрастание доли критической информацией, связанной с перемещением определённых финансовых средств, а также информации, обеспечивающей безопасность функционирования искусственных систем. Безопасность человечества всё более зависит от надежности телекоммуникаций, используемых в управлении технологическими процессами.

Стоимость ошибок в исходных спецификациях протоколов является наиболее существенной, поскольку спецификации воплощаются затем в конкретных устройствах и программном обеспечении, выступающими в качестве носителей ошибок. Кроме того, отмеченные процессы проектирования устройств и программного обеспечения сами выступают в роли источников ошибок. Трудно переоценить последствия блокирования Интернет в нынешнее время на период в несколько дней, аналогичный последствиям известного червя Мориса, использовавшего ошибки программного обеспечения, реализующего протоколы электронной почты.

Таким образом, возникает научная проблема доказательства корректности и оценки эффективности телекоммуникационных систем. Решению отмеченной проблемы на основе разработанной соискателем теории функциональных сетей Петри и посвящена диссертационная работа.

Связь работы с научными программами, планами, темами. Работа выполнена в рамках Рабочих программ IETF по совершенствованию протоколов Интернет, специфицированных в RFC; Программы развития протокола Bluetooth общества Bluetooth-SIG; Программы создания встраиваемых модулей системы Tina автоматизированного анализа сетей Петри и временных сетей Петри лаборатории LAAS, Тулуза, Франция.

Цель и задачи исследования. Целью исследования является создание эффективных методов анализа и синтеза моделей Петри телекоммуникационных систем. Для достижения поставленной цели необходимо решить следующие задачи:

- разработать методы синтеза сетей Петри по стандартным спецификациям протоколов; выполнить синтез сетей Петри и верификацию протоколов, от корректности которых зависит надёжность глобального и корпоративного информационного обмена, таких как TCP, BGP, IOTP;

- разработать методы обеспечивающие существенное ускорение процессов верификации телекоммуникационных протоколов с помощью функциональных сетей Петри;

- разработать основы теории функциональных сетей Петри и кланов систем линейных алгебраических уравнений;

- разработать методы синтеза моделей Петри телекоммуникационных систем на основе композиции моделей компонентов, а также методы измерения функциональных характеристик модели в процессе имитации её динамики;

- выполнить построение типовых моделей коммутируемых и маршрутизируемых сетей, сетей с коммутацией меток, мобильных сенсорных сетей, а также специальных компонентов, обеспечивающих измерение основных функциональных характеристик, таких как, например, время отклика и трафик.

Объектом исследования являются процессы функционирования телекоммуникационных систем, в особенности, телекоммуникационные протоколы, представляющие собой наборы правил функционирования телекоммуникационных систем.

Предмет исследования составляют формальные методы верификации телекоммуникационных протоколов на основе функциональных сетей Петри, методы анализа и синтеза телекоммуникационных систем, устройств, сетей, программного обеспечения, реализующих протоколы.

Методы исследования. Основными математическими методами исследования являются методы высшей алгебры и теории решёток, линейной алгебры и

теории чисел, теории графов. Кроме того, использованы стандартные методы теории сетей Петри, такие как методы фундаментального уравнения и линейных инвариантов, графов достижимых и покрывающих маркировок, декомпозиции и редукции. Для оценки эффективности протоколов и определения функциональных характеристик телекоммуникационных систем и сетей применены методы имитационного моделирования и математической статистики.

Научная новизна полученных результатов. Научная новизна полученных результатов состоит в том, что в работах соискателя впервые введены концепции функциональной сети Петри и клана линейной системы; построены основы теории функциональных сетей Петри и кланов линейных систем.

Получены следующие новые научные результаты:

- разработаны методы синтеза моделей телекоммуникационных протоколов, заданных спецификациями в форме взаимодействующих последовательных процессов Хоара; выполнен синтез модели протокола IOTP;

- разработаны методы верификации телекоммуникационных протоколов на основе композиционного анализа моделей Петри; выполнена верификация телекоммуникационных протоколов ECMA, Ethernet, BGP, TCP, IOTP с помощью композиционного анализа сетей Петри;

- выполнено построение и исследование моделей Петри коммутируемых, маршрутизируемых сетей, сетей с коммутацией меток и мобильных сенсорных сетей; разработан метод измерительных фрагментов для оценки функциональных характеристик моделей в процессе имитации динамики раскрашенных временных сетей Петри.

- доказано, что порождающее семейство (базис) функциональных подсетей составляет множество минимальных функциональных подсетей, а также, что сеть функциональных подсетей является маркированным графом;

- предложен универсальный метод декомпозиции на минимальные функциональные подсети с помощью логических уравнений, который может быть применён также к произвольным классам подсетей с контактными позициями;

предложен специальный метод линейной сложности для декомпозиции на минимальные функциональные подсети;

- получено представление передаточной функции временной сети Петри; разработаны методы эквивалентных преобразований и редукции сетей Петри для слабых типов эквивалентности на основе законов специально введенной для представления передаточной функции алгебры;

- разработаны методы синтеза функций непрерывной логики, заданных таблично;

- разработаны методы вычисления линейных инвариантов, решения фундаментального уравнения, определения ловушек и сифонов сетей Петри в процессе композиции функциональных подсетей, обеспечивающие существенные ускорения вычислений;

- выполнено обобщение методов композиционного анализа для произвольных линейных систем в кольцах со знаком, представленное в терминах кланов линейных систем; предложена последовательная организация процессов композиции кланов, изучены свойства последовательной композиции на основе стягивания подграфов и стягивания рёбер; задача последовательной композиции формализована в терминах теории графов и названа оптимальным коллапсом взвешенного графа, получены оценки верхней и нижней границ ширины коллапса;

- выполнено теоретическое обоснование и оценка вычислительной сложности метода Тудика решения линейных диофантовых систем в неотрицательных числах, традиционно применяемого для поиска инвариантов сетей Петри;

Практическое значение полученных результатов. Разработаны методы, модели и алгоритмы синтеза моделей Петри, верификации протоколов и оценки эффективности сложных телекоммуникационных систем и сетей.

Разработано программное обеспечение декомпозиции и композиционного анализа сетей Петри Deborah и Adriana. Программное обеспечение предназначено для работы с сетями Петри большой размерности, обеспечивает существенные ускорения вычислений и распространяется как встраиваемые модули к

известной системе Tina, используемой в качестве интерфейса для графического ввода и редактирования сетей Петри.

С помощью указанных модулей формально доказана корректность таких широко известных протоколов как Ethernet, ECMA, TCP, BGP, IOTP. Программное обеспечение может быть применено для верификации других протоколов, а также для исследования моделей Петри в других предметных областях.

Разработана методика построения моделей на основе композиции раскрашенных сетей Петри с контактными позициями, которая может быть применена для произвольных телекоммуникационных систем. Построены типовые модели компонентов и измерительных фрагментов для коммутируемых, маршрутизируемых сетей, сетей с коммутацией меток, мобильных сенсорных сетей, которые предназначены для автоматизированного синтеза моделей конкретных сетей и измерения их характеристик. Методика применена при проектировании магистральных сетей ОАО Укртелеком, а также при проектировании локальных сетей диспетчерских центров железной дороги, оснащённых программным обеспечением ГИД-Урал ВНИИЖТ.

Личный вклад соискателя. Основы теории функциональных сетей Петри, включающие методы и алгоритмы декомпозиции на функциональные подсети, композиционный анализ и его обобщения, теоретическое обоснование и оценку сложности метода Тудика, метод измерительных фрагментов, композиционный анализ протоколов ECMA, Ethernet, синтез моделей и композиционный анализ протоколов TCP, BGP являются личным результатом соискателя. Методы синтеза функций непрерывной логики, заданных таблично разработаны совместно с А.И.Слепцовым и В.Г.Сарбеем. Представление передаточной функции структурно-бесконфликтных сетей Петри получено совместно с А.И.Слепцовым. В построении моделей коммутируемых сетей, сетей с коммутацией меток, моделей протоколов Bluetooth и IOTP принимали участие Т.Р.Шмелёва, А.Л.Сакун, М.В.Березнюк, Е.Я.Чорногала.

Апробация результатов диссертации. Результаты работы представлены на семинаре по Дискретной математике, Одесса, Украина, 1995; 10-м семинаре

"Алгоритмы и программы для сетей Петри", Айхштадт, Германия, 2003; Европейской конференции по моделированию, Неаполь, Италия, 2003; 25-й международной конференции по приложениям и теории сетей Петри, Болонья, Италия, 2004; Международной конференции по кибернетике и информационным технологиям, системам и приложениям, Орландо, США, 2004; Международной научно-технической конференции "Искусственный интеллект. Интеллектуальные и многопроцессорные системы". Таганрог, 2004; 11-м семинаре "Алгоритмы и программы для сетей Петри", Падерборн, Германия, 2004; 12-м ежегодном симпозиума IEEE / ACM по Моделированию и анализу компьютерных и телекоммуникационных систем, Фолендам, Нидерланды, 2004; 5-м семинаре и школе по практическому использованию раскрашенных сетей Петри и CPN Tools, Орхус, Дания, 2004; специальных лекциях-семинарах в университете Париж-Дофин и в лаборатории LAAS, Тулуза, Франция, 2005; симпозиуме по проектированию, анализу и моделированию распределённых систем, Сан Диего, США, 2005; 10-й юбилейной конференции по Физике и технологии тонких плёнок, Яремче, Украина, 2005; международной конференции “Моделирование и компьютерная графика”, Донецк, Украина, 2005.

Публикации. Результаты опубликованы в одной монографии, 38 статьях в научных журналах и сборниках научных трудов, из них 29, указанных в перечне ВАК Украины, 9 – в материалах и тезисах конференций.

РАЗДЕЛ 1

ЗАДАЧИ МОДЕЛИРОВАНИЯ ПРОЦЕССОВ В ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМАХ И СЕТЯХ

Основные требования к современным телекоммуникационным системам могут быть кратко охарактеризованы двумя терминами: корректность и эффективность. Тогда множество задач моделирования можно классифицировать как задачи доказательства корректности (верификация) и задачи оценки эффективности. Стандартизация в телекоммуникациях, призванная обеспечить совместимость оборудования различных производителей и открытость технологических решений, повысила значимость телекоммуникационных протоколов, как исходных спецификаций технологий. В большинстве случаев изучают именно верификацию протоколов, хотя процесс проектирования телекоммуникационных систем и сетей также является источником ошибок и верификацию следует выполнять для каждой из спецификаций, полученных в процессе проектирования. Главными характеристиками эффективности телекоммуникационной системы либо сети являются пропускная способность и качество обслуживания, соотнесённые со стоимостью. Управляемое моделью проектирование, всё более успешно применяемое в телекоммуникациях, представляет собой процесс последовательного преобразования моделей протоколов в спецификации оборудования и программного обеспечения. Для построения качественных систем и сетей необходимо решать задачи верификации и оценки эффективности на каждом из этапов проектирования. Разнородность языков спецификаций, используемых в процессе проектирования, создаёт существенные трудности. Иерархия сетей Петри от элементарных до нагруженных (раскрашенных) позволяет применить их в качестве универсального языка спецификаций. Аппаратно-программная реализация процессоров сетей Петри завершает технологическую цепочку проектирования.

1.1. Верификация телекоммуникационных протоколов

Сложность телекоммуникационных систем в настоящее время в значительной степени обуславливается сложностью протоколов [84], которые они реализуют. Протокол представляет собой набор правил, в соответствии с которыми взаимодействуют системы. При разработке телекоммуникационных систем протокол реализуется аппаратно либо программно [170, 171, 174, 175]. Спецификации протоколов представленные в стандартах, которые являются наиболее критичной информацией при разработке систем, должны быть корректными и обеспечивать эффективное взаимодействие. Процесс разработки протоколов становится всё более динамичным, так, например, количество известных протоколов удваивается каждые пять лет. Кроме того, возрастает сложность самих протоколов, что косвенно подтверждается ростом объема их стандартных спецификаций [15]. Современный мир становится всё более зависимым от надёжности коммуникаций, что связано с увеличением количества передаваемой информации и повышением её значимости для общества. Электронная коммерция [12], технологическое управление [180] предъявляют высокие требования к надёжности коммуникаций. Таким образом, формальное доказательство корректности телекоммуникационных протоколов представляет собой важную научную задачу.

Ведущими организациями, обеспечивающих разработку и сопровождение протоколов в настоящее время являются: ISO (International Organization for Standardization – Международная организация по стандартизации, www.iso.org) выполняющая в основном методологические функции; IETF (Internet Engineering Task Force – Оперативная группа разработчиков Интернет, www.ietf.org), издающая RFC (References for comments – Справочники пояснений), специфицирующие протоколы применяемые в Интернет; IEEE (Institute of Electrical and Electronics Engineers – Институт инженеров электротехники и электроники, www.ieee.org), разрабатывающий в основном протоколы канального и транспортного уровней; ITU (International Telecommunication Union – Международ-

ный союз телекоммуникаций, www.itu.int) специфицирующий протоколы физического уровня. Кроме того, известен целый ряд специальных групп SIG (Special Interest Group – Специальная группа интересов), обеспечивающих разработку и сопровождение отдельных протоколов либо их семейств. В такие группы входят представители известных компаний производителей аппаратных средств и программного обеспечения для телекоммуникаций, а также представители университетов и исследовательских организаций. Например, SIG Bluetooth (www.bluetooth.org) занимается сопровождением протокола Bluetooth мобильных сенсорных сетей. Имеется также большое число протоколов, разработанных отдельными компаниями. Как правило, такие протоколы вначале используются только фирмами производителями, а затем приобретают статус мировых стандартов. Одной из наиболее известных фирм, выполняющей разработку собственных протоколов является фирма CISCO, а в качестве примера известного стандарта можно указать протокол TACACS.

ISO представляет собой универсальную организацию, занимающуюся стандартизацией в различных областях деятельности человека. Известен ряд конкретных телекоммуникационных протоколов, разработанных ISO, но они не нашли широкого практического применения. Наибольшую известность получила эталонная модель взаимодействия открытых систем OSI.

IETF разрабатывает и сопровождает протоколы, которые находят наиболее широкое практическое применение в современном мире. Перечень официальных протоколов IETF, представленный в RFC 3600 в 2003 году насчитывает 214 протоколов. IETF определяет язык спецификаций в RFC 825, RFC 1111 в документах Request for comments on Request for Comments: Instructions to RFC authors (Инструкции для авторов RFC). Документ содержит требование к формату текстовой информации, заголовкам, порядку описания. Процесс стандартизации описан в RFC 2026.

IEEE имеет в своей структуре специальное подразделение Standards Association (SA) – Ассоциация стандартов, которое занимается разработкой протоколов. Наиболее известными в настоящее время являются протоколы семейства

Ethernet IEEE 802.3, а также протоколы беспроводных сетей IEEE 802.11. Для разработчиков стандартов IEEE-SA предлагает такие документы как IEEE Standards Association Operations Manual (Руководство к действию) и IEEE Standards Style Manual (Руководство по стилю стандартов).

Сектор ITU, разрабатывающий стандарты телекоммуникационных протоколов, носит название Telecommunication Standardization (Стандартизация телекоммуникаций) и обозначается как ITU-T. Он выпускает документы под названием ITU-T Recommendations (Рекомендации). Для разработчиков стандартов предлагается документ Author's Guide for drafting ITU-T Recommendations (Руководство для авторов рекомендаций).

Рассмотрим язык стандартных спецификаций протоколов, описывающий исходную информацию для разработчиков телекоммуникационных систем. Как правило, это подмножество естественного английского языка, дополненное специальными правилами спецификации стандартов, обеспечивающими сравнительную точность формулировок. Кроме того, для дополнительных описаний применяют таблицы и диаграммы состояний, позволяющие уточнить вербальные спецификации. ISO стандартизовала [190] четыре основных примитива услуг: запрос, признак, ответ и подтверждение. Для спецификации предложено использовать временные диаграммы взаимодействия систем. Пример временной диаграммы с использованием основных примитивов представлен на рис. 1.1.

Известен также ряд специальных языков спецификации [84] протоколов. Они не нашли широкого применения в действующих стандартах, но используются при верификации протоколов. Исходная вербальная стандартная спецификация протокола преобразуется в спецификации на формальном языке. Затем выполняется верификация протокола математическими либо имитационными методами. Рассмотрим наиболее распространённые языки спецификации протоколов [159].

Язык Ina Jo [159] основан на расширенном исчислении предикатов первого порядка. С помощью аксиом системы Ina Jo описывается поведение системы,

а с помощью критериев – требования, которым должна удовлетворять система. Доказательство корректности протокола выполняется с помощью средств доказательства теорем в описанной аксиоматической теории.

Язык LOTOS [159] разработан для протоколов OSI и основан на алгебре процессов. Протокол специфицируется как множество процессов; процессы представлены последовательностями событий. Имеются средства, позволяющие специфицировать поведение злоумышленника. Известно применение LOTOS для верификации криптографических протоколов.

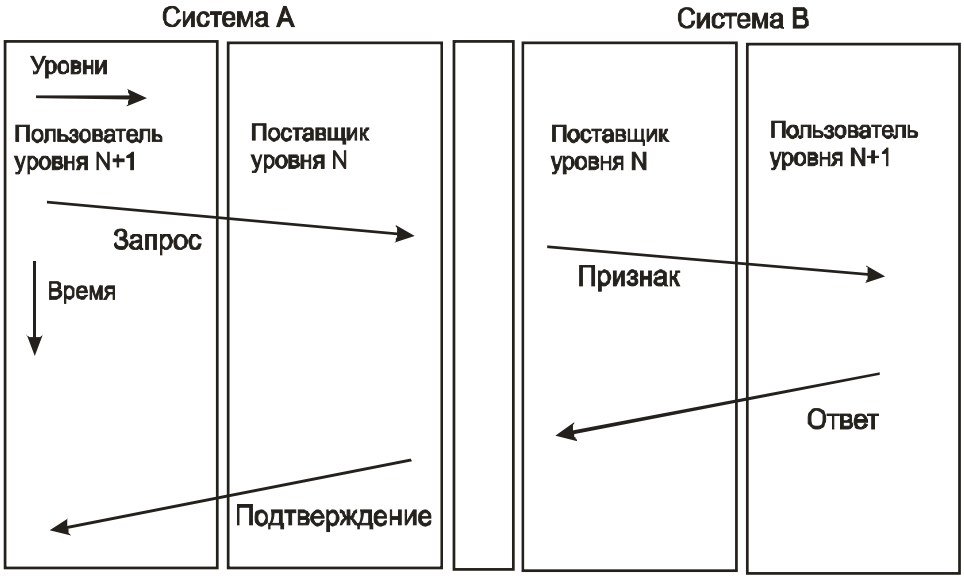


Рис. 1.1. Временная диаграмма основных примитивов протоколов

ASTRAL [159] представляет собой язык для спецификации систем реального времени и позволяет выполнить верификацию протоколов, чувствительным к временным задержкам. На его основе создан специальный анализатор моделей.

Целый ряд языков спецификации, таких как CAPSL, CPAL, NPATRL, предназначен для верификации криптографических протоколов и протоколов аутентификации [159]. Они содержат специальные средства описания криптографических алгоритмов.

Язык спецификаций SDL [124] рекомендован ITU и основан на концепции взаимодействующих конечных автоматов. Существует графическая версия языка, позволяющая представить спецификации в наглядной форме. Набор графических элементов насчитывает более 20 блоков. Известно также применение универсального языка спецификаций программ UML [114] для верификации протоколов.

Язык Estelle, [64] рекомендованный ISO, основан на спецификациях в форме конечных автоматов. Язык HLPSL [91] содержит средства описания ролей, переходов и окружения для спецификации протоколов, а также средства описания целей верификации. Основой языка является Лампортова темпоральная логика действий. Система имеет графический интерфейс; известны примеры её применения такими компаниями, как SIEMENS, SAP. Известен также язык EPSL [21], основанный на специальной форме линейных темпоральных логик.

Разнообразие языков спецификации протоколов во многом обуславливается отличием в синтаксисе, а также особенностями реализации алгоритмов доказательства корректности. Общим является стремление использовать математический формализм в основе спецификаций, позволяющий выполнить дальнейшую верификацию протокола формальными методами. Рассмотрим основные математические системы, применяемые в языках спецификации протоколов.

Конечные автоматы [122] являются наиболее простым формализмом, используемым для верификации протоколов. Теория конечных автоматов позволяет эффективно реализовать доказательство корректности модели. Однако формализм конечных автоматов является недостаточно полным для спецификации протоколов. Известен ряд расширений конечных автоматов, таких как временные автоматы [3], взаимодействующие конечные автоматы, автоматы с вводом и выводом [159] и другие, широко используемых в языках спецификаций.

Исчисления предикатов [4] позволяет аксиоматически специфицировать поведение системы, хотя представление взаимодействия является достаточно сложным. Для верификации протоколов используют системы автоматического доказательства теорем, такие как Prolog [4].

Темпоральные, модальные [159] и другие логики содержат средства описания причинно-следственных связей между событиями, поведения системы во времени; предикаты модальной логики позволяют выразить степень достоверности (доверия). В [159] отмечается слишком высокий абстрактный уровень модальных логик и преимущества подходов, основанных на анализе пространства состояний моделей (model-checking).

Модели исчисления процессов введены и исследованы Хоаром [189]. Разработана теория взаимодействующих последовательных процессов (ВПП), известная в англоязычной нотации как CSP. Модель основана на представлении процессов формулами, элементами которой являются события, а множество операций состоит из операций следования, итерации, альтернативы и параллельной композиции. Хоаром введен также ряд вспомогательных операций, обеспечивающих удобство доказательства теорем.

Перечисленные математические системы обладают определёнными недостатками, затрудняющими их применение для решения задач верификации протоколов. Конечные автоматы имеют ограниченную изобразительную мощность, а большинство систем, основанных на исчислениях, не гарантируют завершение доказательства корректности для произвольной системы аксиом. Проблема выбора приемлемого формализма состоит в обеспечении баланса изобразительной мощности и множества разрешимых задач. Выбор универсальной алгоритмической системы существенно сужает множество разрешимых задач, делая формальную верификацию практически неосуществимой.

Классические сети Петри [173] представляют собой пример достаточно хорошего баланса изобразительной мощности и множества разрешимых задач. Многие исследователи отмечают их удобство для спецификации протоколов

[9,29,39,160]. Сеть Петри позволяет описывать последовательные, конвейерные, параллельные и альтернативные процессы, а также сложные ресурсные отношения, содержит средства представления взаимодействия и синхронизации асинхронных априори процессов. Особенности верификации протоколов с помощью моделей Петри рассмотрены в подразделе 1.4.

1.2. Оценка эффективности телекоммуникационных систем

Эффективность телекоммуникационных систем в большинстве случаев оценивается с помощью характеристик производительности и пропускной способности. Однако современные мультимедиа протоколы и приложения реального времени существенно повышают значение характеристик качества обслуживания, которые, в ряде случаев выступают как основные параметры системы.

Аналитические методы важны для исследования эффективности, так как позволяют представить результат в виде формулы, удобной для выполнения расчётов конкретных систем. В большинстве случаев аналитический результат получен на основе существенного упрощения поведения системы, что позволяет применить его лишь для предварительных грубых оценок.

Известны фундаментальные оценки ширины пропускной способности канала без шумов по Найквисту, а также для канала с шумами по Шенону [182]. Для определения интенсивности трафика известна фундаментальная оценка [182] в безразмерных единицах – эрлангах:

$$A = \lambda h,$$

где λ – средняя частота поступления запросов, T – средняя продолжительность обслуживания.

Однако представленные соотношения не учитывают многие особенности функционирования реальных телекоммуникационных систем. В более сложных

оценках применяются методы теории очередей, именуемой также теорией массового обслуживания, либо теорией телетрафика в применении к задачам расчёта сетей связи [118,190]. Как правило, методы теории телетрафика применяются в расчётах систем с коммутацией сообщений (пакетов). Простейшая модель обслуживания представлена однолинейной системой (рис. 1.2).

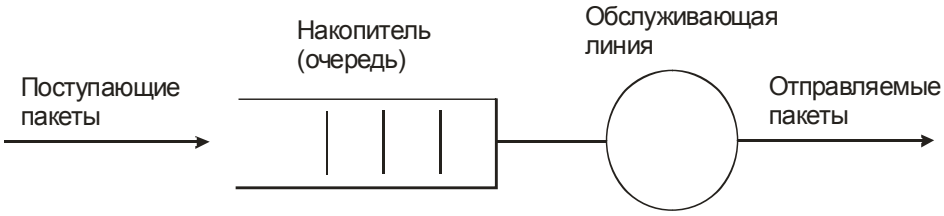


Рис. 1.2. Модель однолинейной системы обслуживания

Пакеты поступают случайным образом со средней скоростью λ пакетов в единицу времени, Они ожидают обслуживания в накопителе и обслуживаются в соответствии с некоторой конкретной дисциплиной со средней скоростью μ пакетов в единицу времени. Обслуживающая линия может представлять канал передачи пакетов, либо некоторое устройство, например, сервер. В качестве основных характеристик системы рассматриваются интенсивность загрузки либо коэффициент использования канала $\rho = \lambda / \mu$, пропускная способность, время задержки, время ожидания в очереди и другие.

Для аналитического расчёта характеристик детализируются следующие параметры системы: процесс поступления пакетов; распределение времени обслуживания; дисциплина обслуживания. Для многолинейных систем вводятся дополнительные параметры, которые в простейшем случае могут быть представлены числом обслуживающих линей. Стандартное представление СМО содержит обозначения перечисленных параметров. Например, M/M/1 является обозначением системы с пуассоновским входным потоком, экспоненциальным распределением времени обслуживания и одной линией обслуживания; буква “M” обозначает Марковский процесс.

Наиболее простым для аналитических расчётов является пуассоновский процесс, который во многих случаях является адекватным описанием реальных потоков запросов. Распределение Пуассона представлено формулой:

$$p(k) = (\lambda T)^k e^{-\lambda T} / k!,$$

где $p(k)$ – вероятность того, что на промежутке времени T поступят k запросов. Среднее значение числа запросов $E(k)$ имеет вид:

$$E(k) = \lambda T .$$

Величина τ, равная интервалу времени между поступлением соседних запросов распределена по экспоненциальному закону и представлена следующей плотностью распределения:

$$f_\tau(\tau) = \lambda e^{-\lambda \tau} .$$

Для системы M/M/1 получены следующие аналитические результаты [190]. Вероятности p_n того, что в системе находится n клиентов в стационарном режиме ($\rho < 1$) описывается формулой:

$$p_n = (1-\rho)\rho^n .$$

Вероятность блокировки (отказа) P_B зависит от ограничений на размер очереди и оценивается выражением:

$$P_B = p_N = (1-\rho)\rho^N /(1-\rho^{N-1}),$$

где N – ограничение на длину очереди. Пропускная способность системы равна

$$\gamma = \mu(1 - p_0).$$

Среднее время задержки в системе $E(T)$ оценивается как:

$$E(T) = 1/\mu(1 - \rho).$$

Одним из наиболее широко применяемых результатов является формула Литтла:

$$L = \lambda W,$$

где L – среднее число пакетов в системе, W – среднее время ожидания.

Аналитические результаты известны также для систем M/M/m, D/D/m, где D обозначает равномерное распределение; для M/G/1, где G обозначает распределение общего вида, известны результаты расчёта средних времён; для систем с приоритетами и других. Часто в обозначение СМО добавляют также ограничения на длину очереди, например: M/M/n/m.

При проектировании сложного оборудования, состоящего из множества взаимодействующих узлов, а также телекоммуникационных сетей применяют теорию сетей систем массового обслуживания [118,190]. В сети систем массового обслуживания каждый узел представляет собой СМО, а взаимосвязи узлов представляют в виде графа либо задают маршрутной матрицей. Различают однородные и неоднородные сети СМО; в однородной сети каждый узел представлен СМО одного типа. Кроме того, исследуют замкнутые и разомкнутые сети. Пример разомкнутой сети СМО представлен на рис. 1.3.

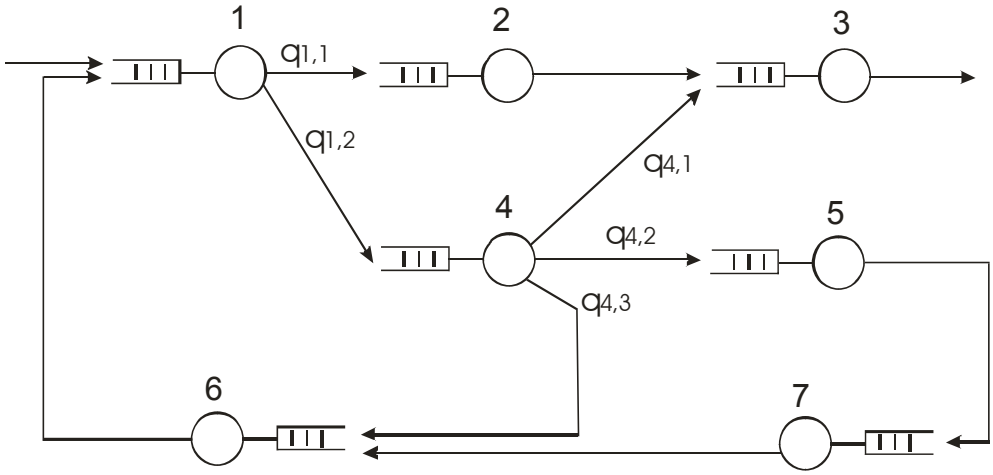


Рис. 1.3. Пример разомкнутой сети СМО

Наиболее простыми моделями являются разомкнутые цепи СМО и замкнутые циклические цепи. Для их исследования используется теорема Нортона, позволяющая заменить линейную последовательность СМО одной системой. В [190] исследована модель управления с помощью скользящего окна в архитектуре SNA, представленная в виде простого цикла СМО. Найдена производительность системы:

$$u(n) = n\mu/(n+(M-1)),$$

где M – количество СМО в цикле, n – количество пакетов в сети.

Модели, в которых потоки пакетов пуассоновские, времена обслуживания в узлах распределены по экспоненциальному закону, а маршруты от одной системы к другой выбираются случайно с фиксированной вероятностью $q_{i,j}$, называют сетями Джексона. Для сетей Джексона получены аналитические решения в форме произведений. Например, для разомкнутой системы:

$$p(\overline{n}) = \prod_{i=1}^{M} p_i(n_i),\ p_i(n_i) = (1-\rho_i)\rho_i^{n_i},\ \rho_i = \lambda_i/\mu_i < 1.$$

Средняя задержка на обслуживание в сети оценивается как:

$$E(T) = \frac{1}{\gamma} \sum_{i=1}^{M} \frac{\lambda_i}{\mu_i - \lambda_i}.$$

Для расчёта более сложных сетей систем массового обслуживания применяется метод анализа средних [190] и ряд приближённых методов расчёта, например диффузионная и декомпозиционная аппроксимации [118]. Сети и системы массового обслуживания широко применяются для анализа и синтеза телекоммуникационных систем [118].

В тех случаях, когда непосредственное применение теории очередей затруднено, выполняют построение специальных моделей для решения конкретных задач в форме обобщённых стохастических либо Марковских процессов. В [118] изучено решение задач расчёта систем с пакетной коммутацией: определения межконцевых задержек, оптимизации пропускной способности, управления потоками, анализа необходимой буферной памяти и другие. Применение немарковских стохастических процессов для исследования телекоммуникационных систем изучено в [45]; получено аналитическое представление функции распределения для стационарного режима. В [1] предложена специальная аналитическая модель для оценки объёма ресурсов кэширующего сервера в пиринговых системах. Модель для оптимизации сети с двумя маршрутизаторами построена в работе [41]. Гетерезисная модель Web-сервиса в TCP сети построена в работе [60].

Недостатком классической теории очередей является элементарность пакета, в то время как в реальных телекоммуникационных сетях заголовок пакета содержит информацию, существенно влияющую на процессы его обработки. Попытки различать приоритеты пакетов, либо маршруты следования приводят, как правило, к громоздким результатам, а исследование более тонких механизмов взаимодействия пакета с сетевой средой приводит к необходимости применения имитационного моделирования.

Задачи выбора оптимальных маршрутов и топологии сети, в большинстве случаев, решают с помощью методов теории графов. Для выбора кратчайшего маршрута применяют алгоритм Дейкстры [184,188], для оптимизации пропускной способности при выборе топологии используют метод Форда-Фалкерсона [186], при проектировании линий связи применяют методы построения остова графа минимального веса либо покрытия множеством циклов [5,184] для резервирования линий связи. Однако сложные критерии оптимизации в реальных задачах маршрутизации, учитывающие множество характеристик линий связи и узлов затрудняют применение классических методов. Проектирование топологии, а также решение задач маршрутизации для современных сетей также приводят к необходимости применения имитационных методов, позволяющих отобразить сложные процессы взаимодействия полей заголовков пакетов с устройствами, предусмотренные протоколами.

Следует отметить, что классические методы теории очередей предполагают потоковый трафик, и не позволяют отобразить многошаговое взаимодействие систем в соответствии с протоколами. Таким образом, априори для оценки производительности, сложный трафик, являющийся результатом взаимодействия систем, трансформируется в некоторое его потоковое приближение. Точное описание трафика, исходя из действующих протоколов, является важной задачей моделирования процессов в телекоммуникационных системах.

Современные мультимедиа протоколы и приложения реального времени существенно повышают значение характеристик качества обслуживания, которые, в ряде случаев выступают как основные параметры системы. Характеристикой качества обслуживания мультимедийного трафика является гарантированное время доставки пакета, которое может быть также представлено как гарантированная задержка между доставкой соседних пакетов. Для приложений реального времени качество обслуживания определяется гарантированным временем отклика сети. Задачи аналитической оценки качества обслуживания возможны лишь для простейших систем [1,60]. Получение оценок для реальных систем возможно лишь с помощью методов имитационного моделирования.

1.3. Имитационное моделирование телекоммуникационных систем

Первой промышленной системой имитационного моделирования общего назначения считают GPSS (General Purpose Simulation System) [192]. В системе применена модель дискретных событий. Язык GPSS представляет собой текстовый язык программирования, дополненный средствами описания устройств и транзактов, а также маршрутов перемещения транзактов внутри модели, их создания, поглощения, размножения и синхронизации. Транзакт является динамическим элементом, перемещаемым внутри модели, и описывается массивом целых чисел, позволяющих различать типы транзактов и их атрибуты. Устройства представляют собой массив элементов, для которых выполняется накопление статистической информации. Язык содержит операторы занятия (seize) и освобождения (release) устройств транзактами. Оператор задержки (delay) позволяет моделировать время обслуживания; предоставляется широкий набор случайных функций с различными распределениями для задания параметров модели. Имеется возможность одновременного занятия нескольких устройств, что обеспечивает существенное расширение модели по сравнению с сетью систем массового обслуживания.

Моделирование дискретных событий требует специальных процедур управления модельным временем. Списки транзактов организуются в виде очереди событий и упорядочиваются в соответствии с временами их активизации. Запускаются все события для текущего момента модельного времени. Затем вычисляется следующий момент модельного времени как минимальный из всех времён будущих событий. Перечисленные действия представляют собой основной цикл работы моделирующей системы, изображённый на рис. 1.4.

Статистическая информация, накапливаемая моделирующей системой, содержит коэффициенты загрузки устройств, средние времена обслуживания, позволяющие затем вычислить характеристики моделируемой системы специфические для области применения.

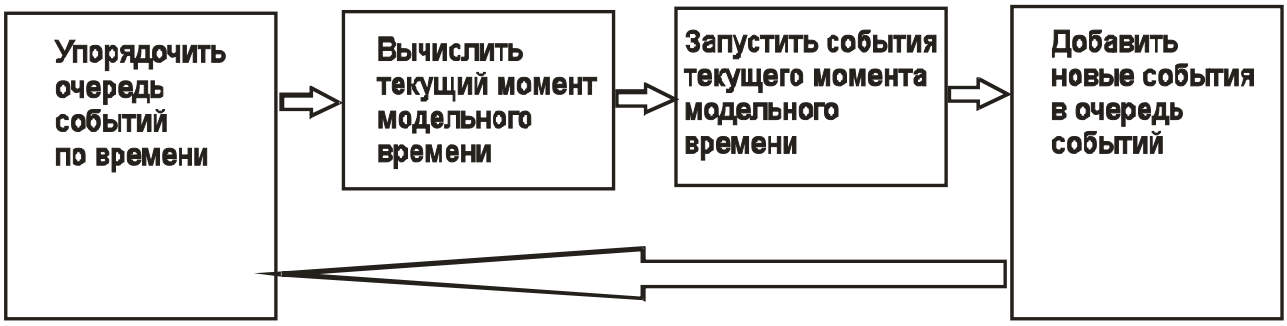


Рис. 1.4. Основной цикл работы системы имитационного моделирования

Распространённым является подход к построению систем имитационного моделирования как расширений некоторого алгоритмического языка. Одной из первых таких систем является Simula [185]. В настоящее время также широко применяется подход объектно-ориентированного моделирования [185], в котором моделирующая система является расширением некоторого объектно-ориентированного языка программирования. Известны примеры построения моделей широковещания (multicasting) в мобильных сотовых сетях при исследовании алгоритмов маршрутизации [17] с помощью систем объектно-ориентированного программирования Java2 SDK, ObjectSpace JGL для оценки среднего времени передачи и частоты широковещательных сообщений. Выбор языка программирования в качестве среды, в которую встраиваются библиотеки, реализующие процесс имитации дискретных событий, является наиболее перспективным. Современные языки программирования предоставляют широкие возможности для описания специфики функционирования телекоммуникационного оборудования с помощью абстрактных типов данных и классов объектов. Дополнительные библиотеки обеспечивают ведение списков событий, вычисление и продвижение модельного времени, запуск модулей, реализующих события. Кроме того, создаются проблемно-ориентированные библиотеки, содержащие классы, описывающие определённую сетевую технологию.

Для построения имитационных моделей телекоммуникационных систем и сетей разработана специализированная система имитационного моделирования NS (Network Simulator) [87], представляющая собой расширение объектно-

ориентированных языков программирования C++, OTcl. NS (NS2) является наиболее популярной системой имитационного моделирования телекоммуникационных систем и сетей. Известны примеры применения системы NS для оценки влияния процессов переупорядочения пакетов в TCP/IP сетях [70]; построения крупномасштабных моделей магистральной маршрутизации на основе BGP [32] для балансировки загрузки маршрутизаторов и минимизации передаваемой маршрутной информации; для проверки аналитических моделей при исследовании переполнений в TCP сетях [31]; анализа эффективности мобильных сетей (MANET) на основе технологии IEEE 802.11e, оценки времени доставки пакета и пропускной способности [16]; анализа производительности транспортных протоколов в геостационарных спутниковых сетях с оценкой времени установления связи и загрузки линий [20]; моделирования маршрутизации в сетях HighSpeedTCP с использованием TailDrop/RED магистральных маршрутизаторов для оценки пропускной способности сети [106] и другие. Многие исследователи выполняют расширение системы NS, добавляя собственные модули либо библиотеки, предназначенные для моделирования различных технологий. Так для моделирования геостационарных спутниковых сетей [20] разработана специализированная библиотека классов. При исследовании влияния переупорядочения пакетов [70] разработано расширение системы NS, названное Hiccup.

Таким образом, система NS становится ядром имитационного моделирования телекоммуникационных систем и сетей. Процесс её расширения специализированными модулями становится всё более динамичным. Однако, при моделировании гетерогенных систем и стыковке различных расширений возникают трудности, поскольку каждый исследователь разрабатывает собственные форматы классов объектов. Система обеспечивает преемственность базовых классов, но расширения зачастую приводят к разнотипным моделям одной и той же технологии. В настоящий момент назрела необходимость стандартизации расширений системы и управления процессом создания новых библиотек, обеспечивающих совместимость моделей.

На втором месте по использованию в проектах находится моделирующая система GTNeS (Georgia Tech Network Simulator) [81]. В [105] описано применение GTNeS для исследования режимов энергосбережения в мобильных сенсорных сетях Bluetooth. Оценивается среднее количество узлов, находящихся в состоянии Hold, низкого энергопотребления. Система обеспечивает эффективную работу с крупномасштабными моделями. В [81] описано применение GTNeS для изучения поведения Интернет-червей на крупномасштабных моделях ядра Интернет, насчитывающих более 50000 узлов. Модель содержит детализированный сценарий поведения червя, представленный фазами распространения, активации, инфицирования. Вектор характеристик червя позволяет представить поведение множества известных вирусов, например Slammer, Code Red II и других. Модель описывает ядро Интернет, представленное маршрутизаторами и прокси-серверами. Оценивается процесс распространения червя во времени (количество инфицированных узлов), дополнительная нагрузка на сеть, общий ущерб и интервал времени до полного блокирования сети.

Моделирующая система OPNET успешно применена во многих проектах [59]. В [104] построена модель для исследования алгоритмов управления очередями в маршрутизаторах Cisco в сетях DiffSer, обеспечивающих заданное качество обслуживания (QoS). Оцениваются время доставки пакета и пропускная способность в сетях ориентированных на соединение (connection-oriented). В [59] OPNET применена для исследования алгоритмов маршрутизации в мобильных сетях. Оценивается эффективность NDMR (Node-Disjoint Multipath Routing) маршрутизации с использованием множественных путей через несмежные узлы. Определяется среднее время доставки пакета, загрузка маршрутизаторов.

Количество реально используемых в проектах специализированных моделирующих систем насчитывает несколько десятков. В [40] представлен обзор, в котором сравниваются такие моделирующие системы, как NISTNET, DummyNet, ModelNet, Ohio Network Emulator, ENDE, Emulab, EMPOWER, NSE, Vint/NS, NETWARS. Обсуждается выбор моделирующей системы для исследо-

вания систем клиент-сервер с целью обеспечения заданного времени отклика сети. Описываются принципы построения модели, состоящей из подмоделей клиента, сервера и трафика. Обсуждаются преимущества использования моделирующей системы GTNeS.

Рассмотрим примеры применения специализированных систем имитационного моделирования в телекоммуникационных проектах. Системы ASPEN, SoftSDV применены для исследования кэш-памяти в сетях SNA [69]. Системы представляют собой пример подхода полного моделирования (full-system) исполнения программного обеспечения в среде конкретных операционных систем EDS (Execution Driven Simulation). Системы SimplePipe, SimpleScalar использованы для анализа производительности в приложениях с сетевыми процессорами (Network Processors) [36] и представляют собой набор модулей, транслируемых в язык Си. Для оценки гарантированных задержек в сетях Ethetnet/IP в распределённых производственных системах применена OMNeT++ [40]. Для скоростного моделирования маршрутизации в сетях масштаба Интернет применена ModelNet [18]; исследуется эффективность построения покрывающих деревьев (spanning-tree) различными алгоритмами маршрутизации. Для оценки энергосбережения в мобильных сетях применена система Odyssey [35]. Для моделирования различных режимов энергопотребления в мобильных сетях IEEE 802.11 применены GloMoSim, QualNet [65]. Для моделирования мобильных сетей Bluetooth применяют Bluehoc, Blueware [105]. Для моделирования процессов распространения Интернет-червей использована SSFNet [81]. Для исследования широковещательных колец протоколов VRing, ALM, NICE использован J-Sim [85]; выполнена сравнительная оценка протоколов, оценено время доставки данных и количество служебной информации. Для моделирования кластеров Web-серверов применён QNAPL [38]. Для моделирования систем сетевой памяти SAN и NAS использован ParIOSim [92]; оценено время отклика сети.

Для классификации систем имитационного моделирования используют такие признаки как: формальная модель процессов; степень реалистичности

описания объектов; графический либо текстовый язык описания модели; автономная либо встраиваемая в среду языка программирования и другие. Модель дискретных событий DEVS используется в большинстве систем, хотя в последнее время широкое распространение находят модели мобильных агентов DynDEVS. По степени реалистичности различают модели с частичным и полным (full-system) реализмом. Применение графического языка является предпочтительным для использования в автоматизированном проектировании. Автономные моделирующие системы являются дорогостоящими, поэтому большинство систем реализовано как расширения известных языков программирования. Важным признаком является наличие подсистемы накопления и обработки статистической информации, а также её графического представления.

Следует отметить, что применение методов имитационного моделирования позволяет выполнить оценку эффективности телекоммуникационной системы, в то время как задача верификации протоколов требует формального доказательства корректности поведения систем.

Таким образом, в области имитационного моделирования телекоммуникационных систем и сетей наметилась необходимость стандартизации языка описания моделей, обеспечивающего переносимость моделей и их совместимость при моделировании гетерогенных сред. Язык должен быть достаточно мощным, графическим, содержать небольшое число элементов и обеспечивать масштабируемость. В качестве такого языка в диссертационной работе предлагается использовать раскрашенные сети Петри. Раскрашенные сети Петри представляют как формализм дискретных процессов, так и формализм мобильных агентов, имеют три базовых графических элемента (позиция, переход, дуга) и содержат язык программирования для описания атрибутов, а также средства обеспечения масштабируемости с помощью подстановки переходов.

1.4. Моделирование телекоммуникационных систем сетями Петри

Применение сетей Петри в качестве моделей телекоммуникационных систем позволяет решить как задачи доказательства их корректности, так и оценки эффективности. Классические, либо низкоуровневые (low-level) сети Петри [160,169,173] позволяют выполнить анализ модели формальными методами, такими как методы дерева достижимых и покрывающих маркировок, алгебраическими методами, основанными на фундаментальном уравнении сети и инвариантах. В ряде случаев рассматривают подклассы низкоуровневых сетей, например, сети со свободным выбором [27], структурно-бесконфликтные сети [169]. Для подклассов сетей Петри известны эффективные алгоритмы решения проблем достижимости маркировки и живости [27]. Простые расширения, такие как ингибиторные, приоритетные, временные сети, являются универсальной алгоритмической системой [169,173], что позволяет применить их для анализа произвольных систем. Однако такие сети громоздки и неудобны для спецификации реальных систем. Высокоуровневые (раскрашенные) сети Петри (hi-level) [50,180], представляющие собой, по существу, иерархические временные стохастические сети, являются удобным инструментом моделирования телекоммуникационных систем. Они позволяют применить методы имитационного моделирования для оценки эффективности телекоммуникационных систем и сетей.

В диссертационной работе предложено использовать сети Петри как универсальный язык спецификации телекоммуникационных систем с целью доказательства их корректности формальными методами и оценки эффективности методами имитационного моделирования.

1.4.1. Верификация протоколов с помощью моделей Петри

Сети Петри [169] успешно применяют для исследования систем и процессов в различных прикладных областях [24, 30, 74, 116, 180]. Верификация протоколов является традиционной областью применения сетей Петри [9, 29]. При-

менение сетей Петри для исследования протоколов впервые было осуществлено М.Диазом [29] и Ж.Берселотом [9] в 1982-м году. В их работах были сформулированы свойства моделей идеальных протоколов. В Советском Союзе основателем научного направления, разрабатывавшего специальные классы сетей Петри для верификации протоколов, выступил В.Е.Котов, возглавивший затем лабораторию параллельных вычислений фирмы HP. Методы проектирования систем управления, в том числе для компьютерных сетей и телекоммуникационных устройств на основе нагруженных сетей Петри разработаны А.И.Слепцовым и представлены в его монографии [180] в 1986 году. Развитием теории регулярных сетей В.Е.Котова [160] выступили работы Н.А.Анисимова [108], посвящённые композиции моделей Петри протоколов с помощью специального множества алгебраических операций. Эти работы по существу завершают множество публикаций посвященных формальным методам верификации протоколов, поскольку исследователи столкнулись с проблемой размерности сетей Петри, сформулированной в работах А.Марсана [66] при исследовании протоколов Ethernet. Детализированная модель реального протокола имеет, как правило, размерность, не позволяющую выполнить её анализ за приемлемое время. Не смотря на многочисленные эвристики, предложенные для решения линейных диофантовых систем в целых неотрицательных числах М.Силвой и Дж.Коломом [22], для повышения эффективности метода Тудика [88], существенные ускорения вычислений не были получены.

При исследовании протоколов решают две основные задачи: доказательство корректности протокола и оценку его эффективности. Первую из задач называют ещё верификацией протокола. Она действительно является приоритетной, поскольку наличие дефектов в исходных спецификациях представляет собой один из наиболее дорогостоящих типов ошибок. Так как, если некорректный протокол будет реализован программно либо аппаратно, то затраты на исправление ошибок могут быть весьма значительными.

Показано [9,29,39], что модель идеального (корректного) телекоммуникационного протокола должна быть ограниченной, консервативной и живой се-

тью Петри. Задача определения ограниченности может быть решена с помощью классического метода, основанного на построении и анализе дерева покрывающих маркировок [160,173]. Структурная консервативность может быть установлена с помощью инвариантов позиций [169] сети Петри. Живость сети Петри является одной из трудноразрешимых задач, эквивалентных задаче достижимости маркировки [173]. Традиционно для исследования живости используют инварианты переходов сети Петри [169]. Таким образом, инвариантный анализ с помощью решения систем линейных диофантовых уравнений в целых неотрицательных числах является перспективным направлением исследования свойств сетей Петри, необходимых для доказательства корректности протоколов.

Детализированные модели реальных телекоммуникационных протоколов, представленные сетями Петри и построенные по исходным спецификациям, насчитывают, как правило, тысячи элементов. Кроме того, трудоёмкость верификации всё более обуславливается трудоёмкостью построения моделей большой размерности. Возникает задача синтеза модели Петри по стандартным спецификациям протоколов. В разделе 4 диссертационной работы представлены методы синтеза моделей Петри [129] с использование промежуточного языка взаимодействующих последовательных процессов Хоара (ВПП) [189]. В большинстве случаев, построение формулы ВПП по исходным стандартным спецификациям менее трудоемко, так как последовательные вербальные описания естественным образом трансформируются в формулы ВПП. В качестве вспомогательной задачи рассмотрен синтез сети Петри, заданной конечным автоматом [26,28,33]; для её решения предложены новые методы, основанные на решении линейных систем уравнений и неравенств. Особенности применения методов синтеза моделей протоколов проиллюстрированы на примере синтеза модели Петри известного протокола электронной коммерции IOTP. Верификация протокола [125] выполнена с помощью композиционного анализа сетей Петри [133], что позволило получить значительные ускорения вычислений при нахождении инвариантов.

Модели Петри сложных телекоммуникационных протоколов имеют большую размерность, в то время как основные методы анализа свойств сетей Петри имеют экспоненциальную вычислительную сложность, что обуславливает практическую неосуществимость исследования реальных систем. Таким образом, возникает задача, состоящая в разработке эффективных методов анализа моделей Петри большой размерности. Методы композиции функциональных подсетей (кланов), разработанные в разделах 2, 3 позволяют решить указанную проблему. В разделе 4 композиционный анализ применен для верификации известных телекоммуникационных протоколов, таких как BGP, TCP, IOTP [94, 96, 125, 130, 131, 141, 147, 150].

Типичным примером моделей телекоммуникационных сетей является структура некоторой регулярной топологии [66]. При этом количество взаимодействующих узлов может быть произвольно. Соответствующие модели Петри являются бесконечными, что делает невозможным их анализ традиционными методами. Для исследования бесконечных моделей регулярной структуры в разделе 4 представлен метод композиционного вычисления инвариантов в параметрической форме, который позволил доказать инвариантность ранее известной модели Ethernet с топологией общей шины для произвольного числа подключенных рабочих станций [151].

1.4.2. Раскрашенные сети Петри как средство имитационного моделирования телекоммуникационных систем

Раскрашенные сети Петри, изученные в монографиях К.Йенсена в 1997 году [50], по существу представляют собой подмножество нагруженных сетей, исследованных А.И.Слепцовым [180]. Практический успех применения раскрашенных сетей, прежде всего, обусловлен разработкой и свободным распространением промышленных моделирующих систем, таких как Design/CPN [2] и CPN Tools [6], выполненными при поддержке фирм Metasoft и Microsoft. К настоящему времени эти системы применены в десятках реальных проектов, более трети которых связаны с телекоммуникациями. Показательно, что в на-

стоящее время система CPN Tools применяется как основная моделирующая система для управляемого моделью проектирования новых поколений мобильных телефонов фирмы Nokia [68]. Научное направление, осуществляющее применение раскрашенных сетей Петри для моделирования телекоммуникационных протоколов основано Дж.Биллингтоном. Термином "раскрашенные сети Петри" в соответствии с трёхтомной монографией К.Йенсена [50] в настоящее время характеризуют временные, нагруженные, иерархические сети Петри. Раскрашенные сети являются универсальной алгоритмической системой и могут быть использованы для спецификации произвольных систем и процессов.

CPN Tools представляет собой интегрированную среду для графического ввода моделей, имитации их динамики и определения основных свойств. Кроме средств рисования сетей система содержит встроенный язык программирования ML предназначенный для описания атрибутов графических объектов. Следует отметить, что в проекте CPN Tools применена новая технология работы с графическими объектами, базирующаяся на стандартах Open GL, позволяющая создавать крупномасштабные модели в короткие сроки. Обеспечена совместимость систем: модели, ранее построенные в среде Design/CPN, могут быть автоматически конвертированы в CPN Tools.

Традиционное применение CPN Tools направлено на генерацию и анализ пространства состояний модели. Этот подход применён Дж. Биллингтоном для верификации ряда телекоммуникационных протоколов [72]. Для раскрашенных сетей единственным известным формальным методом исследования является тривиальная генерация полного пространства состояний, что практически осуществимо лишь для моделей малой размерности. Соискателем предложено использовать CPN Tools как классическую систему имитационного моделирования. Имитируется поведение моделей, описывающих стохастические процессы функционирования телекоммуникационных систем и сетей, на продолжительных интервалах модельного времени, гарантирующих наблюдение стационарного режима. Накапливается статистическая информация, которая затем явля-

ется основой для вычисления функциональных характеристик, таких как время отклика сети, трафик и других.

В разделе 5 представлена методика построения моделей телекоммуникационных систем и сетей в среде CPN Tools в процессе композиции подмоделей их компонентов, являющихся либо функциональными подсетями, либо более общим классом подсетей с контактными позициями [95, 100, 101, 127, 128, 135-137, 140, 146]. При моделировании телекоммуникационной системы наряду с основной задачей обеспечения адекватности исследуемому объекту возникает задача измерения характеристик модели. Встроенные средства CPN Tools позволяют построить пространство состояний модели и определить такие свойства сети Петри как ограниченность, безопасность, отсутствие тупиков, живость [6, 50]. Следует отметить, что модели реальных систем имеют количество состояний, измеряемое сотнями тысяч, что делает анализ пространства состояний практически неосуществимым. Кроме того, практическую значимость имеют нетривиальные характеристики модели, такие, например, как время отклика сети, количество коллизий, вероятности перегрузок. Имитация динамики модели и визуальное наблюдение позволяют выполнить лишь отдельные измерения в отладочных целях. Таким образом, возникает задача, состоящая в организации измерения характеристик моделей Петри телекоммуникационных систем.

Расширенные сети Петри представляют собой универсальную алгоритмическую систему [180, 181]. При проектировании систем автоматизированного управления А.И.Слепцовым было предложено представлять в форме сети Петри как управляемую систему, так и алгоритм управления [180]. Были показаны способы реализации с помощью расширенных сетей Петри основных логических и арифметических операций [181]. Эта методология была, затем, обобщена для временных сетей Петри, множество базисных операций которых включает специальную операцию временной задержки [154]. В разделе 5 представлен метод измерения характеристик моделей Петри телекоммуникационных систем с помощью специальных измерительных фрагментов сетей Петри [95, 140]. Такой подход позволяет организовать измерение нетривиальных ха-

рактеристик модели. Детальное изучение принципов построения измерительных фрагментов выполнено на примере измерения времени отклика в модели коммутируемой локальной сети.

Перспективной технологией разработки сложных телекоммуникационных систем является, так называемое управляемое моделью проектирование [7,68]. Выбор формального языка [114], удобного как для представления модели, так и для описания проекта, позволяет организовать процесс проектирования как последовательную детализацию первоначальной абстрактной модели. Раскрашенные временные сети Петри [50, 154, 180] представляют собой пример такого языка, а система CPN Tools [6] – пример моделирующей системы, реализующей язык сетей Петри. Моделирование является важным этапом построения компьютерных сетей, позволяющим выполнить точную оценку функциональных характеристик. Такая оценка особенно важна для сетей, применяемых в управлении процессами реального времени [117].

Управляемое моделью проектирование [7б,68] представляет собой, по существу, методологию синтеза телекоммуникационных систем и сетей близких к оптимальным [113, 155, 161, 176, 190]. Задача синтеза, ввиду того, что используется универсальная алгоритмическая система, является трудноформализуемой. Предложено выполнять её решение в процессе итерационной композиции моделей из стандартных компонентов и последующего анализа характеристик моделей с помощью метода измерительных фрагментов [95, 140]. Раскрашенные иерархические временные сети Петри [50, 180] является мощным и удобным инструментом для построения и исследования моделей телекоммуникационных систем. Эти сети успешно применены для моделирования протоколов телефонной связи, новых поколений мобильных телефонов, компьютерных сетей Token Ring и Ethernet [50, 68, 95].

Таким образом, соискателем предложено выполнять синтез моделей телекоммуникационных систем как композицию подсетей с контактными позициями (функциональных подсетей) для класса раскрашенных сетей Петри в целях построения эффективных телекоммуникационных систем. Для оценки функ-

циональных характеристик моделей предложен метод измерительных фрагментов, позволяющий вычислять характеристики в процессе имитации динамики сети Петри. Такой подход имеет существенные преимущества по сравнению с использованием других систем имитационного моделирования. Во-первых, обеспечивается преемственность языков разной изобразительной мощности, имеющих в своей основе двудольный ориентированный граф сети Петри. Во-вторых, обеспечивается моделирование гетерогенных сетей, что существенно затруднено при использовании специализированных моделирующих систем. Кроме того, сети Петри также используются для представления алгоритмов вычисления функциональных характеристик моделируемых систем.

1.4.3. Декомпозиция и редукция сетей Петри

Детализированные модели Петри телекоммуникационных протоколов имеют большую размерность. Для работы с крупномасштабными сетями Петри были предложены [8] два основных подхода: декомпозиция и редукция. Известны различные способы реализации этих фундаментальных подходов. Кроме того, декомпозиция и редукция применяются не только к сетям, но также и к пространствам их состояний. Редукция [8] представляет множество правил, уменьшающих размерность сети с сохранением её свойств. Затем применяют обычные методы анализа к редуцированной сети. Декомпозиция и композиция [178] являются абстрактными методами, успешно применяемыми в различных областях науки и техники. С одной стороны, большинство искусственных систем формируют из компонентов, и этот процесс является иерархическим. Таким образом, декомпозиция системы обеспечивается правилами её построения и множеством её компонентов и элементов [24, 39, 50, 52]. Простейший способ состоит в использовании такой конструктивной декомпозиции. Если известны свойства компонентов и используются специальные правила композиции (синтеза), сохраняющие требуемые свойства, то конструируется идеальная система [51, 160]. Но, к сожалению, это не является доминирующей возможностью при создании реальных систем. С другой стороны, цели конкретного анализа часто

требуют специальную декомпозицию. Декомпозиция оправдана лишь в том случае, когда существуют методы, позволяющие определить свойства системы на основе свойств её компонентов. Таким образом, методы декомпозиции всегда предполагают последующую композицию системы.

Рассмотрим подходы к декомпозиции, применяемые в теории сетей Петри. Первая попытка была предпринята М.Хаком [42] для декомпозиции сетей со свободным выбором в автоматные сети. Были исследованы условия сохранения живости сети при композиции живых автоматных сетей. Ж.Берселот [8] изучал декомпозицию на S- и T-компоненты: S-компонент разделяет переходы с другими S-компонентами, в то время, как T-компонент разделяет позиции. Была исследована поведенческая эквивалентность сетей. Дж.Эспарза и М.Силва [34] ввели два специальных типа композиции: синхронизация и слияние. Они рассматривали синхронизацию, сохраняющую живость сетей со свободным выбором. В [11] декомпозиция на T-компоненты была применена для генерации домашних состояний в сетях со свободным выбором. Результаты, связанные с композицией сетей со свободным выбором были представлены в монографии Дж.Десела и Дж.Эспарзы [27]. В.Е.Котов [160] предложил выполнять композицию сетей Петри из элементарных сетей, используя алгебраические операции. Такие сети были названы регулярными. Регулярные сети рассматривают раздельные множества входных и выходных позиций. Алгебраический подход нашёл своё дальнейшее развитие в работах А.И.Слепцова [181] и Н.А.Анисимова [108]. Различные виды композиции, сохраняющие живость, ограниченность и другие свойства были изучены в [19, 56, 57, 86]. Метод последовательной верификации, основанный на композиции подсетей, покрывающих формулу временной логики был предложен в [44].

Несмотря на отличия в определениях компонентов, основная идея композиционного подхода достаточно проста: выделить интерфейс компонента, скрыв его реализацию [173]. Для T-компонентов композиция обеспечивается слиянием контактных позиций. Такая композиция использована для построения иерархических высокоуровневых сетей [50, 52, 54, 75]. Сети, содержащие кон-

тактные позиции, были также названы IO-сетями [51]. При исследовании управляемости сетей было предложено различать входные и выходные позиции [37]. Декомпозиция графов достижимых и покрывающих маркировок была успешно применена для решения проблемы разрастания пространства состояний в низкоуровневых [89] и высокоуровневых [43, 46] сетях Петри. Кроме того, были изучены [51] правила композиции пространств состояний при композиции компонентов сетей.

Функциональные сети Петри были впервые введены соискателем в [103]. Как и T-компоненты, функциональные подсети задают разбиение множества позиций сети. Но они отличаются от T-компонентов и IO-сетей: рассматриваются раздельные подмножества входных и выходных позиций и определение не требует, чтобы позиция имела только одну входящую и одну исходящую дугу. С другой стороны, среди различных предложенных ранее ограничений для множеств инцидентных дуг входных и выходных позиций функциональные сети используют наиболее строгие: входные позиции имеют только входящие внешние дуги, а выходные позиции – только исходящие [103].

Введенный соискателем класс функциональных сетей Петри применён для композиции, как классических, так временных и раскрашенных сетей Петри. Причём в отличие от композиционных методов, изученных В.Е.Котовым и А.А.Анисимовым, использована лишь одна операция совмещения контактных позиций. Кроме того, класс функциональных сетей существенно отличается от S- и T-компонентов, изученных М.Хаком и известных других способов декомпозиции. Композиция сети Петри из её функциональных подсетей [144] была успешно применена для ускорения процессов вычисления инвариантов [99, 152] и решения уравнения состояний [97, 142]. Основной результат, обусловленный осуществлённым впервые соискателем построением композиционного анализа [133] на основе функциональных сетей, состоит в существенном ускорении исследования классических сетей Петри методами линейной алгебры. Ускорения вычислений в ряде случаев позволяют выполнить анализ моделей, ранее рассматривавшийся как практически неосуществимый. Таким образом,

решена задача доказательства корректности (верификации) детализированных моделей большой размерности для сложных телекоммуникационных протоколов.

Построение соискателем передаточной функции сети Петри позволило разработать теоретические основы для эквивалентных преобразований сетей, причём результаты сформулированы для класса временных сетей с многоканальными переходами. Разновидностью эквивалентных преобразований является редукция сетей, снижающая размерность модели, сохраняя её свойства. Правила редукции, сформулированные Ж.Берселотом [8], могут рассматриваться как частный случай редукции на основе алгебраических преобразований передаточной функции сети. Эквивалентные преобразования являются методом, предназначенным для ускорения анализа временных моделей Петри телекоммуникационных систем.

1.4.4. Решение систем линейных диофантовых уравнений

Методы линейной алгебры [30, 79, 169], основанные на уравнении состояний и инвариантах являются мощным инструментарием для анализа классических и временных сетей Петри [145]. Но, для того, чтобы найти линейные инварианты и решить фундаментальное уравнение сети Петри, следует решить систему линейных диофантовых уравнений в целых неотрицательных числах. Решения системы интерпретируются как вектора счёта срабатываний допустимых последовательностей переходов и поэтому должны быть целыми и неотрицательными числами, что обуславливает специфику задачи.

К решению систем линейных алгебраических уравнений (СЛУ) могут быть сведены многие задачи разработки телекоммуникационных систем, проектирования оборудования и программного обеспечения [24, 61, 74, 155, 164, 174]. Специальные методы решения систем диофантовых уравнений СЛДУ изучены в [183]. Эффективные алгоритмы основаны на построении множеств решений в полях классов вычетов по модулям простых чисел с последующим восстановлением искомого решения в кольце целых [120, 187]. Наиболее слож-

ными с вычислительной точки зрения являются задачи нахождения решений СЛДУ в неотрицательной области (СЛДУН). Следует отметить практическую важность задачи решения линейных диофантовых систем в неотрицательной области. Большинство задач исследования свойств сетей Петри может быть представлено в виде систем неравенств и сведено к решению СЛДУН [30, 79, 110, 169, 180]. Кроме теории сетей Петри аналогичные задачи возникают в области искусственного интеллекта, функционального программирования, схемотехники [4, 24, 61]. Все известные методы решения СЛДУН [22, 23, 67, 88, 162-166] имеют экспоненциальную вычислительную сложность, что делает анализ крупномасштабных моделей телекоммуникационных протоколов практически неосуществимым и требует поиска новых методов, обеспечивающих существенное ускорение вычислений.

Представляется целесообразным разработать методы, позволяющие ускорить решение линейных систем, в наиболее общей форме. Для этих целей в разделе 3 вводится понятие клана линейной системы, являющееся обобщением понятия функциональной подсети для произвольной системы линейных алгебраических уравнений над кольцом со знаком. Композиция кланов [139] успешно применена соискателем для ускорения процессов решения однородных и неоднородных систем линейных алгебраических уравнений. Использование композиции позволяет получить ускорения вычислений при решении линейных систем методами вычислительной сложности, превышающей кубическую, так как сложность декомпозиции и последующей композиции оцениваются полиномом третьей степени от размера системы [93,139]. Тем не менее, наиболее существенные ускорения вычислений получены при решении диофантовых систем в целых неотрицательных числах, так как все известные методы решения таких систем [22, 23, 67, 77, 88, 162] имеют экспоненциальную сложность.

В [139] соискателем изучена одновременная композиция всех кланов системы. Тем не менее, в случаях, когда количество контактных переменных превышает количество переменных наибольшего из кланов возможно получение дополнительного ускорения вычислений за счёт последовательной организации

процесса композиции [93, 132, 143]. Эта задача была формализована в терминах теории графов [5, 184, 188] и названа коллапсом взвешенного графа. Предложены эффективные методы её решения. Последовательная композиция применена в разделе 4 для ускорения процессов верификации телекоммуникационных протоколов [94, 96, 98, 130, 131, 141, 147, 150].

Наиболее часто для решения СЛДУН в настоящее время применяют методы Тудика [88] и Контежан [23]. Метод Тудика был представлен без соответствующего теоретического обоснования [88] и рассматривался как эвристический [110-112]. В Приложении Б соискателем выполнено теоретическое обоснование метода Тудика и получены оценки его вычислительной сложности [148,149]. В разделе 3 представлены методы ускорения решения систем линейных алгебраических уравнений, основанные на композиции их кланов [132, 139], а также эффективные алгоритмы, реализующие композиционное решение линейных систем [134,138].

Выводы к 1 разделу

1. Выполнен анализ языков спецификации и методов верификации телекоммуникационных протоколов. Обосновано применение сетей Петри как языка спецификации протоколов предназначенного для формального доказательства их корректности. Обоснована необходимость разработки методов автоматизированного синтеза моделей Петри сложных телекоммуникационных протоколов

2. Выполнен обзор и классификация аналитических методов оценки эффективности телекоммуникационных систем. Показано, что большинство аналитических методов основано на потоковом представлении трафика и упрощённом описании поведения систем. Моделирование особенностей функционирования телекоммуникационных систем и оценки характеристик качества обслуживания требует применения имитационного моделирования.

3. Выполнен обзор и классификация систем имитационного моделирования телекоммуникационных систем и сетей, а также их применения в реальных проектах. Обоснован выбор раскрашенных сетей Петри как универсального языка имитационного моделирования телекоммуникационных систем.

4. Выполнен анализ применения сетей Петри для верификации телекоммуникационных протоколов. Показано, что традиционные методы не позволяют выполнить верификацию сложных протоколов за приемлемое время. Обоснована необходимость разработки методов для ускорения анализа моделей Петри большой размерности. Обоснована необходимость разработки методов анализа бесконечных сетей Петри, моделирующих протоколы с неограниченным числом взаимодействующих устройств.

5. Показано, что исследование свойств классических и временных сетей Петри может быть сведено к решению систем линейных диофантовых уравнений в целых неотрицательных числах. Показано, что все методы решения таких систем асимптотически экспоненциальны. Обоснована необходимость разработки методов ускорения решения линейных систем уравнений для верификации сложных телекоммуникационных протоколов.

РАЗДЕЛ 2

ФУНКЦИОНАЛЬНЫЕ СЕТИ ПЕТРИ

Класс функциональных сетей Петри введен и исследован для ускорения процессов анализа моделей Петри большой размерности. Разработанные методы предназначены для верификации сложных телекоммуникационных протоколов.

2.1. Основные понятия и определения

Понятия функциональной сети Петри и функциональной подсети вводятся для ординарной сети Петри. Поэтому они применимы для различных классов сетей Петри, использующих двудольный ориентированный граф. Кратные дуги рассматриваются при вычислении линейных инвариантов и решении фундаментального уравнения сети Петри.

Определение 2.1. *Сеть Петри*

Сеть Петри – это тройка $N = (P, T, F),$ где $P = \{p_1, p_2, ..., p_m\}$ – конечное множество позиций, $T = \{t_1, t_2, ..., t_n\}$ – конечное множество переходов, и $P \bigcap T = \varnothing$, потоковое отношение $F \subseteq P \times T \bigcup T \times P$ определяет множество дуг, соединяющих позиции и переходы.

Пример сети Петри N_1 представлен на рис. 2.1.

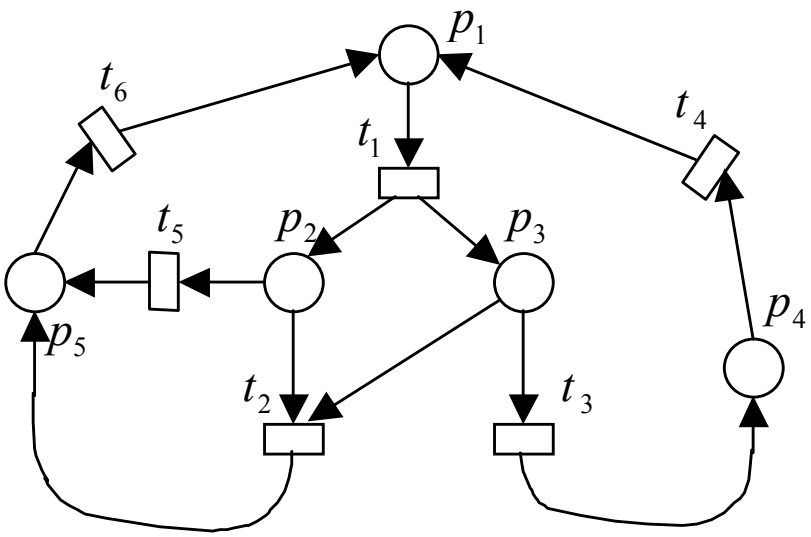


Рис. 2.1. Сеть Петри N_1

Используем специальные обозначения для множеств входных, выходных и инцидентных вершин позиции:

$$^{\bullet}p = \{t \mid \exists(t, p) \in F\},\ \ p^{\bullet} = \{t \mid \exists(p, t\} \in F\}\text{,}\ \ ^{\bullet}p^{\bullet} = {}^{\bullet}p \cup p^{\bullet}\text{.}$$

Аналогично могут быть определены множества входных, выходных и инцидентных вершин перехода и, более того, произвольного подмножества позиций (переходов).

Определение 2.2. *Сеть с входными и выходными позициями*

Сеть с входными и выходными позициями – это тройка $Z = (N, X, Y),$ где N – сеть Петри, $X \subseteq P$ – входные позиции, $Y \subseteq P$ – выходные позиции, кроме того, множества входных и выходных позиций не пересекаются: $X \bigcap Y = \varnothing$. Позиции из множества $Q = P \setminus (X \bigcup Y)$ назовём *внутренними*. Входные и выходные позиции $C = X \bigcup Y$ назовём *контактными*.

Известны также определения [19, 37] сети Петри с контактными позициями, которые не подразделяются на входные и выходные подмножества.

Определение 2.3. *Функциональная сеть*

Функциональная сеть – это сеть с входными и выходными позициями, такая что входные позиции не имеют входящих дуг, а выходные позиции – исходящих: $\forall p \in X: {}^\bullet p = \varnothing$, $\forall p \in Y: p^\bullet = \varnothing$. Обозначим функциональную сеть как $Z = (N, X, Q, Y)$ либо $Z = (X, Q, Y, T, F)$ в соответствии с обозначениями элементов сети Петри *N*.

Утверждение 2.1. Произвольная сеть Петри *N* может рассматриваться как функциональная сеть, где множество *X* формируют источники сети *N*, а множество *Y* – стоки сети *N*:

$$Z(N) = (X, Q, Y, T, F),\ Q = {}^\bullet T \cap T^\bullet,\ X = {}^\bullet T \setminus Q,\ X = T^\bullet \setminus Q.$$

Таким образом, в дальнейшем изложении без ограничения общности будем рассматривать только функциональные сети Петри, допуская пустые множества контактных позиций. В [154] изучена передаточная функция функциональной временной сети Петри и исследованы методы эквивалентных преобразований, основанные на алгебраических преобразованиях передаточной функции.

Используем следующие понятия в соответствии со стандартными определениями теории графов [5, 188]:

- Сеть Петри $N' = (P', T', F')$ является *подсетью N*, если
$$P' \subseteq P, T' \subseteq T, F' \subseteq F \bigcap ((P' \times T') \bigcup (T' \times P')).$$
- *Подсетью порождённой указанным множеством вершин* $B(P', T')$ является подсеть $N' = (P', T', F')$, где F' содержит все дуги, соединяющие вершины P', T' в исходной сети:
$$F' = \{(p, t) \mid p \in P', t \in T', (p, t) \in F\} \bigcup \{(t, p) \mid p \in P', t \in T', (t, p) \in F\}.$$
- *Подсетью порождённой указанным множеством переходов* $B(T')$ является подсеть $B(P', T')$, где

$$P' = {}^{\bullet}T' \cup T'^{\bullet}.$$

Иными словами, вместе с переходами из множества T' подсеть $B(T')$ содержит все инцидентные позиции и порождается этими вершинами. Далее будем рассматривать, как правило, все дуги, соединяющие указанные множества вершин, изучая подсети, порождённые множествами вершин. Поэтому, для краткости будем опускать потоковое отношение, подразумевая исходное отношение *F*.

Определение 2.4. *Функциональная подсеть*

Функциональная сеть $Z = (N', X, Q, Y)$ является *функциональной подсетью* сети *N* и обозначается как $Z \succ N$, если N' является подсетью *N* порождённой множеством T': $N' = B(T')$ и, кроме того, *Z* соединена с оставшейся частью сети только посредством дуг, инцидентных контактным позициям таким образом, что входные позиции имеют только входящие дуги, а выходные позиции – только исходящие:

$$\forall p \in X : \{(p,t) \mid t \in T \setminus T'\} = \varnothing,\ \forall p \in Y : \{(t,p) \mid t \in T \setminus T'\} = \varnothing,$$
$$\forall p \in Q : \{(p,t) \mid t \in T \setminus T'\} = \varnothing \wedge \{(t,p) \mid t \in T \setminus T'\} = \varnothing.$$

Указанные условия можно также представить как:

$$X^{\bullet} \cap (T \setminus T') = \varnothing,\ {}^{\bullet}Y \cap (T \setminus T') = \varnothing,\ {}^{\bullet}Q^{\bullet} \cap (T \setminus T') = \varnothing.$$

Заметим, что аналогичным образом можно ввести понятие двойственной функциональной сети, порождённой указанным множеством позиций и использующей контактные переходы. Но далее предпочтительным является изучение функциональной сети в соответствии с определением 2.4 для двойственной сети Петри [169].

Разность сети Петри N и её функциональной подсети Z' обозначим как сеть $Z'' = N - Z'$, где

$$Z'' = B(T \setminus T') = (Y, P \setminus (X \bigcup Y \bigcup Q), X, T \setminus T').$$

Утверждение 2.2. (Симметрия). $Z' \succ N$ тогда и только тогда, когда $N - Z' \succ N$.
Для доказательства утверждения заметим, что $N - Z'$ соединена с оставшейся частью сети только посредством контактных позиций и, кроме того, ограничения на дуги в определении функциональной подсети соответствуют ограничениям на дуги в определении функциональной сети. □

Определение 2.5. *Минимальная функциональная подсеть*
Функциональная подсеть $Z' \succ N$ является *минимальной*, если она не содержит некоторую другую функциональную подсеть сети Петри *N*.

Множество минимальных функциональных подсетей сети Петри N_1 (рис. 2.1) представлено на рис. 2.2. Заметим, что эти функциональные подсети имеют только входные и выходные позиции (множество внутренних позиций пусто). Например, подсеть $Z_2 = B(\{t_2, t_3, t_5\})$ имеет $X = \{p_2, p_3\}$, $Y = \{p_4, p_5\}$. Более сложные примеры декомпозиции рассмотрены в разделе 4.

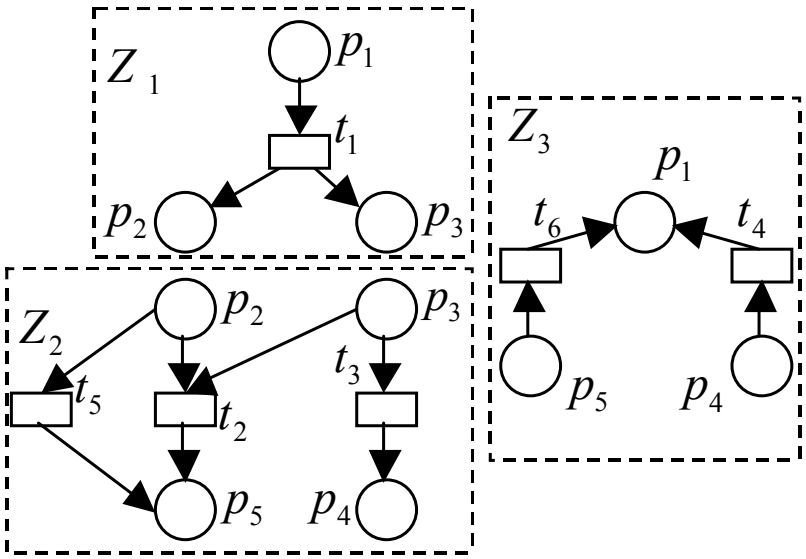


Рис. 2.2. Декомпозиция сети Петри N_1 на минимальные функциональные подсети

Аналогичным образом можно вводить и изучать различные подклассы сетей Петри с контактными позициями. Причём контактные позиции могут быть подразделены не только в подмножества входных и выходных позиций. Предложено классифицировать такие сети следующим образом. Во-первых, рассматриваем связи контактной позиции с внутренними элементами (I) подсети и внешними элементами (O). Во-вторых, различаем три типа связей: только входящие дуги, только исходящие дуги, дуги обоих направлений. Таким образом, можно ввести девять (3^2) типов контактных позиций, представленных на рис. 2.3. Заметим, что функциональная подсеть использует позиции типа d) как входные X и позиции типа b) как выходные Y.

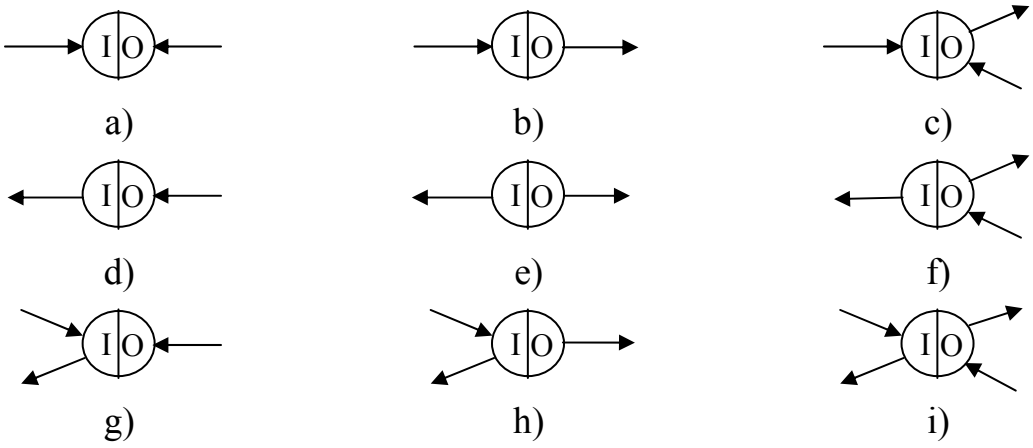


Рис. 2.3. Типы контактных позиций

2.2. Свойства функциональных подсетей

Лемма 2.1. Подсеть $B(R)$, $R \subseteq T$ является функциональной подсетью тогда и только тогда, когда выполняется условие ${}^\bullet(R^\bullet) \cup ({}^\bullet R)^\bullet \subseteq R$.

Доказательство. А) Достаточность. Пусть $Z = B(R)$ функциональная подсеть. Докажем, что ${}^\bullet(R^\bullet) \subseteq R$ от противного. Предположим, что $\exists t : t \in {}^\bullet(R^\bullet) \wedge t \notin R$ и рассмотрим позицию $p \in R^\bullet$ такую что $t \in {}^\bullet p$. Во всех возможных случаях: $q \in X$,

$q \in Y$, $q \in Q$ получаем противоречие. Аналогичным образом докажем, что $({}^\bullet R)^\bullet \subseteq R$.

В) Необходимость. Пусть выполняется условие ${}^\bullet(R^\bullet) \cup ({}^\bullet R)^\bullet \subseteq R$. Если $t \in Q^\bullet \cup X^\bullet$ и так как $Q \cup X \subset {}^\bullet R$, то $t \in ({}^\bullet R)^\bullet \subset T$. С другой стороны, если $t \in {}^\bullet Q \cup {}^\bullet Y$ и так как $Q \cup Y \subset R^\bullet$, то $t \in {}^\bullet(R^\bullet) \subset T$. Следовательно, в двух рассмотренных выше случаях условия определения функциональной подсети ${}^\bullet Q^\bullet \cap (T \setminus T') = \varnothing$, $X^\bullet \cap (T \setminus T') = \varnothing$, ${}^\bullet Y \cap (T \setminus T') = \varnothing$ выполнены. □

Замечание. Если не рассматриваются функциональные подсети, состоящие из изолированных переходов, то условие леммы 2.1 можно представить как ${}^\bullet(R^\bullet) \cup ({}^\bullet R)^\bullet = R$.

Теорема 2.1. Множества переходов двух произвольных минимальных функциональных подсетей Z' и Z'' сети Петри *N* не пересекаются.

Доказательство. Пусть $Z' = B(T')$ и $Z'' = B(T'')$ минимальные функциональные подсети. Предположим противное, а именно $T' \cap T'' \neq \varnothing$ и рассмотрим сеть порождённую множеством $T' \cap T''$.

Используя монотонность операции, представленной точкой, рассмотрим следующую последовательность рассуждений:

$$T' \cap T'' \subset T'$$

тогда

$$(T' \cap T'')^\bullet \subset T'^\bullet$$

следовательно, в соответствии с леммой 2.1

$${}^\bullet((T' \cap T'')^\bullet) \subset {}^\bullet(T'^\bullet) \subset T'.$$

Аналогичным образом получим

$${}^\bullet((T' \cap T'')^\bullet) \subset {}^\bullet(T'^\bullet) \subset T''$$

тогда

$${}^\bullet((T' \cap T'')^\bullet) \subset T' \cap T''.$$

Заметим, что также выполняется следующее условие

$$({}^\bullet(T' \cap T''))^\bullet \subset T' \cap T''.$$

Таким образом, получим

$$^{\bullet}((T' \cap T'')^{\bullet}) \cup (^{\bullet}(T' \cap T''))^{\bullet} \subset T' \cap T'' .$$

Следовательно $B(T' \cap T'')$ является функциональной подсетью сети Петри N, что противоречит с минимальностью подсетей $Z' = B(T')$ и $Z'' = B(T'')$. □

Следствие 1. Множества внутренних позиций двух произвольных минимальных функциональных подсетей Z' и Z'' сети Петри *N* не пересекаются.

Следствие 2. Множество минимальных функциональных подсетей $\Im = \{Z^j\}$, $Z^j \succ N$ задаёт разбиение множества T на непересекающиеся подмножества T^j такие, что $T = \bigcup_j T^j$, $T^j \bigcap T^k = \varnothing$, $j \neq k$.

Для представления взаимосвязей минимальных функциональных подсетей построим высокоуровневую сеть минимальных функциональных подсетей. Переходы этой сети соответствуют минимальным функциональным подсетям. Множество позиций состоит из контактных позиций исходной сети. Высокоуровневая сеть сети Петри N_1 представлена на рис. 2.4. Выполним формальное определение таких сетей.

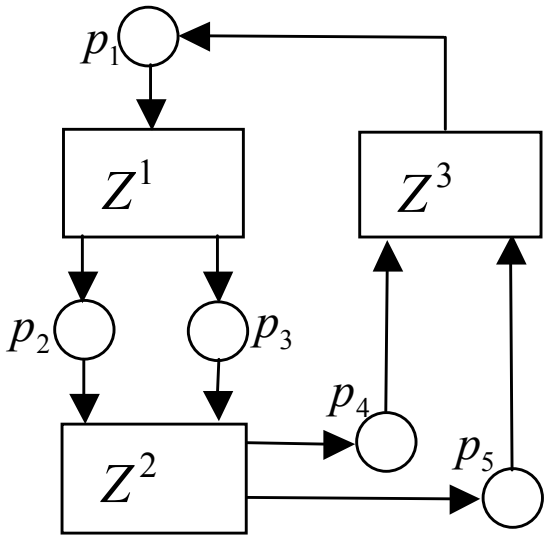


Рис. 2.4. Сеть N_1^z минимальных функциональных подсетей сети Петри N_1

Определение 2.6. *Сеть функциональных подсетей*

Сеть функциональных подсетей заданной сети Петри N – это сеть Петри N' такая, что:

$$P' = C,\ T' = \{t^Z \mid t^Z \leftrightarrow Z, Z \succ N\},$$

$$(p', t^Z) \in F' \Leftrightarrow \exists t \in T(Z) : (p', t) \in F,\ (t^Z, p') \in F' \Leftrightarrow \exists t \in T(Z) : (t, p') \in F.$$

Заметим, что сеть функциональных подсетей может быть определена для декомпозиции, содержащей не минимальные функциональные сети, задающей разбиение множества переходов.

Определение 2.7. *Полнота*

Подсеть $Z = B(R) = (X, Q, Y, R)$ сети Петри N полна в N если следующие условия: $X^\bullet \subseteq R$, $^\bullet Y \subseteq R$, $^\bullet Q^\bullet \subseteq R$ выполняются в N.

Лемма 2.2. Подсеть Z полна в сети Петри N тогда и только тогда, когда она является функциональной подсетью N.

Доказательство. Начнём с доказательства необходимости полноты. Пусть Z функциональная подсеть N: $Z \succ N$. Тогда условия полноты выполняются для инцидентных переходов внутренних позиций Q в соответствии с определением внутренних позиций, а для выходных переходов входных позиций X и входных переходов выходных позиций Y – в соответствии с определением функциональной подсети.

Докажем достаточность. Известно, что Z является подсетью N порождённой множеством переходов R и Z функциональная подсеть. Остаётся доказать, что выполняются ограничения на дуги, соединяющие позиции подсети Z с оставшейся частью сети. Обозначим оставшуюся часть сети как $Z' = N - B(R) = (Y, Q', X, R')$, где $Q' = P \setminus (X \cup Q \cup Y)$, $R' = T \setminus R$. Предположим противное. Пусть N содержит одну либо несколько запрещённых дуг одного из возможных шести типов: a) (x, r'); b) (r', y); c) (r, q'); d) (q', r); e) (q, r'); f) (r', q), где

$x \in X, y \in Y, q \in Q, q' \in Q', r \in R, r' \in R'$. Рассмотрим каждый из указанных типов дуг отдельно:

a) Если $(x, r') \in F$, тогда $r' \in x^{\bullet}$, следовательно $X^{\bullet} \not\subset R$.

b) Если $(r', y) \in F$, тогда $r' \in {}^{\bullet}y$, следовательно ${}^{\bullet}Y \not\subset R$.

c) Если $(r, q') \in F$, тогда $q' \in Y$.

d) Если $(q', r) \in F$, тогда $q' \in X$.

e) Если $(q, r') \in F$, тогда $r' \in q^{\bullet}$, следовательно ${}^{\bullet}Q^{\bullet} \not\subset R$.

f) Если $(r', q) \in F$, тогда $r' \in {}^{\bullet}q$, следовательно ${}^{\bullet}Q^{\bullet} \not\subset R$.

Таким образом, в каждом из указанных случаев получаем противоречие. Этот факт завершает доказательство достаточности полноты подсети. □

Лемма 2.3. Каждая контактная позиция разложенной сети Петри имеет не более одной входной минимальной функциональной подсети и не более одной выходной минимальной функциональной подсети.

Доказательство. Предположим противное. Следует рассмотреть два случая:

a) Существует контактная позиция $p \in C$ которая имеет более одной входной минимальной функциональной подсети;

b) Существует контактная позиция $p \in C$ которая имеет более одной выходной функциональной подсети.

В случае a) имеются минимальные функциональные подсети Z', Z'' такие, что $(\exists t' \in Z',\ t' \in {}^{\bullet}p) \wedge (\exists t'' \in Z'',\ t'' \in {}^{\bullet}p)$.

Так как в соответствии с леммой 2.2 каждая минимальная функциональная подсеть полна в N, то переходы t', t'' в соответствии с определением полноты принадлежат одной и той же минимальной функциональной подсети. Таким образом, получаем противоречие.

В случае b) имеются минимальные функциональные подсети Z', Z'' такие, что $(\exists t' \in Z',\ t' \in p^{\bullet}) \wedge (\exists t'' \in Z'',\ t'' \in p^{\bullet})$.

Тогда получаем противоречие так же, как и в случае a).

Полученное в двух рассмотренных случаях противоречие завершает доказательство леммы. □

Непосредственным следствием леммы 2.3 и определения маркированного графа [30, 169, 173] как сети Петри, все позиции которой имеют не более одной входящей и не более одной исходящей дуги, является следующая теорема.

Теорема 2.2. Сеть функциональных подсетей заданной сети Петри является маркированным графом.

Описанная ранее сеть N^z является слишком детализированным представлением взаимосвязей подсетей. Она содержит параллельные маршруты в случаях если несколько контактных позиций соединяют пару подсетей. Для более компактного представления можно скрыть контактные позиции, рассматривая лишь связи подсетей. В таком случае получим следующий граф.

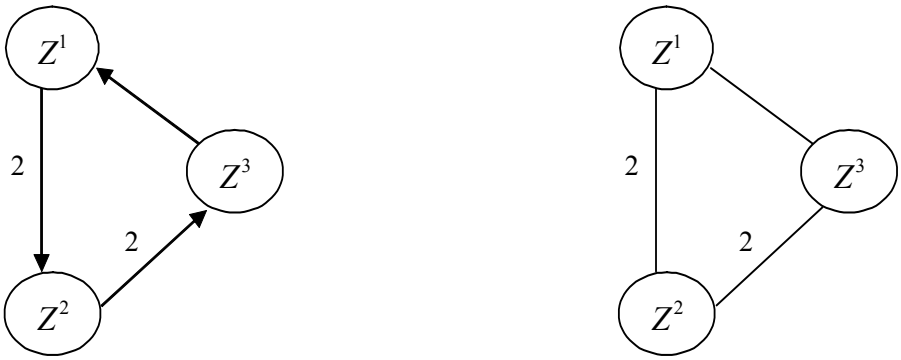
Определение 2.8. *Граф функциональных подсетей*

Граф функциональных подсетей заданной сети Петри N – это ориентированный взвешенный граф $G = (\Im, E, W)$, где множество вершин $\Im$ сформировано множеством минимальных функциональных подсетей сети N а дуги E соединяют вершины в случае, если соответствующие подсети имеют общие контактные позиции таким образом, что: $E = \{(Z^j, Z^k) \mid \exists p : p \in Y^j, p \in X^k\}$, $w(Z^j, Z^k) = \left|\left\{q \middle| \exists t \in T^j, \exists r \in T^k : (t,q) \in F, (q,r) \in F\right\}\right|$.

Кратность дуги равна количеству общих контактных позиций в соответствующем направлении.

Граф позволяет выполнить схематическое представление связей функциональных подсетей исходной сети Петри. На рис. 2.5 а) представлен граф функциональных подсетей сети Петри N_1.

Далее будем использовать также неориентированный граф декомпозиции (рис. 2.5 б), суммируя веса дуг противоположных направлений.



а) ориентированный б) неориентированный

Рис. 2.5. Граф G_1 функциональных подсетей сети Петри N_1

Следует отметить, что минимальность в общем случае не означает небольшое количество позиций и переходов, а предполагает лишь невозможность дальнейшего разделения на (внутренние) функциональные подсети. Более того, неразделимая сеть может содержать произвольное количество вершин. Пример неразделимой сети представлен на рис. 2.6. Цепь позиций и переходов, соединённых дугами указанного вида может соединять неограниченное количество вершин.

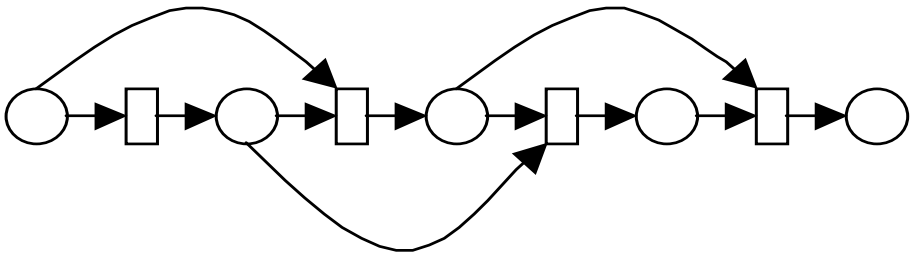


Рис. 2.6. Неразделимая сеть N_2

Теорема 2.3. Любая функциональная подсеть заданной сети Петри *N* является суммой (объединением) конечного числа её минимальных функциональных подсетей.

Доказательство. Предположим противное, а именно, что существует функциональная подсеть Z' сети Петри *N,* которая не является объединением минимальных функциональных подсетей. Так как $\Im$ определяет разбиение множества *T*, то T' содержит части подмножеств T^j. Формально это можно представить как: $T' = \bigcup_{i \in I} R^i$, где *I* – множество номеров подсетей, переходы которых содержатся в T', $R^i \subseteq T^i$ и, кроме того, существует, по крайней мере, одно множество $R^j \subset T^j$ для некоторого $j \in I$. Рассмотрим множество переходов $S = T^j \setminus R^j$ и покажем, что оно порождает функциональную подсеть $B(S)$ сети Петри *N*. Так как Z' – функциональная подсеть *N*, то $B(S)$ соединена с вершинами подсети Z' только посредством контактных позиций и, кроме того, так как Z^j является функциональной подсетью *N*, то $B(S)$ соединена с вершинами подсети $N - Z^j$ также только посредством контактных позиций. Кроме того, для входных и выходных позиций $B(S)$ выполняются ограничения функциональной подсети. Таким образом, функциональная подсеть Z^j содержит функциональную подсеть $B(S)$, что противоречит минимальности Z^j. Полученное противоречие доказывает ложность исходного предположения о том, что $S \neq \varnothing$. Таким образом, T' содержит множество T^j целиком, что в соответствии с произвольностью выбора T^j доказывает теорему. □

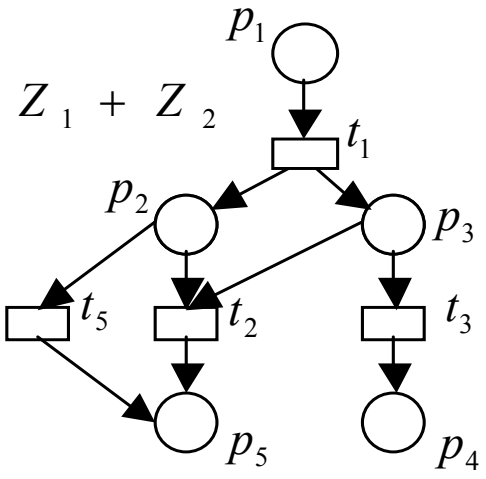


Рис. 2.7. Функциональная подсеть $Z_1 + Z_2$ сети Петри N_1

Представленную теорему можно проиллюстрировать рис. 2.7, на котором изображена сеть, являющаяся суммой двух минимальных функциональных подсетей сети Петри N_1, представленных на рис. 2.2.

Следствие. Разбиение множества *T,* заданное множеством минимальных функциональных подсетей, является порождающим семейством (базисом) множества функциональных подсетей сети Петри *N*.

2.3. Методы декомпозиции на функциональные подсети

Рассмотрим функциональную подсеть $Z^0 = (P^2, P^0, P^3, T^0)$ сети Петри $N = (P, T, F)$. Тогда $N - Z^0$, в соответствии с утверждением 2.2, является функциональной подсетью $Z^1 = (P^3, P^1, P^2, T^1)$. Взаимосвязи указанных подсетей проиллюстрированы на рис. 2.8. Построим уравнения в исчислении предикатов первого порядка, задающие какой из подсетей принадлежит переход либо позиция. Используем определение функциональной подсети, а также то, что в соответствии с утверждением 2.2 сеть Z^1 также является функциональной подсетью *N*. Для переходов имеем:

$$\begin{cases} (t \in T^0) \equiv (\forall p \in {}^\bullet t)((p \in P^0) \vee (p \in P^2)) \wedge (\forall p \in t^\bullet)((p \in P^0) \vee (p \in P^3)) \\ (t \in T^1) \equiv (\forall p \in {}^\bullet t)((p \in P^1) \vee (p \in P^3)) \wedge (\forall p \in t^\bullet)((p \in P^1) \vee (p \in P^2)) \end{cases} \quad (2.1)$$

Аналогичным образом построим уравнения, задающие множество, которому принадлежат позиции:

$$\begin{cases} (p \in P^0) \equiv (\forall t \in {}^\bullet p)(t \in T^0) \wedge (\forall t \in p^\bullet)(t \in T^0) \\ (p \in P^1) \equiv (\forall t \in {}^\bullet p)(t \in T^1) \wedge (\forall t \in p^\bullet)(t \in T^1) \\ (p \in P^2) \equiv (\forall t \in {}^\bullet p)(t \in T^1) \wedge (\forall t \in p^\bullet)(t \in T^0) \\ (p \in P^3) \equiv (\forall t \in {}^\bullet p)(t \in T^0) \wedge (\forall t \in p^\bullet)(t \in T^1) \end{cases} \quad (2.2)$$

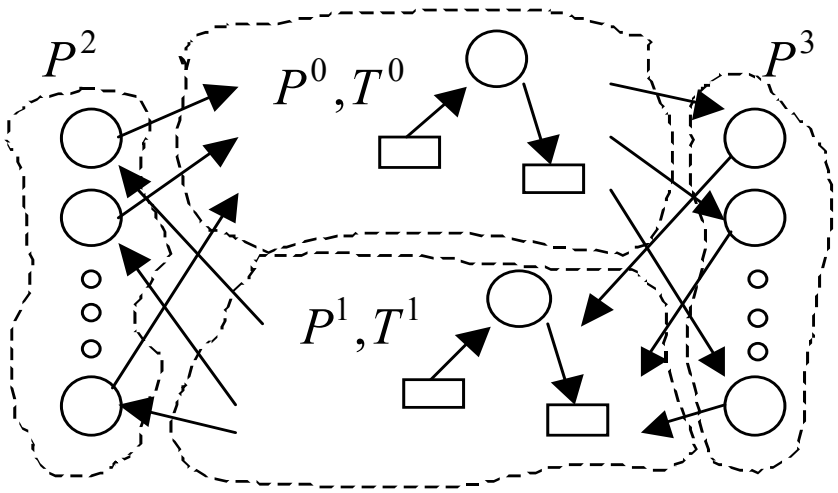


Рис. 2.8. Взаимосвязи функциональных подсетей

Подставим уравнения (2.2) в (2.1) и заметим, что так как $T^0 \bigcup T^1 = T$, а также $T^0 \bigcap T^1 = \varnothing$, то достаточно рассмотреть только одно из уравнений (2.1), например, задающее какие переходы принадлежат подмножеству T^1. Получим следующую систему:

$$
\begin{aligned}
&(t \in T^1) \equiv \\
&(\forall p \in {}^\bullet t)(((\forall s \in {}^\bullet p)(s \in T^1) \wedge (\forall s \in p^\bullet)(s \in T^1)) \vee ((\forall s \in {}^\bullet p)(s \in T^0) \wedge (\forall s \in p^\bullet)(t \in T^1))) \wedge \\
&(\forall p \in t^\bullet)(((\forall s \in {}^\bullet p)(s \in T^1) \wedge (\forall s \in p^\bullet)(s \in T^1)) \vee ((\forall s \in {}^\bullet p)(s \in T^1) \wedge (\forall s \in p^\bullet)(t \in T^0)))
\end{aligned}
\qquad (2.3)
$$

Используя конечность множеств позиций и переходов, заменим кванторы общности конъюнкцией по соответствующим подмножествам элементов. Кроме того, введём индикаторы τ_t принадлежности переходов подмножествам таким образом, что $\tau_t = j \Leftrightarrow t \in T^j$. Заметим, что $\tau_t \in \{0,1\}$; следовательно, эти величины могут быть использованы в логических уравнениях. А, так как $T^0 \bigcap T^1 = \varnothing$, то $\tau_t \Leftrightarrow (t \in T^1)$, и также $\bar{\tau}_t \Leftrightarrow (t \in T^0)$. Поэтому уравнения (2.3) могут быть представлены в Булевой алгебре в следующей форме:

$$
\tau_t \equiv \mathop{\&}_{p \in {}^\bullet t} \Big(\big(\mathop{\&}_{s \in {}^\bullet p} \tau_s \wedge \mathop{\&}_{s \in p^\bullet} \tau_s \big) \vee \big(\mathop{\&}_{s \in {}^\bullet p} \bar{\tau}_s \wedge \mathop{\&}_{s \in p^\bullet} \tau_s \big) \Big) \wedge \mathop{\&}_{p \in t^\bullet} \Big(\big(\mathop{\&}_{s \in {}^\bullet p} \tau_s \wedge \mathop{\&}_{s \in p^\bullet} \tau_s \big) \vee \big(\mathop{\&}_{s \in {}^\bullet p} \tau_s \wedge \mathop{\&}_{s \in p^\bullet} \bar{\tau}_s \big) \Big) \qquad (2.4)
$$

Таким образом, доказана следующая теорема.

Теорема 2.4. Разбиение произвольной сети Петри на функциональные подсети полностью задаётся системой логических уравнений (2.4).

В процессе решения система (2.4) может быть заменена одним уравнением, которое представляет собой конъюнкцию уравнений, соответствующих каждому переходу сети. Методы решения логических уравнений достаточно хорошо изучены, например в [122].

Рассмотрим пример декомпозиции сети N_1 (рис. 2.1). Построим систему логических уравнений вида (2.4):

$$\begin{cases} \tau_1 \equiv (\tau_6\tau_4\tau_1 \vee \bar{\tau}_6\bar{\tau}_4\tau_1)(\tau_1\tau_5\tau_2 \vee \tau_1\bar{\tau}_5\bar{\tau}_2)(\tau_1\tau_2\tau_3 \vee \tau_1\bar{\tau}_2\bar{\tau}_3) \\ \tau_2 \equiv (\tau_1\tau_5\tau_2 \vee \bar{\tau}_1\tau_5\tau_2)(\tau_1\tau_2\tau_3 \vee \bar{\tau}_1\tau_2\tau_3)(\tau_5\tau_2\tau_6 \vee \tau_5\tau_2\bar{\tau}_6) \\ \tau_3 \equiv (\tau_1\tau_2\tau_3 \vee \bar{\tau}_1\tau_2\tau_3)(\tau_3\tau_4 \vee \tau_3\bar{\tau}_4) \\ \tau_4 \equiv (\tau_3\tau_4 \vee \bar{\tau}_3\tau_4)(\tau_1\tau_4\tau_6 \vee \bar{\tau}_1\tau_4\tau_6) \\ \tau_5 \equiv (\tau_1\tau_5\tau_2 \vee \bar{\tau}_1\tau_5\tau_2)(\tau_5\tau_2\tau_6 \vee \tau_5\tau_2\bar{\tau}_6) \\ \tau_6 \equiv (\tau_5\tau_2\tau_6 \vee \bar{\tau}_5\bar{\tau}_2\tau_6)(\tau_1\tau_4\tau_6 \vee \bar{\tau}_1\tau_4\tau_6) \end{cases}$$

Построим конъюнкцию уравнений, упростим её и приведём к дизъюнктивной нормальной форме. Получим:

$$\bar{\tau}_1\bar{\tau}_2\bar{\tau}_3\bar{\tau}_4\bar{\tau}_5\bar{\tau}_6 \vee \tau_1\bar{\tau}_2\bar{\tau}_3\bar{\tau}_4\bar{\tau}_5\bar{\tau}_6 \vee \bar{\tau}_1\tau_2\tau_3\bar{\tau}_4\tau_5\bar{\tau}_6 \vee \bar{\tau}_1\bar{\tau}_2\bar{\tau}_3\tau_4\bar{\tau}_5\tau_6 \vee$$
$$\tau_1\bar{\tau}_2\bar{\tau}_3\tau_4\bar{\tau}_5\tau_6 \vee \tau_1\tau_2\tau_3\bar{\tau}_4\tau_5\bar{\tau}_6 \vee \bar{\tau}_1\tau_2\tau_3\tau_4\tau_5\tau_6 \vee \tau_1\tau_2\tau_3\tau_4\tau_5\tau_6$$

Заметим, что первый из термов соответствует пустой подсети; следующие три терма соответствуют минимальным функциональным подсетям, представленным на рис. 2.2: $T^1 = \{t_1\}$, $T^2 = \{t_2, t_3, t_5\}$, $T^3 = \{t_4, t_6\}$. Оставшиеся термы описывают суммы минимальных подсетей. Так, например, шестой терм описывает подсеть, представленную на рис. 2.7.

Следует отметить, что алгоритмическая сложность декомпозиции произвольной сети Петри на функциональные подсети с помощью описанных логических уравнений в общем случае асимптотически экспоненциальная, что связано с оценками сложности решения логических уравнений [122]. Но это метод является универсальным и может быть применён также для декомпозиции сети Петри в другие типы подсетей с контактными позициями, представленными в подразделе 2.1 (рис. 2.3).

Рассмотрим следующий алгоритм.

Алгоритм 2.1:

Шаг 0. Выберем произвольный переход $t \in T$ сети N и включим его во множество выбранных переходов $R := \{t\}$.

Шаг 1. Построим подсеть *Z,* порождённую множеством *R*: $Z = B(R) = (X, Q, Y, R)$.

Шаг 2. Если *Z* полна в *N*, то *Z* искомая подсеть, стоп.

Шаг 3. Построить множество поглощаемых переходов:

$$S = \{t \mid t \in X^{\bullet} \wedge t \notin R \vee t \in {}^{\bullet}Y \wedge t \notin R \vee t \in {}^{\bullet}Q^{\bullet} \wedge t \notin R\}.$$

Шаг 4. Положить $R := R \cup S$ и перейти на шаг 1.

Теорема 2.5. Подсеть *Z,* построенная алгоритмом 2.1 является минимальной функциональной подсетью сети Петри *N*.

Доказательство. В соответствии с леммой 2.2 алгоритм 2.1 строит функциональную подсеть. Следует доказать её минимальность. Предположим противное: пусть *Z* не является минимальной. Тогда существует минимальная функциональная подсеть Z' сети Петри *N* такая что $Z' = B(T'')$ и $T'' \subset T'$. Следовательно, подсеть *Z* содержит Z'. Рассмотрим два возможных варианта выполнения алгоритма 2.1: a) запуск с перехода $t \in T'$, такого что $t \in T''$; b) запуск с перехода $t \in T'$, такого что $t \notin T''$. Рассмотрим каждый из отмеченных вариантов отдельно.

a) Пусть $t \in T''$. Рассмотрим первый переход *v* множества $T' \setminus T''$, который был включен в множество *S* на некотором проходе основного цикла алгоритма 2.1. Тогда, в соответствии с описанием шага 3, возможен один из трёх случаев: $v \in X^{\bullet}$ либо $v \in {}^{\bullet}Y$, либо $v \in {}^{\bullet}S^{\bullet}$. В первом случае позиция $x \in X$, такая что $v \in x^{\bullet}$ не может быть ни входной, ни выходной, ни внутренней позицией сети Z'. Получено противоречие. Аналогично приходим к противоречию во втором и третьем случаях.

b) Пусть $t \notin T''$. Рассмотрим первый переход *v* множества T'', который был включен во множество *S* на некотором проходе главного цикла алгоритма 2.1. Тогда, в соответствии с описанием шага 3, возможен один из трёх случаев: $v \in X^{\bullet}$ либо $v \in {}^{\bullet}Y$, либо $v \in {}^{\bullet}S^{\bullet}$. В первом случае позиция $x \in X$, такая что $v \in x^{\bullet}$ не может быть ни входной, ни выходной, ни внутренней позицией сети Z'. Получено противоречие. Аналогично приходим к противоречию во втором и третьем случаях. □

Таким образом, алгоритм 2.1 позволяет построить минимальную функциональную подсеть *Z* сети Петри *N*. Положим $i := 1$ и $Z^i := Z$. Затем положим $N := N - Z$ и повторим выполнение алгоритма 2.1 в случае, если множество *T* не пусто. Продолжая аналогичным образом и полагая $i := i + 1$ построим множество минимальных функциональных подсетей $Z^1, Z^2, ..., Z^k$ сети *N,* которое является искомым разбиением исходной сети.

Применение алгоритма к сети, представленной на рис. 2.1, даёт результат, совпадающий с полученным в предыдущем подразделе с помощью логических уравнений и представленный на рис 2.2.

Так как каждая дуга сети Петри обрабатывается алгоритмом только один раз, верна следующая теорема.

Теорема 2.6. Временная сложность алгоритма 2.1 линейна по отношению к размеру сети Петри.

2.4. Передаточная функция сети Петри

В [154] построено уравнение состояний временной сети Петри с многоканальными переходами и получено представление передаточной функции для подкласса структурно-бесконфликтных сетей Петри. Показано, что алгебраические преобразования передаточной функции соответствуют эквивалентным преобразованиям сетей.

Существенным ограничением в применении полученных результатов является малая изобразительная мощность подкласса структурно-бесконфликтных сетей, допускающих не более одной исходящей дуги для каждой позиции. Кроме того, при исследовании систем целесообразно применение слабых типов эквивалентности по отношению к специальным формам входных последовательностей фишек и отдельным заданным моментам времени.

Маркировка сети – это отображение $\mu : P \to \mathrm{N}_0$, определяющее распределение динамических элементов, именуемых фишками, на множестве позиций; N_0 – множество неотрицательных целых чисел $\mathrm{N}_0 = \mathrm{N} \cup \{0\}$. *Маркированная сеть Петри* – это пара $M = (N, \mu_0)$, либо пятёрка $M = (P, T, F, \mu_0)$, где μ_0 – её начальная маркировка.

Поведение сети Петри представляет собой процесс перемещения фишек между позициями в результате срабатывания переходов. Правила срабатывания переходов определяются классом исследуемых сетей [30, 39, 50, 180]. При исследовании передаточной функции будем рассматривать класс временных сетей Петри с кратными дугами и многоканальными переходами [154]. В этом случае представление сети включает дополнительные отображения: $\mathrm{W} : F \to \mathrm{N}$ – кратности дуг и $\mathrm{D} : T \to \mathrm{N}$ – времена срабатывания переходов. Поведение такой сети описывается следующим *уравнением состояний* [154]:

$$\begin{cases} \lambda_p(\tau) = \mu_p(\tau-1) + \sum\limits_{t \in {}^\bullet p} w_{t,p} \cdot u_t(\tau - d_t) + \alpha_p^\tau, \\ \mu_p(\tau) = \lambda_p(\tau) - \sum\limits_{t \in p^\bullet} w_{t,p} \cdot u_t, \\ \mu_p(\tau) \geq 0, \\ \\ 0 \leq u_t(\tau) \leq \upsilon_t(\tau), \\ \upsilon_t(\tau) = \mathop{\&}\limits_{q \in {}^\bullet t} \lambda_q(\tau) / w_{q,t}, p \in P, t \in T, \\ S(0) = S_0, \tau = 1,2,... \end{cases} \tag{2.5}$$

где величина $u_t(\tau)$ равна количеству экземпляров (каналов) перехода t, запущенных в такте τ; $\nu_t(\tau)$ - количество экземпляров перехода t, возбуждённых в такте τ; $\lambda_p(\tau)$ - промежуточная маркировка позиции p в момент смены такта $\tau - 1$ тактом τ, получающаяся при завершении ранее запущенных переходов. Операция конъюнкции (&) интерпретируется, как в многозначной логике [153]: $x\&y = \min(x,y).$ а операция деления является делением нацело.

Заметим, что невременные синхронные и асинхронные сети Петри [169] являются частным случаем рассматриваемого класса сетей Петри при временах срабатывания переходов, равном единице, и одном экземпляре каждого из переходов. Явные ограничения числа каналов переходов не рассматриваются, так как они могут быть введены с помощью маркировок вспомогательных позиций [154].

Рассмотрим функциональную сеть Петри как дискретную систему (рис. 2.9), преобразующую входную последовательность фишек, направленную во входные позиции X, в выходную последовательность фишек, наблюдаемую в выходных позициях Y [154].

Входной последовательностью α сети Z будем называть множество целых неотрицательных чисел $\alpha = \{\alpha_x^\tau \mid x \in X, \tau = 1,2,...\}$. *Выходной последовательностью* ϕ сети Z будем называть последовательность маркеров $\{\phi_y^\tau \mid y \in Y, \tau = 1,2,...\}$, поступающих в её выходные позиции в результате функционирования сети Z.

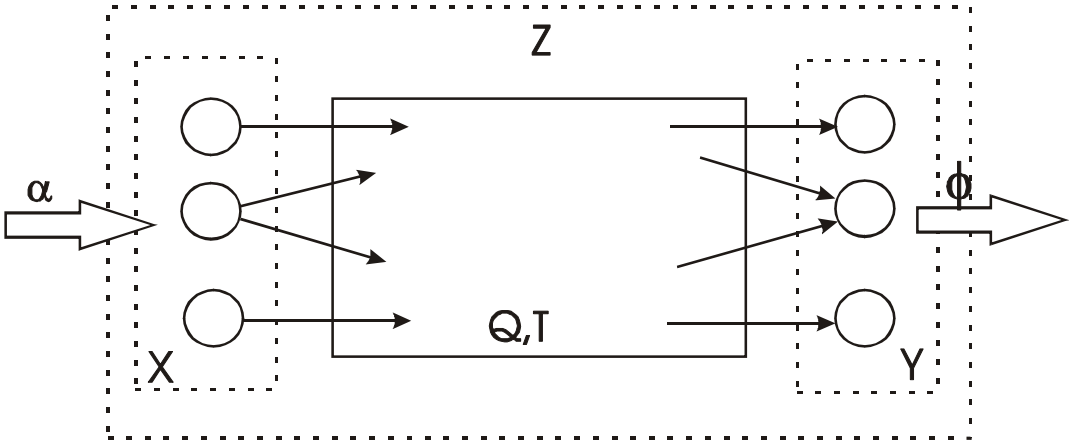


Рис. 2.9. Передаточная функция сети Петри

Отображение f_Z множества входных последовательностей $\{\alpha\}$ сети Z во множество множеств её выходных последовательностей $\{\phi\}$ будем называть *передаточной функцией сети* и обозначать $\Phi = f_Z\{\alpha\}$.

Неявное представление передаточной функции произвольной сети Петри задано её уравнением состояний (2.5). В [154] получено явное представление передаточной функции относительно частичных сумм входных последовательностей для подкласса структурно-бесконфликтных сетей. В настоящей работе для представления передаточной функции произвольной сети используем следующее соотношение:

$$\phi_y(\tau) = \mu_y(\tau) - \mu_y(\tau - 1), y \in Y$$

и далее, используя тот факт, что для выходной позиции её маркировка и промежуточная маркировка в момент смены тактов совпадают (так как отсутствуют исходящие дуги) $\mu_y(\tau) = \lambda_y(\tau), y \in Y$, представим передаточную функцию как

$$\phi_y(\tau) = \lambda_y(\tau) - \lambda_y(\tau - 1), y \in Y.$$

Таким образом, последовательности $\lambda_y(\tau), \tau = 1,2,...$ однозначно определяют передаточную функцию сети Петри. Тогда следующая теорема задаёт представление передаточной функции.

Теорема 2.7. Передаточная функция временной сети Петри Z описывается следующей системой:

$$\begin{cases} \lambda_p(\tau) = \lambda_p(\tau-1) - \sum\limits_{t \in {}^\bullet p} w_{p,t} \cdot u_t(\tau-1) + \sum\limits_{t \in {}^\bullet p} w_{t,p} \cdot u_t(\tau - d_t), p \in Q \cup Y, \\ \sum\limits_{t \in p^\bullet} w_{p,t} \cdot u_t(\tau) \le \lambda_p(\tau), p \in X \cup Q, \\ u_t(\tau) \le \mathop{\&}\limits_{q \in {}^\bullet t} \lambda_q(\tau) / w_{q,t}, p \in X \cup Q. \end{cases} \quad (2.6)$$

Доказательство. Подставим второе уравнение системы (2.6) для такта $\tau - 1$ в первое и получим:

$$\lambda_p(\tau) = \lambda_p(\tau-1) - \sum_{t \in {}^\bullet p} w_{p,t} \cdot u_t(\tau-1) + \sum_{t \in {}^\bullet p} w_{t,p} \cdot u_t(\tau - d_t), p \in Q \cup Y .$$

Из второго уравнения и неравенства $\mu_p(\tau) \ge 0$ получим:

$$\sum_{t \in p^\bullet} w_{p,t} \cdot u_t(\tau) \le \lambda_p(\tau), p \in X \cup Q .$$

Из четвёртого уравнения и неравенства $0 \le u_t(\tau) \le \upsilon_t(\tau)$ получим:

$$u_t(\tau) \le \mathop{\&}_{q \in {}^\bullet t} \lambda_q(\tau) / w_{q,t} .$$

□

Для алгебраических преобразований передаточной функции синхронных сетей положим $u_t(\tau) = v_t(\tau)$. Тогда с использованием операций алгебры временных сетей [154] передаточная функция (2.6) в произвольном такте времени τ может быть представлена в виде:

$$\begin{cases} \lambda_p = \lambda_p \triangleright 1 - \sum\limits_{t \in {}^\bullet p} w_{p,t} \cdot \&\limits_{q \in {}^\bullet t} (\lambda_q \triangleright 1)/w_{q,t}) + \sum\limits_{t \in {}^\bullet p} w_{t,p} \cdot \&\limits_{q \in {}^\bullet t} (\lambda_q \triangleright d_t)/w_{q,t}), p \in Q \cup Y, \end{cases} \quad (2.7)$$

где операция $\triangleright$ представляет временную задержку.

Сети Z и Z' будем называть *функционально эквивалентными* [154] и обозначать $Z \equiv Z'$, если для любой входной последовательности α множества их выходных последовательностей совпадают $f_Z(\alpha) = f_{Z'}(\alpha)$.

Приведенное определение функциональной эквивалентности является наиболее сильным, поскольку оно требует совпадения множеств выходных последовательностей для любой входной последовательности и любого момента времени. Определим слабые типы эквивалентности.

Определение 2.9. *Слабая функциональная эквивалентность*

А) Пусть $\Omega = \{\alpha\}$ – заданный класс входных последовательностей. Например, $\alpha_\tau = 1$ – последовательность, добавляющая одну фишку в каждый момент времени. Сети Z и Z' будем называть *эквивалентными по отношению к классу входных последовательностей* Ω и обозначать $Z \overset{\Omega}{\equiv} Z'$, если для любой входной последовательности $\alpha \in \Omega$ множества их выходных последовательностей совпадают $f_Z(\alpha) = f_{Z'}(\alpha)$.

Б) Пусть $\Delta = \tau_1, \tau_2, \ldots$ – последовательность (возможно бесконечная) моментов времени наблюдения. Например, $\Delta = 10$ – единственный момент времени наблюдения равный 10. Сети Z и Z' будем называть *эквивалентными по отношению к моментам наблюдения* Δ и обозначать $Z \overset{\Delta}{\equiv} Z'$, если для любой входной последовательности α множества их входных последовательностей в моменты времени $\tau \in \Delta$ совпадают $f_Z^\tau(\alpha) = f_{Z'}^\tau(\alpha)$.

В) Комбинированный тип эквивалентности назовём слабой функциональной эквивалентностью сетей Петри. Сети Z и Z' будем называть *слабо эквивалентными* по отношению к классу входных последовательностей Ω и момен-

там наблюдения Δ и обозначать $Z \overset{\Omega,\Delta}{\equiv} Z'$, если для любой входной последовательности $\alpha \in \Omega$ множества их входных последовательностей в моменты времени $\tau \in \Delta$ совпадают $f_Z^\tau(\alpha) = f_{Z'}^\tau(\alpha)$.

Исследование слабой эквивалентности целесообразно при разработке систем управления, в которых результат наблюдается только в моменты выдачи управляющих воздействий на объект. Кроме того, специфика работы датчиков, отображаемых на входные позиции, определяет конкретный класс входной последовательности. При композиции функциональных сетей [93] класс входной последовательности подсети задан классом выходной последовательности её входной подсети.

Преобразования сетей для сильного типа эквивалентности могут быть выполнены с помощью эквивалентных преобразований формул уравнений передаточной функции (2.7) на основе законов алгебры временных сетей [154], включающей арифметические, логические и временные операции. Рассмотрим ряд преобразований конкретных сетей Петри, изображенных на рис. 2.10.

Пример 1) $Z_1 \equiv Z_1'$:

$$\begin{cases} \lambda_3 = \lambda_3 \triangleright 1 - w_3 \cdot ((\lambda_3 \triangleright 1)/w_3) + w_3 \cdot ((\lambda_1 \triangleright d_1)/w_1), \\ \lambda_2 = \lambda_2 \triangleright 1 + w_2 \cdot ((\lambda_3 \triangleright d_2)/w_3). \end{cases}$$

$$\lambda_2 = \lambda_2 \triangleright 1 + w_2 \cdot (((\lambda_3 \triangleright 1 - w_3 \cdot ((\lambda_3 \triangleright 1)/w_3) + w_3 \cdot ((\lambda_1 \triangleright d_1)/w_1)) \triangleright d_2)/w_3) =$$

$$\lambda_2 \triangleright 1 + w_2 \cdot (\lambda_3 \triangleright (d_2 + 1))/w_3) - w_2 \cdot w_3 \cdot ((\lambda_3 \triangleright (d_2 + 1)/w_3)/w_3 +$$
$$w_2 \cdot w_3 \cdot ((\lambda_1 \triangleright (d_1 + d_2)/w_1)/w_3) =$$

$$\lambda_2 \triangleright 1 + w_2 \cdot (\lambda_3 \triangleright (d_2 + 1))/w_3) - w_2 \cdot ((\lambda_3 \triangleright (d_2 + 1)/w_3) +$$
$$w_2 \cdot ((\lambda_1 \triangleright (d_1 + d_2)/w_1)) =$$

$$\lambda_2 \triangleright 1 + w_2 \cdot ((\lambda_1 \triangleright (d_1 + d_2)/w_1)).$$

Пример 2) $Z_2 \equiv Z_2'$:

$$\begin{cases} \lambda_3 = \lambda_3 \triangleright 1 - w_3 \cdot ((\lambda_3 \triangleright 1)/w_3) + w_3 \cdot ((\lambda_1 \triangleright d_1)/w_1, \\ \lambda_4 = \lambda_4 \triangleright 1 - w_4 \cdot ((\lambda_4 \triangleright 1)/w_4) + w_4 \cdot ((\lambda_1 \triangleright d_1)/w_1, \\ \lambda_2 = \lambda_2 \triangleright 1 + w_2 \cdot (((\lambda_3 \triangleright d_2)/w_3) \,\&\, ((\lambda_4 \triangleright d_2)/w_4). \end{cases}$$

$$\lambda_2 = \lambda_2 \triangleright 1 + w_2 \cdot ((((\lambda_3 \triangleright 1 - w_3 \cdot ((\lambda_3 \triangleright 1)/w_3) + w_3 \cdot ((\lambda_1 \triangleright d_1)/w_1) \triangleright d_2)/w_3) \,\&$$
$$(((\lambda_4 \triangleright 1 - w_4 \cdot ((\lambda_4 \triangleright 1)/w_4) + w_4 \cdot ((\lambda_1 \triangleright d_1)/w_1) \triangleright d_2)/w_4) =$$

$$\lambda_2 \triangleright 1 + w_2 \cdot ((((\lambda_3 \triangleright 1 - w_3 \cdot ((\lambda_3 \triangleright 1)/w_3) + w_3 \cdot ((\lambda_1 \triangleright d_1)/w_1) \triangleright d_2)/w_3)\ \&$$
$$(((\lambda_4 \triangleright 1 - w_4 \cdot ((\lambda_4 \triangleright 1)/w_4) + w_4 \cdot ((\lambda_1 \triangleright d_1)/w_1) \triangleright d_2)/w_4) =$$

$$\lambda_2 \triangleright 1 + w_2 \cdot (((\lambda_1 \triangleright (d_1 + d_2))/w_1)\ \&\ ((\lambda_1 \triangleright (d_1 + d_2))/w_1) =$$

$$\lambda_2 \triangleright 1 + w_2 \cdot (((\lambda_1 \triangleright (d_1 + d_2))/w_1)\ .$$

Заметим, что примеры 1,2 были ранее рассмотрены в работе [154], где преобразования получены с использованием представления передаточной функции структурно-бесконфликтной сети для частичных сумм последовательностей; результаты преобразований совпадают.

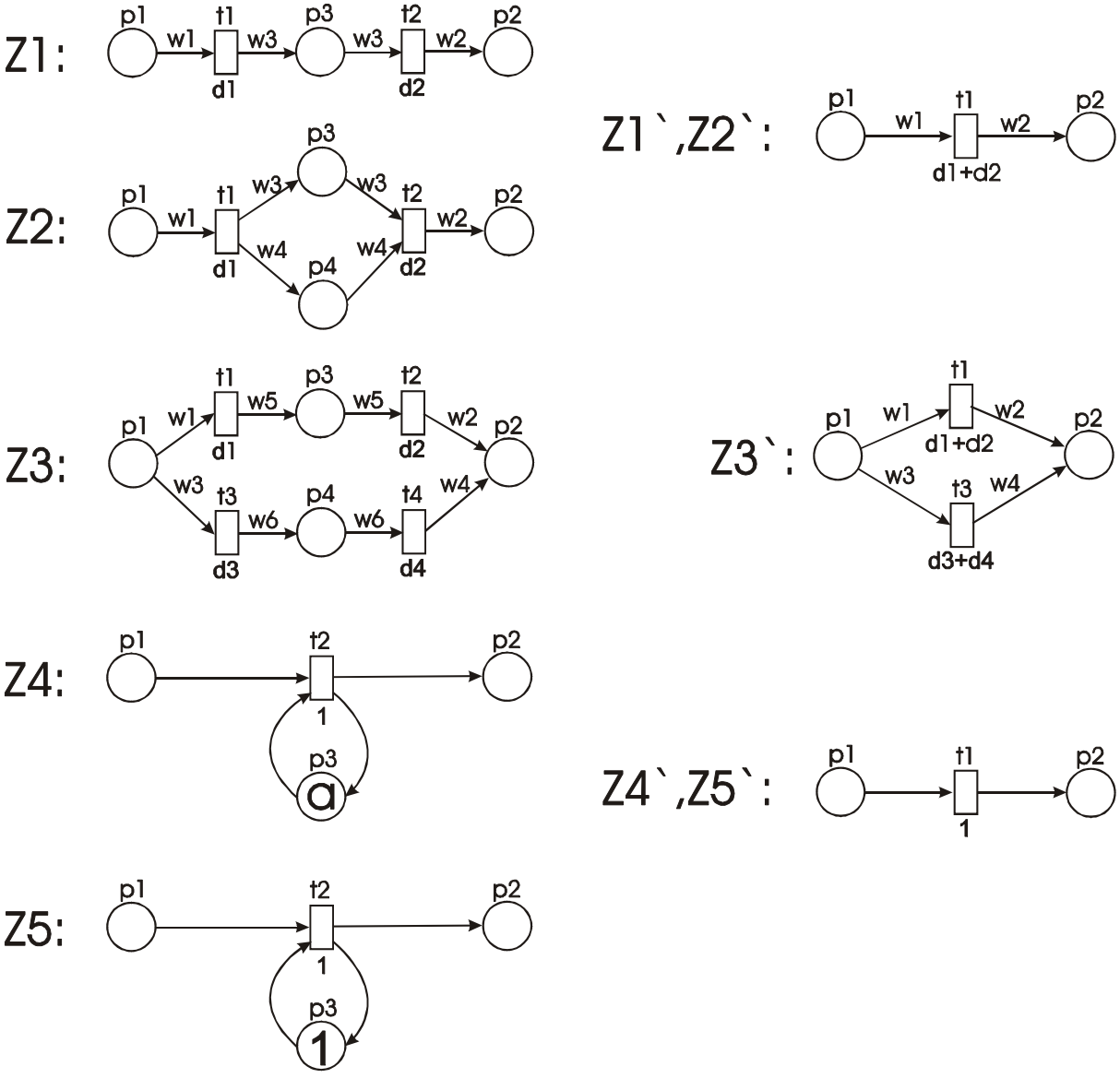


Рис. 2.10. Примеры эквивалентных преобразований

Пример 3) $Z_3 \equiv Z_3'$:

$$\begin{cases} \lambda_3 = \lambda_3 \triangleright 1 - w_5 \cdot ((\lambda_3 \triangleright 1)/w_5) + w_5 \cdot ((\lambda_1 \triangleright d_1)/w_1, \\ \lambda_4 = \lambda_4 \triangleright 1 - w_6 \cdot ((\lambda_4 \triangleright 1)/w_6) + w_6 \cdot ((\lambda_1 \triangleright d_3)/w_1, \\ \lambda_2 = \lambda_2 \triangleright 1 + w_2 \cdot (((\lambda_3 \triangleright d_2)/w_5) + w_4 \cdot ((\lambda_4 \triangleright d_4)/w_6). \end{cases}$$

$$\lambda_2 = \lambda_2 \triangleright 1 + w_2 \cdot ((((\lambda_3 \triangleright 1 - w_5 \cdot ((\lambda_3 \triangleright 1)/w_5) + w_5 \cdot ((\lambda_1 \triangleright d_1)/w_1) \triangleright d_2)/w_5) + w_4 \cdot (((\lambda_4 \triangleright 1 - w_6 \cdot ((\lambda_4 \triangleright 1)/w_6) + w_6 \cdot ((\lambda_1 \triangleright d_3)/w_1) \triangleright d_4)/w_6) =$$

$$\lambda_2 \triangleright 1 + w_2 \cdot ((\lambda_1 \triangleright (d_1 + d_2)/w_1)) + w_4 \cdot ((\lambda_1 \triangleright (d_3 + d_4)/w_1)).$$

Заметим, что в случае конфликтов в сетях использование уравнений (2.7) является корректным только с точки зрения структурных преобразований; при рассмотрении последовательностей фишек они должны быть дополнены неравенствами, представленными в системе (2.6).

Рассмотрим особенности преобразований для слабых типов эквивалентности. Специфические виды входных последовательностей и последовательностей моментов наблюдения позволяют получить дополнительные правила преобразований, приводящие к существенному уменьшению размерности сетей.

Пример 4) Постоянное маркирование входных позиций.

$Z_4 \overset{\Omega_1}{\equiv} Z_4'$, $\Omega_1 = \{\alpha_x^\tau = a \mid \tau \geq 0, a = const\}$:

$$\lambda_3 = \lambda_3 \triangleright 1 - (\lambda_3 \triangleright 1) \,\&\, (\lambda_1 \triangleright 1) + (\lambda_3 \triangleright 1) \,\&\, (\lambda_1 \triangleright 1) = \lambda_3 \triangleright 1 = a.$$

$$\lambda_1 = \lambda_1 \triangleright 1 - (\lambda_1 \triangleright 1) \,\&\, (\lambda_3 \triangleright 1) + a = \lambda_1 \triangleright 1 - (\lambda_1 \triangleright 1) \,\&\, a + a = \lambda_1 \triangleright 1 = a.$$

$$\lambda_2 = \lambda_2 \triangleright 1 + (\lambda_1 \triangleright 1) \,\&\, (\lambda_3 \triangleright 1) = \lambda_2 \triangleright 1 + (\lambda_1 \triangleright 1).$$

Пример 5) Периодическая входная последовательность и периодическое наблюдение.

$$Z_5 \overset{\Omega_2, \Delta_1}{\equiv} Z_5', \ \Omega_2 = \{(\alpha_x^\omega = a \mid \omega = 0, b, 2b, 3b, ...) \,\&\, (\alpha_x^\tau = 0, \tau \neq \omega, a = const)\},$$

$\Delta_1 = \{b, 2b, 3b, ... \mid b = const\}$, $a \leq b$:

Z_5:

$$\lambda_3 = \lambda_3 \triangleright 1 - (\lambda_3 \triangleright 1) \,\&\, (\lambda_1 \triangleright 1) + (\lambda_3 \triangleright 1) \,\&\, (\lambda_1 \triangleright 1) = \begin{cases} 1, \tau \bmod b \le a, \\ 0, \tau \bmod b > a. \end{cases}$$

$$\lambda_2 = \lambda_2 \triangleright 1 + (\lambda_1 \triangleright 1) \,\&\, (\lambda_3 \triangleright 1) = \lambda_2 \triangleright 1 + \left\{ \begin{matrix} 1, \tau \bmod b \le a \\ 0, \tau \bmod > a \end{matrix} \right\}.$$

$$\lambda_2 = \lambda_2 \triangleright b + a \,.$$

Z_5' :

$$\lambda_2' = \lambda_2' \triangleright 1 + \left\{ \begin{matrix} a, \tau \bmod b = 0 \\ 0, \tau \bmod b \ne 0 \end{matrix} \right\}.$$

$$\lambda_2' = \lambda_2' \triangleright b + a \,.$$

Таким образом, в настоящем подразделе получено алгебраическое представление передаточной функции произвольной временной сети Петри. Введены и исследованы слабые типы функциональной эквивалентности, изучены дополнительные правила преобразований сетей для слабых типов эквивалентности, приводящие к существенному уменьшению размера сети.

2.5. Синтез функций непрерывной логики, заданных таблично

Одним из способов представления функций непрерывной логики является таблица выбора [121]. Задачи синтеза функции по таблицам выбора возникают при проектировании гибридных [191] и аналоговых [156] вычислительных устройств, а также при построении уравнений состояний сетей Петри и выполнении их эквивалентных преобразований в соответствии с представленными в предыдущем подразделе методами. Структура исходной таблицы при этом определяется графом сети Петри.

Известны алгоритмы синтеза функций непрерывной логики по таблицам выбора специального вида, например, по таблицам упорядоченного выбора [156]. В [119] отмечено, что общий алгоритм синтеза функции непрерывной логики по произвольной таблице выбора неизвестен. В настоящем подразделе получен критерий, позволяющий определить, задает ли таблица выбора

некоторую функцию непрерывной логики и построен простой алгоритм, выполняющий синтез функции по таблице.

В соответствии с [119] определим алгебру непрерывной логики как квазибулеву алгебру $\Delta = (C, \wedge, \vee, \neg)$, где $C = [A, B]$ – непрерывный отрезок множества вещественных чисел с заданным на нем обычным образом отношением линейного порядка $\leq$; $M = (A + B)/2$ – середина отрезка: называемая медианой. Базовые операции конъюнкции $\wedge$ дизъюнкции $\vee$ и отрицания $\neg$ вводятся для любых $x, y \in C$ следующим образом:

$$x \wedge y = \min(x, y),\ x \vee y = \max(x, y),\ \neg x = 2M - x. \qquad (2.8)$$

Отрезок C рассматривается как множество допустимых значений степени истинности логических переменных; при этом A представляет значение степени истинности абсолютно ложного высказывания: а B – значение степени истинности абсолютно истинного высказывания. Через базовые операции (2.8) можно ввести дополнительные операции: используемые в традиционной логике: импликацию, эквивалентность, исключающую дизъюнкцию и другие. Знак операции конъюнкции далее в формулах алгебры непрерывной логики будем опускать.

Функцией непрерывной логики (ФНЛ) будем называть любую функцию $f : C^n \to C$, образованную путем суперпозиции конечного числа базовых операций, примененных к независимым переменным $x_1, x_2, ..., x_n \in C$.

Из определений базовых операций (2.8) следует, что ФНЛ всегда принимает значение одного из своих аргументов либо его отрицания. Кроме того, значение функции полностью определяется способом упорядочения аргументов и их отрицаний с помощью отношения $\leq$. Таким образом, произвольная ФНЛ может быть однозначно представлена таблицей: в которой перечислены все варианты упорядочения аргументов и их отрицаний и указаны значения функции для каждого варианта. Такие таблицы, в соответствии с [119], будем

называть таблицами выбора. В табл. 2.1 представлена таблица выбора для функции $f = x_1 \vee x_2 \bar{x}_2$.

Таблица 2.1 -

Пример таблицы выбора

Номер области	Область	Значение функции
1	$x_1 \le x_2 \le \bar{x}_2 \le \bar{x}_1$	x_2
2	$x_1 \le \bar{x}_2 \le x_2 \le \bar{x}_1$	$\bar{x}_2$
3	$\bar{x}_1 \le x_2 \le \bar{x}_2 \le x_1$	x_1
4	$\bar{x}_1 \le x_2 \le \bar{x}_2 \le x_1$	x_1
5	$x_2 \le x_1 \le \bar{x}_1 \le x_2$	x_1
6	$x_2 \le \bar{x}_1 \le x_1 \le \bar{x}_2$	x_1
7	$x_2 \le \bar{x}_1 \le x_1 \le x_2$	x_1
8	$\bar{x}_2 \le \bar{x}_1 \le x_1 \le x_2$	x_1

Квазибулева алгебра Δ является дистрибутивной решеткой с псевдодополнениями [191]; в ней выполняется большинство законов, характерных для двоичной логики [119]: коммутативный, ассоциативный, дистрибутивный, Де-Моргана, Клини, поглощения, двойного отрицания, идемпотентности элементов. Справедливость указанных законов может быть установлена, например, с помощью таблиц выбора. Особенностью непрерывной логики является невыполнение законов исключения третьего и противоречия: $x \wedge \bar{x} \neq B$, $x \wedge \bar{x} \neq A$.

В качестве стандартных форм ФНЛ принимают дизъюнктивные (ДНФ) и конъюнктивные (КНФ) нормальные формы [119]. В отличии от аналогичных форм функций двоичной логики элементарные конъюнкты (дизъюнкты) могут вместе с аргументом x содержать и его отрицание $\bar{x}$. Из основных законов непрерывной логики следует, что любая ФНЛ может быть представлена в дизъюнктивной (конъюнктивной) нормальной форме.

Заметим, что способ определения алгебры непрерывной логики как квазибулевой алгебры Δ является в настоящее время наиболее распространенным [119, 156, 191], но не единственным. Так, в бесконечнозначной логике Мак-Нотона [167] использованы две базисные операции отрицания и влечения,

причем результат операции влечения в общем случае не совпадает со значением одного из аргументов либо отрицанием аргумента.

Как было отмечено ранее: любая ФНЛ может быть однозначно представлена соответствующей таблицей выбора. Всего существует $(2n)!$ способов упорядочения n аргументов и их отрицаний. Однако, значения переменной x и ее отрицания $\bar{x}$ симметричны относительно медианы M, т.е. $x_i \le x_j$ тогда и только тогда, когда $\bar{x}_j \le \bar{x}_i$. Таким образом, общее число строк таблицы выбора функции n аргументов [191] равно $L = 2^n \cdot n!$. Для каждого варианта упорядочения функция может принимать независимо одно из $2n$ возможных значений. Поэтому существует $N = (2n)^L$ различных таблиц выбора, задающих некоторую функцию n аргументов.

В работах [119, 191] получены оценки количества различных ФНЛ одного, двух и трех аргументов. В табл. 2.2 эти значения сопоставлены с количествами различных таблиц выбора для соответствующего числа аргументов. Так как в [119, 191] при подсчете учитывались также функции, принимающие значения A и B, количество таблиц выбора вычислялось по формуле $N = (2n+2)^L$. Рассмотрение табл. 2.2 позволяет сделать вывод, что лишь немногие из функций, задаваемых таблицами выбора, являются функциями непрерывной логики.

Таблица 2.2 -

Количество ФНЛ и таблиц выбора

Количество аргументов	Количество ФНЛ	Количество таблиц выбора
1	6	8
2	84	1679616
3	43918	$\approx 223 \cdot 10^{41}$

Во избежание использования в дальнейшем изложении нескольких уровней индексов введем следующий способ представления таблиц выбора. Пусть $\mathbf{x} = (x_1, x_2, ..., x_n)$ – набор аргументов и переменные x_i с индексами i в пре-

делах от $n+1$ до $2n$ обозначают отрицания аргументов: $x_i = \bar{x}_{i-n}$ для $i = \overline{n+1, m}$, где $m = 2n$. Таблицу выбора T будем рассматривать как множество строк $T = \{t\}$, $|T| = L$; $t = (p, a)$ – строка таблицы, $p = (i_1, ..., i_m)$ – последовательность индексов аргументов в описываемом варианте упорядочения: a – индекс аргумента: являющегося значением функции. Таким образом:

$$f(\mathbf{x}) = x_a \text{ при } x \in D_t,$$

где область $D_t \in C^n$ задается неравенством $x_{i_1} \le x_{i_2} \le ... \le x_{i_m}$.

Будем говорить: что строка t таблицы выбора T *перекрывает* строку t', если для подмножеств индексов Φ_t и $\widetilde{\Phi}_{t'}$ имеет место включение $\Phi_t \subseteq \widetilde{\Phi}_{t'}$, где

$$\begin{gathered} t = ((i_1, ..., i_{k-1}, a, i_{k+1}, ..., i_m), a), \\ t' = ((j_1, ..., j_{l-1}, b, j_{l+1}, ..., j_m), b), \\ \Phi_t = \{a, i_{k+1}, ..., i_m\} \ (x_a \le x_i \text{ для } \forall i \in \Phi_t), \\ \widetilde{\Phi}_{t'} = \Phi_{t'} \setminus \{b\} = \{j_{l+1}, ..., j_m\}. \end{gathered} \tag{2.9}$$

Строки t и t' таблицы выбора T будем называть *перекрывающимися*, если одна из них перекрывает другую.

Теорема 2.8. Таблица выбора, задающая функцию непрерывной логики, не содержит перекрывающихся строк.

Доказательство. Выберем произвольно функцию непрерывной логики f. Не ограничивая общности, можно считать, что функция f представлена в дизъюнктивной нормальной форме:

$$f(\mathbf{x}) = \bigvee_q K_q \tag{2.10}$$

где $K_q = \bigwedge_{i \in I_q} x_i$ – элементарные конъюнкты, I_q – множество индексов i переменных x_i, входящих в q-й конъюнкт.

Построим таблицу выбора T для функции f. Предположим, что в таблице T некоторая строка t перекрывает строку t', т.е. $\Phi_t \subseteq \widetilde{\Phi}_{t'}$ в соответствии с (2.9). Пусть наборы аргументов $\mathbf{x} \in D_t$ и $\mathbf{x}' \in D_t$ удовлетворяют строкам t и t' таблицы T соответственно в смысле строгого выполнения неравенств.

Так как $f(\mathbf{x}) = x_a$, то по определению дизъюнкции как максимума значений аргументов, по крайней мере, один из конъюнктов ДНФ (2.10) на наборе $\mathbf{x}$ примет значение x_a. Пусть это будет конъюнкт K_u:

$$K_u = \bigwedge_{i \in I_u} x_i, \; K_u(\mathbf{x}) = x_a$$

Тогда по определению конъюнкции как минимума значений аргументов имеем $I_u \subseteq \Phi_t$ и далее из $\Phi_t \subseteq \widetilde{\Phi}_{t'}$ заключаем

$$I_u \subseteq \widetilde{\Phi}_{t'} \tag{2.11}$$

Рассмотрим значение конъюнкта K_u на наборе $\mathbf{x}'$. Так как $f(\mathbf{x}') = x_b$, то $K_u(\mathbf{x}') \le x_b$, значит, существует $j \in I_u$ такое, что $x_j \le x_b$, следовательно $j \notin \widetilde{\Phi}_{t'}$. Таким образом имеем

$$\neg(I_u \subseteq \widetilde{\Phi}_{t'}) \tag{2.12}$$

Сопоставляя (2.11) и (2.12), приходим к противоречию, что и доказывает невозможность присутствия в таблице выбора, задающей ФНЛ, перекрывающихся строк. □

Пусть задана таблица выбора T, не содержащая перекрывающихся строк. Построим алгоритм, позволяющий выполнить синтез ФНЛ по таблице T.

Определение 2.10. *Конституэнта максимума*

Конституэнтой максимума строки t таблицы T будем называть конъюнкт

$$\phi_t = \bigwedge_{i \in \Phi_t} x_i, \ (x_i \le x_a \ \text{для} \ \forall x_i \in \Phi_t). \tag{2.13}$$

Из определения (2.13) вытекает следующее свойство конституэнт: для любого набора аргументов $\mathbf{x}$, удовлетворяющего строке t ($\mathbf{x} \in D_t$):

$$\phi_t(\mathbf{x}) = x_a. \tag{2.14}$$

Лемма 2.4. Для двух неперекрывающихся строк t и t' таблицы выбора T и для набора аргументов $\mathbf{x}'$, удовлетворяющего t' ($\mathbf{x}' \in D_{t'}$) выполняется неравенство

$$\phi_t(\mathbf{x}') \le \phi_{t'}(\mathbf{x}'). \tag{2.15}$$

Доказательство. Предположим противное. Пусть $\phi_t(\mathbf{x}') > \phi_{t'}(\mathbf{x}')$.

По свойству конституэнт максимума (2.14) имеем

$$\phi_{t'}(\mathbf{x}') = x_b.$$

Тогда

$$x_b < \bigwedge_{x_i \in \Phi_t} x_i.$$

Или, по определению конъюнкции как минимума значений аргументов:

$$x_b < x_i \text{ для } \forall i \in \Phi_t . \tag{2.16}$$

Множество $\widetilde{\Phi}_{t'}$ в силу определения 2.10 составляют такие индексы j аргументов, что

$$x_b < x_j \text{ для } \forall j \in \widetilde{\Phi}_{t'} . \tag{2.17}$$

Тогда, сопоставляя (2.16) и (2.17): имеем $\Phi_t \subseteq \widetilde{\Phi}_{t'}$. Таким образом: строка t таблицы T перекрывает строку t'. Полученное противоречие и завершает оказательство леммы. □

Теорема 2.9. Произвольная таблица выбора T, не содержащая перекрывающихся строк, задает функцию непрерывной логики, равную дизъюнкции конституэнт максимума строк таблицы.

Доказательство. В силу определения конституэнты максимума (2.13) для любой таблицы выбора T можно построить функцию

$$f(\mathbf{x}) = \bigvee_{t \in T} \phi_t . \tag{2.18}$$

Так как таблица T не содержит перекрывающихся строк, то конституэнты ϕ_t обладают свойствами (2.14) и (2.15). Рассмотрим значение функции f на произвольном наборе аргументов $\mathbf{x}' \in C^n$. Любой набор аргументов всегда удовлетворяет одной из строк таблицы T. Пусть набор $\mathbf{x}'$ удовлетворяет строке t' ($\mathbf{x}' \in D_{t'}$). Тогда

$$\phi_{t'}(\mathbf{x}') = x_b \text{ и для } t \in T,\ t \neq t' :\ \phi_t(\mathbf{x}') \leq x_b .$$

Поэтому, в силу определения дизъюнкции как максимума значений аргументов

$$f(\mathbf{x}') = \bigvee_{t \in T} \phi_t(\mathbf{x}') = x_b .$$

Следовательно, значение функции f, заданной соотношением (2.18) совпадает со значением, определяемым таблицей выбора T. В силу произвольности выбора набора аргументов $\mathbf{x}'$, значения совпадают для $\forall \mathbf{x}' \in C^n$. □

Аналогичным образом можно ввести *конституэнты минимума* строк таблицы T:

$$\psi_t = \bigvee_{i \in \Psi_t} x_i \text{, где } \Psi_t = \{i_1, ..., i_{k-1}, a\} \ (x_i \le x_a \text{ для } \forall x_i \in \Psi_t). \tag{2.19}$$

Тогда теорему 2.9 можно представить в двойственной форме, определяющей ФНЛ как конъюнкцию конституэнт минимума строк таблицы:

$$f(\mathbf{x}) = \bigwedge_{t \in T} \psi_t . \tag{2.20}$$

Теорема 2.10. Таблица выбора задает функцию непрерывной логики тогда и только тогда, когда не содержит перекрывающихся строк (теорема является непосредственным следствием теорем 2.8 и 2.9).

По теореме 2.9 алгоритм синтеза ФНЛ по таблице выбора состоит в последовательном выполнении двух шагов:

Шаг 1. Построить конституэнты минимума (2.13) (максимума (2.19)) строк таблицы.

Шаг 2. Построить ДНФ (КНФ) функции как дизъюнкцию (2.18) (конъюнкцию (2.20)) конституэнт максимума (минимума).

Средний размер конституэнты равен n ($n = m/2$). Поэтому общая сложность алгоритма имеет порядок $\circ(L \cdot n)$. Для минимизации полученной функции необходимо использовать методы, изложенные в [191]. Размер строящейся ДНФ (КНФ) можно сократить, если совместить выполнение шагов 1 и 2 алгоритма с применением законов поглощения:

$$x \vee xy = x, \; x \wedge (x \vee y) = x.$$

Непосредственный поиск перекрывающихся строк (2.19) в таблице выбора требует выполнения проверок для каждой из $L^2 - L$ пар различных строк. Однако, в силу определений конституэнт и дизъюнктивной (конъюнктивной) нормальных форм, алгоритм позволяет выполнить синтез функции и для таблиц, содержащих перекрывающиеся строки. В этом случае значения полученной функции на наборах аргументов, удовлетворяющих перекрывающимся строкам, будут отличаться от значений, указанных в таблице выбора.

Этот факт можно использовать для практического синтеза функций. Не проверяя предварительно таблицу выбора T на наличие перекрывающихся строк, выполним синтез ДНФ (КНФ) функции f непрерывной логики. Затем построим таблицу выбора T' функции f. Если T и T' совпадут, то исходная таблица задает ФНЛ f. В противном случае функция, задаваемая таблицей T, не является функцией непрерывной логики. Для аналитического представления таких функций необходимо использовать гибридную логику [156] либо предикатную алгебру выбора [119].

В предикатной алгебре выбора с помощью полученного алгоритма строим последовательность ФНЛ. Для этого повторяем алгоритм для тех строк таблицы, на которых значение построенной функции не совпало с исходным. Каждая ФНЛ задает исходную функцию на подмножестве наборов, не содержащем перекрывающихся строк. Вопросы нахождения оптимального с точки зрения практически значимых критериев разбиения связаны с общими вопросами

минимизации функций предикатной алгебры выбора и выходят за рамки настоящей работы.

Пусть функции F_1 и F_2 двух аргументов заданы таблицами выбора T_1 и T_2 соответственно, где

T_1={ ((1,2,4,3),2), ((1,4,2,3),2), ((3,2,4,1),4), ((3,4,2,1),4),
((2,1,3,4),1), ((2,3,1,4),1), ((4,1,3,2),3), ((4,3,1,2),3) };
T_2={ ((1,2,4,3),2), ((1,4,2,3),3), ((3,2,4,1),2), ((3,4,2,1),4),
((2,1,3,4),1), ((2,3,1,4),3), ((4,1,3,2),4), ((4,3,1,2),3) }.

Табл. 2.3 содержит представление F_1 и F_2 в наглядной форме. Выполним синтез указанных функций.

Конституэнты максимума функции F_1 имеют вид:

$$\phi_1 = x_2\bar{x}_2\bar{x}_1,\ \phi_2 = x_2\bar{x}_1,\ \phi_3 = \bar{x}_2x_1,\ \phi_4 = \bar{x}_2x_2x_1,$$
$$\phi_5 = x_1\bar{x}_1\bar{x}_2,\ \phi_6 = x_1\bar{x}_2,\ \phi_7 = \bar{x}_1x_2,\ \phi_8 = \bar{x}_1x_1x_2.$$

В силу коммутативности операции конъюнкции, ϕ_7 равна ϕ_2; ϕ_6 равна ϕ_3; ϕ_2 поглощает ϕ_1 и ϕ_8; ϕ_3 поглощает ϕ_4 и ϕ_5. Таким образом, получим ДНФ

$$f_1 = \phi_2 \vee \phi_3 = x_1\bar{x}_2 \vee \bar{x}_1x_2.$$

Таблица функции f_1 совпадает с таблицей функции F_1 (табл. 2.3). Таким образом, F_1 является функцией непрерывной логики, а f_1 – её дизъюнктивная нормальная форма. Отсутствие перекрывающихся строк в таблице T_1 можно установить также путем непосредственной проверки.

Построим конституэнты максимума функции F_2:

$$\phi_1 = x_2\bar{x}_2\bar{x}_1,\ \phi_2 = \bar{x}_1,\ \phi_3 = x_2\bar{x}_2x_1,\ \phi_4 = \bar{x}_2x_2x_1,$$

$\phi_5 = x_1\bar{x}_1\bar{x}_2$, $\phi_6 = x_1\bar{x}_1\bar{x}_2$, $\phi_7 = \bar{x}_2x_1\bar{x}_1x_2$, $\phi_8 = \bar{x}_1x_1x_2$.

Конституэнта ϕ_3 равна ϕ_4; ϕ_5 равна ϕ_6; ϕ_2 поглощает ϕ_1, ϕ_5, ϕ_7, ϕ_8.

Таблица 2.3 -

Таблица выбора ФНЛ

Номер	Область	F_1	F_2	f_1	f_2	f_1^1	f_1^2	g_2^1	g_2^2	g_2^3
1	$x_1 \le x_2 \le \bar{x}_2 \le \bar{x}_1$	x_2	x_2	x_2	$\bar{x}_1$		x_2			x_2
2	$x_1 \le \bar{x}_2 \le x_2 \le \bar{x}_1$	x_2	$\bar{x}_1$	x_2	$\bar{x}_1$	$\bar{x}_1$		$\bar{x}_1$		
3	$\bar{x}_1 \le x_2 \le \bar{x}_2 \le x_1$	$\bar{x}_2$	x_2	$\bar{x}_2$	x_2	x_2				x_2
4	$\bar{x}_1 \le x_2 \le \bar{x}_2 \le x_1$	$\bar{x}_2$	$\bar{x}_2$	$\bar{x}_2$	$\bar{x}_2$	$\bar{x}_2$				$\bar{x}_2$
5	$x_2 \le x_1 \le \bar{x}_1 \le x_2$	x_1	x_1	x_1	$\bar{x}_1$		x_1			x_1
6	$x_2 \le \bar{x}_1 \le x_1 \le \bar{x}_2$	x_1	$\bar{x}_1$	x_1	$\bar{x}_1$	$\bar{x}_1$				$\bar{x}_1$
7	$x_2 \le \bar{x}_1 \le x_1 \le x_2$	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_1$	$\bar{x}_1$		$\bar{x}_2$		$\bar{x}_2$	
8	$\bar{x}_2 \le \bar{x}_1 \le x_1 \le x_2$	$\bar{x}_1$	$\bar{x}_1$	$\bar{x}_1$	$\bar{x}_1$	$\bar{x}_1$				$\bar{x}_1$

Построим ДНФ

$f_2 = \phi_2 \vee \phi_3 = \bar{x}_1 \vee x_1x_2\bar{x}_2$.

Таблица функции f_2 не совпадает с таблицей F_2 в областях 1, 5, 7 (табл. 2.3). Действительно, строка 2 таблицы T_2 перекрывает строки 1, 5, 7, а строка 8 перекрывает строку 7:

$\Phi_2 \subseteq \tilde{\Phi}_1$, $\Phi_2 \subseteq \tilde{\Phi}_5$, $\Phi_2 \subseteq \tilde{\Phi}_7$, $\Phi_8 \subseteq \tilde{\Phi}_7$,

где Φ_2={3}, Φ_8={3,1,2}, $\tilde{\Phi}_1$={4,3}, $\tilde{\Phi}_5$={3,4}, $\tilde{\Phi}_7$={1,3,2}.

Таким образом, F_2 не является функцией непрерывной логики. Для представления F_2 в предикатной алгебре выбора можно воспользоваться одной из следующих форм записи:

$$F_2 = \begin{cases} f_2^1 = \bar{x}_1 \vee x_1 x_2 \bar{x}_2, & \mathbf{x} \in D_2, D_3, D_4, D_6, D_8, \\ f_2^2 = x_2 \bar{x}_2 x_1 \vee x_1 \bar{x}_1 x_2, & \mathbf{x} \in D_1, D_5, D_7. \end{cases} \quad (2.21)$$

$$F_2 = \begin{cases} g_2^1 = \bar{x}_1, & \mathbf{x} \in D_2, \\ g_2^2 = \bar{x}_2, & \mathbf{x} \in D_7, \\ g_2^3 = x_1 \bar{x}_2 \vee x_2 \bar{x}_2, & \mathbf{x} \in D_1, D_3, D_4, D_5, D_6, D_8. \end{cases} \quad (2.22)$$

Представления (2.21) и (2.22) функции F_2 соответствуют различным способам разбиения таблицы T_2 на подмножества неперекрывающихся строк. Функции непрерывной логики f_2^i и g_2^j синтезируются отдельно для каждого из подмножеств неперекрывающихся строк таблицы.

2.6. Эффективная реализация алгоритма декомпозиции на функциональные подсети

Программная реализация алгоритма декомпозиции 2.1 требует выбора структур данных, обеспечивающих эффективную обработку информации [107]. Традиционным является представление сети Петри с помощью матрицы инцидентности, либо, для сетей с петлями – парой матриц, задающими отдельно входящие и исходящие дуги переходов. Декомпозиция предназначена для обработки сетей большой размерности. Как правило, матрицы инцидентности таких сетей является разрежёнными. Стандартное представление разрежённых матриц использует массив (список) ненулевых элементов, соответствующий множеству дуг сети Петри. Например, возможно описание сети двумя массивами с элементами вида:

$$(np,\ nt),\ (nt,\ np),$$

где np – номер позиции, а nt – номер перехода; значение ненулевого элемента не указывается, так как для сетей без кратных дуг оно равно единице.

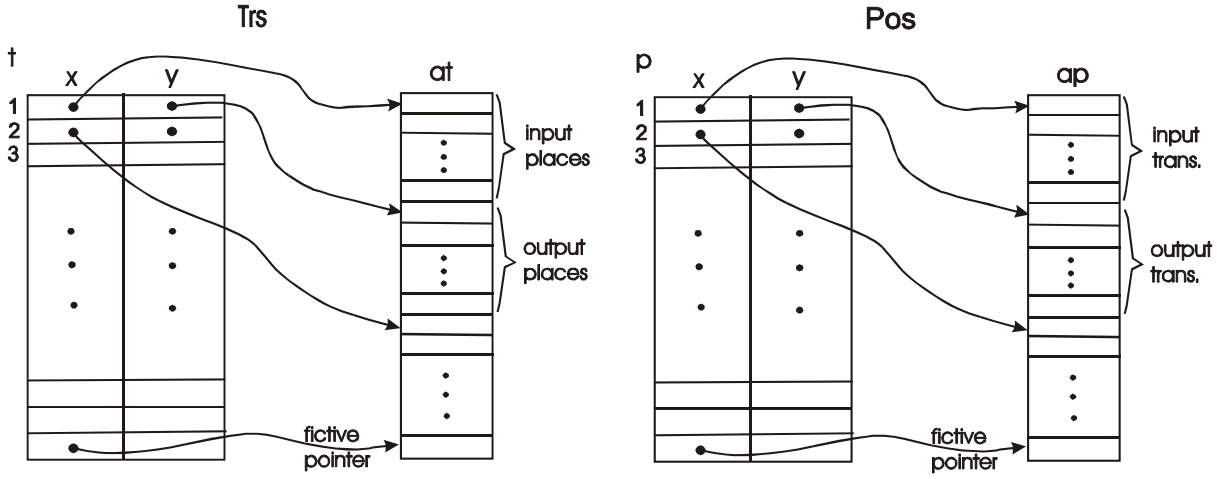


Рис. 2.11. Основные структуры данных

Рассмотрим операции, наиболее часто выполняемые алгоритмом декомпозиции 2.1. При построении порождённой подсети $B(R)$ на шаге 1 – это поиск входящих и исходящих позиций для переходов множества R: $X=\{p\,|\,p\in{}^\bullet R\wedge p\notin R^\bullet\}$, $Y=\{p\,|\,p\notin{}^\bullet R\wedge p\in R^\bullet\}$, $Q=\{p\,|\,p\in{}^\bullet R\wedge p\in R^\bullet\}$. При формировании множества поглощаемых переходов на шаге 2 – это поиск исходящих, входящих и смежных переходов для множеств X,Y,Q соответственно: $X^\bullet$, ${}^\bullet Y$, ${}^\bullet Q^\bullet$. Таким образом, основными операциями алгоритма являются операции нахождения множеств ${}^\bullet t$, $t^\bullet$, ${}^\bullet t^\bullet$, ${}^\bullet p$, $p^\bullet$, ${}^\bullet p^\bullet$.

Применение упорядоченных массивов входящих и исходящих дуг требует дополнительные массивы указателей, для быстрого обращения к элементам, описывающим дуги, инцидентные выбранной вершине. Поэтому, для представления сети Петри выбраны структуры данных, представленные на Рис. 2.11. Каждый из двух типов `Trs` и `Pos` является полным представлением сети Петри, но тип `Trs` обеспечивает быстрый доступ к дугам, инцидентным переходам, а тип `Pos` – к дугам, инцидентным позициям. Такое незначительное дублирование информации оправдано, так как позволяет избежать выполнения поиска информации при реализации основных операций алгоритма.

Рассмотрим более подробно тип данных `Trs`. Массив `t` индексируется номером перехода и содержит указатели на входные `x` и выходные `y` позиции.

Инцидентные позиции для всех переходов располагаются в массиве `at`. При этом позиции следуют в таком порядке: сначала входные позиции первого перехода, затем выходные позиции первого перехода, затем входные позиции второго перехода и выходные позиции второго перехода и так далее. Таким образом, следующие операторы циклов

$p \in {}^{\bullet}t_i$: `for( j=t[i].x; j<t[i].y; ++j ) {p=at[j];}`

$p \in t_i^{\bullet}$: `for( j=t[i].y; j<t[i+1].x; ++j ) {p=at[j];}`

$p \in {}^{\bullet}t_i^{\bullet}$: `for( j=t[i].x; j<t[i+1].x; ++j ) {p=at[j];}`

обеспечивают обработку входных, выходных и инцидентных позиций перехода с номером `i`. Аналогичным образом организован тип данных `Pos`. Для обеспечения описанных способов использования типов `Trs` и `Pos` необходима одна фиктивная последняя запись указателя `x`.

Выберем структуры данных для представления минимальных функциональных подсетей. Поскольку в соответствии с теоремой 2.2 множество минимальных функциональных подсетей задаёт разбиение множества переходов на непересекающиеся подмножества, в качестве основного результата работы алгоритма будем рассматривать индикатор подсети, представленный массивом `SubNet` имеющим размерность множества переходов. Массив задаёт принадлежность переходов минимальным функциональным подсетям таким образом, что значение элемента `SubNet[i]` равняется номеру подсети, к которой принадлежит переход t_i. Подсети нумеруются начиная с единицы. В программной реализации дополнительная информация о декомпозиции представлена массивами `SX` и `SY`, являющимися индикаторами входных и выходных позиций соответственно.

Кроме того, на текущем проходе алгоритма формируемая минимальная функциональная подсеть, представлена массивами `R`, `X`, `Q`, `Y`, соответст-

вующими элементам определения функциональной подсети. А множество поглощаемых переходов – массивом S.

Описанный алгоритм декомпозиции (рис. 2.12) реализован в программе Deborah (Приложение Д), которая разработана как встраиваемый модуль (плагин) для известной системы анализа сетей Петри Tina [10].

```
SubNet=0; iSubNet=1;
цикл (построение подсетей)
    для (t∈T : SubNet[t]=0) положить { R={t}; SubNet[t]=iSubNet }
    цикл (поглощение переходов)
        Построение порождённой подсети B(R):
           X=Q=Y=∅
           для всех позиций p∈P:
                pinp=pout=false;
                для всех переходов t∈R:
                     если p∈•t то pinp=true;
                     если p∈t• то pout=true;
                если pinp=true & pout=true то Q=Q∪{p}
                если pinp=true то X=X∪{p}
                если pout=true то Y=Y∪{p}
        Построение множества поглощаемых переходов S:
        S=∅
        Проверка выходных переходов X:
           для всех позиций p∈X
                для всех переходов t∈p•
                     если t∉R∧t∉S то { S=S∪{t}; SubNet[t]=iSubNet }
        Проверка входных переходов Y:
        для всех позиций p∈Y
                для всех переходов t∈•p
                     если t∉R∧t∉S то { S=S∪{t}; SubNet[t]=iSubNet }
        Проверка инцидентных переходов Q:
                для всех позиций p∈Q
                     для всех переходов t∈•p•
                          если t∉R∧t∉S то { S=S∪{t}; SubNet[t]=iSubNet }
        R=R∪S
    повторять пока S≠∅
     iSubNet++
повторять пока существует переход t∈T такой что iSubNet[t]=0
```

Рис. 2.12. Детализированный алгоритм декомпозиции

Характеристики программы при работе с сетями большой размерности исследованы на специально генерированных случайных двудольных орграфах. Кроме того, выполнена декомпозиция моделей Петри таких реальных объектов как: телекоммуникационные протоколы BGP, TCP, IOTP (раздел 4); химические реакции [47] апоптоза, метаболизма, гликолиза; процессы обработки транзакций в системе Transit [90] регистрации движения ценностей в странах Европейского союза и Европейского региона свободной торговли (EFTA).

Выводы к 2 разделу

1. Впервые введены понятия функциональной сети Петри и функциональной подсети. Показано, что порождающее семейство (базис) функциональных подсетей составляет множество минимальных функциональных подсетей.

2. Для представления декомпозиции впервые введены сеть функциональных подсетей и графы функциональных подсетей. Показано, что сеть функциональных подсетей является маркированным графом.

3. Разработан универсальный метод декомпозиции сетей Петри на основе логических уравнений, который может быть применён для декомпозиции на различные классы подсетей с контактными позициями. Разработан алгоритм декомпозиции на минимальные функциональные подсети с линейной временной сложностью.

4. Впервые введены понятия слабой эквивалентности функциональных подсетей. Получено представление передаточной функции функциональной временной сети Петри, позволяющее выполнять редукцию сетей Петри для слабых типов эквивалентности.

5. Впервые разработан метод синтеза функций непрерывной логики, заданных таблично. Метод применён при построении уравнения состояний и формул передаточной функции временных сетей Петри с кратными дугами.

6. Построены эффективные алгоритмы декомпозиции на функциональные подсети (кланы). Выполнена их программная реализация Deborah. Обеспечена переносимость модуля Deborah на различные платформы, а также встраивание в известную систему анализа сетей Петри Tina.

РАЗДЕЛ 3

КЛАНЫ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

Выполнено обобщение функциональной подсети для произвольной системы линейных алгебраических уравнений над кольцом со знаком, названное кланом. Разработаны методы ускорения процессов решения линейных систем с помощью композиции кланов. Методы предназначены для ускорения процессов верификации сложных телекоммуникационных протоколов.

3.1. Алгебраические методы анализа сетей Петри

Пусть $N=(P,T,F,W),$ – сеть Петри; $|P|=m$, $|T|=n$, и множества позиций и переходов занумерованы. Следующие матрицы A^-, A^+

$$A^- = \left\| a^-{}_{i,j} \right\|,\ i=\overline{1,m},\ j=\overline{1,n};\ a^-{}_{i,j} = \begin{cases} w(p_i,t_j),\ (p_i,t_j)\in F, \\ 0,\ (p_i,t_j)\notin F. \end{cases}$$

$$A^+ = \left\| a^+{}_{i,j} \right\|,\ i=\overline{1,m},\ j=\overline{1,n};\ a^+{}_{i,j} = \begin{cases} w(t_j,p_i),\ (t_j,p_i)\in F, \\ 0,\ (t_j,p_i)\notin F. \end{cases}$$

называют матрицами входящих и исходящих дуг переходов соответственно. Матрицу $A=A^+-A^-$ называют *матрицей инцидентности* сети Петри.

Уравнение

$$\overline{\mu}=\overline{\mu}_0+A\cdot\overline{\sigma} \text{ либо } \overline{y}\cdot A^T=\Delta\overline{\mu} \qquad (3.1)$$

называют *фундаментальным уравнением сети Петри*, где $\Delta\overline{\mu}=\overline{\mu}-\overline{\mu}_0, \overline{y}=\overline{\sigma}$, $\overline{\sigma}$ – вектор счёта допустимых последовательностей срабатываний переходов. Раз-

решимость фундаментального уравнения в целых неотрицательных числах является необходимым условием достижимости заданной маркировки.

p-инвариантом сети Петри [169] называют целые неотрицательные решения системы

$$\bar{x} \cdot A = 0 . \qquad (3.2)$$

t-инвариантом называют целые неотрицательные решения системы

$$\bar{y} \cdot A^T = 0 .$$

Инварианты играют ключевую роль при исследовании таких свойств сетей как ограниченность, консервативность, живость [30, 39, 79, 169, 180]. *Сеть инвариантна*, если она имеет инвариант, все компоненты которого натуральные числа. Известно, что p-инвариантная сеть консервативна и ограничена, причём эти свойства являются структурными, то есть выполняются при любой начальной маркировке. t-инвариант представляет постоянные последовательности срабатываний переходов. Наличие таких последовательностей является необходимым условием живости ограниченной сети. Так как в соответствии с [169] каждый t-инвариант сети Петри является p-инвариантом двойственной сети, далее, не ограничивая общности, мы будем рассматривать только p-инварианты.

Следует отметить, что к решению систем линейных алгебраических уравнений и неравенств сводится большинство известных задач анализа свойств сетей Петри [30, 39, 79, 169, 180]. Необходимые и достаточные условия для основных структурных свойств сетей Петри представлены в табл. 3.1.

Таблица 3.1 -

Необходимые и достаточные условия для структурных свойств сетей Петри

Структурное свойство	Необходимые и достаточные условия
Ограниченность	$\exists \bar{x} > 0, \ \bar{x} \cdot A \le 0$
Консервативность	$\exists \bar{x} > 0, \ \bar{x} \cdot A = 0$
Повторяемость	$\exists \bar{y} > 0, \ A \cdot \bar{y} \ge 0$
Постоянство	$\exists \bar{y} > 0, \ A \cdot \bar{y} = 0$

При исследовании свойств сетей Петри со свободным выбором [27, 30, 169] широко применяют такие структурные элементы как сифоны и ловушки. Непустое подмножество позиций S сети N называют *сифоном*, если ${}^{\bullet}S \subseteq S^{\bullet}$. Непустое подмножество позиций Q сети N называют *ловушкой*, если $S^{\bullet} \subseteq {}^{\bullet}S$. Сеть со свободным выбором жива тогда и только тогда, когда каждый сифон в ней содержит маркированную ловушку. Характеристические векторы сифонов и ловушек могут быть найдены как $\{0,1\}$ решения систем неравенств

$$\bar{s} \cdot D \le 0 \ \text{ и } \ \bar{q} \cdot D' \le 0,$$

где D и D' – модифицированные матрицы инцидентности сети.

В [145] построено фундаментальное уравнение и введены понятия частичных и полных инвариантов состояния и поведения для временных сетей Петри.

Допустимой последовательностью запусков переходов временной сети Петри названа последовательность $\sigma = U(1), U(2), ..., U(\tau)$, удовлетворяющая уравнению состояний (2.5), где $U(\tau) = \{u_t(\tau - \theta) \mid \theta = \overline{0, d_t - 1}, t \in T\}$.

Следующее уравнение названо *фундаментальным уравнением временной сети Петри*.

$$\bar{\mu}(\tau) = \bar{\mu}_0 + A^+ \cdot \bar{\gamma}^+ - A^- \cdot \bar{\gamma}^-, \qquad (3.3)$$

где $\bar{\gamma}^-(\sigma) = \left\| \gamma_t^-(\sigma) \mid t \in T \right\|$ и $\bar{\gamma}^+(\sigma) = \left\| \gamma_t^+(\sigma) \mid t \in T \right\|$ – *векторы счёта запусков и завершений переходов* соответственно и

$$\gamma_t^-(\sigma) = \sum_{\theta=1,\tau} u_t(\theta),\ \gamma_t^+(\sigma) = \sum_{\theta=1,\tau} u_t(\theta - d_t).$$

Объединив матрицы входящих и исходящих дуг, получим более компактное представление уравнения (3.3):

$$\bar{\mu}(\tau) = \bar{\mu}_0 + B \cdot \bar{\gamma}, \qquad (3.4)$$

где $B = \left\| A^+, -A^- \right\|$, $\bar{\gamma} = \left\| \begin{matrix} \bar{\gamma}^+ \\ \bar{\gamma}^- \end{matrix} \right\|$.

Заметим, что хотя уравнение (3.4) по форме совпадает с фундаментальным уравнением базовой сети Петри, матрица B и вектор $\bar{\gamma}$ имеют отличную размерность и структуру. Для классической сети в уравнении (3.1) используется матрица инцидентных нагрузок $A = A^+ - A^-$ и вектор счёта срабатываний $\bar{y} = \bar{\gamma}^+ = \bar{\gamma}^-$.

Частичный инвариант состояния временной сети Петри введен как инвариант позиций соответствующей базовой сети Петри, а именно как целое неотрицательное решение уравнения

$$\bar{x} \cdot A = 0$$

Взвешенная сумма фишек частично инвариантной временной сети не возрастает в процессе её функционирования:

$$\bar{x} \cdot \bar{\mu} \le \bar{x} \cdot (\bar{\mu}_0 + A^+ \cdot \bar{c}) = L_{\bar{x}} = const,$$

где вектор $\bar{c}$ задаёт ограничения на количество каналов переходов:

$$\sum_{\theta=0,d_t-1} u_t(\tau-\theta) \le c_t, t \in T, \tau = 1,2,... \qquad (3.5)$$

Полный инвариант состояния временной сети Петри определен как целый неотрицательный вектор $\bar{z}$, являющийся решением уравнения

$$\bar{z} \cdot H = 0,$$

где $H = \left\| \begin{matrix} B \\ R \end{matrix} \right\| = \left\| \begin{matrix} A^+, -A^- \\ -E, E \end{matrix} \right\|$, $R = \|-E, E\|$, E – единичная матрица.

Основное свойство полного инварианта состояния заключается в сохранении взвешенной суммы компонентов состояния:

$$\bar{z} \cdot \bar{\pi} = \bar{z} \cdot \bar{\pi}_0 = L_{\bar{z}} = const,$$

где $\bar{\pi} = \|\bar{\mu}, \bar{\rho}\|$, $\rho_t(\tau) = \sum_{\theta=0,d_t-1} u_t(\tau-\theta)$.

Полный инвариант поведения временной сети Петри определен как инвариант переходов соответствующей базовой сети, а именно, как целое неотрицательное решение уравнения

$$A \cdot \bar{y} = 0.$$

Полный инвариант поведения временной сети описывает векторы счёта запусков и завершений последовательностей срабатывания переходов, приводящих сеть в начальное состояние $S_0 \xrightarrow{\sigma} S_0$.

Частичный инвариант поведения временной сети Петри определен как целый неотрицательный вектор, являющийся решением уравнения

$$B \cdot \overline{g} = 0 .$$

При наличии ограничений на количество каналов переходов сети (3.5) также требуется, чтобы инвариант удовлетворял неравенству

$$\left| \overline{g}^{+} - \overline{g}^{-} \right| \le \overline{c} .$$

Частичный инвариант представляет собой вектор счёта запусков и завершений переходов для последовательностей запусков переходов, приводящих к повторению лишь начальной маркировки.

Правильной последовательностью частичных инвариантов поведения временной сети Петри будем называть такую последовательность, сумма элементов которой является полным инвариантом сети:

$$\overline{g}_1, \overline{g}_2, ..., \overline{g}_k : \sum_{i=1,k} \overline{g}_i^{+} = \sum_{i=1,k} \overline{g}_i^{-} = \overline{y} .$$

Правильные последовательности частичных инвариантов позволяют организовать полные циклические процессы с повторением начальной маркировки внутри цикла. Такая последовательность действий является основой для построения эффективных штатных режимов функционирования систем [25, 180].

Таким образом, задачи нахождения решений уравнения состояний, полных и частичных инвариантов состояния и поведения, построения правильных последовательностей частичных инвариантов временных сетей Петри также сводятся к решению систем линейных алгебраических уравнений и неравенств.

Как показано в [162-166], система, содержащая уравнения и неравенства, может быть сведена к системе уравнений. Отметим, что указанным преобразо-

ваниям соответствуют модификация исходной сети таким образом, что нахождение того либо иного свойства можно рассматривать, как определение p-инвариантов модифицированной сети. Поэтому, в дальнейшем изложении, не ограничивая общности, мы будем решать однородное уравнение вида (3.2) для нахождения структурных свойств и неоднородное уравнение вида (3.1) для нахождения поведенческих свойств сетей Петри.

Кроме того, в соответствии с [162] будем представлять общее решение однородной системы, как линейную комбинацию с целыми неотрицательными коэффициентами базисных решений. Заметим, что базис состоит из минимальных в целой неотрицательной решётке решений системы. В отличие от классической теории линейных систем для представления общего решения неоднородной системы в целых неотрицательных числах необходимо использовать не одно произвольное, а множество минимальных частных решений.

Все известные методы решения систем линейных диофантовых уравнений в целых неотрицательных числах [22, 23, 67, 77, 88, 162] имеют асимптотически экспоненциальную сложность, что затрудняет, а в некоторых случаях делает практически неосуществимым их применение для анализа реальных систем. Далее выполнено построение композиционных методов решения линейных систем, позволяющих существенно ускорить вычисления.

3.2. Кланы линейных систем как обобщение функциональных сетей Петри

Рассмотрим линейную однородную систему из m уравнений с n неизвестными

$$A \cdot \bar{x} = 0, \tag{3.6}$$

где A – матрица коэффициентов размерности $m \times n$, $\bar{x}$ – вектор-столбец неизвестных размерности n. Не будем указывать точно множества значений переменных и коэффициентов. Предположим только, что известен метод, позволяющий решить систему (3.6) и представить общее решение в форме

$$\bar{x} = G \cdot \bar{y}, \tag{3.7}$$

где G – матрица базисных решений, а $\bar{y}$ – вектор-столбец свободных переменных. Каждый из столбцов матрицы G является базисным решением. Для получения частного решения системы, значения компонентов вектора $\bar{y}$ могут быть выбраны произвольно из множества значений переменных $\bar{x}$. Заметим, что система (3.6) всегда имеет, по крайней мере, тривиальное нулевое решение. Для краткости мы будем далее называть однородную систему несовместимой, если она имеет только тривиальное решение.

Рассмотрим неоднородную систему

$$A \cdot \bar{x} = \bar{b}, \tag{3.8}$$

где $\bar{b}$ – вектор-столбец свободных членов размерности m. Будем предполагать также, что существует метод решения системы (3.8), позволяющий представить общее решение в форме

$$\bar{x} = \bar{x}' + G \cdot \bar{y}, \tag{3.9}$$

где $G \cdot \bar{y}$ – общее решение соответствующей однородной системы (3.6), а $\bar{x}'$ – минимальное частное решение неоднородной системы (3.8).

В соответствии с классической алгеброй [115], приведенные выше результаты справедливы для произвольных колец (полей) со знаком. Кроме того, в соответствии с [162] они также справедливы для решений структуры моноида

при структуре кольца элементов матрицы A и вектора $\bar{b}$. В последнем случае, для неоднородной системы следует рассматривать множество минимальных решений $\{\bar{x}'\}$.

Представим систему (3.6) в виде предиката

$$S(\bar{x}) = L_1(\bar{x}) \wedge L_2(\bar{x}) \wedge ... \wedge L_m(\bar{x}), \tag{3.10}$$

где $L_i(\bar{x})$ – уравнения системы:

$$L_i(\bar{x}) = (\bar{a}^i \cdot \bar{x} = 0),$$

$\bar{a}^i$ – i-я строка матрицы A. Будем предполагать также, что $\bar{a}^i$ – ненулевой вектор, то есть, по крайней мере, один из компонентов $\bar{a}^i$ ненулевой. Обозначим X множество неизвестных системы. Рассмотрим множество уравнений $\Im = \{L_i\}$ системы S. Введём отношения на множестве $\Im$.

Определение 3.1. *Отношение близости.* Два уравнения $L_i, L_j \in \Im$ *близки* и обозначаются как $L_i \circ L_j$ если и только если $\exists x_k \in X:\ a_{i,k}, a_{j,k} \neq 0$, $sign(a_{i,k}) = sign(a_{j,k})$.

Утверждение 3.1. Отношение близости рефлексивно и симметрично.

Доказательство.

a) Рефлексивность: $L_i \circ L_i$. Так как $\bar{a}^i$ – ненулевой вектор, то существует $x_k \in X:\ a_{i,k} \neq 0$. Тогда тривиально $sign(a_{i,k}) = sign(a_{i,k})$.

b) Симметричность: $L_i \circ L_j \Rightarrow L_j \circ L_i$. Отношение симметрично в соответствии с симметричностью отношения равенства: $sign(a_{i,k}) = sign(a_{j,k}) \Rightarrow sign(a_{j,k}) = sign(a_{i,k})$.

□

Определение 3.2. *Отношение клана.* Два уравнения $L_i, L_j \in \Im$ *связаны отношением клана*, что обозначается $L_i \circ L_j$, если и только если существует последовательность (возможно пустая) уравнений $L_{l_1}, L_{l_2}, ..., L_{l_k}$ таких что: $L_i \circ L_{l_1} \circ ... \circ L_{l_k} \circ L_j$. Заметим, что отношение клана представляет собой транзитивное замыкание отношения близости.

Теорема 3.1. Отношение клана является отношением эквивалентности.

Доказательство. Требуется доказать, что отношение клана рефлексивно, симметрично и транзитивно.

a) Рефлексивность: $L_i \circ L_i$. Так как определение 3.1 допускает пустые последовательности уравнений и отношение близости рефлексивно в соответствии с утверждением 3.1, то и отношение клана рефлексивно.

b) Симметричность: $L_i \circ L_j \Rightarrow L_j \circ L_i$. Так как $L_i \circ L_j$, то в соответствии с определением 3.2 существует последовательность $L_{l_1}, L_{l_2}, ..., L_{l_k}$ такая что $L_i \circ L_{l_1} \circ ... \circ L_{l_k} \circ L_j$. Так как отношение близости симметрично в соответствии с утверждением 3.1, то для обратной последовательности $L_{l_k}, L_{l_{k-1}}, ..., L_{l_1}$ выполняется $L_j \circ L_{l_k} \circ ... \circ L_{l_1} \circ L_i$ тогда $L_j \circ L_i$.

c) Транзитивность: $L_i \circ L_j, L_j \circ L_l \Rightarrow L_i \circ L_l$. Так как $L_i \circ L_j$, $L_j \circ L_l$, то в соответствии с определением 3.2 существуют $\sigma = L_{i_1}, L_{i_2}, ..., L_{i_k}$ и $\varsigma = L_{l_1}, L_{l_2}, ..., L_{l_r}$ такие что $L_i \circ L_{i_1} \circ ... \circ L_{i_k} \circ L_j$ и $L_j \circ L_{l_1} \circ ... \circ L_{l_r} \circ L_l$. Рассмотрим конкатенацию $\sigma L_j \varsigma$. Эта последовательность связывает элементы L_i и L_l цепочкой из элементов, удовлетворяющих отношению близости. Таким образом $L_i \circ L_l$.

□

Следствие. Отношение клана задаёт разбиение множества $\Im$: $\Im = \bigcup_j C^j$, $C^i \bigcap C^j = \varnothing$, $i \neq j$.

Определение 3.3. *Клан.* Элемент разбиения $\{\Im, \circ\}$ будем называть *кланом* и обозначать C^j.

Определение 3.4. Переменные $X^j = X(C^j) = \{x_i \,|\, x_i \in X, \exists L_k \in C^j : a_{k,i} \neq 0\}$ будем называть *переменными клана* C^j. Переменные $x_i \in X(C^j)$ являются *внутренними переменными клана* C^j, если и только если для всех остальных кланов C^l, $l \neq j$ выполняется $x_i \notin X^l$. Множество внутренних переменных клана C^j будем обозначать $\hat{X}^j$. Переменная $x_i \in X$ является *контактной переменной* если и только если существуют такие кланы C^j и C^l, что $x_i \in X^j$, $x_i \in X^l$. Множество всех контактных переменных обозначим X^0. Обозначим также множество контактных переменных клана C^j как $\breve{X}^j$ таким образом что $X^j = \hat{X}^j \bigcup \breve{X}^j$ и $\hat{X}^j \bigcap \breve{X}^j = \varnothing$.

Лемма 3.1. Контактная переменная $x_i \in X^0$ не может принадлежать различным кланам с одним и тем же знаком.

Доказательство. Предположим противное. Пусть переменная $x_i \in X$ содержится в различных кланах $C^{j_1}, C^{j_2}, ..., C^{j_k}$ с одним и тем же знаком, причём $k > 1$. Следовательно в соответствии с Определением 3.2 существуют уравнения $L_{l_1} \in C^{j_1}$, $L_{l_2} \in C^{j_2}, \ldots$, $L_{l_k} \in C^{j_k}$ такие что $a_{l_1,i} \neq 0$, $a_{l_2,i} \neq 0, \ldots$, $a_{l_k,i} \neq 0$ и $sign(a_{l_1,i}) = sign(a_{l_2,i}) = ... = sign(a_{i_k,i})$. Тогда в соответствии с Определениями 3.1 и 3.2 уравнения $L_{l_1}, L_{l_2}, ..., L_{l_k}$ принадлежат к одному и тому же клану. Таким образом, получаем противоречие.

□

Теорема 3.2. Контактная переменная $x_i \in X^0$ содержится ровно в двух кланах.

Доказательство. Предположим противное. Пусть контактная переменная $x_i \in X$ содержится в q различных кланах, и $q \neq 2$. Рассмотрим отдельно два случая:

a) $q < 2$. Тогда в соответствии с Определением 3.4 переменная x_i не является контактной.

b) $q > 2$. Получаем противоречие с леммой 3.1, так как существуют только два различных знака: плюс и минус.

□

Определение 3.5. Клан C^j будем называть *входным кланом контактной переменной* $x_i \in X^0$ и обозначать $I(x_i)$, если и только если он содержит эту переменную со знаком плюс. Клан C^j будем называть *выходным кланом контактной переменной* $x_i \in X^0$ и обозначать $O(x_i)$, если и только если он содержит эту переменную со знаком минус. Аналогичную классификацию на входные и выходные можно ввести также и для контактных переменных.

Таким образом, получено с одной стороны разбиение множества уравнений на кланы, а с другой стороны, разбиение переменных на внутренние и контактные. Введём новую нумерацию уравнений и переменных. Нумерацию уравнений начнём с уравнений первого клана и так далее до последнего клана разбиения. Нумерацию переменных начнём с контактных переменных и продолжим далее для внутренних переменных в порядке возрастания номеров кланов. Упорядочим множества уравнений и переменных в соответствии с новой нумерацией. В результате получим следующую блочную форму представления матрицы A:

$$A = \begin{Vmatrix} A^{0,1} & \widehat{A}^1 & 0 & 0 & 0 \\ A^{0,2} & 0 & \widehat{A}^2 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A^{0,k} & 0 & 0 & 0 & \widehat{A}^k \end{Vmatrix}.$$

Более наглядным является представление матрицы A в табл. 3.2, в которой явно указаны кланы и подмножества переменных, соответствующие блокам матрицы.

Таблица 3.2 -

Блочное представление матрицы системы

Клан/Переменные	X^0	$\widehat{X}^1$	$\widehat{X}^2$	. . .	$\widehat{X}^k$
C^1	$A^{0,1}$	$\widehat{A}^1$	0	. . .	0
C^2	$A^{0,2}$	0	$\widehat{A}^2$	. . .	0
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
C^k	$A^{0,k}$	0	0	. . .	$\widehat{A}^k$

Строка матрицы представляет клан, а столбцы матрицы соответствуют вектору разбиения множества переменных $\left(X^0 \quad \widehat{X}^1 \quad \widehat{X}^2 \quad \ldots \quad \widehat{X}^k\right)$. Рассмотрим более подробно структуру матрицы для контактных переменных. В соответствии с Определением 3.4 $\breve{X}^j$ обозначает контактные переменные клана C^j. Тогда $X^0 = \bigcup_j \breve{X}^j$, но это не является разбиением множества X^0, так как каждая контактная переменная $x_i \in X^0$ в соответствии с теоремой 3.2 принадлежит двум кланам. Далее мы будем также использовать матрицы $\breve{A}^j$. Заметим, что в отличие от $A^{0,j}$, которая содержит значения для всех контактных переменных X^0, матрица $\breve{A}^j$ содержит значения только для контактных переменных $\breve{X}^j$

клана C^j. Другими словами матрица $\breve{A}^j$ содержит только ненулевые столбцы матрицы $A^{0,j}$. В результате применения теоремы 3.2 к матрице A мы заключаем, что каждая контактная переменная $x_i \in X^0$ содержится с ненулевыми коэффициентами ровно в двух матрицах множества $A^{0,j}$, $j = \overline{1,k}$ и, кроме того, появляется в одной из матриц с положительными коэффициентами, а в другой – с отрицательными коэффициентами.

Отметим, что для заданной системы (3.6) по её матрице может быть построена матрица направлений $D = sign(A)$, которая может рассматриваться как матрица инцидентности некоторой сети Петри. В этом случае кланы соответствую функциональным подсетям рассматриваемой сети Петри.

3.3. Композиция кланов линейных систем

Решим систему отдельно для каждого клана. Если рассматривать только переменные клана, то имеем систему уравнений

$$A^j \cdot \bar{x}^j = 0, \tag{3.11}$$

где

$$A^j = \left\| \breve{A}^j \quad \hat{A}^j \right\|, \ \bar{x}^j = \left\| \begin{matrix} \breve{\bar{x}}^j \\ \hat{\bar{x}}^j \end{matrix} \right\|.$$

Систему (3.11) обозначим также $S^{C^j}(\bar{x})$. Значения $X \setminus X^j$ могут быть выбраны произвольно. Более подробно

$$S^{C_j}(\bar{x}) = \mathop{\&}_{L_l \in C^j} L_l(\bar{x}).$$

Пусть общее решение системы (3.11) в соответствии с (3.7) имеет вид

$$\bar{x}^j = G^j \cdot \bar{y}^j \tag{3.12}$$

Каждая внутренняя переменная $x_i \in \widehat{X}^j$ входит ровно в одну систему (3.11); таким образом, для всех внутренних переменных кланов справедливо

$$\hat{\bar{x}}^j = \hat{G}^j \cdot \bar{y}^j .$$

Каждая контактная переменная $x_i \in \breve{X}^j$ в соответствии с теоремой 3.2 принадлежит ровно двум системам $S^{C^j}(\bar{x})$ и $S^{C^l}(\bar{x})$, где $C^j = O(x_i)$, $C^l = I(x_i)$. Следовательно, её значения должны совпадать

$$\bar{x}_i^j = \bar{x}_i^l \text{ или } G_i^j \cdot \bar{y}^j = G_i^l \cdot \bar{y}^l ,$$

где G_i^j обозначает строку матрицы G^j соответствующую переменной x_i. Таким образом, мы получаем систему

$$\begin{cases} \bar{x}^j = \bar{y}^j \cdot G^j, \;\; j = \overline{1,k}, \\ G_i^j \cdot \bar{y}^j = G_i^l \cdot \bar{y}^l, \;\; x_i \in X^0, \;\; C^j = O(x_i), \;\; C^l = I(x_i). \end{cases} \tag{3.13}$$

Так как выражение (3.13) является эквивалентным представлением исходной системы (3.6) и имеется разбиение множества X на контактные и внутренние переменные, то приведенные выше рассуждения доказывают следующую теорему.

Теорема 3.3. Система (3.13) эквивалентна системе (3.6).

Уравнения системы (3.13) для контактных переменных

$$G_i^j \cdot \bar{y}^j = G_i^l \cdot \bar{y}^l$$

можно представить как

$$\left\| G_i^j \quad -G_i^l \right\| \cdot \left\| \begin{matrix} \bar{y}^j \\ \bar{y}^l \end{matrix} \right\| = 0 .$$

Занумеруем все переменные $\bar{y}^j$ так чтобы получить общий вектор

$$\bar{y} = \left\| \bar{y}^1 \quad \bar{y}^2 \quad \ldots \quad \bar{y}^k \right\|^T$$

и объединим матрицы G_i^j, $-G_i^l$ в общую матрицу F. Тогда получим систему

$$F \cdot \bar{y} = 0 .$$

Полученная система имеет вид (3.6) следовательно, её общее решение имеет вид (3.7):

$$\bar{y} = R \cdot \bar{z} . \qquad (3.14)$$

Построим объединённую матрицу G решений (3.12) системы (3.11) для всех кланов таким образом, что

$$\bar{x} = G \cdot \bar{y} . \qquad (3.15)$$

Матрица имеет следующую блочную структуру

$$G = \begin{Vmatrix} J^1 & \widehat{G}^1 & 0 & 0 & 0 \\ J^2 & 0 & \widehat{G}^2 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ J^k & 0 & 0 & 0 & \widehat{G}^k \end{Vmatrix}^T .$$

Поясним структуру первого столбца блочного представления, который соответствует контактным переменным. Для каждой контактной переменной $x_i \in X^0$ строим столбец так, что либо блок J^j, либо J^l содержит соответствующий столбец из $\breve{G}^j$, либо $\breve{G}^l$, где $C^j = O(x_i)$, $C^l = I(x_i)$. Действительно, так как контактная переменная принадлежит двум кланам, то её значения могут быть вычислены как в соответствии с общим решением для входного клана, так и в соответствии с общим решением для выходного клана.

Подставим (3.14) в (3.15):

$$\bar{x} = G \cdot R \cdot \bar{z} .$$

Таким образом

$$\bar{x} = H \cdot \bar{z} , \; H = G \cdot R . \tag{3.16}$$

Так как только эквивалентные преобразования были использованы, представленные выше рассуждения доказывают следующую теорему.

Теорема 3.4. Выражение (3.16) представляет общее решение однородной системы (3.6).

Выполним аналогичные преобразования для неоднородной системы (3.8). Общее решение системы для каждого клана в соответствии с (3.9) имеет вид

$$\bar{x}^j = \bar{x}'^j + G^j \cdot \bar{y}^j . \tag{3.17}$$

Уравнения для контактных переменных можно представить следующим образом

$$\bar{x}_i'^j + G_i^j \cdot \bar{y}^j = \bar{x}_i'^l + G_i^l \cdot \bar{y}^l$$

и далее

$$G_i^j \cdot \bar{y}^j - G_i^l \cdot \bar{y}^l = \bar{b}_i',\ \bar{b}_i' = \bar{x}_i'^l - \bar{x}_i'^j$$

либо в матричной форме

$$F \cdot \bar{y} = \bar{b}'\ .$$

Общее решение этой системы в соответствии с (3.8) можно представить как

$$\bar{y} = \bar{y}' + R \cdot \bar{z}\ .$$

Используя объединённую матрицу G, представим (3.17) как

$$\bar{x} = \bar{x}' + G \cdot \bar{y}$$

или

$$\bar{x} = \bar{x}' + G \cdot \left(\bar{y}' + R \cdot \bar{z}\right) = \bar{x}' + G \cdot \bar{y}' + G \cdot R \cdot \bar{z}$$

и, далее

$$\bar{x} = \bar{y}'' + H \cdot \bar{z}\ ,\ \bar{y}'' = \bar{x}' + G \cdot \bar{y}'\ ,\ H = G \cdot R\ . \qquad (3.18)$$

Так как были использованы только эквивалентные преобразования, доказана следующая теорема.

Теорема 3.5. Выражение (3.18) представляет общее решение неоднородной системы (3.8).

Таким образом, построены общие решения однородных и неоднородных линейных систем, полученные с помощью декомпозиции системы на кланы.

В соответствии с теоремами 3.3, 3.4, решение линейной однородной системы уравнений (3.6) с помощью декомпозиции включает в себя следующие этапы:

Этап 1. Выполнить декомпозицию системы (3.6) на множество кланов: $\{C^j\}$.

Этап 2. Для каждого из кланов C^j решить систему (3.11): $\bar{x}^j = \bar{y}^j \cdot G^j$.

Этап 3. Построить и решить систему (3.13) для контактных переменных: $\bar{y} = \bar{z} \cdot R$.

Этап 4. Скомпоновать матрицу G и вычислить матрицу H базисных решений: $H = R \cdot G$.

Заметим, что этапы 2, 3 используют некоторый известный метод решения линейной системы. Выбор этого метода определяется используемым множеством чисел. Например, это может быть метод Гаусса для рациональных чисел, приведение к нормальной форме Смита для целых чисел и метод Тудика при нахождении целых неотрицательных решений. Кроме того, аналогичный подход в соответствии с теоремой 3.5 может быть применён также и для неоднородных систем. Для выполнения этапа 1 могут быть применены методы декомпозиции, рассмотренные в подразделе 2.3.

Пусть $M(q)$ – сложность решения линейной системы размера q. Оценим общую сложность решения линейной системы с помощью декомпозиции. Пусть исходная система (3.6) состоит из k кланов. Тогда размер каждого из кланов можно оценить, как $p = q/k$. Остаток от деления не будем рассматривать. Предположим также, что количество контактных переменных не превышает p. Следующее выражение представляет сложность решения системы с помощью декомпозиции:

$$V(q) = V(k \cdot p) = (k \cdot p) + k \cdot M(p) + M(p) + (k \cdot p)^3 .$$

Каждое из слагаемых этого выражения представляет собой оценку вычислительной сложности соответствующего этапа. Упростим выражение:

$$V(k \cdot p) = 2 \cdot k^3 \cdot p^3 + (k+1) \cdot M(p) \approx k^3 \cdot p^3 + k \cdot M(p) .$$

Оценим ускорение вычислений от использования декомпозиции. Искомое выражение имеет вид:

$$Acc(k \cdot p) = \frac{M(k \cdot p)}{k^3 \cdot p^3 + k \cdot M(p)} .$$

Таким образом, даже для полиномиальных методов степени, превышающей куб, получаем ускорение большее единицы. Оценим ускорение для методов, имеющих экспоненциальную сложность $M(q) = 2^q$, таких, как, например, метод Тудика [88, 110, 111]:

$$AccE(q) = \frac{2^q}{k^3 \cdot p^3 + k \cdot 2^p} \approx \frac{2^q}{2^p} = 2^{q-p} .$$

Получено экспоненциальное ускорение.

Решим однородную систему вида (3.6) из 9 уравнений с 10 переменными. Пусть матрица системы имеет вид

$$A = \begin{Vmatrix} 1 & 0 & 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 2 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 2 & -1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \end{Vmatrix} .$$

Этап 1. *Декомпозиция.* Система состоит из двух кланов

$$C^1 = \{L_1, L_2, L_5, L_6\},\ C^2 = \{L_3, L_4, L_7, L_8, L_9\}.$$

Она имеет следующие множества переменных кланов

$$X^1 = \{x_3, x_6, x_8, x_{10}, x_1, x_2, x_7\},\ X^2 = \{x_3, x_6, x_8, x_{10}, x_4, x_5, x_9\},$$

контактные переменные

$$X^0 = \breve{X}^1 = \breve{X}^2 = \{x_3, x_6, x_8, x_{10}\}$$

и внутренние переменные

$$\hat{X}^1 = \{x_1, x_2, x_7\},\ \hat{X}^2 = \{x_4, x_5, x_9\}.$$

В соответствии с новой нумерацией переменных

$$nx = \begin{pmatrix} 3 & 6 & 8 & 10 & 1 & 2 & 7 & 4 & 5 & 9 \end{pmatrix}$$

и новой нумерацией уравнений

$$nL = \begin{pmatrix} 1 & 2 & 5 & 6 & 3 & 4 & 7 & 8 & 9 \end{pmatrix}$$

матрица A имеет вид

$$A = \begin{Vmatrix} 2 & -1 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 & -1 & 1 & 0 & 0 & 0 \\ -1 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \end{Vmatrix}.$$

Этап 2. *Решение систем для кланов.* Применение алгоритма Тудика [88, 110, 111] даёт следующие матрицы базисных решений для кланов:

$$G^1 = \begin{Vmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{Vmatrix}^T,\ G^2 = \begin{Vmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{Vmatrix}^T$$

по отношению к векторам свободных переменных $\bar{y}^1 = (y_1^1, y_2^1, y_3^1)^T$ и $\bar{y}^2 = (y_1^2, y_2^2)^T$.

Этап 3. *Решение системы для контактных переменных.* Система уравнений для контактных переменных имеет вид:

$$\begin{cases} y_1^1 - y_1^2 = 0, \\ y_1^1 - y_1^2 = 0, \\ y_2^1 - y_1^2 = 0, \\ y_2^1 - y_1^2 = 0. \end{cases}$$

Отметим, что уравнения соответствуют контактным переменным x_3, x_6, x_8, x_{10}; первое уравнение совпадает со вторым; третье уравнение совпадает с четвёртым. Базисные решения системы имеют вид

$$R = \begin{Vmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{Vmatrix}^T .$$

Этап 4. *Компоновка решения исходной системы.* Как было ранее отмечено, матрица G может быть построена различными способами:

$$G = \begin{Vmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{Vmatrix}^T \text{ либо } G = \begin{Vmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{Vmatrix}^T .$$

И, наконец, результирующая матрица базисных решений системы (3.6) имеет вид

$$H = R \cdot G = \begin{Vmatrix} 1 & 1 & 1 & 1 & 0 & 2 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{Vmatrix}^T .$$

Полученный результат совпадает с базисными решениями, вычисленными обычным способом с помощью алгоритма Тудика [88, 110, 111].

3.4. Последовательная композиция кланов линейных систем

(m,n)-системой будем называть линейную систему вида (3.6), содержащую m уравнений и n неизвестных. В качестве общей оценки размерности системы, как правило, используют один параметр, равный максимуму количества уравнений и количества переменных: $q = \max(m,n)$. Если числа m и n отличаются незначительно возможно использование одного из них в качестве параметра в оценках сложности. Далее будем предполагать, что сложность системы характеризуется количеством уравнений m.

Вычислительная сложность решения систем линейных уравнений существенно зависит от области значений переменных и коэффициентов. Так для решения систем в полях известны [115] полиномиальные методы сложности порядка q^3. Для решения систем в кольцах предложены [120, 183, 187] полиномиальные методы порядка q^4. Отметим, что известные методы решения линейных диофантовых систем в целых неотрицательных числах [162] являются экспоненциальными; их временная сложность оценивается как 2^q. Как показано в [148], приведенная оценка является оптимистичной, поскольку сложность в худшем случае может быть сопоставима с двойной экспонентой. Большинство полученных далее результатов имеют место, как для полиномиальных, так и для экспоненциальных методов решения систем. Если различия являются существенными, то это оговорено дополнительно.

Предположим, что выполнена декомпозиция системы на k кланов. Рассмотрим размерности систем для полученных кланов. Итак, требуется решить k систем размерности $(m_1,n_1),(m_2,n_2),\ldots,(m_k,n_k)$. Пусть для клана C^i получена матрица базисных решений G^i, насчитывающая l_i решений. Для полей и колец

имеет место известная оценка количества базисных решений как $l_i = n_i - r_i$, где r_i – ранг матрицы A^i соответствующей системы. Отметим, что при решении систем в целых неотрицательных числах оценка количества базисных решений является нетривиальной задачей. Известны примеры систем [10], содержащих пару уравнений и пять переменных, для которых базис насчитывает 240 решений. Однако такое разрастание базиса наблюдается только при использовании целочисленных свободных переменных. Базисы для рациональных генераторов, либо, что означает то же самое, при использовании операции сокращения на общий делитель, получаются более компактными. Например, для упомянутой системы он состоит из 4-х решений.

После нахождения общих решений для кланов необходимо решить систему композиции для контактных переменных. Оценка размерности этой системы $(p, \sum_i l_i)$, поскольку уравнения системы соответствуют контактным переменным, а неизвестными являются свободные переменные базисных решений для кланов. Заметим, что $\sum_i n_i = n$, $\sum_i m_i = m + p$, где p – количество контактных переменных в полученной декомпозиции: $p = \left|X^0\right|$, где X^0 – множество всех контактных переменных исходной системы. Как было отмечено ранее, будем предполагать, считая $p \approx \sum_i l_i$, что сложность системы определяется количеством её уравнений p. Далее считаем характеристикой размерности системы уравнений клана количество его переменных, а характеристикой размерности композиции кланов – количество используемых в композиции контактных переменных. Контактные переменные при таком подсчёте учитываются дважды для каждого из смежных кланов $m_i = \left|\hat{X}^i\right| + \left|\breve{X}^i\right|$, где $\hat{X}^i$ – множество внутренних переменных, а $\breve{X}^i$ – множество контактных переменных клана C^i. Поэтому, как правило, выполняется неравенство $n_i \le m_i$.

В предыдущем подразделе показано [139], что каждая контактная переменная используется для связи ровно двух кланов. Поэтому для представления декомпозиции удобно применять ориентированный граф, кратность дуг которо-

го соответствует количеству контактных переменных, используемых для связи пары кланов в определённом направлении. В настоящей работе направление связей несущественно, поэтому в качестве основного средства представления декомпозиции линейной системы выбран взвешенный неориентированный граф, веса рёбер которого равны количеству контактных переменных, используемых для связи соответствующих кланов. Численные характеристики вершин определяются парой (m_i, n_i). Далее будет показано, что в последовательной композиции характеристики вершин могут быть опущены.

Рассмотрим основные способы организации композиции кланов:

I. Одновременная.

II. Последовательная:

1) Парная (рёберная);

2) Подграфов.

Одновременная композиция изучена в предыдущем подразделе, полученное ускорение вычислений оценивается как 2^{q-p}. Наиболее простой последовательной является парная композиция, при которой пара смежных вершин заменяется одной вершиной в результате решения системы, построенной для контактных переменных, используемых для связи соседних кланов; количество контактных переменных равно весу соответствующего ребра. По существу, такая операция может быть представлена как слияние (стягивание) смежных вершин графа. Пусть выполняется слияние двух смежных вершин с номерами i и j, представляющими системы уравнений сложности (m_i, n_i) и (m_j, n_j) соответственно. Тогда сложность системы, решаемой при парной композиции равна $(p_{i,j}, l_i + l_j)$, где $p_{i.j}$ – количество контактных переменных в композиции кланов C^i и C^j. При композиции подграфов на шаге решается система для выбранного подмножества вершин, затем подграф, порождённый этими вершинами, заменяется единственной вершиной. Примеры решения системы с помощью одновременной и последовательной композиции рассмотрены в разделе 4.

Одновременную композицию целесообразно применять в тех случаях, когда количество контактных переменных не превышает количества внутренних

переменных максимальной подсети $\max_i(m_i) \geq p$, либо в случаях незначительного превышения. Поскольку обязательным этапом композиционного метода [139] является решение уравнения для каждого клана, дальнейшие построения направлены на снижение сложности решения системы для контактных переменных.

Для сравнения различных вариантов композиции по отношению к выбранному параметру сложности системы введём следующий взвешенный граф, описывающий декомпозицию системы на кланы. Граф декомпозиции – это тройка $G = (V, E, W)$, где вершины множества $V = \{v\}$ соответствуют кланам: $v \leftrightarrow C$; рёбра $E \subseteq V \times V$ соединяют кланы, имеющие общие контактные переменные: $v_1 v_2 \in E \Leftrightarrow \exists x \in X^0 : (I(x) = C^1 \wedge O(x) = C^2) \vee (I(x) = C^2 \wedge O(x) = C^1)$; функция взвешивания $W : (V \to \mathrm{N}) \bigcup (E \to \mathrm{N})$ сопоставляет вершине количество переменных соответствующего клана, а ребру, количество контактных переменных. Кроме того, для каждой вершины выполняется неравенство $w(v) \geq \sum_u w(v,u)$. Неравенство отражает тот факт, что контактные переменные учитываются совместно с внутренними переменными при оценке размера клана.

Представим последовательную композицию следующим образом. Пусть задан граф G декомпозиции линейной системы S. Выберем некоторое подмножество вершин $V^1 \subseteq V$, порождающее связный подграф $H^1 = (V^1)$ графа G; заменим его одной вершиной и перейдём затем к рассмотрению полученного графа G^1. Продолжая описанный процесс, мы преобразуем исходный граф в единственную вершину. Процесс последовательной композиции можно представить следующей последовательностью:

$$G = G^0 \underset{d_1}{\overset{V^1}{\to}} G^1 \underset{d_2}{\overset{V^2}{\to}} G^2 \ldots \underset{d_k}{\overset{V^k}{\to}} G^k,$$

причём заключительный граф последовательности G^k состоит из единственной вершины и соответствует завершению процесса композиционного решения.

Число d_i равно сумме весов рёбер графа H^i и соответствует размерности системы композиции, решаемой на шаге i. Пример коллапса подграфов представлен на рис. 3.1; на рис. 3.2 представлен пример рёберного коллапса для последовательности подграфов, изображённой на рис. 3.1.

Пусть $M(q)$ сложность решения системы размерности q. Тогда сложность последовательной композиции можно оценить как $Y(q) = \sum_{i=1,k} M(d_i)$. Следует отметить, что $\sum_{i=1,k} d_i = p$, где p представляет собой сумму весов рёбер исходного графа (общее количество контактных переменных) $p = \sum_{e} w(e)$. Таким образом, как для полиномиальной, так и для экспоненциальной сложности решения систем последовательная композиция не хуже одновременной.

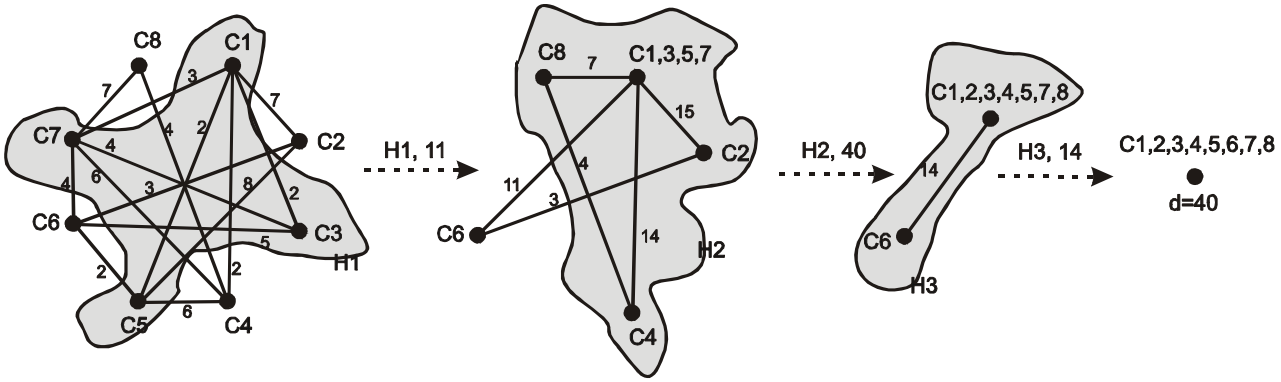


Рис. 3.1. Пример коллапса подграфов

Так как исходный граф декомпозиции системы на кланы сжимается в единственную вершину, по аналогии с процессами, исследуемыми в астрофизике, процесс последовательной композиции назван коллапсом графа. С точки зрения сложности решения систем нас интересует коллапс, обеспечивающий минимальную вычислительную сложность решения исходной системы. Такой коллапс назовём оптимальным. В зависимости от того, рассматриваются ли методы полиномиальной либо экспоненциальной сложности, имеются два основных варианта постановки соответствующей оптимизационной задачи:

$$\text{I.}\begin{cases}\sum\limits_i d_i^{\,u} \to \min,\\ \sum\limits_i d_i = q,\\ d_i > 0.\end{cases} \qquad \text{II.}\begin{cases}\sum\limits_i u^{d_i} \to \min,\\ \sum\limits_i d_i = q,\\ d_i > 0.\end{cases} \qquad (3.19)$$

Как правило, для полиномиальных методов в задаче I рассматривают $u \ge 3$, а для экспоненциальных методов в задаче II рассматривают $u \ge 2$. В соответствии с теорией сложности в оценках экспоненциальных функций слагаемые низших степеней могут быть опушены. Таким образом, для экспоненциальных методов становиться актуальным параметр $d = \max\limits_i(d_i)$, названный шириной коллапса и равный максимальному из чисел d_i. Тогда сложность последовательной композиции можно оценить как u^d, а дополнительное ускорение вычислений по сравнению с одновременной композицией – как u^{p-d}. Хотя для методов полиномиальной сложности аналогичные упрощения являются довольно грубыми, выражение $k \cdot d^u$ можно рассматривать как оценку верхней границы сложности. В ряде случаев в качестве квазиоптимального коллапса целесообразно рассматривать коллапс, имеющий минимальную ширину. Действительно, при произвольном разбиении числа p на k частей, как в задаче I, так и в задаче II, оптимальным является равномерное разбиение $d_i = p/k$. Таким образом, снижение максимума последовательности d_i способствует приближению к равномерному разбиению.

Сравнительные оценки сложности различных способов организации последовательного коллапса приведены в табл. 3.3. Композиция подграфов соответствует последовательности, изображённой на рис. 3.1; рёберная композиция 1 соответствует последовательности, изображённой на рис. 3.2; рёберная композиция 2 соответствует оптимальному коллапсу; рёберная композиция 3 соответствует наихудшему коллапсу. Графические изображения рёберных композиций 2, 3 представлены далее на рис 3.5. Следует отметить, что использование ширины коллапса позволяет получить простые и достаточно хорошие оценки сложности. Заметим, что ускорения вычислений для наилучшего коллапса по

сравнению с одновременной композицией имеют порядок 10^{15} для методов экспоненциальной сложности.

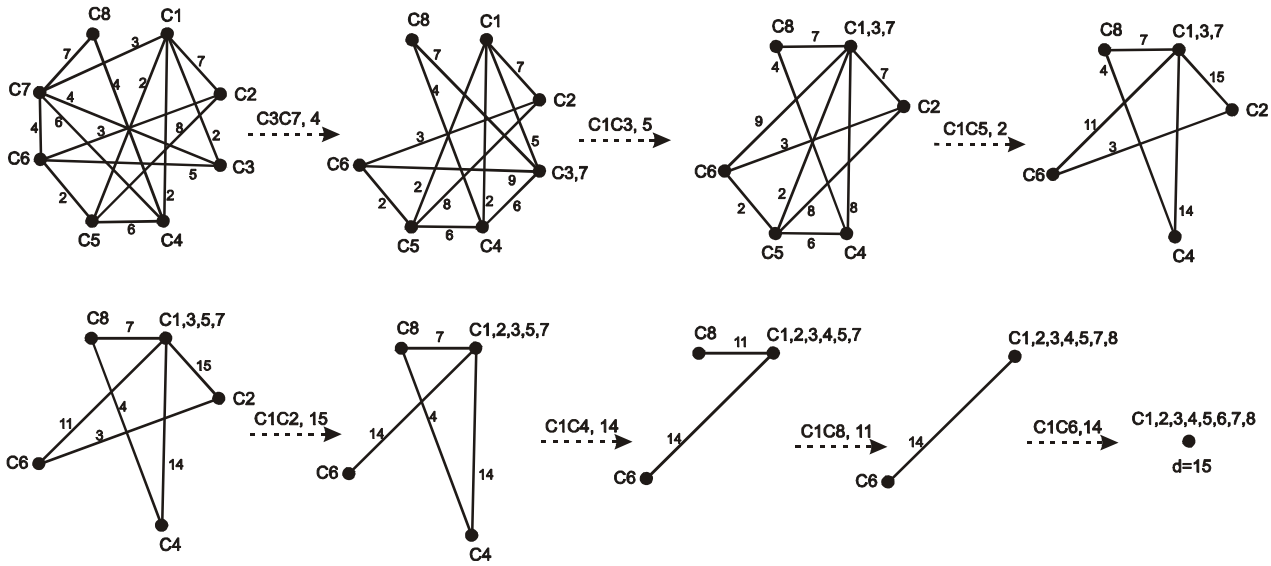


Рис. 3.2. Пример рёберного коллапса

При выборе способа реализации последовательной композиции следует рассмотреть два основных вопроса:

1. Может ли применение не минимальных кланов ускорить процесс решения.

2. Может ли композиция подграфов быть более эффективной, чем парная композиция.

Выбор не минимальных кланов предполагает решение единственной системы уравнений для составного клана без решения промежуточных систем для внутренних минимальных кланов. Композиция подграфов предполагает одновременную композицию всех вершин подграфа. Полученные в результате графы совпадают. Отличие состоит только в сложности решения системы для подграфа, которая в одном случае решается обычными методами (без применения композиции), а в другом случае – с помощью одновременной композиции.

Таблица 3.3 -

Сравнительные оценки сложности композиционного решения систем

Композиция	Последовательность	d	Сложность решения систем			
			Полином		Экспонента	
			d^4	$\sum d_i^4$	2^d	$\sum 2^{d_i}$
Подграфы	H1,11-H2,40-H3,14	40	$2{,}56\cdot10^6$	$2{,}61\cdot10^6$	$1{,}1\cdot10^{12}$	$1{,}99\cdot10^{12}$
Рёберная 1	C3C7,4-C1C3,5-C1C5,2-C1C2,15-C1C4,14-C1C8,11-C1C6,14	15	$5{,}06\cdot10^4$	$1{,}43\cdot10^5$	$3{,}28\cdot10^4$	$6{,}76\cdot10^4$
Рёберная 2	C2C5,8-C1C2,9-C1C4,8-C1C7,9-C1C8,11-C1C6,9-C1C3,11	11	$1{,}46\cdot10^4$	$5{,}72\cdot10^4$	$2{,}05\cdot10^3$	$6{,}14\cdot10^3$
Рёберная 3	C5C6,2-C1C3,2-C1C4,2-C1C8,4-C5C7,4-C1C2,7-C1C4,44	44	$3{,}75\cdot10^6$	$3{,}75\cdot10^6$	$1{,}76\cdot10^{13}$	$1{,}76\cdot10^{13}$
Одновременная	C1C2C3C4C5C6C7C8,65	65	$1{,}79\cdot10^7$	$1{,}79\cdot10^7$	$3{,}69\cdot10^{19}$	$3{,}69\cdot10^{19}$

Имеют место следующие оценки сложности: подграф без композиции – $q = \sum_{v\in H} w(v) - \sum_{e\in H} w(e)$; подграф с одновременной композицией – $q = \max(\max_{v\in H} w(v), \sum_{e\in H} w(e))$.

Лемма 3.2. Для любого графа декомпозиции выполняется неравенство:

$$\max(\max w(v), \sum w(e)) \le \sum w(v) - \sum w(e). \qquad (3.20)$$

Доказательство. Неравенство (3.20) можно представить в виде эквивалентной системы неравенств:

$$\begin{cases} \max w(v) \le \sum w(v) - \sum w(e), \\ \sum w(e) \le \sum w(v) - \sum w(e). \end{cases}$$

Докажем по отдельности выполнение каждого из неравенств системы. В соответствии с определением графа декомпозиции

$$w(v) \geq \sum_{u} w(v,u) .$$

Просуммируем неравенство по всем вершинам графа

$$\sum_{v} w(v) \geq \sum_{v} \sum_{u} w(v,u) .$$

Учитывая известное из теории графов [184] равенство

$$\sum_{v} \sum_{u} w(v,u) = 2 \cdot \sum_{e} w(e) , \tag{3.21}$$

получим

$$\sum_{v} w(v) \geq 2 \cdot \sum_{e} w(e) .$$

И, далее,

$$\sum w(e) \leq \sum w(v) - \sum w(e) .$$

Справедливость второго из неравенств доказана. Докажем справедливость первого неравенства. Покажем, что неравенство

$$w(v') \leq \sum w(v) - \sum w(e)$$

выполняется для произвольной вершины $v' \in V$ графа G. Имеем

$$0 \leq \sum_{v \neq v'} w(v) - \sum w(e) .$$

И, далее,

$$\sum w(e) \leq \sum_{v \neq v'} w(v) .$$

При доказательстве этого неравенства будем учитывать тот факт, что вес ребра уже содержится в весе каждой инцидентной ему вершины таким образом, что

$$w(v) \geq \sum_{u} w(v,u).$$

Тогда

$$\sum_{v \neq v'} w(v) \geq \sum_{v \neq v'} \sum_{u} w(v,u).$$

Учитывая соотношение (3.21), получим

$$\sum_{v \neq v'} \sum_{u} w(v,u) = 2 \cdot \sum_{v \neq v'} w(e) + \sum_{v'} w(e) = \sum w(e) + \sum_{v \neq v'} w(e).$$

Заметим, что индекс суммирования рёбер задан множеством инцидентных вершин. Принимая во внимание неравенство

$$\sum_{v \neq v'} w(e) \geq 0,$$

получим

$$\sum w(e) \leq \sum_{v \neq u} w(v),$$

что и требовалось доказать.

□

Следствие. Использование минимальных кланов в процессе последовательной композиции является более эффективным.

Таким образом, решение систем уравнений для всех минимальных кланов является обязательным этапом. Поэтому, в дальнейшем изложении веса вершин графа декомпозиции могут быть опущены. В качестве графа декомпозиции на кланы будем рассматривать взвешенный граф $G = (V, E, W)$, где отображение $W : E \to \mathrm{N}$ задаёт кратность его рёбер.

Рассмотрим последовательный коллапс графа путём слияния (коллапса) подграфов, порождённых указанным множеством вершин. Не ограничивая

общности, будем рассматривать связные подграфы. В качестве основного параметра коллапса будем рассматривать его ширину d. Задача состоит в построении такой последовательности стягивания подграфов, которая обеспечит минимальную ширину коллапса. Возможны два упомянутых ранее способа организации этого процесса: одновременный и последовательный. Последовательный коллапс может быть организован как парный (рёберный) либо как коллапс произвольных подграфов, порождённых указанным множеством вершин.

Лемма 3.3. Парный (рёберный) коллапс не хуже произвольного коллапса подграфов.

Доказательство. Во-первых, результат применения рёберного коллапса подграфа совпадает с результатом применения одновременного коллапса по отношению к окружению подграфа. То есть в обоих случаях будет получен один и тот же граф.

Отличие состоит в ширине коллапса выбранного подграфа. Так как ширина одновременного коллапса известна и равна сумме весов рёбер, необходимо оценить ширину рёберного коллапса подграфа. Покажем, что в результате рёберного коллапса не может появиться ребро веса превышающего сумму весов всех рёбер.

Выберем произвольное ребро $e' \in E$. При слиянии вершин веса рёбер, инцидентных общей вершине, суммируются. Поэтому, сумма весов рёбер полученного графа равна

$$\sum_{e} w(e) - w(e').$$

Эта сумма и является верхней границей оценки сложности рёберного коллапса полученного графа. Продолжение коллапса не приведёт к появлению ребра большего веса, а ширина рёберного коллапса равна максимальному весу ребра. Имеем:

$$\sum_e w(e) = w(e') + \sum_{e \neq e'} w(e).$$

Тогда

$$\sum_e w(e) \geq \max\left(w(e), \sum_{e \neq e'} w(e) \right).$$

□

Таким образом, в дальнейшем будем рассматривать рёберный коллапс, как более эффективный способ композиции при экспоненциальной сложности решения системы. Следует отметить, что приведенные оценки сложности являются асимптотическими. В частных случаях при небольшой размерности кланов, когда конкретные значения оценок экспоненциальной сложности сопоставимы с полиномиальными сомножителями, коллапс подграфов может иметь меньшую вычислительную сложность.

3.5. Оптимальный коллапс взвешенного графа

Пусть задан взвешенный граф $G = (V, E, W)$. Не ограничивая общности, считаем, что G связный, иначе выполним коллапс по компонентам.

Определение 3.6. *Операция рёберного коллапса*
Определим *операцию рёберного коллапса* $G \setminus e$ для некоторого $e \in E$ следующим образом. Пусть $e = v_1 v_2$. Тогда $G \setminus e = G' = (V', E', W')$, где $V = (V \setminus v_1 v_2) \bigcup v$, где v - новая вершина, представляющая собой слияние (коллапс) вершин v_1, v_2:

$$E' = (E \setminus (v_1 v_2 \bigcup \{v_1 u | u \in V, v_1 u \in E\} \bigcup \{v_2 u | u \in V, v_2 u \in E\})) \bigcup \{vu | u \in V', v_1 u \in E \vee v_2 u \in E\},$$

$$W'(vu) = \begin{cases} W(v_1 u) + W(v_2 u), & v_1 u \in E \wedge v_1 u \in E, \\ W(v_1 u), & v_1 u \in E \wedge v_2 u \notin E, \\ W(v_2 u), & v_2 u \in E \wedge v_1 u \notin E. \end{cases}$$

Таким образом, при слиянии вершин ребра объединяются рёбра инцидентные обеим вершинам.

Утверждение 3.2. Операция рёберного коллапса сохраняет связность графа.

Утверждение 3.3. Верно следующее равенство для суммы весов рёбер

$$S(G) = S(G') + w(e).$$

В соответствии с терминологией [184], граф, в котором $|V| = k$, а $|E| = p$ будем называть (k, p)-графом либо *k*-графом. Так как при выполнении операции рёберного коллапса объединяется пара смежных вершин графа, то рёберный коллапс всего графа состоит в последовательном выполнении $(k-1)$ операций рёберного коллапса.

Определение 3.7. *Процесс рёберного коллапса*

Процесс последовательного рёберного коллапса взвешенного k-графа – это последовательность $(k-1)$ операций рёберного коллапса:

$$G^0 = G \to G^1 = G^0 \setminus e_1 \to G^2 = G^1 \setminus e_2 \to \cdots \to G^{k-1} = G^{k-2} \setminus e_k$$

Заметим, что полученный в результате граф G^{k-1} состоит из единственной вершины. Это вполне соответствует названию процесса, сжимающего исходный граф в вершину. В качестве основного параметра коллапса будем рассматривать его ширину, равную максимальному весу стягиваемого ребра.

Определение 3.8. *Ширина коллапса*

Ширина коллапса – это максимальный вес ребра в процессе коллапса:

$$d(\sigma) = \max_{e \in \sigma} w(e).$$

Выбор различных последовательностей рёбер $e_1e_2...e_k$ в общем случае приводит к различным значениям ширины коллапса. *Оптимальным процессом коллапса* будем называть последовательность рёбер, обеспечивающую минимизацию общей сложности решения систем (3.19). *Квазиоптимальным процессом коллапса* будем называть коллапс, имеющий наименьшую ширину. Отметим, что минимальная ширина коллапса является свойством заданного графа. Введём рекуррентное определение минимальной ширины коллапса. Обозначим $d(G)$ ширину рёберного коллапса графа G. Тогда

$$\begin{cases} d(G) = \min\limits_{e} d(G,e), \\ d(G,e) = \max(w(e), d(G \setminus e)), \end{cases}$$

где функция двух аргументов $d(G,e)$ определяет минимальную ширину рёберного коллапса графа G при условии, если первоначально будет выполнен коллапс ребра e.

Рёберный коллапс представляет собой комбинаторную задачу, для решения которой применим универсальный метод полного перебора всех возможных последовательностей рёбер. Пример дерева полного перебора рёберного коллапса графа представлен на рис. 3.3; заметим, что минимальная ширина коллапса составляет 14, максимальная 23, а сумма весов рёбер (ширина одновременного коллапса) равна 35. Точное число различных последовательностей равно $K(G) = \prod\limits_{i=0,k-2} \left|E^i\right|$. На каждом шаге стягивается пара смежных вершин, а наибольшее количество смежных вершин имеет место в полном графе. Количество рёбер полного k-графа равно $\frac{k \cdot (k-1)}{2}$. Тогда $\hat{K}(G) = \prod\limits_{i=2,k} \frac{i \cdot (i-1)}{2} = \frac{k! \cdot (k-1)!}{2^{k-1}} = \frac{(k!)^2}{k \cdot 2^{k-1}}$. Например, $\hat{K}(10) = 2{,}6 \cdot 10^9$, а $\hat{K}(20) = 5{,}7 \cdot 10^{29}$, $\hat{K}(100) = 1{,}4 \cdot 10^{284}$. Таким образом, необходим поиск эффективных методов решения задачи оптимального (квазиоптимального) рёберного коллапса.

Теорема 3.6. Ширина коллапса ациклического графа равна максимальному весу ребра.

Доказательство. Операция рёберного коллапса ациклического графа приводит к получению нового ациклического графа, содержащего количество рёбер меньшее на единицу, чем исходный граф. Кроме того, эта операция не изменяет веса оставшихся рёбер. Таким образом, ширина коллапса не зависит от порядка выбора рёбер и равна максимальному весу ребра. □

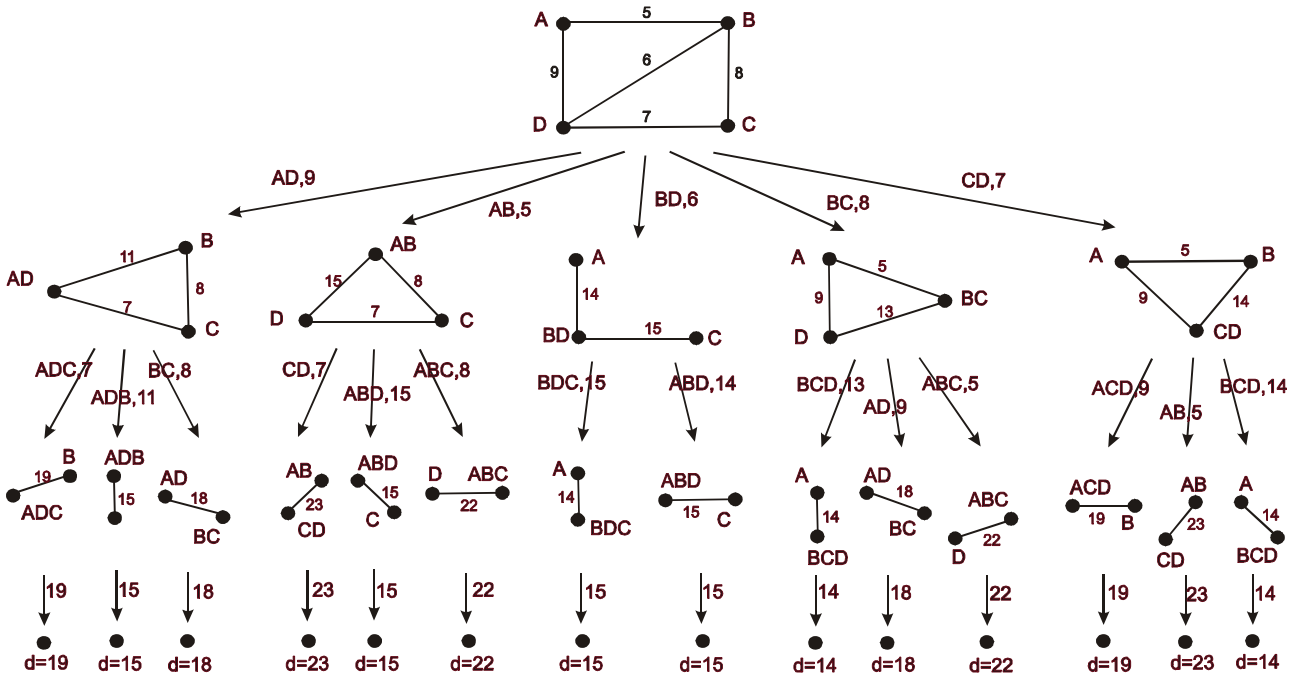


Рис. 3.3. Пример дерева полного перебора рёберного коллапса

Любая простая цепь может быть заменена ребром минимального веса при ширине её коллапса, равного максимальному весу ребра. Это соответствует выбору на шаге коллапса ребра максимального веса. Не ограничивая общности можно рассматривать компактные графы, не содержащие простых цепей и висячих вершин.

Утверждение 3.4. Если граф имеет точки сочленения, то ширина коллапса равна максимальной ширине среди его двусвязных компонентов (блоков).

Теорема 3.7. Ширина коллапса простого цикла равна $\max_{e,e_1,e_2}(w(e), \min(w(e_1)+w(e_2))$.

Доказательство. Простой цикл, преобразуется в цикл меньшей размерности до тех пор, пока не будет получен треугольник. При коллапсе треугольника будет получен граф, состоящий из одного ребра, вес которого равен сумме весов рёбер отличных от стягиваемого. Таким образом, ширина коллапса определяется с одной стороны максимальным весом ребра перед стягиванием треугольника, а с другой стороны, весом последнего ребра. Следовательно, нижней границей ширины является как вес максимального ребра, так и суммарный вес пары рёбер. □

Теорема 3.8. Оптимальный коллапс простого цикла соответствует выбору на шаге коллапса ребра максимального веса.

Доказательство. Коллапс простого цикла продолжается без изменения весов рёбер до тех пор, пока не будет получен треугольник. Выбор максимального ребра гарантирует, что при получении треугольника останется три рёбра минимального веса. Кроме того, при коллапсе треугольника выбор ребра максимального веса обеспечит минимальный суммарный вес оставшихся рёбер. Действительно, имеет место соотношение $\min_{e_1,e_2}(w(e_1)+w(e_2)) = \min_{e_1}(e_1) + \min_{e_2 \neq e_1}(e_2)$. □

Графы, полученные в результате стягивания рёбер, называют минорами исходного графа [184, 188]. Рассмотрим решётку миноров, полученных в результате коллапса, которую назовём *частичной решёткой коллапса*. Решётка состоит из $(k-1)$ уровней. На i-м уровне точками представлены рёбра текущего графа G^i. Линии соответствуют отношению частичного порядка << рёбер текущего и предыдущего уровней таким образом, что:

$$e_1^i << e_3^{i+1} \Leftrightarrow e_3^{i+1} = e_1^i \vee e_3^{i+1} = e_1^i + e_2^i .$$

Частичная решётка является наглядным представлением процесса коллапса. В соответствии с определением операции коллапса на каждом шаге стягивается одно ребро. Если концы этого ребра не имеют общих смежных вершин (не формируют треугольников вместе с другими рёбрами), то на следующем уровне присутствуют все рёбра за исключением стягиваемого. Если ребро образует один либо несколько треугольников, то каждая пара рёбер треугольника заменяется одним ребром. Рекуррентное соотношение для числа рёбер $p_i = p_{i-1} - 1 - t$, где t – количество треугольников, определяемых стягиваемым ребром. Решётка иллюстрирует взаимосвязи рёбер. Таким образом, каждое ребро на шаге коллапса представляет собой либо ребро исходного графа, либо сумму некоторых рёбер. Решётки двух различных последовательностей коллапса, представленных на рис. 3.3, изображены на рис. 3.4. Стягиваемые рёбра помечены диагональным крестом.

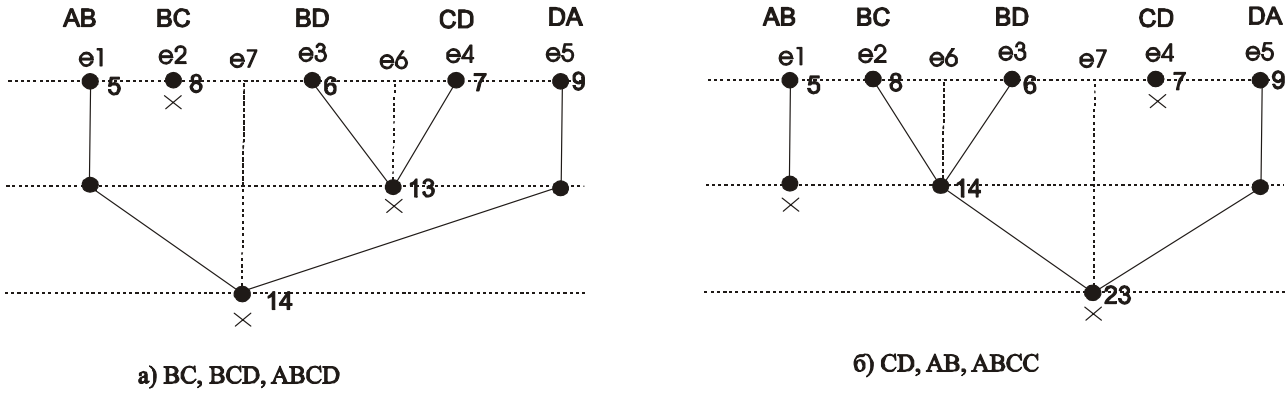


Рис. 3.4. Примеры частичных решёток коллапса

Утверждение 3.5. Каждое ребро на шаге коллапса является суммой некоторых рёбер исходного графа.

Таком образом, ширина коллапса равна весу ребра, полученного на некотором шаге. Такое ребро будем называть *критическим ребром коллапса*. Критическое ребро либо просто стягивается в процессе коллапса, либо остаётся его последним ребром.

Верхние и нижние оценки ширины коллапса могут быть использованы при поиске оптимального коллапса с помощью метода ветвей и границ [49, 58, 168, 177]. В соответствии с определением ширины рёберного коллапса:

$$\max_e w(e) \le d(G) \le \sum_e w(e).$$

Построим более точные верхние оценки ширины коллапса. В лемме 3.2 была представлена оценка

$$d(G) \le \max\left(w(e), \sum_{e \ne e'} w(e) \right).$$

Действительно, по крайней мере, одно ребро будет аннулировано и ширина коллапса оставшегося графа не превысит сумму его рёбер. Продолжая описанный процесс не более $(k-1)$ раз можно прийти к следующей оценке. Пусть $w^{\max} = \max_{e \in G} w(e)$ – максимальный вес ребра исходного графа, а $w^{\min} = \min_{e \in G} w(e)$ – минимальный. На первом шаге не может появиться ребро большего веса, чем $2 \cdot w^{\max}$, на третьем $2 \cdot w^{\max} + 2 \cdot w^{\max} = 4 \cdot w^{\max}$ и так далее. Получим рекуррентное соотношение:

$$\begin{cases} w_0^{\max} = w^{\max}, \\ w_i^{\max} = 2 \cdot w_{i-1}^{\max}, \quad i = \overline{1, k-2}. \end{cases}$$

Тогда

$$d(G) \le w_{k-2}^{\max} = 2^{k-2} \cdot w^{\max}.$$

С другой стороны, после первого шага, сумма весов рёбер оставшейся части графа не превысит $\sum w(e) - w^{\min}$, после второго: $\sum w(e) - 2 \cdot w^{\min}$. Продолжая оценки до завершения коллапса, получим:

$$d(G) \le \max\left(2^{k-2} \cdot w^{\max}, \sum w(e) - (k-2) \cdot w^{\min}\right).$$

Хотя оценка является довольно грубой, она дополняет ранее полученные оценки. Для получения более точных оценок рассмотрим процесс добавления рёбер, соединяющих пару несмежных вершин. Изучим, влияние этой операции на ширину коллапса.

Теорема 3.9. Добавление ребра, соединяющего пару несмежных вершин графа, увеличивает ширину коллапса не более чем на вес добавленного ребра.

Доказательство. Пусть ширина коллапса графа G равна $d(G)$ и достигается с помощью последовательности σ. Рассмотрим граф $G+e$ и выполним его коллапс с помощью той же последовательности σ. Пусть $e = v_1 v_2$.

При выполнении операции коллапса будем помечать символом v_1 все вершины, стягиваемые с вершиной v_1 и символом v_2 все вершины, стягиваемые с v_2. Во-первых, вершины v_1, v_2 несмежные в исходном графе. Во-вторых, граф связный. Следовательно, на некотором шаге коллапса получаем вершину u, смежную как с v_1, так и с v_2. Стягивание этой вершины в графе G с одной из вершин v_1, v_2 приводит к образованию ребра $e' = v_1 v_2$. В дальнейшем это ребро может участвовать в образовании критического ребра либо будет просто аннулировано.

Рассмотрим выполнение рассмотренной операции в графе $G+e$. Перед получением треугольника, образованного ребром e и некоторой вершиной u процесс не отличается от ранее рассмотренного. При стягивании вершины u с одной из вершин v_1, v_2 вместо ребра веса $w(e')$ будет получено ребро веса $w(e') + w(e)$. Далее это ребро либо войдёт в критическое ребро коллапса, либо

будет просто аннулировано. В первом случае ширина коллапса увеличится на величину $w(e)$, во втором случае не изменится. □

Для получения более точных верхних оценок рассмотрим процесс добавления недостающих рёбер к некоторому остову R графа G. Поскольку ширина коллапса ациклического графа в соответствии с теоремой 3.7 равна максимальному весу ребра, то может быть представлена следующая оценка ширины коллапса.

Теорема 3.10. Ширина коллапса не превышает сумму веса максимального ребра остова и весов оставшихся рёбер:

$$d(G) \le \max_{e \in R} w(e) + \sum_{e \notin R} w(e), \text{ где } R - \text{остов графа } G.$$

Чтобы улучшить оценки можно выбирать остов, содержащий рёбра максимального веса, таким образом, чтобы минимизировать сумму. В качестве хорошего приближения можно рассматривать стандартную задачу выбора остова максимального веса [5, 184, 188]. Заметим, что количество оставшихся рёбер равно цикломатическому числу графа $\nu(G) = p - k + 1$. Тогда оценка может быть представлена как

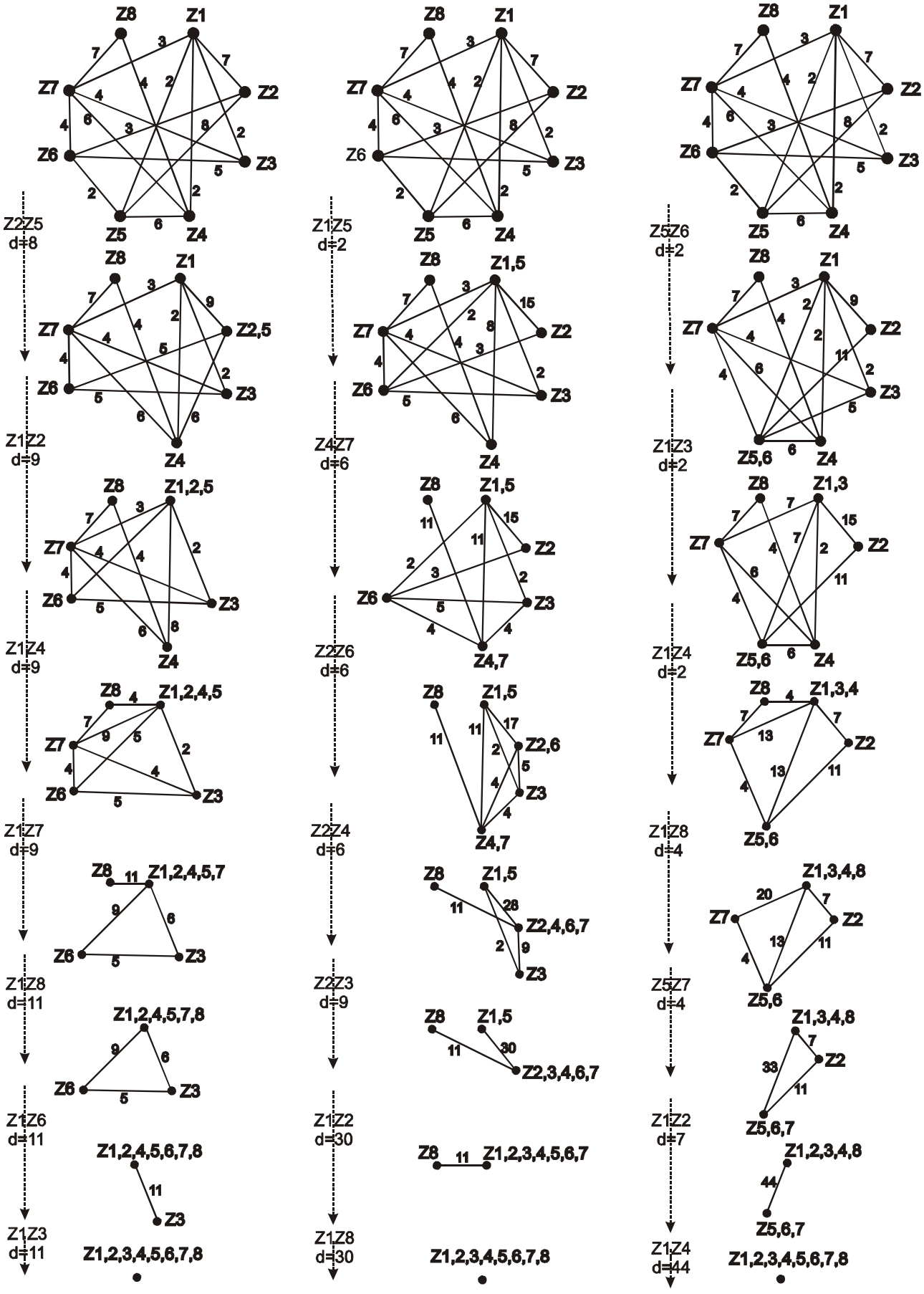
$$d(G) \le (\nu(G) + 1) \cdot w^{\max} = (p - k + 2) \cdot w^{\max}.$$

Оптимальный коллапс для полиномиальной и экспоненциальной оценок сложности, а также квазиоптимальный коллапс минимальной ширины являются задачами, для решения которых ранее предложено применить универсальный метод полного перебора, имеющий экспоненциальную сложность по отноше-

нию к размеру графа. Являются ли указанные задачи NP-полными, представляет собой открытый вопрос. Построим и статистически обоснуем эвристический алгоритм решения задач оптимального коллапса, основанный на одношаговых стратегиях выбора стягиваемого ребра. Применение эвристических алгоритмов целесообразно также по той причине, что исходная информация для последовательной композиции не является достаточно точной, поскольку отсутствуют точные оценки количества базисных решений подсистем композиции.

В соответствии с результатами, полученными для простой цепи и простого цикла можно предложить выбор ребра максимального веса на шаге коллапса. Можно выбирать первое, либо случайное ребро максимального веса в случае, если имеется несколько таких рёбер. Алгоритм состоит в простой реализации операции коллапса в соответствии с определением, дополненной правилом выбора ребра максимального веса. Сложность такого алгоритма можно оценить как $k \cdot p^2$. Действительно, необходимо выполнить $k-1$ шагов и на каждом шаге следует обработать не более чем p рёбер, для которых при коллапсе треугольников обрабатывается не более чем p смежных рёбер. Следует отметить, что описанный «жадный» алгоритм не всегда гарантирует оптимальный коллапс, хотя и обеспечивает достаточно хорошее приближение; так для графа, изображённого на рис. 3.1 «жадный» алгоритм даёт ширину 15, в то время как оптимальная ширина равна 14. Примеры коллапса графа, изображённого на рис. 3.1, с использованием выбора максимального, минимального и случайного ребра представлены на рис. 3.5.

Для сравнения различных правил выбора ребра на шаге коллапса генерировались случайные графы, и выполнялся их рёберный коллапс. Сравнивался выбор максимального, минимального и случайного ребра на шаге. Полученные результаты представлены в табл. 3.4.



а) Максимальное ребро б) Случайное ребро в) Минимальное ребро

Рис. 3.5. Процессы рёберного коллапса взвешенного графа

Таблца 3.4 -

Сравнение стратегий коллапса случайных графов

Количество вершин графа	Плотность графа (%)	Ширина одновременного коллапса	Ширина последовательного коллапса					
			Максимальное Ребро		Случайное ребро		Минимальное ребро	
			Ширина	%	Ширина	%	Ширина	%
20	20	442	35	7.9	191	44.6	231	52.3
	40	869	66	7.6	367	42.2	533	61.3
	60	1372	102	7.4	651	47.4	829	60.4
	80	1825	160	8.8	876	48.0	990	54.2
40	20	1836	73	4.0	632	34.4	1002	54.6
	40	3699	139	3.8	1664	45.0	2133	57.7
	60	5539	214	3.9	2665	48.1	2948	53.2
	80	7354	314	4.3	3608	49.0	3908	53.1
100	20	11602	160	1.4	4827	41.6	5829	50.2
	40	22973	316	1.4	7617	33.2	12341	53.7
	60	34334	501	1.5	13282	38.7	17559	51.1
	80	45582	754	1.7	17144	37.6	23008	50.5
200	20	46073	288	0.63	19673	42.7	23781	51.6
	40	91715	612	0.67	42260	46.0	91715	50.5
	60	137684	997	0.72	67609	49.1	68957	50.0
	80	183652	1486	0.81	91015	49.6	91669	49.9

Для построения табл. 3.4 использованы случайные равномерно распределённые веса рёбер в диапазоне от 4 до 20. Использование других диапазонов приводит к другим абсолютным величинам, но сохраняет процентные соотношения. Оценивалась ширина коллапса, а также сложность последовательной композиции для полиномиальных и экспоненциальных методов решения систем.

Можно сделать вывод, что худшим является выбор ребра минимального веса. Он приближается к случайному выбору ребра с ростом количества вершин. Лучшим выбором на шаге является выбор ребра максимального веса, который обеспечивает существенно меньшую ширину коллапса. Для более точно-

го приближения к оптимальному решению можно рекомендовать использование многошаговых стратегии на основе процедур мини-макс [58, 168].

3.6. Алгоритмы композиционного решения систем линейных алгебраических уравнений

Представленные далее алгоритмы могут быть применены для ускорения решения систем линейных уравнений (СЛУ), как в кольцах, так и в полях, однако наиболее существенные ускорения вычислений получены для диофантовых систем, в особенности, при нахождении их решений в неотрицательной области.

Целью настоящего подраздела является построение алгоритмов композиционного решения СЛУ и представление их программной реализации Adriana (Приложение Д) инвариантной по отношению к выбранному базовому алгоритму решения СЛУ. Следует отметить, что обеспечено встраивание программы в известную систему автоматизированного исследования свойств сетей Петри Tina [10].

Метод композиционного решения линейной системы представлен в подразделе 3.3. Построим алгоритмы нахождения композиционного решения (3.16) однородных систем, так как поиск частного решения неоднородной системы может быть выполнен с помощью алгоритмов решения однородных систем [162-166]. Композиционные алгоритмы предназначены для решения систем большой размерности, как правило, представленных разрежёнными матрицами. В качестве основной формы представления данных будем использовать поэлементное представление разрежённых матриц. Таким образом, файлы данных являются последовательностью строк вида:

i j A(i,j)

где i – номер строки, j – номер столбца, $A(i,j)$ – значение элемента. Будем указывать только значения ненулевых элементов. Кроме того, не будем предъяв-

лять дополнительные требования к порядку следования строк и нумерации элементов. Описанный формат представление матриц назовём SPM (sparse matrix) и будем использовать его как для представления матриц систем, так и для представления матриц базисных решений.

Для минимизации требований к оперативной памяти все промежуточные решения будем сохранять в отдельных файлах. Схема взаимосвязи программных модулей и файлов данных представлена на рис. 3.6.

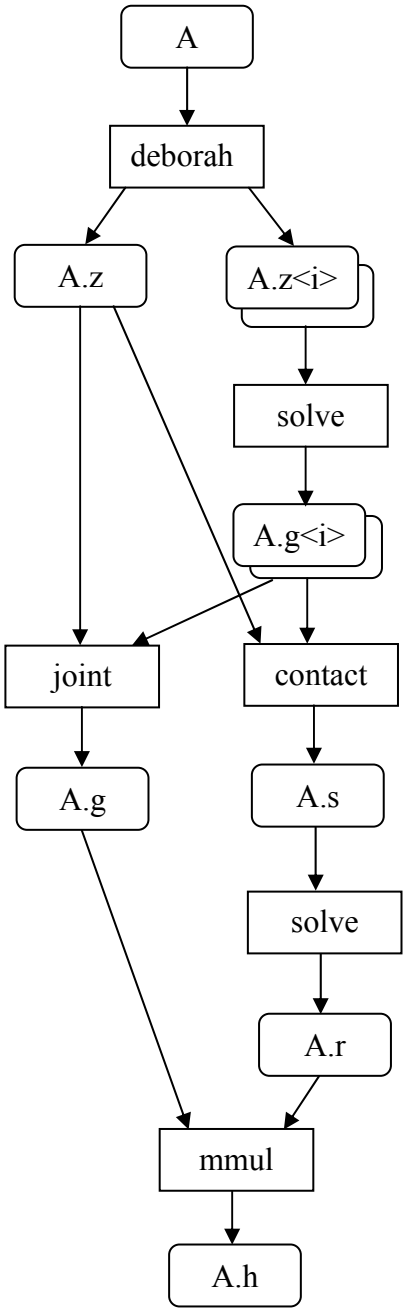


Рис. 3.6. Общая схема алгоритма композиционного решения системы

Рассмотрим назначение модулей, использованных в указанной схеме. Модуль **deborah** выполняет декомпозицию исходной системы на кланы. Каждый из кланов сохраняется в отдельном файле с расширением *.z<i>*, где *i* равно порядковому номеру клана, кроме того, создаётся файл с расширением *.z*, содержащий описание входных и выходных переменных кланов. Модуль **solve** находит базисные решения системы. Модуль **contact** строит систему уравнений для контактных переменных. Он считывает файл *.z* с номерами контактных переменных, а также файлы базисных решений кланов *.g<i>* и выводит систему композиции в файл с расширением *.s*. Модуль **joint** строит объединённую матрицу базисных решений кланов. Он считывает файл *.z* и файлы *.g<i>* и создаёт файл *.g*. Модуль **mmul** выполняет умножение разрежённых матриц. Кроме того, при решении диофантовых систем в неотрицательной области может быть использован дополнительный модуль для фильтрации полученных решений в целях минимизации базиса в решётке неотрицательных целых. Как указано в [22, 67, 162] минимальный базис состоит их минимальных в решётке элементов.

Следует обратить внимание, что схема, изображённая на рис. 3.6, инвариантна по отношению к применяемому методу решения линейной системы, инкапсулированному в модуле **solve**. Кроме того, алгоритмы работы остальных модулей инвариантны по отношению к множеству значений элементов матриц систем. Главным требованием является возможность определения и изменения знака элемента *sign(e)*, а также возможность вычисления суммы и произведения элементов при умножении матриц. Таким образом, с помощью композиционных алгоритмов можно решать системы на множествах целых и вещественных чисел, а также в произвольных полях и кольцах.

Опишем файлы данных, используемые при композиционном решении систем:

A – файл матрицы исходной системы, тип SPM;

A.z – файл контактных переменных декомпозиции, тип CVA;

A.z<i> – файл матрицы системы клана *i*, тип SPM;

A.g<i> – файл базисных решений клана *i*, тип SPM;

A.g – файл объединённой матрицы базисных решений кланов, тип SPM;

A.s – файл матрицы системы для контактных переменных, тип SPM;

A.r – файл базисных решений системы для контактных переменных, тип SPM;

A.h – файл базисных решений исходной системы, тип SPM.

Тип файла CVA (contact variable) используется для представления дополнительной информации о контактных переменных декомпозиции на кланы. Он представляет собой последовательность строк вида:

v x y

где *v* – номер контактной переменной, *x* – номер её входного клана, а *y* – номер её выходного клана. Выбор такой формы представления информации позволяет в дальнейшем оптимизировать алгоритмы работы модулей **contact** и **joint**.

```
static int is=0;
#define lnum( j ) (is+(j))

contact( nz; файл A.z; файлы A.g<i> )
{
 считать файл A.z в C; положить nc равным количеству записей;
 открыть выходной файл sFile;

 для z=1..nz
 {
   открыть файл A.g<z> как zFile;

   nl=0;
   пока( не конец_файла( zFile ) )
   {
     считать из zFile строку i, j, e;
     для ic=1..nc
     {
       если( C[ic].v==j && C[ic].x==z )
         записать в sFile строку lnum(i), j, opposite_sign( e );
       если( C[ic].v==j && C[ic].y==z )
         записать в sFile строку lnum(i), j, e;
     }
     nl=max(nl,i);
   }
   is+=nl;
 }
} /* contact */
```

Рис. 3.7. Алгоритм работы модуля **contact**

Особенности реализации модуля декомпозиции **deborah** [138] описаны в подразделе 2.6, алгоритм работы модуля умножения разрежённых матриц **mmul** тривиален, поэтому остановимся более подробно на описании модулей **contact** и **joint**.

Алгоритм работы модуля **contact**, выполняющего построение системы уравнений композиции, на Си подобном псевдоязыке представлен на рис. 3.7. Каждое уравнение создаваемой системы соответствует контактной переменной, указанной в файле *A.z*. Переменные системы композиции соответствует базисным решениям для кланов. Файл декомпозиции *A.z*, содержащий описания контактных переменных, считывается целиком и размещается в памяти. Затем последовательно считываются файлы базисных решений кланов, причём в памяти размещается только одна запись. Если запись содержит контактную переменную, являющуюся входной либо выходной для обрабатываемого клана, то выводится терм в файл системы композиции. Использование макроса *lnum(l)* позволяет организовать нумерацию всех базисных решений кланов за счёт накопления количества решений ранее обработанных кланов в переменной *is*. Информация о природе элемента матрицы инкапсулирована в функции *opposite_sign(e)*, которая определяет значение элемента противоположного знака. Сам элемент вводится и храниться как строка, и не интерпретируется алгоритмом.

Алгоритм работы модуля **joint**, выполняющего построение объединённой матрицы базисных решений кланов, на Си подобном псевдоязыке представлен на рис. 3.8. Особенность алгоритма заключается в том, что базисные решения для каждой из контактных переменных должны быть вычислены либо в соответствии с базисными решениями входного клана, либо в соответствии с базисными решениями выходного клана. Для определённости выбрано вычисление в соответствии с базисными решениями входного клана. Таким образом, не все элементы базисных решений кланов попадают в объединённую матрицу: переписывать следует либо решения для внутренних переменных, либо для входных контактных переменных. Кроме того, алгоритмом выполняется перенумерация

базисных решений. Так как элемент матрицы только копируется, он не интерпретируется алгоритмом.

```
static int is=0;
#define vnum( j ) (is+(j))

joint( nz; файл A.z; файлы A.g<i> )
{
 считать файл A.z в C; положить nc равным количеству записей;
 открыть выходной файл gFile;

 для z=1..nz
 {
   открыть файл A.g<z> как zFile;

   nl=0;
   пока( не конец_файла( zFile ) )
   {
     считать из zFile строку i, j, e;
     yes=1;
     для ic=1..nc
       если( C[ic].v==j && C[ic].x!=z ) { yes=0; прерватьцикл; }
     если(yes) вывести в gFile строку vnum(i), j, e;
     nl=max(nl,i);
   }
   is+=nl;
 }
} /* joint */
```

Рис. 3.8. Алгоритм работы модуля **joint**

При реализации композиционных алгоритмов учитывались такие требования как решение систем большой размерности, представленных разрежёнными матрицами, а также инвариантность по отношению к природе элемента матрицы и алгоритму решения системы. Исходя из этих требований каждый из модулей, указанных на рис. 3.6, реализован как отдельная программа, использующая интерфейс командной строки. Программирование выполнено на языке ANSI C в среде операционной системы Unix. Для объединения модулей использован командный файл, позволяющий легко заменять модуль **solve** фактически применяемым для решения линейной системы модулем.

Командный файл (скрипт) оболочки bash операционной системы Unix представлен на рис. 3.9. Значение переменной *nz* равно количеству кланов, по-

лученных при декомпозиции. Переменная цикла *z* используется для нумерации кланов. Предусмотрен специальный случай неразложимости системы на кланы (*nz=1*). Следует отметить, что промежуточные файлы *A.** за исключением результата *A.h* могут быть удалены.

Тестирование командного файла выполнено для модулей **solve** представляющих алгоритм Гаусса для вещественных чисел (**gauss**) и алгоритм Тудика [88], для решения диофантовых уравнений в неотрицательной области (**toudic**). Примеры композиционного решения систем представлены в следующем разделе. Использование ANSI C позволяет легко переносить систему на другие платформы. Кроме того, при необходимости командный файл также может быть закодирован на языке Си и специализирован для определённого множества алгоритмов решения системы.

```
#!/bin/bash
nz=`deborah $1`
echo decomposed into $nz subsystems
z=1
while [ $z -le $nz ]
do
  echo sybsystem $z is solved...
  solve $1.z$z $1.g$z
  z=`expr $z + 1`
done
if [ $nz -eq 1 ]
then
  mv $1.g1 $1.h
  exit
fi
contact $1
echo 'composition system has been created'
solve $1.s $1.r
echo 'composition system has been solved'
joint $1
echo 'joint matrix has been constructed'
mmul $1.r $1.g $1.h
echo 'result has been obtained'
#end
```

Рис. 3.9. Командный файл Unix композиционного решения системы

Рассмотрим более подробно пример такой реализации, предназначенный для встраивания в известную систему автоматизированного анализа сетей Пет-

ри Tina [10]. Программа Adriana (Приложение Д) выполняет поиск инвариантов сетей Петри с помощью композиционных алгоритмов. Для решения систем выбран алгоритм Тудика (модуль **toudic**) [88]. Инварианты сети Петри представляют собой целые неотрицательные решения однородной линейной диофантовой системы уравнений. Матрицей системы является матрица инцидентности сети Петри [169] для инвариантов позиций, либо транспонированная матрица инцидентности для инвариантов переходов. Как было отмечено ранее, решение таких систем является наиболее трудоёмким с вычислительной точки зрения. Поэтому ускорения вычислений, получаемые при решении таких систем, являются наиболее ощутимыми. В литературе [10, 67] отмечены примеры сетей, насчитывающих несколько десятков элементов и требующих для вычисления инвариантов более 10^{20} операций.

При реализации программы Adriana кроме кодирования командного файла на языке ANSI C и встраивания в программу перечисленных модулей, решена также задача перекодирования форматов файлов представляющих сети Петри в системе Tina. Система Tina использует два основных формата файлов для представления сетей Петри: абстрактный NET и графический NDR. Формальное описание форматов приведено в дистрибутиве системы [10]. Отметим, что оба формата используют мнемонические имена элементов сети и содержат ряд дополнительных характеристик, несущественных для вычисления инвариантов. Кроме того, формат NDR содержит положение элементов сети на плоскости. Наиболее простым представляется перекодирование NET/NDR файла в формат SPM с сохранением таблиц имён позиций и переходов сети. После композиционного решения системы таблицы имён используются для мнемонического представления инвариантов, принятого в системе Tina. Алгоритм работы программы Adriana представлен на рис. 3.10.

Файлы *A.p* и *A.t* содержат таблицы имён позиций и переходов сети Петри соответственно и представляют собой последовательность строк вида:

n name

где *n* – номер элемента, а *name* – его имя. Модуль **encode** нумерует элементы сети и создаёт файл матрицы инцидентности *A*, а также таблицы имён. Модуль **decode** представляет инварианты в мнемонической форме системы Tina:

name1[*val1] name2[*val2] ... namek[*valk]

где *namei* – имя элемента, *vali* – значение инварианта; единичные значения элементов опускаются. Промежуточные файлы создаются в каталоге временных файлов и удаляются по завершению работы программы. Программы Deborah и Adriana могут быть загружены с сайта www.geocities.com/zsoftua.

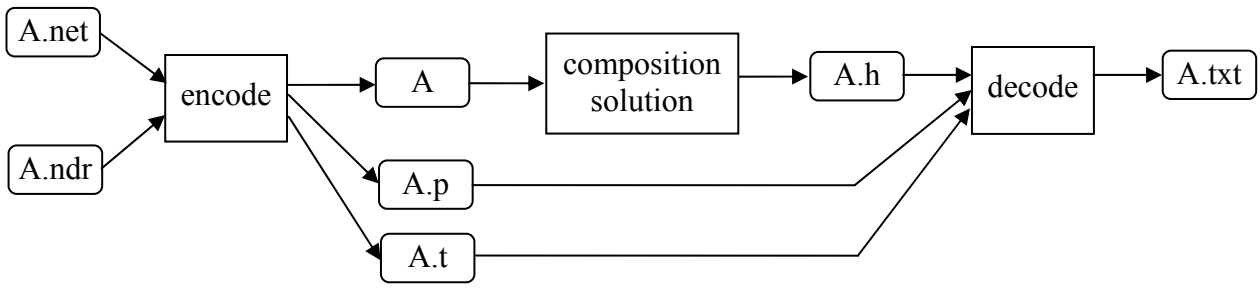


Рис. 3.10. Алгоритм работы программы Adriana

Таким образом, построены композиционнае алгоритмы решения СЛУ, предназначеные для ускорения процессов решения систем методами с вычислительной сложностью, превышающей кубическую. Для экспоненциальных методов решения систем, таких, как, например, методы решения диофантовых уравнений в неотрицательной области, ускорение вычислений экспоненциально.

Выполнена программная реализация построенных алгоритмов. Обеспечено встраивание модуля композиционного поиска инвариантов Adriana в систему автоматизированного анализа сетей Петри Tina.

Выводы к 3 разделу

1. Впервые введено понятие клана системы линейных алгебраических уравнений, являющееся обобщением понятия функциональной сети Петри для произвольных матриц над кольцами со знаком. Исследованы свойства кланов.

2. Разработан метод решения линейных систем с помощью композиции их кланов, заключающийся в том, что базисные решения исходной системы получают из базисных решений систем для кланов. Композиционный метод обеспечивает ускорение вычислений при решении систем исходными методами с временной вычислительной сложностью, превышающей кубическую. Для методов экспоненциальной временной сложности получено экспоненциальное ускорение вычислений.

3. Для получения дополнительных ускорений вычислений предложено организовать последовательный процесс композиции кланов. Соответствующая задача формализована в терминах теории графов и названа оптимальным коллапсом взвешенного графа.

4. Получены оценки верхней и нижней границ ширины оптимального коллапса, которые могут быть применены при решении задачи оптимального коллапса методом ветвей и границ. Построен и статистически обоснован эвристический алгоритм оптимального коллапса линейной временной вычислительной сложности.

5. Построены эффективные алгоритмы композиционного решения линейных систем. Выполнена их программная реализация Adriana инвариантная по отношению к базовому методу решения системы. Обеспечена переносимость модуля Adriana на различные платформы, а также встраивание в известную систему анализа сетей Петри Tina.

РАЗДЕЛ 4

СИНТЕЗ МОДЕЛЕЙ ПЕТРИ И ВЕРИФИКАЦИЯ ТЕЛЕКОММУНИКАЦИОННЫХ ПРОТОКОЛОВ

Построены методы синтеза моделей Петри телекоммуникационных протоколов и методы верификации протоколов на основе композиционного анализа сетей Петри. Выполнен синтез модели протокола IOTP. Выполнена верификация протоколов Ethernet, ECMA, TCP, BGP, IOTP.

4.1. Сеть Петри как универсальный язык спецификации протоколов

Рассмотрим язык стандартных спецификаций протоколов, описывающий исходную информацию для разработчиков телекоммуникационных систем. Как правило, это подмножество естественного английского языка, дополненное специальными правилами спецификации стандартов, обеспечивающими сравнительную точность формулировок. Кроме того, для дополнительных описаний применяют таблицы и диаграммы состояний, позволяющие уточнить вербальные спецификации. Известен также ряд специальных языков спецификации [84] для семейств протоколов, таких как CAPSL, HLPS, SMURPH, JXTA, но они не нашли широкого применения в действующих стандартах.

Сети Петри [169] являются широко используемым средством верификации телекоммуникационных протоколов [9, 29], а расширенные сети Петри [50, 180] позволяют исследовать эффективность протоколов [72]. Как правило, диаграммы состояний и схемы, используемые в стандартных спецификациях протоколов, являются представлениями конечных автоматов. Однако, конечные автоматы не позволяют отобразить параллелизм поведения систем и описать порядок их асинхронного взаимодействия. Указанные аспекты протоколов, как

правило, описывают вербально, что является потенциальным источником возможных ошибок. Сеть Петри можно использовать не только для верификации протокола, но также для его спецификации. В условиях усложнения спецификаций и роста их объема актуальной задачей становится управление степенью детализации моделей Петри.

Вербальная спецификация протоколов в стандартах содержит описания порядка взаимодействия систем, а также форматы передаваемых сообщений. При построении модели Петри может быть применен различный уровень абстрагирования от деталей спецификации и, таким образом, построены модели Петри различной степени детализации. Предложены два основных подхода к управлению степенью детализации модели:

1. Представлять фишкой сети Петри сообщение протокола.
2. Представлять фишкой сети Петри поле сообщения.

Далее построены: модель Петри протокола BGP на основе первого подхода и модель протокола TCP – на основе второго.

4.1.1. Построение модели Петри протокола BGP

Border Gateway Protocol (BGP) [62] представляет собой протокол маршрутизации трафика сети Internet между автономными системами. Он является весьма значимым для функционирования всей сети Internet, так как автономные системы составляют магистраль глобального обмена информацией. Более тридцати RFC (Requests For Comments) посвящены спецификации, уточнениям и дополнениям к протоколу BGP. В последнее время наиболее распространённой является версия BGP-4 [80], но различия в сравнении с первой стандартной спецификацией [62] являются несущественными для построения абстрактной модели приемлемой размерности.

Основной функцией систем, реализующих протокол BGP, является обмен информацией с другими системами о достижимости конкретных сетей в Internet. Эта информация о достижимости включает сведения об автономных системах, через которые необходимо направлять трафик, чтобы попасть в опре-

делённую сеть. Передаваемая информация представляет собой по существу исходные данные для построения графа взаимосвязи автономных систем, из которого могут быть найдены и удалены циклические маршруты, а также приняты решения о политике маршрутизации на уровне автономных систем.

Имеется пять типов стандартных сообщений BGP:

1. OPEN (ОТКРЫТИЕ),
2. UPDATE (МОДИФИЦИКАЦИЯ),
3. NOTIFICATION (ПРЕДУПРЕЖДЕНИЕ),
4. KEEPALIVE (АКТИВНОСТЬ),
5. OPEN CONFIRM (ПОДТВЕРЖДЕНИЕ ОТКРЫТИЯ).

После установления соединения транспортным протоколом первым сообщением, посылаемым одной из сторон должно быть сообщение OPEN. Если сообщение OPEN принято успешно, то для подтверждения этого в обратном направлении посылается сообщение OPEN CONFIRM. После того, как сообщение OPEN подтверждено, возможен обмен сообщениями UPDATE, KEEPALIVE, NOTIFICATION.

Сообщения UPDATE используются для обмена маршрутной информацией между парами систем, взаимодействующих в соответствии с протоколом BGP. Информация в пакете UPDATE используется для построения графа, описывающего взаимосвязи автономных систем. Путём применения специальных правил из этого графа могут быть определены и удалены циклические маршруты и другие аномалии глобальной маршрутизации.

Протокол BGP не использует механизмы проверки нормального функционирования соседней системы, предоставляемые транспортным протоколом, позволяющие определить достижимость соседней системы в сети. Вместо этого пара систем периодически обменивается сообщениями KEEPALIVE. Периодичность обмена указывается в заголовке пакета BGP. Сообщение KEEPALIVE представляет собой лишь заголовок протокола BGP и не содержит данных.

Сообщения NOTIFICATION передаются, когда обнаруживаются ошибки.

Модель Петри протокола BGP представлена на рис. 4.1. Модель описывает асимметричное взаимодействие двух систем. Первая система представлена позициями $p_1 - p_5$ и переходами $t_1 - t_6$, вторая система – позициями $p_6 - p_{10}$ и переходами $t_7 - t_{12}$. Позиции $p_{11} - p_{14}$ соответствуют коммуникационной подсистеме и моделируют стандартные сообщения: OPEN, OPENCONFIRM, KEEPALIVE. Заметим, что модель представляет только процедуры установления и обеспечения соединения, абстрагируясь от описания процессов передачи конкретных данных, настраивающих таблицы маршрутизации. Обмен данными выполняется в состоянии ESTABLISHED с помощью стандартного сообщения UPDATE. Смысловое описание элементов модели представлено в табл. 4.1.

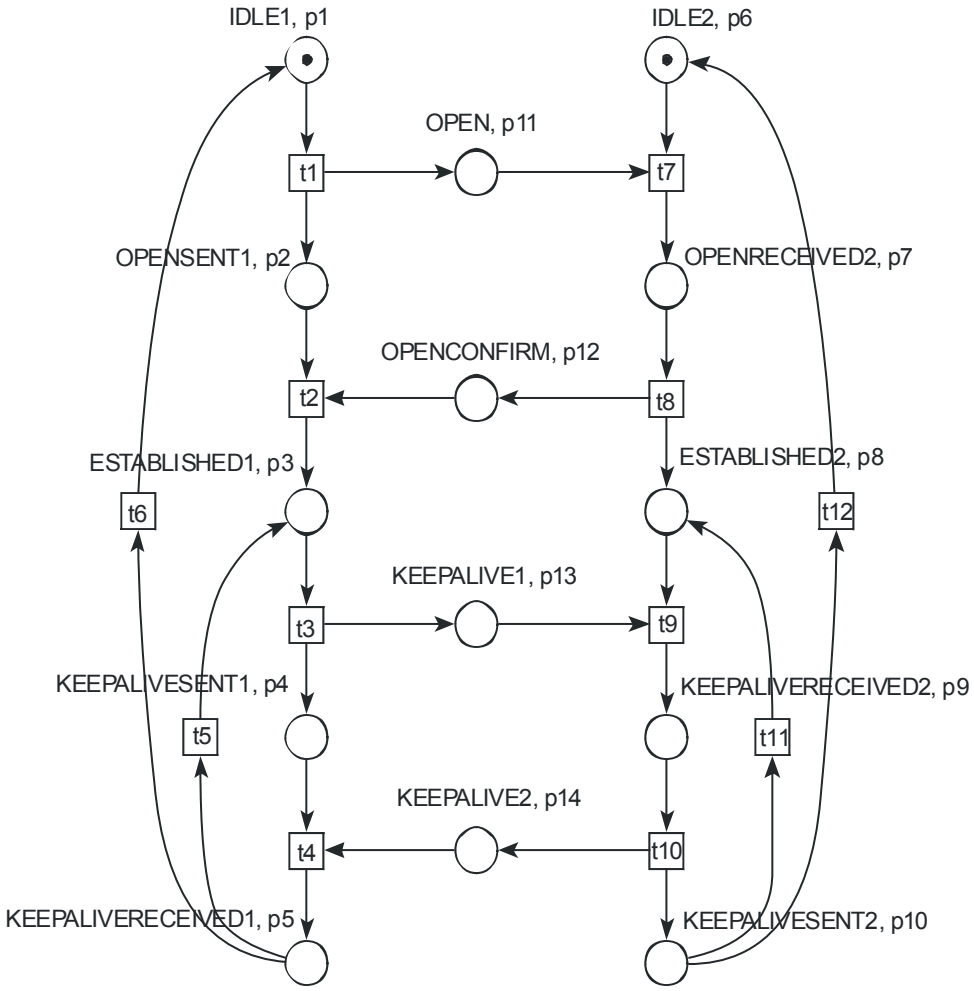


Рис. 4.1. Модель Петри протокола BGP

Таблица 4.1-

Описание элементов модели

Позиция	Описание	Переход	Описание
p_1, p_6	Начальное состояние систем	t_1	Послать сообщение OPEN
p_2	Отправлен запрос на открытие	t_7	Получить сообщение OPEN
p_7	Получен запрос на открытие	t_8	Послать сообщение OPENCONFIRM
p_3, p_8	Соединение установлено	t_2	Получить сообщение OPENCONFIRM
p_4	KEEPALIVE отправлено	t_3, t_{10}	Послать сообщение KEEPALIVE
p_9	KEEPALIVE получено	t_4, t_9	Получить сообщение KEEPALIVE
p_5	KEEPALIVE получено	t_5, t_{11}	Цикл опроса готовности
p_{10}	KEEPALIVE отправлено	t_6, t_{12}	Разъединить
p_{11}	Сообщение OPEN		
p_{12}	Сообщение OPENCONFIRM		
p_{13}, p_{14}	Сообщение KEEPALIVE		

4.1.2. Построение модели Петри протокола TCP

Стандартная спецификация протокола TCP была представлена в 1981 году в RFC 793 [76]. Этот документ явился результатом продолжительных обсуждений, отражённых, например, в RFC с номерами 44, 55, 761. В процессе эксплуатации в протокол были внесены изменения, касающиеся таких вопросов как медленный старт RFC 1122, быстрое восстановление RFC 2001, повторная передача RFC 2988. Совершенствование стандарта не прекращается и в настоящее время, о чём свидетельствуют, например, документы RFC 3360, 3481, 3562, в которых предлагаются средства надежного взаимодействия при сбросе связи, обмен информацией по беспроводным линиям, алгоритмы обмена ключами для обеспечения защиты информации.

Напомним, что протокол TCP предназначен для транспортировки потоков данных в двух направлениях между парой приложений, каждое из которых запущено на компьютере, подключенном к сети. В отличие от протокола UDP, протокол TCP предполагает установление соединения между взаимодействующими приложениями. Для однозначной идентификации приложения используется концепция гнезда (socket). Гнездо представляет собой пару, состоящую из IP-адреса и номера порта. IP-адрес является элементом протокола IP, располо-

женного на сетевом уровне иерархии ISO, и однозначно определяет хост сети. Порт представляет собой логический номер, идентифицирующий процесс внутри хоста.

Протокол TCP имеет три фазы: установление соединения, передача данных и разъединение. Формат заголовка протокола TCP, предусмотренный стандартом, представлен на рис. 4.2. Отметим, что фазы установления соединения и разъединения используют битовые флаги заголовка SYN, FIN, ACK. Остальные поля заголовка предназначены для управления обменом информацией и позволяют контролировать скорость передачи данных, их корректность, а также обеспечивать повторную передачу искажённой информации. Флаг SYN применяют для синхронизации номеров последовательностей (Sequence Number) передаваемых данных при установлении соединения. Флаг FIN инициирует завершение связи. Флаг ACK используют для подтверждения; он может быть указан совместно с другими флагами.

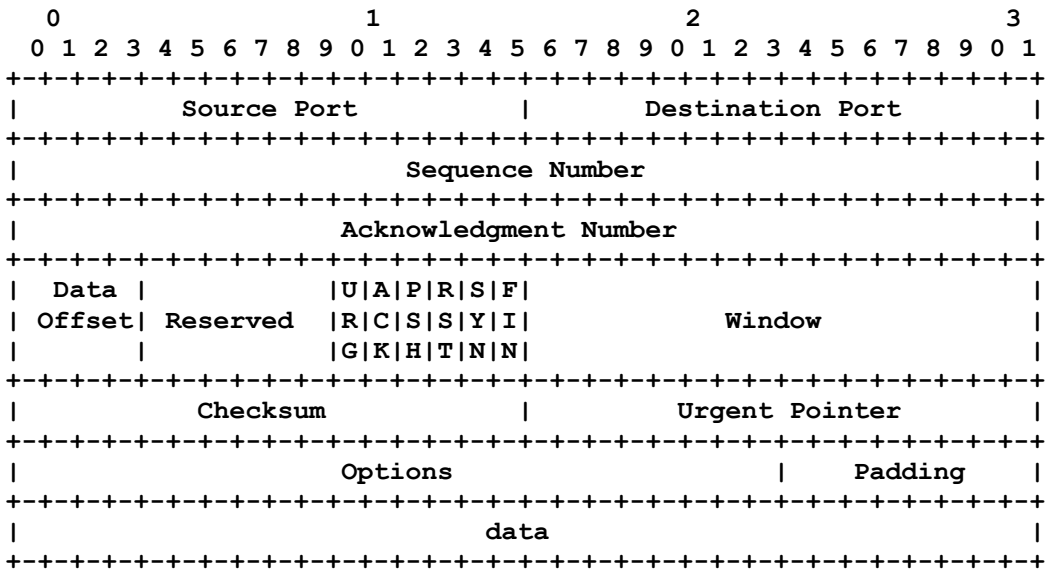


Рис. 4.2. Формат заголовка TCP

Диаграмма состояний протокола TCP, предусмотренная стандартом, представлена на рис. 4.3. Отметим, что состояния изменяются с одной стороны под влиянием команд, поступающих от протоколов прикладного уровня, таких

как OPEN, CLOSE, SEND, RECEIVE, а с другой – под воздействием полей заголовков пакетов, доставляемых протоколом сетевого уровня. Предусмотрено два типа открытия связи с помощью команды OPEN: активная (active) – когда выполняется попытка установления соединения с указанным гнездом и пассивная (passive) – когда осуществляется прослушивание указанного гнезда в целях ожидания входящих запросов на соединение. Активные команды OPEN применяют, как правило, в клиентском программном обеспечении, пассивные – в серверном. Текущее состояние связи полностью контролируется управляющим блоком TCB.

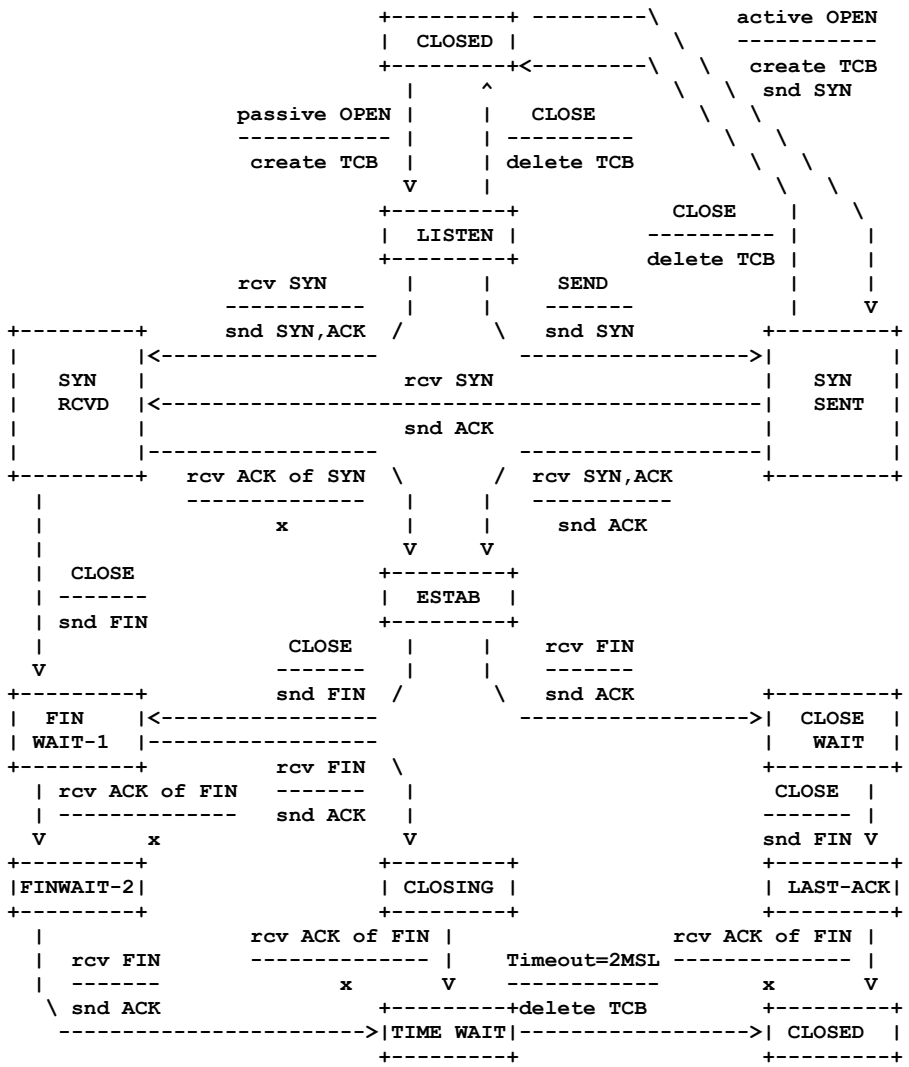


Рис. 4.3. Диаграмма состояний протокола TCP

Сокращения snd и rcvd использованы для представления посылки и получения пакета с указанным признаком соответственно. Основными являются состояния автономной работы (CLOSED), прослушивания (LISTEN), установленного соединения (ESTAB). Отметим, что обмен информацией выполняется в состоянии ESTAB. Процедура установления соединения протокола TCP известна также под названием тройного рукопожатия (three-way handshake) так как требует обмена тремя начальными пакетами. Разъединение также выполняется в три этапа.

Таким образом, даже если не рассматривать фазу обмена информацией, протокол должен обеспечивать корректную обработку каждой из четырёх указанных команд и каждого из трёх флагов в произвольные моменты времени. Наличие кроме трёх основных ещё девяти дополнительных состояний делает процесс достаточно сложным и требует применения специальных формальных методов.

Модель протокола TCP в форме сети Петри изображена на рис. 4.4. В модели (рис. 4.4) можно выделить три основных части: левая взаимодействующая система; правая взаимодействующая система; коммуникационная подсистема. Каждая из взаимодействующих систем в точности соответствует стандартной диаграмме состояний (рис. 4.3) протокола. В обозначениях правой подсистемы присутствует префикс “x”. Состояния диаграммы представлены одноимёнными позициями сети Петри. Дуги диаграммы состояний представлены переходами сети Петри. При этом использованы отдельные позиции, соответствующие флагам SYN, FIN, ACK заголовков пакетов. Эти позиции и образуют коммуникационную подсистему. Флаги пакетов, передаваемых правой взаимодействующей системой имеют префикс “x”. Отметим, что для наглядности модели флаг подтверждения ACK представлен отдельными позициями, соответствующими его получению в ответ на флаг SYN (SYNACK), либо в ответ на флаг FIN (FINACK). Кроме того, поскольку модель не содержит описаний протоколов прикладного уровня, команды OPEN, CLOSE, SEND представлены лишь в обозначениях соответствующих переходов. В качестве имён остальных переходов

выбраны первые буквы ожидаемых флагов, представленные на стандартной диаграмме состояний протокола (рис. 4.3).

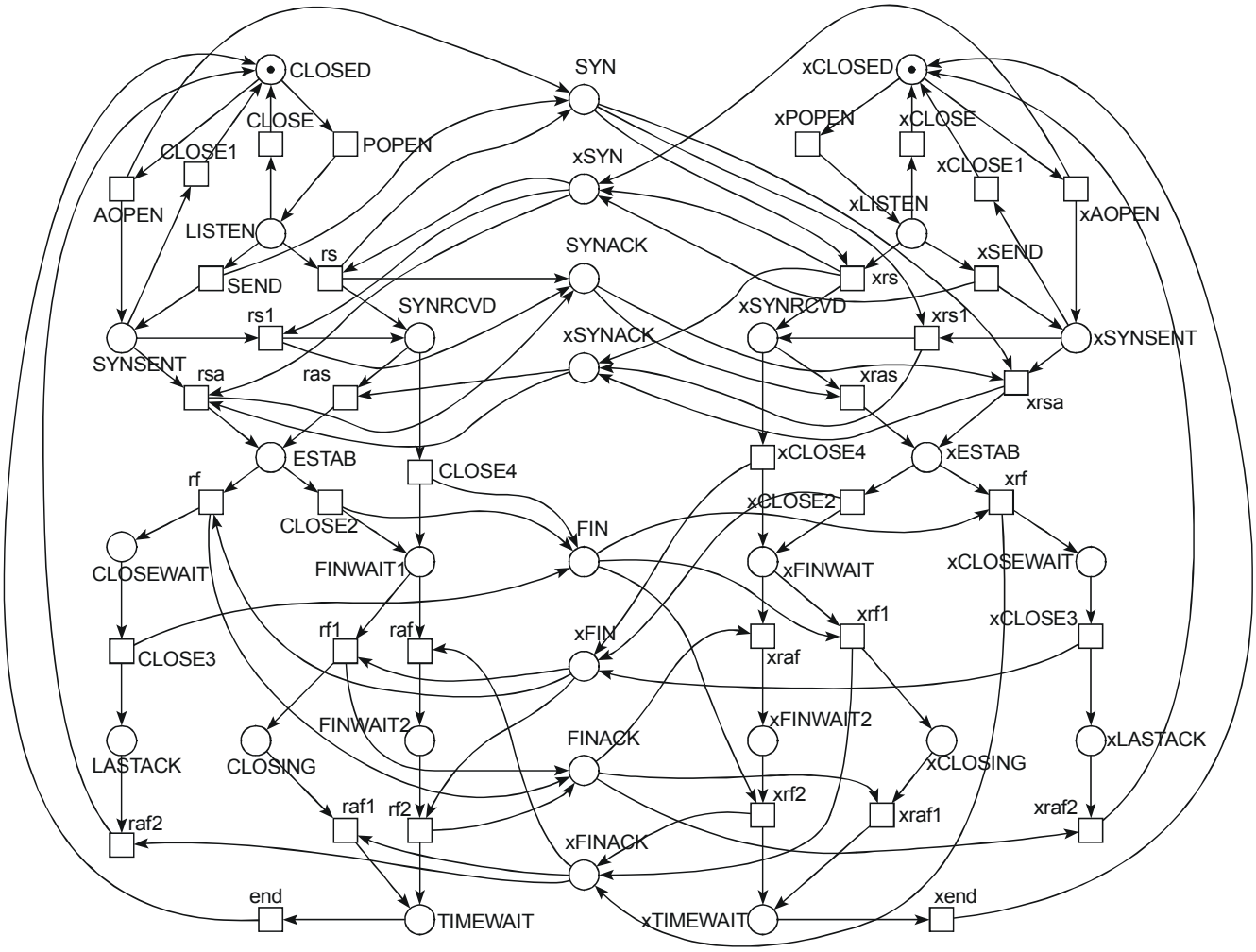


Рис. 4.4. Модель Петри протокола TCP

Исследование модели, изображённой на рис. 4.4, показало, что сеть Петри не является ограниченной. Неограниченность сети вызвана возможностью немедленного завершения не полностью установленного соединения, например, с помощью перехода CLOSE1. Повторение последовательности AOPEN, CLOSE1 приводит к неограниченному росту маркировки позиции SYN. Такие действия, как правило, являются злонамеренными и применяются в программах, предназначенных для дестабилизации систем путём «бомбардировки» пакетами. Методы предотвращения таких ситуаций рассматривались при обсуждении

средств устранения перегрузок (congestion avoidance) в FRC 896, а также при рассмотрении проблематики медленного старта в RFC 1122. В соответствии с указанными рекомендациями построена уточнённая модель протокола TCP, представленная на рис. 4.5.

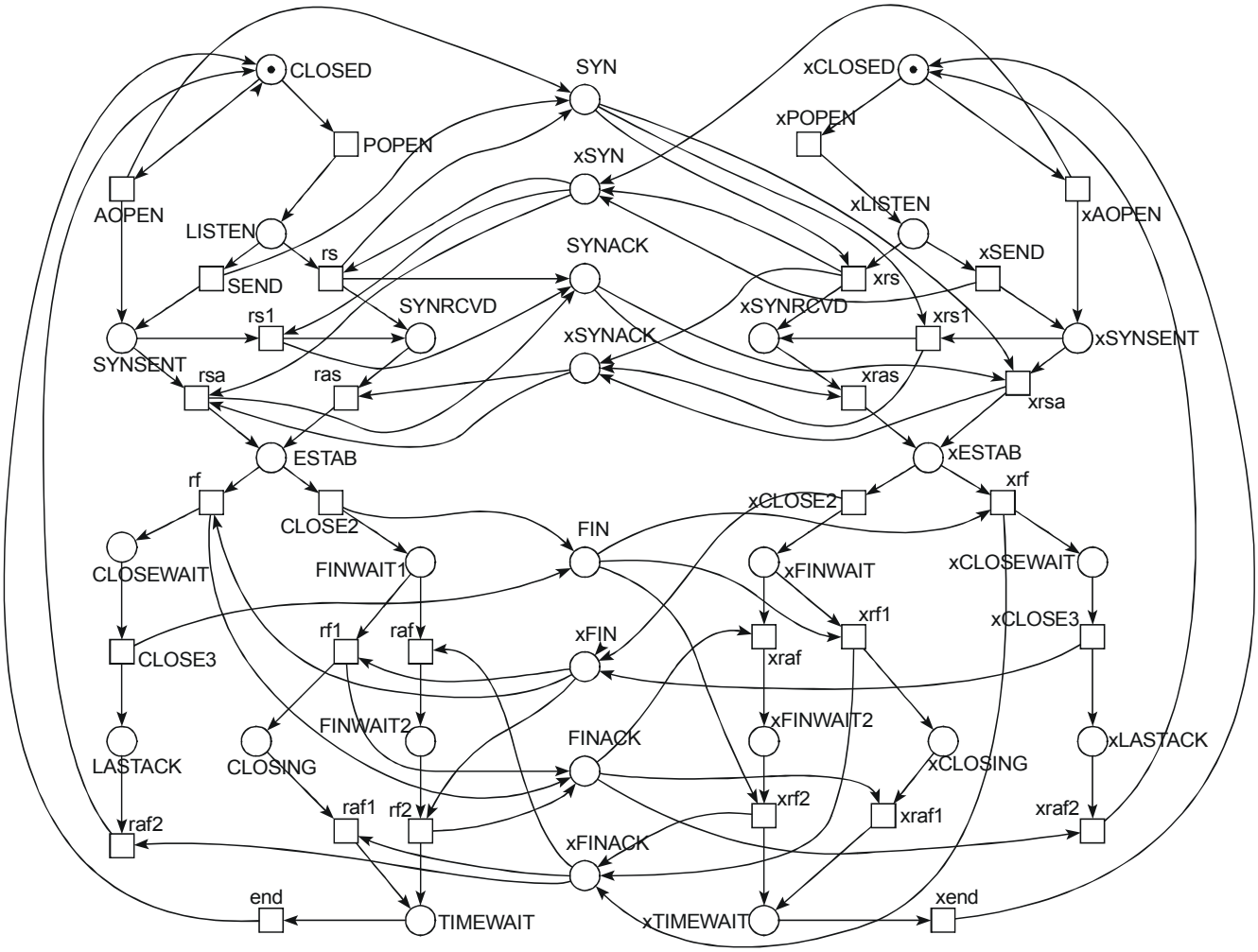


Рис. 4.5. Уточнённая модель Петри протокола TCP

Таким образом, в настоящем подразделе предложено использовать сети Петри как универсальный язык спецификации протоколов; представлены два подхода к управлению степенью детализации модели: моделировать фишкой сообщения либо поля сообщений; выполнено построение модели Петри протокола BGP на основе первого подхода и модели Петри протокола TCP – на основе второго.

4.2. Верификация протокола BGP

Выполним верификацию протокола BGP в процессе композиции функциональных подсетей (кланов) его модели Петри. Задача верификации протокола BGP представляет собой самостоятельный интерес и, кроме того, абстрактная модель Петри, изображённая на рис. 4.1 имеет сравнительно небольшой размер, что позволяет проиллюстрировать особенности применения композиционного анализа сетей Петри для верификации телекоммуникационных протоколов.

Декомпозиция модели Петри протокола BGP, полученная с помощью алгоритма 2.1, представлена на рис. 4.6 и содержит восемь минимальных функциональных подсетей $Z1,...,Z8$.

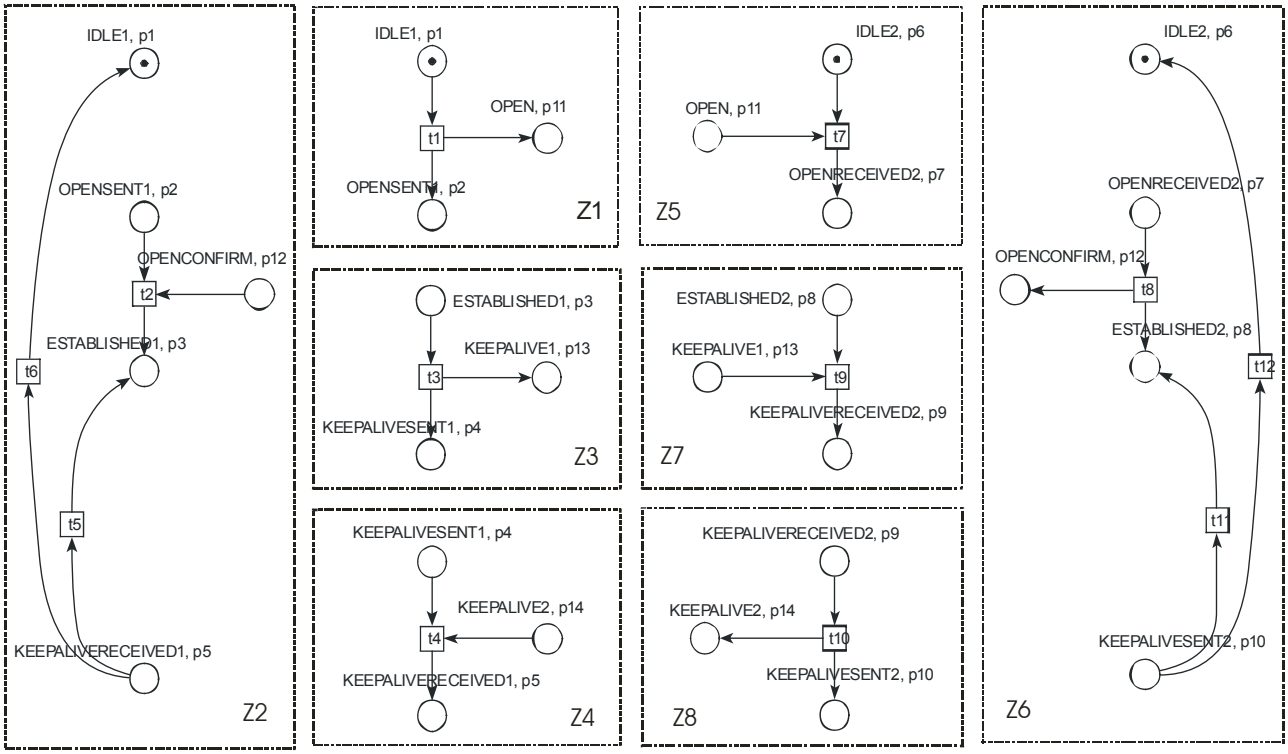
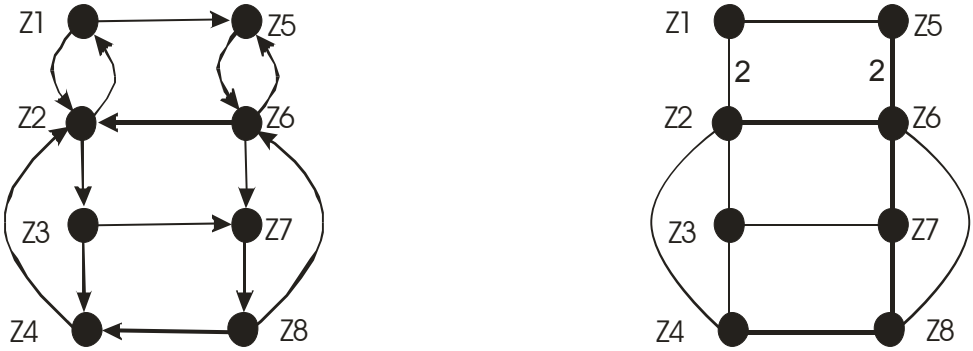


Рис. 4.6. Декомпозиция модели Петри протокола BGP
на минимальные функциональные подсети

Декомпозиция получена с помощью программы Deborah (подраздел 2.6, приложение Д). Декомпозиция, изображённая на рис. 4.6, может быть представлена ориентированным (рис. 4.7 а), либо неориентированным (рис. 4.7 б)

графом. Напомним, что веса рёбер равны количеству используемых для связи подсетей контактных позиций. Так как при решении линейных систем направление связи представлено знаком переменной, а на оценки сложности влияет лишь количество контактных позиций, далее будем использовать неориентированные графы. При композиционном вычислении инвариантов количество уравнений системы соответствует количеству контактных позиций.



а) ориентированный граф б) неориентированный граф

Рис. 4.7. Представление декомпозиции модели протокола BGP

Для вычисления инвариантов укрупним используемые функциональные подсети, объединив минимальные подсети (рис. 4.8). Так, например, подсеть Z^2 представляет собой сумму двух минимальных подсетей, порождаемых переходами t_3 и t_4 соответственно. Четыре изображённые функциональные на рис. 4.8 подсети Z^1, Z^2, Z^3, Z^4, определяют разбиение исходной модели.

С помощью метода Тудика [88, 110] получим следующие базисные инварианты подсетей, представленных на рис. 4.8:

$$Z^1:\ (x_1,x_2,x_3,x_5,x_{11},x_{12}) = (u_1^1,u_2^1)\cdot G^1,\ \ G^1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

$$Z^2:\ (x_3,x_4,x_5,x_{13},x_{14}) = (u_1^2,u_2^2,u_3^2)\cdot G^2,\ \ G^2 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix},$$

$$Z^3:\ (x_6,x_7,x_8,x_{10},x_{11},x_{12}) = (u_1^3,u_2^3)\cdot G^3,\ \ G^3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix},$$

$$Z^4:\ \ (x_8, x_9, x_{10}, x_{13}, x_{14}) = (u_1^4, u_2^4, u_3^4, u_4^4)\cdot G^4,\ \ G^4 = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

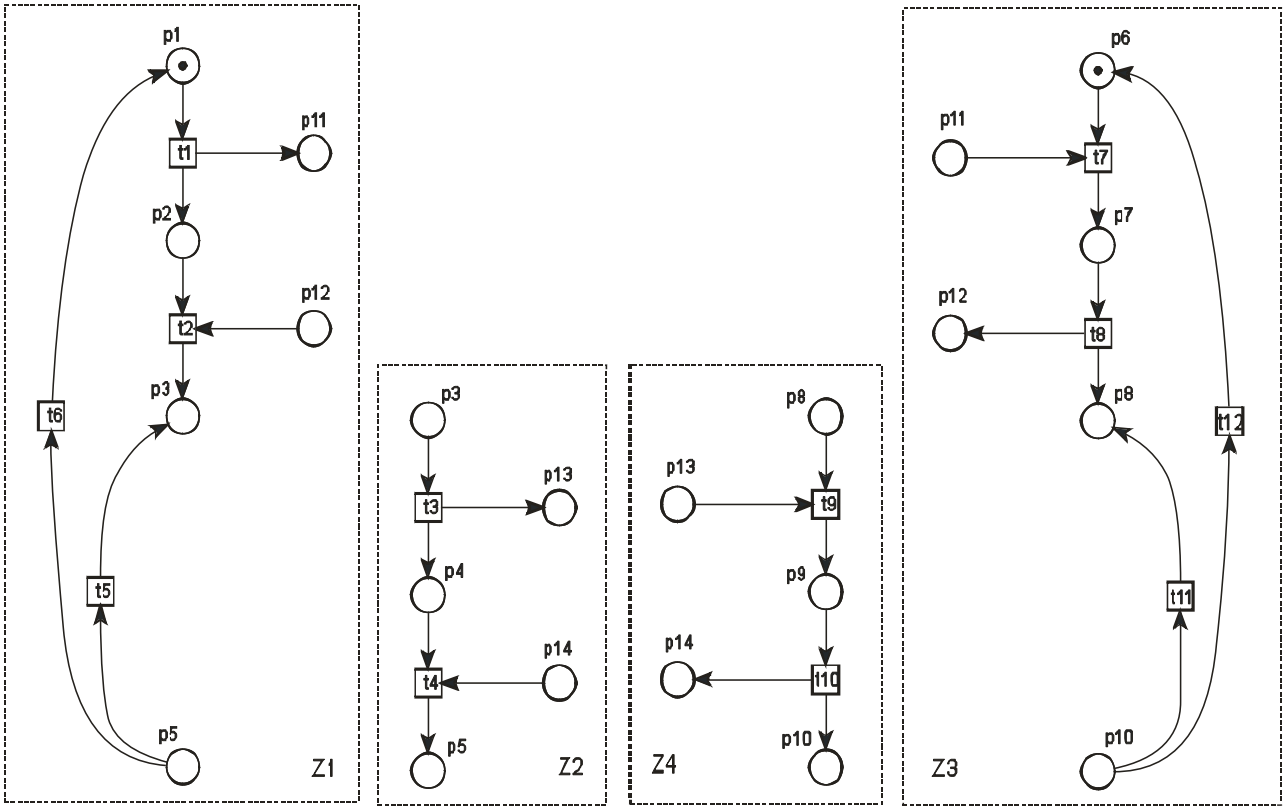


Рис. 4.8. Декомпозиция модели Петри протокола BGP

Композиция модели определяется слиянием восьми контактных позиций, указанных на рис. 4.8. Запишем систему уравнений для контактных позиций:

$$\begin{cases} p_3:\ \ u_1^1 + u_2^1 - u_1^2 - u_3^2 = 0, \\ p_5:\ \ u_1^1 + u_2^1 - u_1^2 - u_2^2 = 0, \\ p_8:\ \ u_1^3 - u_1^4 - u_2^4 = 0, \\ p_{10}:\ \ u_1^3 - u_1^4 - u_3^4 = 0, \\ p_{11}:\ \ u_2^1 - u_2^3 = 0, \\ p_{12}:\ \ u_2^1 - u_2^3 = 0, \\ p_{13}:\ \ u_3^2 - u_3^4 - u_4^4 = 0, \\ p_{14}:\ \ u_2^2 - u_2^4 - u_4^4 = 0. \end{cases}$$

Базисные решения системы по отношению к вектору неизвестных $(u_1^1, u_2^1, u_1^2, u_2^2, u_3^2, u_1^3, u_2^3, u_1^4, u_2^4, u_3^4, u_4^4)$ имеют вид

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

Построим объединённую матрицу G из матриц G^1, G^2, G^3, G^4. Матрица G может быть построена несколькими способами в зависимости от порядка вычисления инвариантов контактных позиций. Так как каждая контактная позиция инцидентна двум подсетям, то её инварианты можно вычислить двумя различными способами.

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

После перемножения матриц получим

$$H = J \cdot G = \begin{pmatrix} 1&1&1&1&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&1&1&1&1&0&0&0&0 \\ 1&0&1&1&1&0&1&0&0&0&1&1&0&0 \\ 1&1&1&0&1&0&0&0&1&0&0&0&1&1 \\ 1&0&1&0&1&0&1&0&1&0&1&1&1&1 \\ 1&1&1&0&1&1&1&1&2&1&0&0&1&1 \\ 1&0&1&0&1&1&2&1&2&1&1&1&1&1 \end{pmatrix}.$$

Исходная система имеет пять базисных решений, так как шестое решение является суммой второго и четвёртого, а седьмое – суммой второго и пятого.

Таким образом, модель протокола BGP является p-инвариантной, так как, например, инвариант

$$\bar{x}^* = (2\ 1\ 2\ 1\ 2\ 1\ 2\ 1\ 2\ 1\ 1\ 1\ 1\ 1),$$

который представляет собой сумму второго, третьего и четвёртого базисных инвариантов, содержит все натуральные компоненты. Следовательно, модель протокола консервативна и ограничена.

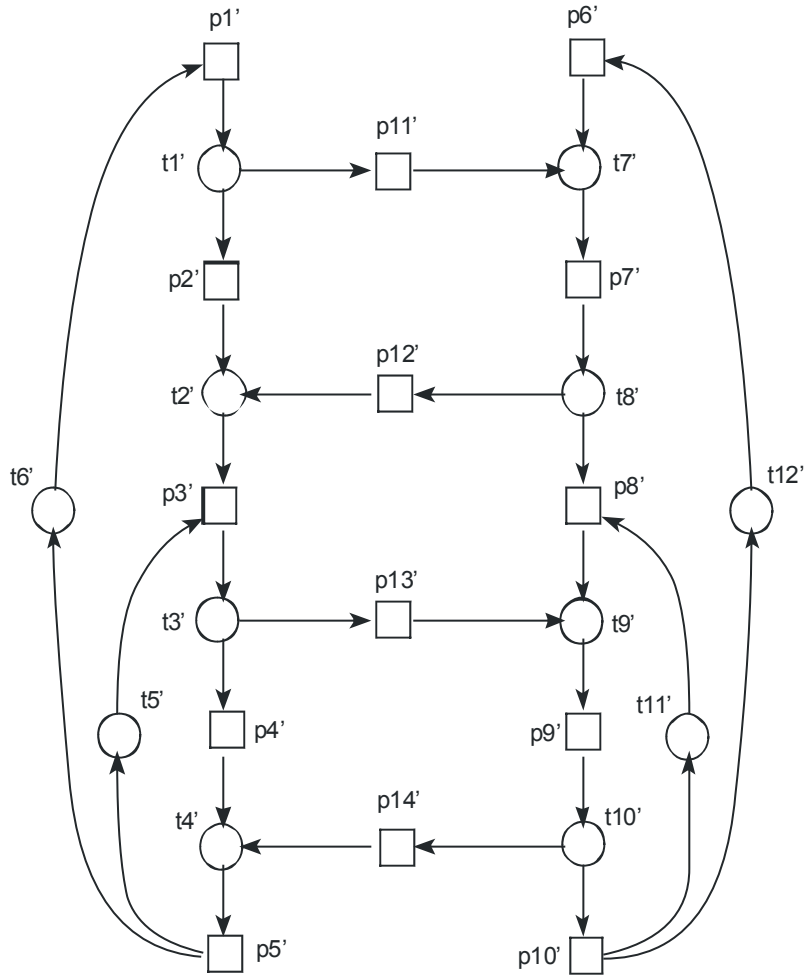


Рис. 4.9. Двойственная сеть Петри модели протокола BGP

Чтобы вычислить инварианты переходов построим двойственную сеть Петри (рис. 4.9), выполним её декомпозицию (рис. 4.10) и применим методику, описанную ранее для инвариантов позиций. Декомпозиция содержит шесть минимальных функциональных подсетей. Удобно объединить эти минимальные подсети в четыре следующих подсети: $Z^{1'}$, $Z^{2'} + Z^{5'} + Z^{6'}$, $Z^{3'}$, $Z^{4'}$. Матрица базисных инвариантов переходов имеет вид

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

Так как, например, сумма двух базисных инвариантов

$$\bar{y}^* = (1 \quad 1 \quad 2 \quad 2 \quad 1 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 1 \quad 1)$$

содержит все натуральные компоненты, то модель протокола BGP является t-инвариантной. Таким образом, модель обладает свойством стационарной повторяемости.

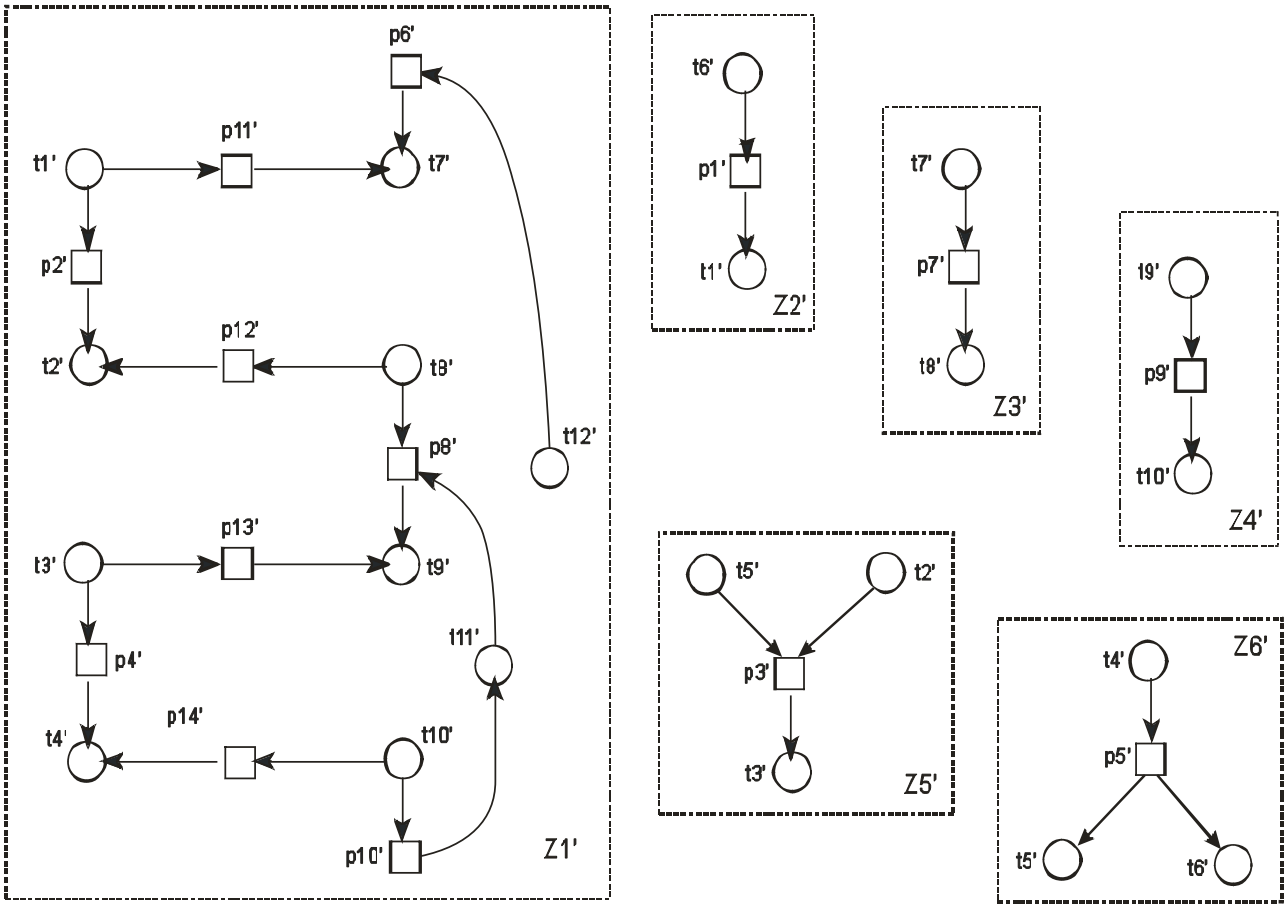


Рис. 4.10. Декомпозиция двойственной сети Петри

Ранее показано, что модель ограничена. Таким образом, можно применить анализ пространства состояний для более точного исследования модели. Пространство состояний модели представлено на рис. 4.11. Оно состоит из 21

состояния, описываемых с помощью перечисления позиций, содержащих фишку.

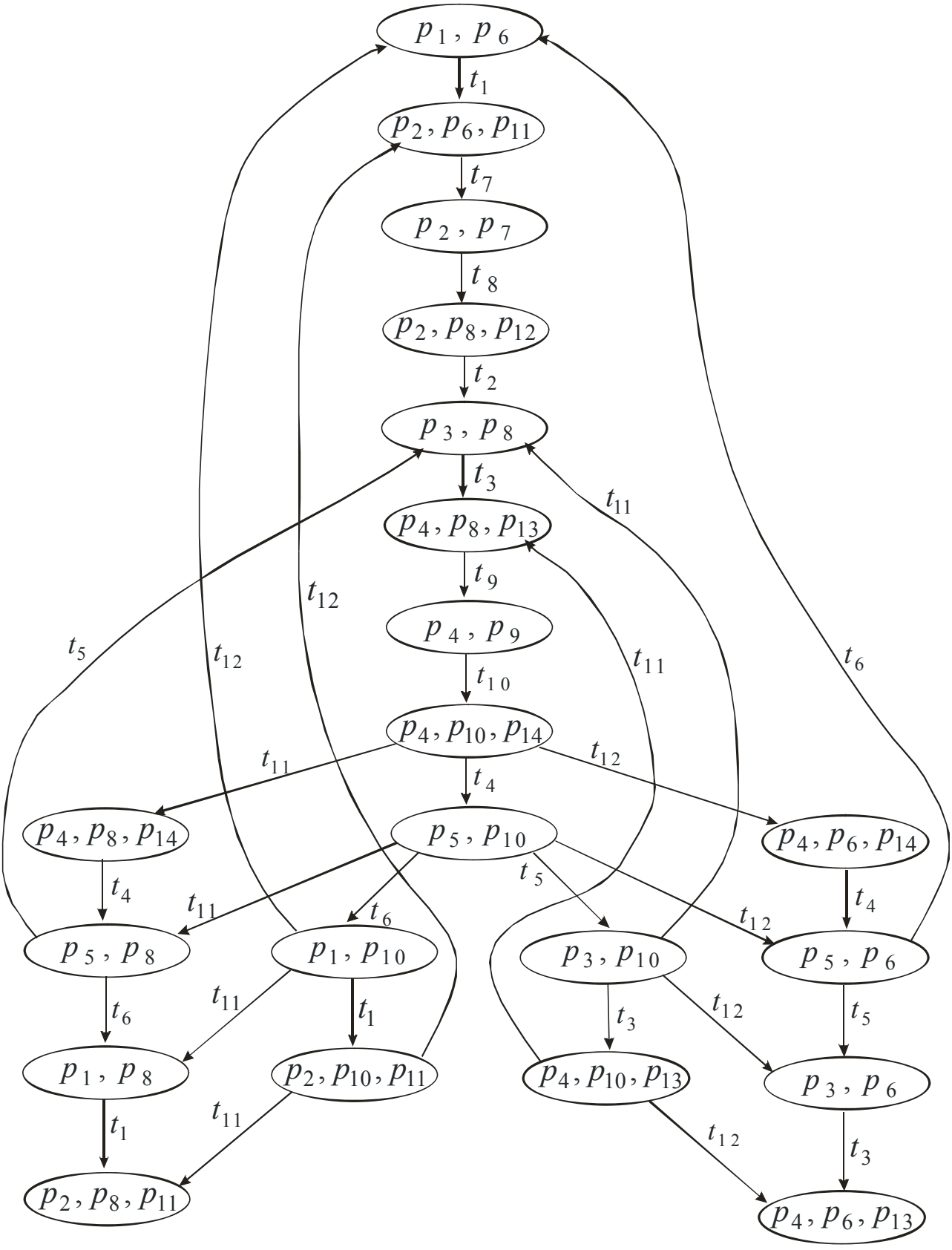


Рис. 4.11. Пространство состояний модели протокола BGP

Проверим свойства полученных инвариантов. Для всех состояний выполняется $\bar{x}^* \cdot \bar{\mu} = 3$. Таким образом, сеть консервативна. Последовательность $\sigma^* = t_1 t_7 t_8 t_2 t_3 t_9 t_{10} t_{11} t_4 t_5 t_3 t_9 t_{10} t_{12} t_4 t_6$, соответствующая инварианту $\bar{y}^*$, обеспечивает $\bar{\mu}_0 \xrightarrow{\sigma^*} \bar{\mu}_0$. Следовательно, сеть обладает свойством стационарной повторяемости.

Заметим, что хотя модель протокола BGP инвариантна, она содержит тупики (p_2, p_8, p_{11}) и (p_4, p_6, p_{13}). Этот факт можно объяснить тем, что модель не представляет таймаутов, предусмотренных стандартными спецификациями. Дополненная переходами, возвращающими каждую из систем из состояния ESTABLISHED в состояние IDLE, модель становится живой.

Выполним вычисление инвариантов модели Петри протокола BGP в процессе последовательной композиции минимальных функциональных подсетей (рис 4.6). Заметим, что одновременная композиция минимальных подсетей требует решения системы 14 уравнений. Последовательная композиция может быть реализована множеством различных способов. Дерево полного перебора насчитывает 91788 вариантов. При этом обеспечивается ширина коллапса от 3 до 7, соответствующая максимальной размерности системы на шаге. Наглядно, что даже самая худшая последовательная композиция обуславливает сложность 2^7 вместо 2^{14}, что обеспечивает ускорение вычислений примерно в 2^7 или в 128 раз. Оптимальный коллапс даёт сложность 2^3 и ускорения в $2^{11} = 2048$ раз.

На рис. 4.12 представлен фрагмент дерева полного перебора, содержащего одну последовательность для каждого возможного значения ширины коллапса. Точное количество вариантов для каждого из случаев приведено в табл. 4.2. В подразделе 3.5 представлен и статистически обоснован эвристический алгоритм оптимального коллапса, состоящий в выборе на шаге ребра с максимальным весом. Алгоритм имеет линейную сложность и даёт достаточно хорошее приближение. Для модели протокола BGP все 48 возможных вариантов

применения правила "ребро максимального веса" дают оптимальную ширину коллапса, равную 3.

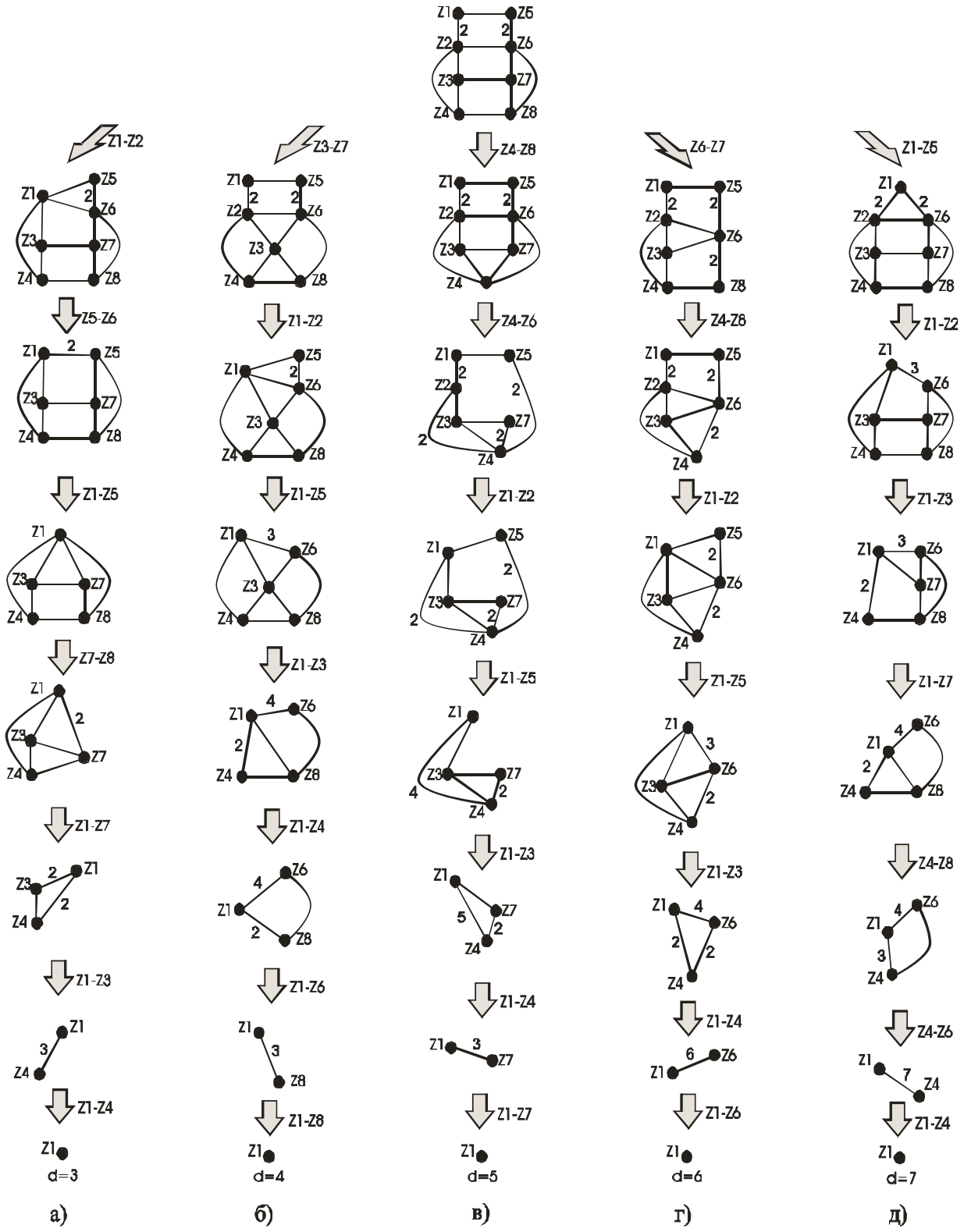


Рис. 4.12. Фрагмент дерева полного перебора коллапса взвешенного графа

Таблица 4.2 -

Количество вариантов коллапса графа

Ширина коллапса	3	4	5	6	7	Всего
Количество вариантов	16040	35956	26832	10080	2880	91788

Вычислим инварианты позиций модели протокола BGP в процессе последовательной композиции из минимальных функциональных подсетей в соответствии с оптимальной последовательностью, представленной на рис. 4.12 а).

I. Вычислим инварианты минимальных функциональных подсетей (табл. 4.3):

Таблица 4.3 -

Базисные инварианты минимальных функциональных подсетей

Подсеть	Базисные инварианты	Подсеть	Базисные инварианты
$Z1$	| 1 2 3 ----+---------- 1| 1 1 0 2| 1 0 1 ----+----------	$Z5$	| 2 10 11 ----+---------- 1| 0 1 1 2| 1 0 1 ----+----------
$Z2$	| 1 3 4 5 9 ----+---------- 1| 1 0 1 1 1 2| 1 1 0 1 1 ----+----------	$Z6$	| 4 10 11 12 14 ----+---------- 1| 1 0 1 0 0 2| 0 1 1 1 1 ----+----------
$Z3$	| 5 6 7 ----+---------- 1| 1 1 0 2| 1 0 1 ----+----------	$Z7$	| 6 12 13 ----+---------- 1| 1 0 1 2| 0 1 1 ----+----------
$Z4$	| 7 8 9 ----+---------- 1| 0 1 1 2| 1 0 1 ----+----------	$Z8$	| 8 13 14 ----+---------- 1| 1 1 0 2| 0 1 1 ----+----------

Для вычисления инвариантов использована программа Adriana, реализующая модифицированный метод Тудика (Приложения Б, Д). Напомним, что в матричном представлении номер строки соответствует номеру базисного инварианта; столбцы соответствуют позициям сети Петри.

II. Выполним последовательную композицию (табл. 4.4):

Таблица 4.4 -

Последовательная композиция минимальных функциональных подсетей

Композиция	Система композиции	Базисные инварианты полученной подсети
$Z1 + Z2$ $\rightarrow Z1,2$	| 1 2 3 4 ----+---------- 1| -1 -1 1 1 3| 0 1 0 -1 ----+----------	| 1 2 3 4 5 9 ----+---------- 1| 1 0 1 0 1 1 2| 1 1 0 1 1 1 ----+----------
$Z5 + Z6$ $\rightarrow Z5,6$	| 1 2 3 4 ----+---------- 10| -1 0 0 1 11| 1 1 -1 -1 ----+----------	| 2 4 10 11 12 14 ----+---------- 1| 0 0 1 1 1 1 2| 1 1 0 1 0 0 ----+----------
$Z1,2 + Z5,6$ $\rightarrow Z1,2,5,6$	| 2 4 ----+------- 2| 1 -1 4| -1 1 ----+-------	| 1 2 3 4 5 9 10 11 12 14 ----+---------- 1| 1 1 0 1 1 1 0 1 0 0 2| 1 0 1 0 1 1 0 0 0 0 3| 0 0 0 0 0 0 1 1 1 1 ----+----------
$Z7 + Z8$ $\rightarrow Z7,8$	| 1 2 3 4 ----+---------- 13| 1 1 -1 -1 ----+----------	| 6 8 12 13 14 ----+---------- 1| 1 1 0 1 0 2| 1 0 0 1 1 3| 0 1 1 1 0 4| 0 0 1 1 1 ----+----------
$Z1,2,5,6 + Z7,8$ $\rightarrow Z1,2,5,6,7,8$	| 3 5 6 7 ----+---------- 12| 1 0 -1 -1 14| -1 1 0 1 ----+----------	| 1 2 3 4 5 6 8 9 10 11 12 13 14 ----+---------- 1| 0 0 0 0 0 0 0 0 1 1 1 1 1 2| 1 1 0 1 1 0 0 1 0 1 0 0 0 3| 1 0 1 0 1 0 0 1 0 0 0 0 0 4| 0 0 0 0 0 1 1 0 0 0 0 1 0 ----+----------
$Z1,2,5,6,7,8 + Z3$ $\rightarrow Z1,2,3,5,6,7,8$	| 2 3 4 5 6 ----+---------- 5| 1 1 0 -1 -1 6| 0 0 -1 1 0 ----+----------	| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 ----+---------- 1| 1 1 0 1 1 0 1 0 1 0 1 0 0 0 2| 1 1 0 1 1 1 0 1 1 0 1 0 1 0 3| 1 0 1 0 1 0 1 0 1 0 0 0 0 0 4| 1 0 1 0 1 1 0 1 1 0 0 0 1 0 5| 0 0 0 0 0 0 0 0 0 1 1 1 1 1 ----+----------
$Z1,2,3,5,6,7,8 + Z4$ $\rightarrow Z1,2,3,4,5,6,7,8$	| 1 2 3 4 6 7 ----+---------- 7| 1 0 1 0 0 -1 8| 0 1 0 1 -1 0 9| -1 -1 -1 -1 1 1 ----+----------	| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 ----+---------- 1| 1 1 0 1 1 0 1 0 1 0 1 0 0 0 2| 1 0 1 0 1 0 1 0 1 0 0 0 0 0 3| 1 1 0 1 1 1 0 1 1 0 1 0 1 0 4| 1 0 1 0 1 1 0 1 1 0 0 0 1 0 5| 0 0 0 0 0 0 0 0 0 1 1 1 1 1 ----+----------

Для вычисления инвариантов в процессе последовательной композиции функциональных подсетей использована программа Adriana (Приложение Д), алгоритм работы которой представлен в подразделе 3.6. Отметим, что решение системы представляет собой множество базисных инвариантов соответствующей подсети, полученной в результате композиции. Таким образом, решение, указанное в последней строке таблицы, представляет собой множество базисных инвариантов позиций исходной сети Петри. Это решение совпадает с решением, ранее полученным с помощью одновременной композиции подсетей, а также с решением, полученным обычными методами, например, с помощью системы Tina [10].

Остановимся более подробно на представлении системы композиции. Строки матрицы соответствуют уравнениям системы и индексированы номерами контактных позиций. Столбцы соответствуют вспомогательным переменным, которые индексируют базисные инварианты функциональных подсетей, участвующих в композиции. Так, например, система композиции, представленная в первой строке таблицы имеет вид:

$$\begin{cases} -z_1 - z_2 + z_3 + z_4 = 0, \\ z_2 - z_4 = 0. \end{cases}$$

Переменная z_1 соответствует первому базисному инварианту подсети $Z1$; переменная z_2 – второму базисному инварианту подсети $Z1$; переменная z_3 – первому базисному инварианту подсети $Z2$; переменная z_4 – второму базисному инварианту подсети $Z2$. Первое уравнение соответствует контактной позиции p_1. Для построения этого уравнения рассмотрим столбец базисных инвариантов подсетей $Z1$ и $Z2$, индексированный номером позиции p_1. Для каждой из подсетей столбец содержит единицы; для входящей подсети значение берётся с плюсом, для исходящей – с минусом. Второе уравнение соответствует контактной позиции p_3. Ненулевым в столбце, индексированном номером позиции p_3, яв-

ляется лишь элемент второй строки, как для подсети $Z1$, так и для подсети $Z2$. В табл. 4.4 приведен лишь конечный результат вычислений в виде матрицы базисных инвариантов.

Для сравнения процессов одновременной и последовательной композиции модели Петри протокола BGP приведём матрицу системы одновременной композиции и полученные базисные инварианты (табл. 4.5):

Таблица 4.5 -

Одновременная композиция минимальных функциональных подсетей

Система одновременной композиции

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	-1	-1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0
3	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0
5	0	0	1	1	-1	-1	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0	0	-1	0	0	0
7	0	0	0	0	0	1	0	-1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0
9	0	0	-1	-1	0	0	1	1	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0
11	0	0	0	0	0	0	0	0	1	1	-1	-1	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	-1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	1	-1	-1
14	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1

Базисные инварианты

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	0	0	0	0	0	0	0	1	1	1	1	1
2	1	0	1	0	1	0	1	0	1	0	0	0	0	0
3	1	1	0	1	1	0	1	0	1	0	1	0	0	0
4	1	0	1	0	1	1	0	1	1	0	0	0	1	0
5	1	1	0	1	1	1	0	1	1	0	1	0	1	0

Заметим, что 16 переменных системы композиции соответствуют 16-ти базисным инвариантам всех минимальных функциональных подсетей.

В настоящем подразделе рассмотрен достаточно абстрактный пример модели Петри, содержащей 14 позиций и 12 переходов. Детализированная модель такого сложного протокола, как BGP, с учётом дополнений и изменений к первоначальному стандарту требует использования сотен вершин сети Петри. Мы абстрагировались от содержимого сообщений, которыми обмениваются системы. Примером того, каким образом поля пакетов могут быть представлены в модели, является модель протокола TCP, изученная в следующем подразделе. Изучение сравнительно небольшой модели позволило представить подробное описание методологии одновременной и последовательной композиции функциональных подсетей на примере конкретного телекоммуникационного протокола.

Таким образом, в настоящем подразделе представлена и детально изучена на примере верификации протокола BGP методология одновременной и последовательной композиции моделей Петри из функциональных подсетей (кланов). Методология предназначена для анализа и синтеза моделей Петри больший размерности, таких, как, например, детализированные модели телекоммуникационных протоколов.

4.3. Верификация протокола TCP

Трудно переоценить значимость протокола TCP [76, 84] для обеспечения коммуникации информации в современном мире. TCP является основным транспортным протоколом Internet. Именно посредством TCP ежедневно передаётся более двухсот петабит информации, как общего пользования, так и корпоративной. В семействе протоколов TCP/IP на долю TCP приходится около 80% всего информационного потока и лишь оставшиеся 20% всего объёма информации доставляется посредством протокола UDP. Таким образом, формальное доказательство корректности протокола TCP имеет ключевое значение для обоснования надёжности функционирования современных глобальных сетей. В условиях вирусных атак необходима уверенность в том, что исходные спецификации протоколов не содержат скрытых дефектов. Это обусловлено тем, что механизм внедрения большинства интеллектуальных вирусов основан на использовании дефектов либо в программном обеспечении, реализующем определённые протоколы, либо в самих спецификациях протоколов.

Выполним декомпозицию уточнённой модели протокола TCP, представленной на рис. 4.5, на минимальные функциональные подсети в соответствии с алгоритмом 2.1, с помощью программы Deborah (подраздел 2.6, Приложение Д).

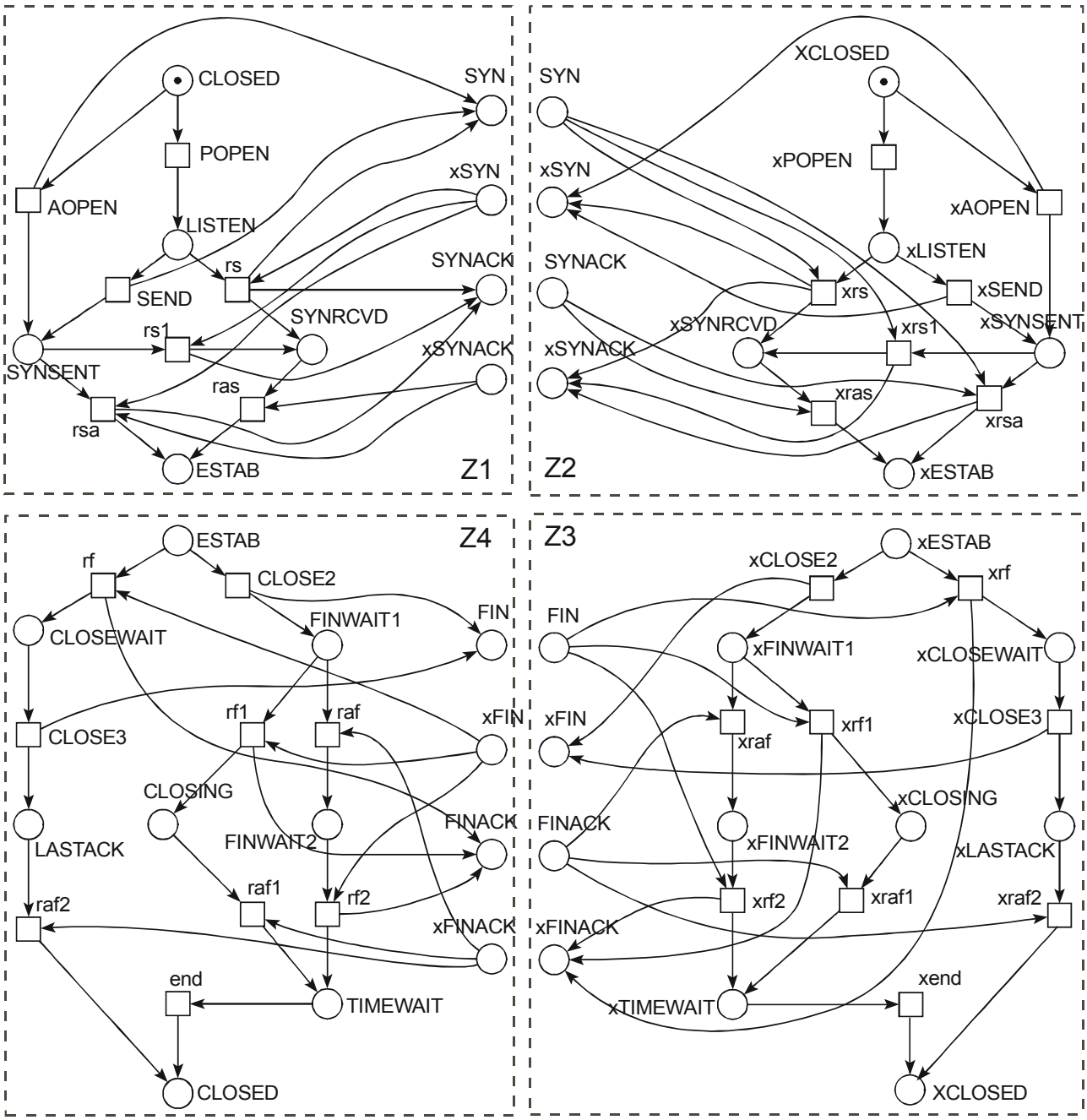


Рис. 4.13. Декомпозиция модели Петри протокола TCP

Применение алгоритма декомпозиции 2.1 [93, 144] к модели протокола TCP (рис. 4.5) приводит к получению множества $\{Z1, Z2, Z3, Z4\}$, состоящего из четырёх минимальных функциональных подсетей, представленных на рис. 4.13. Граф функциональных подсетей [93, 132, 144] изображён на рис. 4.14. Заметим, что в силу симметрии процессов взаимодействия систем, пары подсетей $Z1$ и

$Z2$, а также $Z3$ и $Z4$ являются изоморфными. Поэтому в дальнейшем необходимо исследовать свойства лишь двух из перечисленных четырёх подсетей.

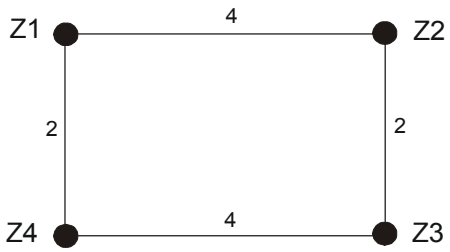
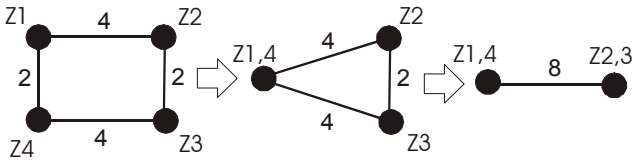


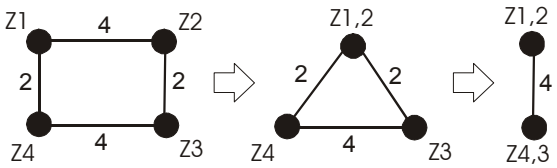
Рис. 4.14. Граф декомпозиции модели Петри протокола TCP

Различные способы композиция минимальных функциональных подсетей позволяет получить декомпозицию исходной модели на левую и правую взаимодействующие системы Z^L и Z^R, а также декомпозицию на сеть, устанавливающую соединение и сеть, выполняющую разъединение Z^C и Z^D, где $Z^L = Z1 + Z4$, $Z^R = Z2 + Z3$, $Z^C = Z1 + Z2$, $Z^D = Z3 + Z4$.

Выполним вычисление инвариантов построенной модели Петри протокола TCP (рис. 4.5) в процессе последовательной композиции функциональных подсетей. Следует отметить, что построенная модель позволяет наглядно проиллюстрировать эффективность композиционных методов. Хотя она имеет сравнительно небольшую размерность – 30 позиций и 32 переходов, вычисление инвариантов для целочисленных генераторов с помощью известной программы Tina [10] не было завершено в течение трёх суток на компьютере Pentium. Для композиционного вычисления инвариантов с помощью специально разработанной программы Adriana (Приложение Д) потребовалось всего лишь десять секунд.



а) Z1+Z4; Z2+Z3; Z1,4+Z2,3



б) Z1+Z2; Z4+Z3; Z1,2+Z4,3

Рис. 4.15. Последовательная композиция протокола TCP

Процесс последовательной композиции применён в [147] на интуитивном уровне; при этом использована последовательность, представленная на рис. 4.15 а). В разделе 3 задача формализована в терминах теории графов и названа оптимальным коллапсом взвешенного графа. Показано, что минимальную ширину, соответствующую размерности промежуточных систем композиции, обеспечивает коллапс, представленный на рис. 4.15 б). Выполним вычисление инвариантов в процессе оптимальной последовательной композиции, изображённой на рис. 4.15 б) и обеспечивающей ширину коллапса, равную 4.

Занумеруем позиции сети в соответствии с табл. 4.6 для вычисления инвариантов. Базисные инварианты подсетей $Z1$ и $Z4$ вычислены с помощью алгоритма Тудика [88, 110]. Инварианты изоморфных подсетей $Z2$ и $Z3$ построены из полученных инвариантов.

Таблица 4.6 -

Нумерация позиций сети

Номер	Имя	Номер	Имя	Номер	Имя
1	CLOSED	11	TIMEWAIT	21	XLISTEN
2	LISTEN	12	SYN	22	XSYNSENT
3	SYNSENT	13	XSYN	23	XSYNRCVD
4	SYNRCVD	14	SYNACK	24	XESTAB
5	ESTAB	15	xSYNACK	25	XCLOSEWAIT
6	CLOSEWAIT	16	FIN	26	xFINWAIT1
7	FINWAIT1	17	XFIN	27	XLASTACK
8	LASTACK	18	FINACK	28	XCLOSING
9	CLOSING	19	xFINACK	29	xFINWAIT2
10	FINWAIT2	20	xCLOSED	30	XTIMEWAIT

1) Композиция: Z1+Z2

По отношению к нумерации позиций, заданной табл. 4.6, инварианты подсетей $Z1$ и $Z2$ могут быть представлены как

$$(x_1, x_2, x_3, x_4, x_5, x_{12}, x_{13}, x_{14}, x_{15}) = (z_1^1, z_2^1, z_3^1, z_4^1, z_5^1, z_6^1) \cdot G^1,$$

$$(x_{20}, x_{21}, x_{22}, x_{23}, x_{24}, x_{13}, x_{12}, x_{15}, x_{14}) = (z_1^2, z_2^2, z_3^2, z_4^2, z_5^2, z_6^2) \cdot G^2,$$

где указанные матрицы имеют вид

$$G^1 = G^2 = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{vmatrix},$$

Компоненты векторов $\bar{x}^j$, соответствующие подсетям $Z1$ и $Z2$ записаны в явной форме. Они определяют индексацию столбцов построенных матриц. Индексы строк соответствую компонентам векторов

$$\bar{z}^1 = (z_1^1, z_2^1, z_3^1, z_4^1, z_5^1, z_6^1), \bar{z}^2 = (z_1^2, z_2^2, z_3^2, z_4^2, z_5^2, z_6^2).$$

Построим систему композиции для контактных позиций:

$$\begin{cases} p_{12}: & z_4^1 - z_3^2 - z_6^2 = 0, \\ p_{13}: & z_4^2 - z_3^1 - z_6^1 = 0, \\ p_{14}: & z_2^1 + z_6^1 - z_5^2 = 0, \\ p_{15}: & z_2^2 + z_6^2 - z_5^1 = 0. \end{cases}$$

В композиции подсетей $Z1$ и $Z2$ использованы такие контактные позиции как p_{12}, p_{13}, p_{14}, p_{15}. Общее решение системы имеет вид

$$(z_1^1, z_2^1, z_3^1, z_4^1, z_5^1, z_6^1, z_1^2, z_2^2, z_3^2, z_4^2, z_5^2, z_6^2) = \bar{y} \cdot R^{1,2},$$

$$R^{1,2} = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{vmatrix}.$$

Для вычисления базисных инвариантов подсети $Z1,2$ построим объединённую матрицу $G^{1,2}$ из инвариантов подсетей G^1 и G^2:

$$G^{1,2} = \begin{vmatrix} 1&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \\ 1&1&1&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&1&1&0&1&0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&1&1&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{vmatrix}.$$

Индексация столбцов соответствует вектору

$$(x_1, x_2, x_3, x_4, x_5, x_{12}, x_{13}, x_{14}, x_{15}, x_{20}, x_{21}, x_{22}, x_{23}, x_{24}).$$

Инварианты контактных позиций вычислены в соответствии с матрицей подсети $Z1$. Матрица базисных решений $H^{1,2} = R^{1,2} \cdot G^{1,2}$ имеет вид

$$H^{1,2} = \begin{vmatrix} 0&0&0&0&0&0&0&0&0&1&1&1&1&1 \\ 1&1&1&1&1&0&0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&1&0&0&0&0&0&0&1&1 \\ 0&0&0&1&1&0&1&0&0&1&1&0&0&0 \\ 0&0&0&0&0&0&1&1&0&1&1&0&0&1 \\ 1&1&1&0&0&0&0&1&0&0&0&0&0&1 \\ 1&1&0&0&1&1&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&1&1&1&1&0&0 \end{vmatrix}.$$

2) Композиция: Z4+Z3

Инварианты подсетей $Z4$ и $Z3$ могут быть представлены как

$$(x_1, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{16}, x_{17}, x_{18}, x_{19}) = (z_1^4, z_2^4, z_3^4, z_4^4, z_5^4, z_6^4) \cdot G^4,$$

$$(x_{20}, x_{24}, x_{25}, x_{26}, x_{27}, x_{28}, x_{29}, x_{30}, x_{17}, x_{16}, x_{19}, x_{18}) = (z_1^3, z_2^3, z_3^3, z_4^3, z_5^3, z_6^3) \cdot G^3,$$

где указанные матрицы имеют вид

$$G^4 = G^3 = \begin{vmatrix} 1&1&1&1&1&1&1&1&0&0&0&0 \\ 1&0&1&0&1&1&0&1&0&1&0&0 \\ 0&1&0&1&0&0&1&0&0&0&1&0 \\ 1&0&0&0&0&0&1&1&0&0&0&1 \\ 0&1&1&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&1&0 \end{vmatrix}.$$

Система композиции для контактных позиций имеет вид

$$\begin{cases} p_{16}: & z_5^4 - z_2^3 - z_6^2 = 0, \\ p_{17}: & z_5^3 - z_2^4 - z_6^2 = 0, \\ p_{18}: & z_3^4 + z_6^4 - z_4^3 = 0, \\ p_{19}: & z_3^3 + z_6^3 - z_4^4 = 0. \end{cases}$$

Общее решение может быть представлено как

$$(z_1^4, z_2^4, z_3^4, z_4^4, z_5^4, z_6^4, z_1^3, z_2^3, z_3^3, z_4^3, z_5^3, z_6^3) = \overline{y} \cdot R^{4,3},$$

$$R^{4,3} = \begin{vmatrix} 0&0&0&0&0&0&1&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&1&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0&0&1&1&0 \\ 0&0&1&0&0&0&0&0&0&1&0&0 \\ 0&0&0&1&1&0&0&0&0&0&0&1 \\ 0&0&0&1&0&0&0&0&1&0&0&0 \end{vmatrix}.$$

Для вычисления базисных инвариантов подсети $Z4{,}3$ построим объединённую матрицу $G^{4,3}$ из инвариантов подсетей G^4 и G^3

$$G^{4,3} = \begin{vmatrix} 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 1&0&1&0&1&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&0&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&1&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{vmatrix}.$$

Индексация столбцов соответствует вектору

$$(x_1, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{24}, x_{25}, x_{26}, x_{27}, x_{28}, x_{29}, x_{30}).$$

Матрица базисных решений $H^{4,3} = R^{4,3} \cdot G^{4,3}$ имеет вид

$$H^{4,3} = \begin{vmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&0&1&0&0&0&1&0&1&0&1&1&0&1 \\ 1&0&1&0&1&1&0&1&0&1&0&0&0&1&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&1&1 \\ 0&1&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0&0&1&1 \\ 1&1&1&0&0&0&1&1&1&0&0&1&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&1&1&0&0&0&1&0&1&0&1&0&0&1&0 \end{vmatrix}.$$

3) Композиция: Z1,2 +Z4,3

Система уравнений для контактных позиций имеет вид

$$\begin{cases} p_1: & z_2^{4,3}+z_4^{4,3}+z_7^{4,3}+z_8^{4,3}-z_2^{1,2}-z_3^{1,2}-z_6^{1,2}-z_7^{1,2}=0, \\ p_5: & z_2^{1,2}+z_4^{1,2}+z_7^{1,2}+z_8^{1,2}-z_2^{4,3}-z_3^{4,3}-z_6^{4,3}-z_7^{4,3}=0, \\ p_{20}: & z_1^{4,3}+z_3^{4,3}+z_5^{4,3}+z_6^{4,3}-z_1^{1,2}-z_4^{1,2}-z_5^{1,2}-z_8^{1,2}=0, \\ p_{24}: & z_1^{1,2}+z_3^{1,2}+z_5^{1,2}+z_6^{1,2}-z_1^{4,3}-z_4^{4,3}-z_5^{4,3}-z_8^{4,3}=0. \end{cases}$$

Эта система имеет 48 базисных решений, представленных матрицей R:

$$R=\left|\begin{array}{cccccccccccccccccccccccccccccccccccccccccccccccc}
0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \\
0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0 \\
0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{array}\right|^T.$$

Построим объединённую матрицу G из инвариантов подсетей $G^{1,2}=H^{1,2}$ и $G^{4,3}=H^{4,3}$:

$$G=\left|\begin{array}{cccccccccccccccccccccccccccccc}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0 \\
1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0 \\
0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&1&1&0&0&1&0&0&0&0&0&0 \\
1&1&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\
1&1&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1 \\
0&0&0&0&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&1&1&0&1 \\
0&0&0&0&0&1&0&1&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&1&1 \\
0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1 \\
0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\
0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1&0
\end{array}\right|.$$

Матрица базисных решений $H^{4,3}=R^{4,3}\cdot G^{4,3}$ после удаления не минимальных решений имеет вид

$$H = \begin{vmatrix}
1&1&1&0&0&0&0&0&0&1&1&0&0&1&0&0&0&0&1&0&0&0&0&1&0&1&0&0&1&0\\
1&1&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&1&0&0&0&1&1&0&1&0&0&1&0\\
1&1&0&0&1&1&0&0&0&1&1&1&0&0&1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&0&0&0&1&1&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&0&0&1&0&1&1&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
1&1&0&0&0&1&0&1&1&0&1&1&0&0&0&0&1&0&0&0&0&0&0&1&1&1&0&0&0&0&0\\
1&1&0&0&1&1&1&1&1&1&1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&1&0&0&1&0&0&0&0&1&0&0&1&0&1&1&1&0&0&0&0&0&0&1&1\\
0&0&0&1&1&0&1&0&0&1&0&0&1&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&1&1&0&0&1&1&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&1&1&1&0&0&0&1&1\\
0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&0&0&0&1&1&1&0&0&1&0&1&1&0&1\\
0&0&0&1&1&1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&1&1&0&0&0&1&0&1&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1&0&0&1&1&1&1&1&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1\\
1&1&0&0&1&0&1&0&0&2&1&1&1&1&1&0&0&1&1&1&1&0&0&1&0&1&0&0&2&1\\
1&1&1&1&1&0&1&0&0&2&1&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0&1&0&0&2&1\\
1&1&0&0&1&2&0&1&1&0&1&1&1&1&1&1&1&1&0&0&1&1&0&0&1&2&0&1&1&0&1\\
1&1&1&1&1&2&0&1&1&0&1&0&0&0&0&1&1&0&0&1&1&1&1&1&2&0&1&1&0&1\\
1&1&1&0&1&1&0&0&0&1&1&0&0&1&1&1&1&1&1&1&1&1&0&1&1&0&0&0&1&1\\
1&1&0&1&1&1&0&0&0&1&1&1&1&0&0&1&1&1&1&1&1&1&0&1&1&1&0&0&0&1&1\\
1&1&1&0&1&1&1&1&1&1&1&1&0&0&1&1&0&0&0&0&1&1&1&0&1&1&1&1&1&1&1\\
1&1&0&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&1&1&0&1&1&1&1&1&1&1&1&1
\end{vmatrix}.$$

Так как, например, сумма всех строк содержит все натуральные компоненты, модель протокола является p-инвариантной и, следовательно, сеть является ограниченной и консервативной. Аналогичным образом [93, 131] с использованием двойственной сети можно показать, что она является также и t-инвариантной; это означает, что сеть Петри стационарно повторяемая и представляет необходимые условия её живости.

Для исследования свойств модели протокола TCP применена также стандартная методика, использующая граф достижимых маркировок сети Петри [173]. Граф достижимых маркировок, представленный на рис. 4.16 содержит 58 вершин и позволяет определить ограниченность и живость модели протокола с помощью метода полного перебора. Граф достижимых маркировок (рис. 4.16) подтверждает достоверность инвариантов, полученных с помощью декомпозиции модели. Описание маркировок графа приведено в табл. 4.7.

Используем граф для проверки свойств полученных инвариантов. Известно [169], что инвариант позиций представляет собой вектор весов, такой, что взвешенная сумма фишек остаётся постоянной в любой достижимой маркировке сети. Например, для инварианта

(2 2 1 1 2 2 1 1 1 2 2 1 1 1 1 1 1 1 2 2 1 1 2 2 1 1 1 2 2)

взвешенная сумма фишек в начальной маркировке равняется 4 и остаётся таковой в любой из достижимых маркировок, приведенных в табл. 4.7.

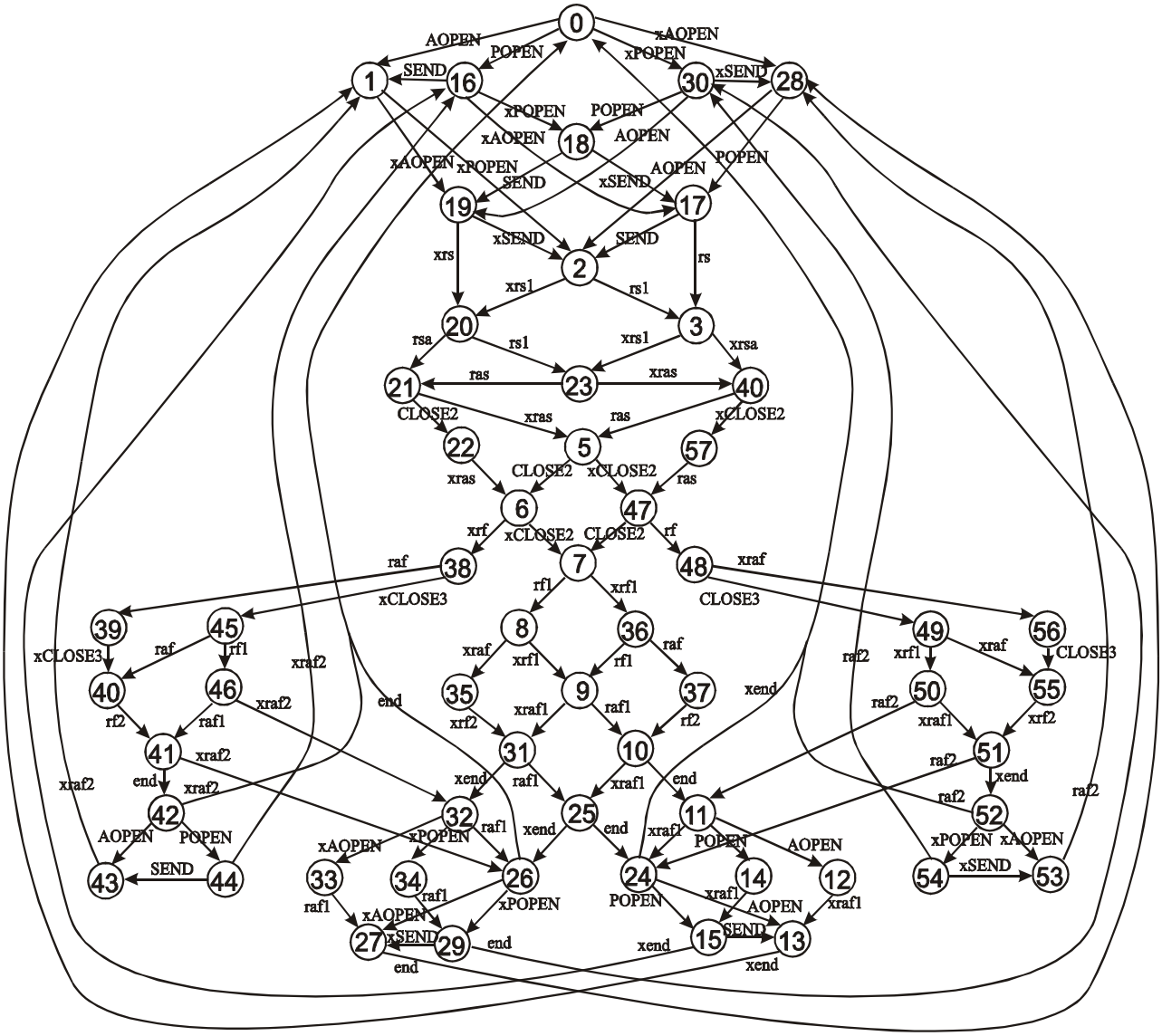


Рис. 4.16. Граф достижимых маркировок модели Петри протокола TCP

Инвариант переходов [169] представляет собой вектор счёта стационарно повторяющихся последовательностей срабатывания переходов. Наличие таких последовательностей является необходимым условием живости сети. С помощью графа достижимых маркировок (рис. 4.16) можно установить, что, например, последовательность

```
AOPEN xPOPEN xrs rsa xras CLOSE2 xrf raf xCLOSE3 rf2 xraf2 end POPEN xAOPEN
rs xrsa res xCLOSE2 rf xraf CLOSE3 xrf2 raf2 xend POPEN SEND xPOPEN SEND rs1
xrs1 ras xras CLOSE2 xCLOSE2 rf1 xrf1 raf1 xraf1 end xend
```

является стационарно повторяющейся и содержит все переходы сети Петри. Она соответствует t-инварианту

```
AOPEN POPEN*2 SEND rs rs1 rsa ras*2 xAOPEN xPOPEN*2 xSEND xrs xrs1 xrsa xras
rf CLOSE2*2 rf1 raf CLOSE3 raf1 rf2 raf2 end*2 xrf xCLOSE2*2 xrf1 xraf xCLOSE3
xraf1 xrf2 xraf2 xend*2
```

Таблица 4.7 -

Описание маркировок сети Петри

№	Маркировка	№	Маркировка
0	CLOSED xCLOSED	29	TIMEWAIT xLISTEN
1	SYN SYNSENT xCLOSED	30	CLOSED xLISTEN
2	SYN SYNSENT xSYN xSYNSENT	31	CLOSING xFINACK xTIMEWAIT
3	SYN SYNACK SYNRCVD xSYNSENT	32	CLOSING xCLOSED xFINACK
4	SYNRCVD xESTAB xSYNACK	33	CLOSING xFINACK xSYN xSYNSENT
5	ESTAB xESTAB	34	CLOSING xFINACK xLISTEN
6	FIN FINWAIT1 xESTAB	35	CLOSING FIN xFINWAIT2
7	FIN FINWAIT1 xFIN xFINWAIT1	36	FINWAIT1 xCLOSING xFIN xFINACK
8	CLOSING FIN FINACK xFINWAIT1	37	FINWAIT2 xCLOSING xFIN
9	CLOSING FINACK xCLOSING xFINACK	38	FINWAIT1 xCLOSEWAIT xFINACK
10	FINACK TIMEWAIT xCLOSING	39	FINWAIT2 xCLOSEWAIT
11	CLOSED FINACK xCLOSING	40	FINWAIT2 xFIN xLASTACK
12	FINACK SYN SYNSENT xCLOSING	41	FINACK TIMEWAIT xLASTACK
13	SYN SYNSENT xTIMEWAIT	42	CLOSED FINACK xLASTACK
14	FINACK LISTEN xCLOSING	43	FINACK SYN SYNSENT xLASTACK
15	LISTEN xTIMEWAIT	44	FINACK LISTEN xLASTACK
16	LISTEN xCLOSED	45	FINWAIT1 xFIN xFINACK xLASTACK
17	LISTEN xSYN xSYNSENT	46	CLOSING FINACK xFINACK xLASTACK
18	LISTEN xLISTEN	47	ESTAB xFIN xFINWAIT1
19	SYN SYNSENT xLISTEN	48	CLOSEWAIT FINACK xFINWAIT1
20	SYNSENT xSYN xSYNACK xSYNRCVD	49	FIN FINACK LASTACK xFINWAIT1
21	ESTAB SYNACK xSYNRCVD	50	FINACK LASTACK xCLOSING xFINACK
22	FIN FINWAIT1 SYNACK xSYNRCVD	51	LASTACK xFINACK xTIMEWAIT
23	SYNACK SYNRCVD xSYNACK xSYNRCVD	52	LASTACK xCLOSED xFINACK
24	CLOSED xTIMEWAIT	53	LASTACK xFINACK xSYN xSYNSENT
25	TIMEWAIT xTIMEWAIT	54	LASTACK xFINACK xLISTEN
26	TIMEWAIT xCLOSED	55	FIN LASTACK xFINWAIT2
27	TIMEWAIT xSYN xSYNSENT	56	CLOSEWAIT xFINWAIT2
28	CLOSED xSYN xSYNSENT	57	SYNRCVD xFIN xFINWAIT1 xSYNACK

Таким образом, выполнение свойства инвариантов, полученных в процессе последовательной композиции модели протокола TCP из функциональных подсетей, подтверждено с помощью графа достижимых маркировок.

Основным результатом настоящего подраздела является формальное доказательство корректности процедур установления соединения и разъединения протокола TCP. Для достижения этой цели построена модель протокола в форме сети Петри с уровнем детализации, представляющим флаги стандартного за-

головка сообщения. Инвариантность модели определена в процессе её последовательной композиции из функциональных подсетей, обеспечивающем существенное ускорение вычислений. Полученные результаты подтверждены также с помощью графа достижимых маркировок сети Петри.

4.4. Методы синтеза моделей Петри протоколов

Методы, представленные в разделах 2, 3 позволяют выполнить верификацию моделей большой размерности. Однако трудоёмкость верификации всё более обуславливается трудоёмкостью построения моделей, имеющей объёмы порядка тысяч элементов сети Петри.

Целью настоящего подраздела является создание эффективных методов синтеза моделей Петри с использованием промежуточного языка спецификаций, разработанного Чарльзом Хоаром – языка взаимодействующих последовательных процессов [189].

Стандартная спецификация протокола описывает порядок взаимодействия систем и форматы передаваемых сообщений (сигналов). Дополнительные требования могут включать временные параметры. Для описания структуры передаваемых сообщений в настоящее время широко используются языки, имеющие XML нотацию.

Концепция взаимодействующих последовательных процессов (ВПП) была предложена выдающимся специалистом в области теории систем, лауреатом премии Тьюринга Чарльзом Энтони Ричардом Хоаром [189] в качестве универсального языка спецификации систем. ВПП представляют собой удобное средство, позволяющее перейти от естестественноязыковых спецификаций к формализованным и содержат минимально возможное количество используемых операций. Существенным ограничением формализма ВПП является абстрагирование от временных характеристик и рассмотрение лишь последовательностей событий. Такая абстракция вполне соответствует классическим сетям Пет-

ри, также абстрагирующимся от концепции времени. Вопросы эффективности протоколов требуют детального анализа временных характеристик и применения расширенных сетей Петри [50, 72, 180]. Таким образом, основная область применения ВПП – исследование корректности процессов, либо, применительно к протоколам, их верификация.

ВПП предназначены для описания поведения *объектов*. Поведение объектов формулируется в терминах *событий*. Событие является элементарным с точки зрения исследователя моментальным действием и полностью характеризуется своим *именем*. Множество всех допустимых событий формирует *алфавит* объекта. Следует отметить, что действия, имеющие некоторую временную протяжённость можно описывать парой событий, соответствующих началу и завершению действия. Термин *процесс* обозначает поведение объекта, представленное последовательностью происходящих событий. Алфавит процесса P обозначают αP.

Последовательный процесс описывается с помощью трёх основных операций:

1) Следование (префикс).

2) Рекурсия (итерация).

3) Выбор (альтернатива).

Пусть x – событие, а P – процесс. Тогда операция следования обозначается как $(x \to P)$ и описывает объект, который вначале участвует в событии x, а затем ведёт себя как процесс P. Например, пусть $\alpha P_1 = \{sendM, receivA\}$, где событие $sendM$ представляет собой отправку сообщения, а событие $receivA$ представляет получение подтверждения. Тогда процесс $P = (sendM \to Q)$ описывает процесс P, который отправляет сообщение, а затем ведёт себя как процесс Q.

Префиксную запись можно использовать для описания процесса, который после выполнения определённой последовательности событий остановится. Для описания поведения объектов, которые повторяют определенные последовательности действий, используется рекурсия. Для обозначения рекурсии имя определяемого процесса включают в его описание. Например, рекурсивное оп-

ределение $P_1 = (sendM \to (receivA \to P_1))$ описывает процесс P_1, который отправляет сообщение, получает подтверждение, а затем ведёт себя точно так же, как процесс P_1, то есть выполняет указанную последовательность неограниченное число раз.

Выбор используют для описания альтернатив в поведении объекта. Пусть x, y – различные события; тогда формула $(x \to P | y \to Q)$ описывает объект, который сначала участвует в одном из событий x либо y, а затем ведёт себя как P, если произошло событие x либо как процесс Q, если произошло событие y. Например, дополним алфавит объекта P_1 событием $fault$, соответствующим его поломке $\alpha P_1' = \{sendM, receivA, fault\}$, тогда процесс $P_1' = (sendM \to (receiveA \to P_1 | fault \to stop))$ описывает объект, который может сломаться после получения подтверждения. Более точным является описание объекта, который может сломаться также и после отправки сообщения $P_1'' = (sendM \to (receiveA \to P_1 | fault \to stop) | fault \to stop)$.

В качестве *протокола поведения процесса* Хоар рассматривает конечную последовательность символов, фиксирующих события, в которых процесс участвует до некоторого момента времени. *Спецификацией* названо описание предполагаемого (идеального) поведения. Именно термин спецификация соответствует термину протокол, используемому в области телекоммуникаций. Стандарты телекоммуникационных протоколов представляют собой множество их спецификаций на естественном языке.

Существенным для описания телекоммуникационных систем является представление параллельных процессов в терминах взаимодействующих последовательных процессов. Для представления взаимодействия Хоаром использована операция параллельной композиции процессов: $P \| Q$ обозначает процесс, ведущий себя как система, в которой события объектов P и Q чередуются в произвольном порядке, если они имеют разные имена и требуют синхронизации, если имена событий совпадают. Таким образом, если событие входит в

алфавиты двух объектов, то для его возникновения необходимо, чтобы оно стало возможным для обоих процессов.

Дополненные операцией параллельной композиции, ранее введенные операции следования, рекурсии и выбора, представляют собой минимальное множество операций, используемых Хоаром для описания произвольных процессов. Указанное множество операций дополняется в монографии [189] рядом вспомогательных операций и функций, обеспечивающих компактность описаний сложных объектов, кроме того, разрабатываются алгебраические методы доказательства корректности процессов. Следует отметить некоторую громоздкость дополнительных конструкций; кроме того, особенностью методов доказательства корректности является их организация в виде правил вывода, не гарантирующих доказательство корректности произвольного процесса. Таким образом, использование ВПП оправдано в качестве промежуточного языка при переходе от естественноязыковых спецификаций к моделям Петри.

Рассмотрим пример построения ВПП простейшего телекоммуникационного протокола, задающего одностороннюю передачу сообщений с подтверждениями:

Передающая подсистема: $P_1 = (sendM \to (receivA \to P_1))$.

Принимающая подсистема: $P_2 = (receivM \to (sendA \to P_1))$.

Канал передачи сообщений: $P_M = (sendM \to (receivM \to P_M))$.

Канал передачи подтверждений: $P_A = (sendA \to (receivA \to P_A))$.

Телекоммуникационный протокол: $S = P_1 \| P_2 \| P_M \| P_A$.

В системе, функционирующей в соответствии с S, допустима единственная последовательность событий вида $sendM$, $receivM$, $sendA$, $receivA$,...

Модели Петри представляют собой простой и мощный формализм для представления и верификации параллельных процессов [24, 39, 50, 169, 180]. Выбор сетей Петри для верификации протоколов обусловлен широким набором известных методов их анализа [30, 39, 79, 110, 169, 180], а также компьютерных моделирующих систем [6, 10], обеспечивающих автоматизацию процессов верификации. Применение методов теории функциональных сетей Петри [93,

133, 144] позволяет выполнять анализ крупномасштабных моделей за приемлемое время. Следует отметить, что именно большая размерность детализированных моделей телекоммуникационных протоколов сдерживала широкое применение методов сетей Петри для верификации протоколов. Целью настоящего подраздела является построение методов синтеза сети Петри заданной формулой ВПП.

Заметим, что последовательный процесс может быть представлен конечным автоматом Мили, входной алфавит которого пуст, а выходной алфавит задан алфавитом событий процесса. Состояниями автомата являются символы процессов в формуле ВПП, а также промежуточные символы, соответствующие операциям следования.

Правила построения конечного автомата A_P по формуле последовательного процесса P:

1) $A_P = (X, Q, Y, q_0, F)$, где $X = \varnothing$ – входной алфавит, $Y = \alpha P$ – выходной алфавит, Q – конечное множество внутренних состояний, $q_0 = P$ – начальное состояние, $F : Q \to Y \times Q$ – функция переходов.

2) Состояния автомата. Разобьем формулу P на эквивалентное множество формул с помощью вспомогательных переменных-процессов P_i следующим образом: если в некоторой операции следования $x \to R$, правая часть R представляет собой формулу (не символ процесса), то заменим формулу R новым символом состояния автомата P_i и продолжим преобразования над формулой $P_i = R$.

3) Функция переходов. Для каждой формулы вида $P_i = y \to P_j$ добавим переход из состояния P_i в состояние P_j, с выходным сигналом y.

Например, для протокола S имеем:

$P_1 = (sendM \to P_{1,1})$, $P_{1,1} = (receivA \to P_1)$;

$P_2 = (receivM \to P_{2,1})$, $P_{2,1} = (sendA \to P_1)$;

$P_M = (sendM \to P_{M,1})$, $P_{M,1} = (receivM \to P_M)$;

$P_A = (sendA \to P_{A,1})$, $P_{A,1} = (receivA \to P_A)$.

Соответствующее множество автоматов представлено на рис. 4.17.

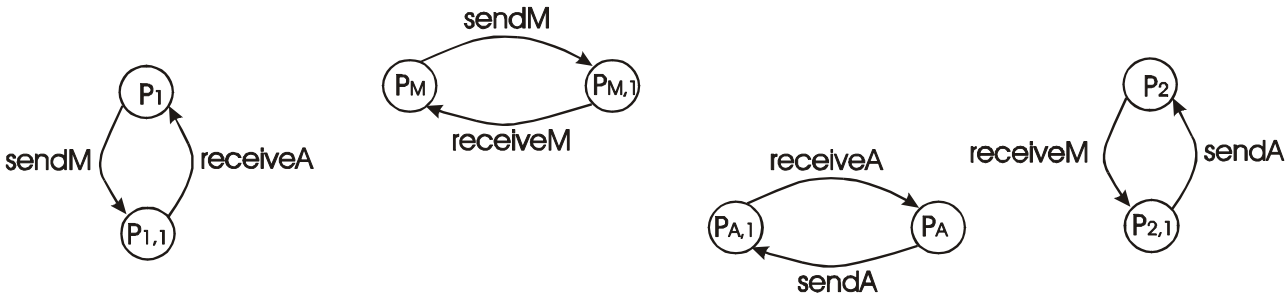


Рис. 4.17. Конечные автоматы последовательных процессов P_1, P_2, P_M, P_A

Теорема 4.1. Построенный конечный автомат A_P является представлением процесса P.

Доказательство. Применение правила 2) обеспечивает построение множества формул вида

$$P_i = x_1 \to P_{i,1} \mid x_2 \to P_{i,2} \mid ... \mid x_l \to P_{i.l}$$

эквивалентного исходной формуле процесса P. Применение правила 3) обеспечивает построение функции переходов автомата, содержащей переход $q_{P_i} \to (x_j, q_{P_{i,j}})$ для каждой формулы $P_i = x \to P_{i,j}$ и не содержащей никаких других переходов. □

После построения конечного автомата может быть выполнена его минимизация с помощью известных алгоритмов [3, 109, 122]. Следует также отметить, что аналогичным образом можно определить автомат с входным алфавитом, совпадающим с алфавитом процесса, и пустым выходным алфавитом; такой автомат можно рассматривать как распознаватель корректных последовательных процессов.

Построим помеченную автоматную сеть Петри APN_P как указано в [169], заменяя каждое состояние автомата позицией сети Петри, а каждую дугу авто-

мата, помеченную символом y последовательностью из дуги, перехода, помеченного символом y и дуги; поместим единственную фишку в позицию, соответствующую начальному состоянию. Представления процессов P_1, P_2, P_M, P_A помеченными автоматными сетями Петри изображены на рис. 4.18.

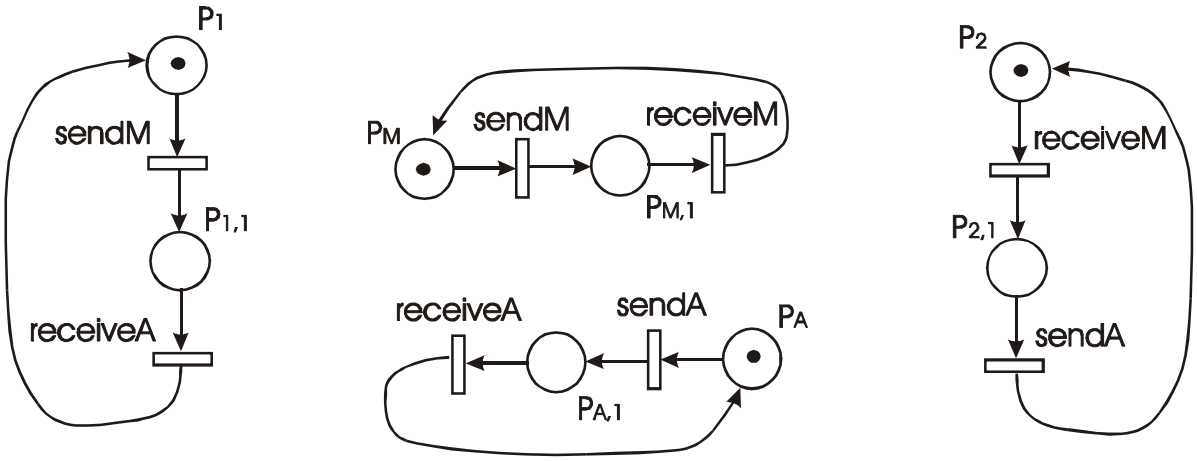


Рис. 4.18. Автоматные сети Петри процессов P_1, P_2, P_M, P_A

Утверждение 4.1. Автоматная помеченная сеть Пери APN_P является представлением процесса P.

Построим представление параллельной композиции для сетей Петри. Пусть автоматные помеченные сети Петри APN_{P_1}, APN_{P_2} являются представлениями последовательных процессов P_1 и P_2 соответственно. Построим сеть Петри APN_P следующим образом: объединим каждый из переходов сети APN_{P_1} с каждым из переходов сети APN_{P_2}, помеченные одинаковым символом события y. Пример построенной модели телекоммуникационного протокола S представлен на рис. 4.19.

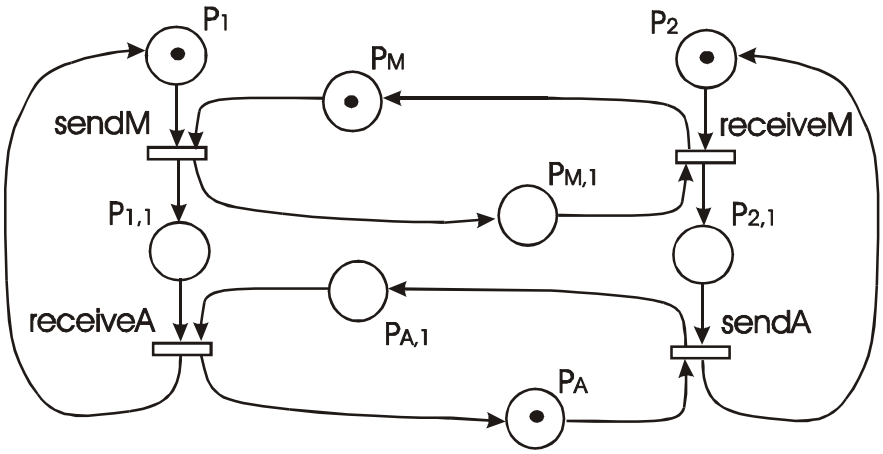


Рис. 4.19. Модель телекоммуникационного протокола S

Теорема 4.2. Сеть Петри APN_P является представлением формулы $P = P_1 \| P_2$.

Доказательство. Так как каждая из сетей APN_{P_1}, APN_{P_2} имеет фишку в своей начальной маркировке, поведение сети APN_P представляет собой произвольное чередование событий процессов P_1 и P_2 до тех пор, пока не появятся условия возбуждения перехода, соответствующего событию, принадлежащему алфавитам обоих процессов. Правила срабатывания перехода сети Петри требуют наличия фишек во входных позициях двух процессов одновременно. После срабатывания перехода фишки появятся в выходных позициях двух процессов, что обеспечивает их дальнейшее независимое выполнение. Таким образом, поведение сети APN_P соответствует определению операции параллельной композиции процессов Хоара. □

Следствие. Сети Петри с помеченными переходами эквивалентны ВПП.

Выполним синтез сети Петри заданной конечным автоматом. Следует отметить, что автоматная сеть Петри с помеченными переходами является наиболее простым, но не единственным способом представления конечного автомата [26,28,33]. В такой сети множества переходов могут иметь одинаковую пометку события y. В настоящем разделе формализована задача синтеза классической сети Пери (без помеченных переходов), эквивалентной заданному ко-

нечному автомату с помощью системы линейных уравнений и неравенств. В такой сети каждое событие y представлено единственным переходом t_y. Задача может быть сформулирована как оптимизационная с целевой функцией, минимизирующей количество элементов (позиций, дуг) сети Петри.

Пусть состояния автомата q_i, $i = \overline{0, k-1}$ закодированы маркировками сети $\bar{\mu}_i$, а переходы сети t_y соответствуют событиям $y \in Y$ и представлены парой векторов $\bar{t}_y^-$ и $\bar{t}_y^+$, задающими входящие и исходящие дуги перехода соответственно. Тогда, с одной стороны, необходимо, чтобы каждый переход t_y, соответствующий событию $y \in Y$, представленному дугой автомата $q_i \xrightarrow{y} q_j$, был разрешен в маркировке $\bar{\mu}_i$, и его срабатывание приводило к маркировке $\bar{\mu}_j$. А с другой стороны, необходимо, что все остальные переходы t_z, события которых невозможны в состоянии q_i были запрещены в маркировке $\bar{\mu}_i$. Имеем:

$$\begin{cases} \bar{\mu}_i \geq \bar{t}^-, \quad \forall t \in T_y, \\ \bar{\mu}_j - \bar{\mu}_i - \bar{t}^- + \bar{t}^+ = 0, \quad \forall t \in T_y, \\ \neg(\bar{\mu}_i \geq \bar{t}^-), \quad \forall t \notin T_y, \\ T_y = \{t \,|\, q_i \xrightarrow{y} q_j\}, \forall y \in Y. \end{cases} \tag{4.1}$$

Теорема 4.3. Сеть Петри, удовлетворяющая системе (4.1) является представлением конечного автомата A_P.

Доказательство. По построению каждый переход сети является представлением выходного символа автомата, а каждая допустимая маркировка – представлением состояния автомата. Для каждой пары состояний q_i, q_j такой, что $\exists y \in Y: \; q_i \xrightarrow{y} q_j$ первый и второй термы системы (4.1) обеспечивают, что сеть Петри в соответствии с уравнением состояний [169] перейдёт из маркировки $\bar{\mu}_i$ в маркировку $\bar{\mu}_j$ в результате срабатывания перехода t_y. Кроме того, третий терм обеспечивает, что никакой другой переход t_z не сработает в маркировке $\bar{\mu}_i$. □

Следует отметить, что система (4.1) содержит алгебраические уравнения и неравенства, а также логические операции. Необходима разработка специальных методов эффективного решения таких систем. Кроме того, размерность векторов, соответствующая количеству позиций сети заранее неизвестна. Пусть количество позиций сети Петри равно m. Тогда задача синтеза сети Петри с минимальным количеством позиций может быть представлена целевой функцией (4.2), а задача синтеза сети Петри с минимальным количеством дуг (учитывая их кратность) – целевой функцией (4.3).

$$m \to \min, \tag{4.2}$$

$$\sum_{y \in Y} \sum_{j=i.m} t^{-}_{y,j} + \sum_{y \in Y} \sum_{j=i.m} t^{+}_{y,j} \to \min. \tag{4.3}$$

В заключение следует отметить, что в настоящем подразделе построены методы синтеза конечных автоматов по формуле последовательных процессов и синтеза помеченной сети Петри по формуле взаимодействующих последовательных процессов, а также формализована задача синтеза непомеченной сети Петри заданной конечным автоматом с помощью систем линейных уравнений и неравенств.

Разработанные методы предназначены для автоматизации процессов построения моделей Петри по стандартным спецификациям телекоммуникационных протоколов. В качестве промежуточного языка спецификаций использованы взаимодействующие последовательные процессы Хоара.

4.5. Синтез модели Петри и верификация протокола электронной коммерции IOTP

Протокол электронной коммерции IOTP является наиболее сложным из более чем четырёх сотен протоколов, разработанных IETF. Его стандартная

спецификация [15] занимает около трёхсот страниц и дополняется вспомогательными документами. Важность протокола для современного мира обуславливается неуклонным ростом объемов продаж, осуществляемых с его помощью [12]. Таким образом, верификация протокола IOTP представляет собой актуальную научную задачу.

Моделирование протокола IOTP раскрашенными сетями Петри [72] не позволяет применить формальные методы за исключением простой генерации пространства состояний для его верификации. Использование классических сетей Петри было затруднено ввиду большой размерности модели и практической неосуществимости инвариантного анализа [169]. Применение композиционных методов вычисления инвариантов [93, 133, 143] обеспечивает верификацию протокола IOTP за приемлемое время.

Целью настоящего подраздела является синтез модели Петри основной транзакции протокола IOTP Приобретение с использованием промежуточного языка взаимодействующих последовательных процессов (ВПП) [189], на основе методологии, представленной в предыдущем подразделе, а также верификация протокола композиционными методами, описанными в разделах 2, 3.

Протокол IOTP предусматривает [15] обмен IOTP-сообщениями между субъектами, играющими определённую торговую роль (Trade Role). Предусмотрено пять основных торговых ролей: Клиент (Customer), Коммерсант (Merchant), Оператор платежей (Payment Handler), Оператор доставки (Delivery Handler), Обслуживание клиента (Merchant Customer Care Provider). Заметим, что несколько ролей могут выполняться одним субъектом одновременно.

Торговые обмены (Trading Exchanges) представляют собой конструктивы для формирования транзакций. Предусмотрено четыре торговых обмена: Предложение (Offer), Оплата (Payment), Доставка (Delivery), Аутентификация (Authentication). Для каждой торговой роли спецификация содержит схему взаимодействия ролей и подробные описания используемых элементов.

Транзакция представляет собой завершённое целенаправленное действие. Предусмотрены следующие транзакции: Приобретение (Purchase), Возмещение

(Refund), Обмен Значений (Value Exchange), Аутентификация (Authentication), Возврат (Withdrawal), Депозит (Deposit), Справка (Inquiry). Основной транзакцией электронной коммерции является Приобретение.

Существенной для понимания протокола является структура IOTP-сообщения, которое представляет собой многоуровневый XML документ (рис. 4.20).

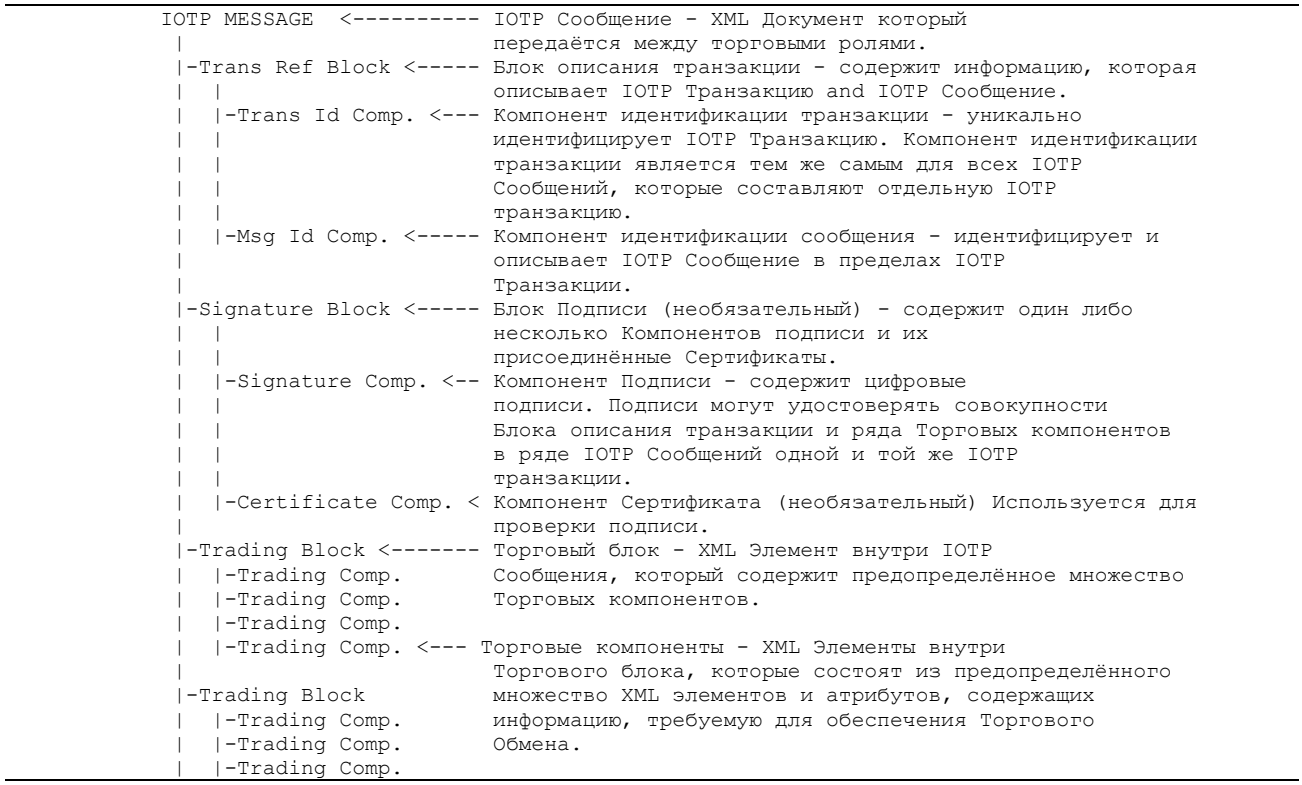


Рис. 4.20. Стандартная структура IOTP-сообщения

Сообщение представляет собой последовательность торговых блоков (Trading Block), предварённую обязательными блоками Описания транзакции (Trans Ref Block) и Блоком подписи (Signature Block). Набор торговых блоков определяется типом торгового обмена; всего предусмотрено 18 типов торговых блоков. Блоки в свою очередь состоят из Торговых компонентов (Trading Component); спецификация определяет 21 торговых компонентов. Каждый торговый компонент собирается из определённых XML элементов с указанием их атрибутов.

Перечислим торговые компоненты, используемые в транзакции Приобретение:

- Статус (Status): содержит информацию об успешном завершении либо ошибках торговых процессов;
- Организация (Organization): содержит информацию об организациях и лицах;
- Заказ (Order): описывает заказ, содержит код заказа (ссылку на базу данных);
- Список способов платежа (Brand List): принимаемые способы платежа, сумма, протоколы платежа, ссылки на Операторав Платежа;
- Выбор способа платежа (Brand Selection): содержит описание выбранного способа платежа, протокола платежа и Оператора платежа;
- Платёж (Payment): содержит информацию о том, как осуществить платёж, временные штампы и ссылку на список допустимых способов платежа;
- Схема платежа (Payment Scheme): содержит информацию для выбранного протокола платежа (например, SET) с указанием конкретных данных;
- Квитанция платежа (Payment Receipt): содержит запись о фактически оплаченной (и полученной) сумме;
- Пояснения платежа (Payment Note): дополнительная информация, описывающая выполненный платёж;
- Доставка (Delivery): содержит информацию, описывающую, куда и как следует доставить товары и сервисы (почтой, курьером, через Интернет);
- Пояснения доставки (Delivery Note): информация, необходимая для получения товаров и сервисов при фактической доставке.

Рассмотрим принципы построения модели. Модель протокола может быть построена с различным уровнем детализации. Абстрактная модель представляет фишкой целое IOTP-сообщение. Наиболее детализированная модель обеспечивает различимое представление XML элементов и их атрибутов. В настоящей работе принят следующий компромисс между размером модели и её адекватностью объекту: различимо представлены Торговые компоненты протокола, задействованные в транзакции Приобретение. Кроме того, не рассматри-

ваются необязательные компоненты и компоненты, связанные с безопасностью протокола.

В соответствии с методологией синтеза модели Петри, описанной подразделе 4.4, построим ВПП Торговых обменов, входящих в транзакцию Приобретение по схемам обменов стандартной спецификации протокола [15]. Затем построим ВПП транзакции Приобретение. Завершает построение модели синтез сети Петри по ВПП Транзакции Приобретение.

Построим ВПП торговых обменов и транзакции. Следующие ВПП построены непосредственно по стандартным схемам Торговых Обменов, представленных на страницах 18-26 в [15]. Заметим, что вложенные скобки в ВПП опущены в предположении правой ассоциативности операции следования $\rightarrow$ таким образом, что $A \rightarrow B \rightarrow C = (A \rightarrow (B \rightarrow C))$.

Предложение:

$ConsumerOffer = MakeChoice \rightarrow PutOffer\,\mathrm{Re}quest \rightarrow GetOffer\,\mathrm{Re}sponse \rightarrow CheckOffer$.

$MercantOffer = GetOffer\,\mathrm{Re}quest \rightarrow Check\,\mathrm{Re}quest \rightarrow PutOffer\,\mathrm{Re}sponse$.

$Offer = MerchantOffer \,||\, ConsumerOffer$.

$Offer\,\mathrm{Re}sponce = Status \,||\, Organization \,||\, Order \,||\, Payment \,||\, Delivery$.

Оплата:

$ConsumerPayment = TradeDecision \rightarrow PutPaidFor \rightarrow GetBrandList \rightarrow SelectBrand \rightarrow PutBrandSelection \rightarrow GetPaymentAmount \rightarrow CheckPaymentAmount \rightarrow PutPayment\,\mathrm{Re}quest \rightarrow PaymentExchange \rightarrow Payment\,\mathrm{Re}sponse \rightarrow CheckPayment\,\mathrm{Re}seipt$

$MerchantPayment = GetPaidFor \rightarrow BrandDecision \rightarrow PutBrandList \rightarrow GetBrandSelection \rightarrow CheckBrandSelection \rightarrow PutPaymentAmount$

$PaymentHandlerPayment = GetPayment\,\mathrm{Re}quest \rightarrow CheckPayment\,\mathrm{Re}quest \rightarrow PaymentExchange \rightarrow PaymentPesponse$

$Payment = ConsumerPayment \,||\, MerchantPayment \,||\, PaymentHandlerPayment$

$PaymentAmount = Payment \,||\, Organization$

$Payment\,\mathrm{Re}quest = Status \,||\, Payment \,||\, Organization \,||\, PaySchemeData$

$PaymentExchange = PaySchemeData$

$Payment\,\mathrm{Re}sponse = Status \,||\, Pay\,\mathrm{Re}ceipt \,||\, PaymentNote$

Доставка:

$$ConsumerDelivery = WhatDeliverDecision \rightarrow PutWhatDeliver \rightarrow GetHowDeliver \rightarrow CheckDeliveryInfo \rightarrow PutDelivery\,\mathrm{Re}quest \rightarrow GetDelivery\,\mathrm{Re}sponse \rightarrow CheckDeliveryNote$$

$$MerchantDelivery = GetWhatDeliver \rightarrow CheckWhatDeliver \rightarrow PutHowDeliver$$

$$DeliveryHandlerDelivery = GetDelivery\,\mathrm{Re}quest \rightarrow CheckDelivery\,\mathrm{Re}quest \rightarrow PutDelivery\,\mathrm{Re}sponse$$

$$Delivery = ConsumerDelivery \,||\, MerchantDelivery \,||\, DeliveryHandlerDelivery$$

$$HowDeliver = Delivery \,||\, Organization \,||\, Order$$

$$Delivery\,\mathrm{Re}quest = Status \,||\, Delivery \,||\, Organization \,||\, Order$$

$$Delivery\,\mathrm{Re}sponse = Status \,||\, DeliveryNote$$

Приобретение:

$$TransPerchase = Offer \rightarrow Payment \rightarrow Deliver \rightarrow TransPurchase\ .$$

Имена переменных в формулах выбраны в соответствии с ранее описанными названиями торговых обменов, торговых ролей и компонентов. Далее приведен тезаурус имён дополнительных переменных:

- Request/Response: запрос/ответ;
- Put/Get/Check: послать/получить/проверить:
- MakeChoice: сделать выбор предложения с помощью HTTP;
- TradeDecision: принять решение о приобретаемых товарах/услугах;
- PaidFor: выбрать товары/услуги, за которые выполняется платёж с помощью HTTP;
- SelectBrand: выбрать способ платежа из списка;
- PaymentAmmount: сформировать сумму платежа;
- PaymentExchange: платёж в соответствии с выбранным способом (протоколом);
- WhatDeliverDecision: принять решение о доставляемых товарах/услугах;
- WhatDeliver: список доставляемых товаров/услуг;
- HowDeliver: описание способа доставки товаров/услуг;
- DeliveryInfo: информация о принятом к исполнению способе доставки.

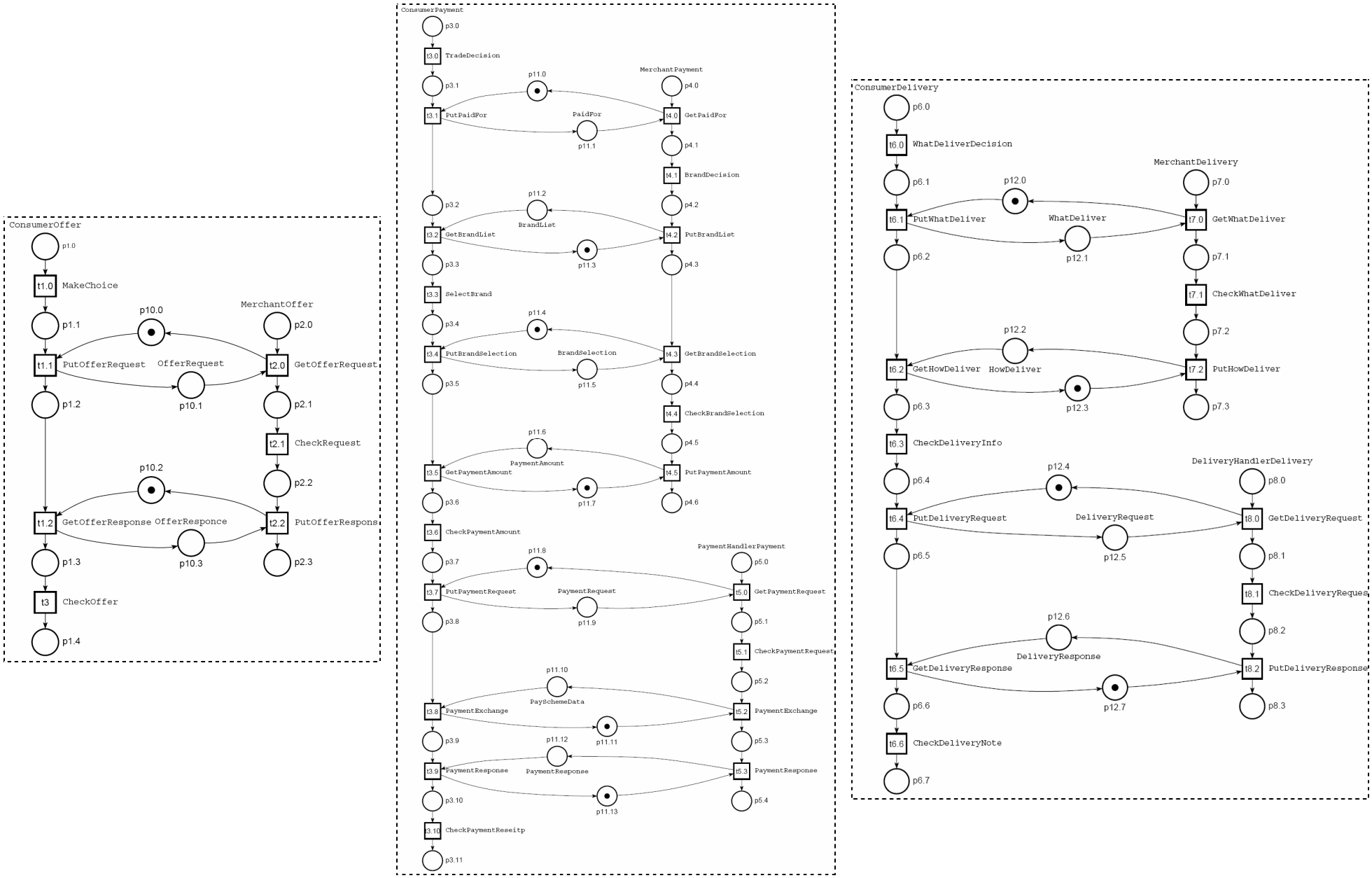


Рис. 4.21. Модели торговых обменов

Выполним синтез модели Петри. Синтез модели предполагает построение отдельной сети Петри по каждой формуле ВПП и их композицию в соответствии с правилами, представленными в подразделе 4.4. Модели отдельных формул являются тривиальными; их рисунки опущены. Модели торговых обменов без компонентов IOTP-сообщения изображены на рис. 4.21. Модель транзакции Приобретение с используемыми компонентами IOTP-сообщения представлена на рис. 4.22; она содержит 111 позиций и 45 переходов сети Петри. Обозначения элементов сети выбраны уникальные; пометки элементов соответствуют наименованиям переменных формул ВПП.

Выполним верификацию протокола. Для верификации протокола применено программное обеспечение Deborah и Adriana, реализующее вычисление инвариантов позиций и переходов сети Петри в процессе последовательной композиции её функциональных подсетей (Приложение Д). Инвариантный анализ классическими методами не был завершен за трое суток вычислений на компьютере Pentium 3,2GHz; композиционное вычисление инвариантов заняло около двух часов.

Граф функциональных подсетей [93, 143] модели транзакции Приобретение (рис. 4.22) изображён на рис. 4.23. Вершины графа соответствуют минимальным функциональным подсетям, порождённым подмножествами переходов сети Петри; табл. 4.8 содержит индикаторы переходов сети Петри, указывающие номера подсетей, которым они принадлежат.

Таблица 4.8 -

Индикаторы (номеров) подсетей переходов

T	1.0	1.1	1.2	2.0	2.1	2.2	3	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9
Z	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	3.10	4.0	4.1	4.2	4.3	4.4	4.5	5.0	5.1	5.2	5.3	3.0	3.1	6.0	6.1
Z	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
T	6.2	6.3	6.4	6.5	6.6	7.0	7.1	7.2	8.0	8.1	8.2	5.4	8.4	7.4	6.7
Z	31	32	33	34	35	36	37	38	39	39	39	40	41	42	43

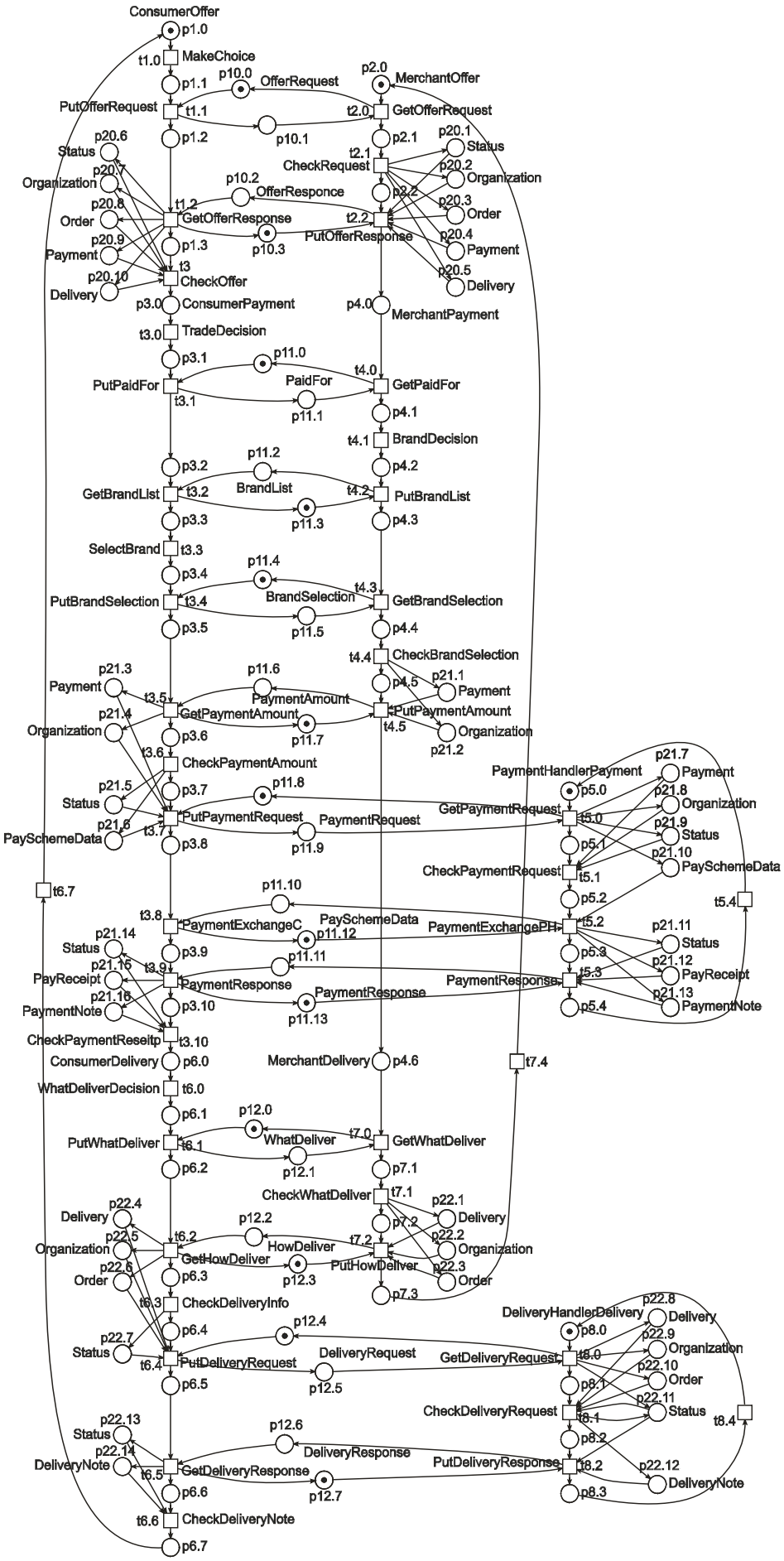


Рис. 4.22. Модель транзакции Приобретение

Инвариант позиций, вычисленный программой Adriana в процессе последовательной композиции функциональных подсетей и содержащий всё ненулевые компоненты, представлен в табл. 4.9 (взвешенная сумма фишек равна 35). Таким образом, модель ограничена и консервативна.

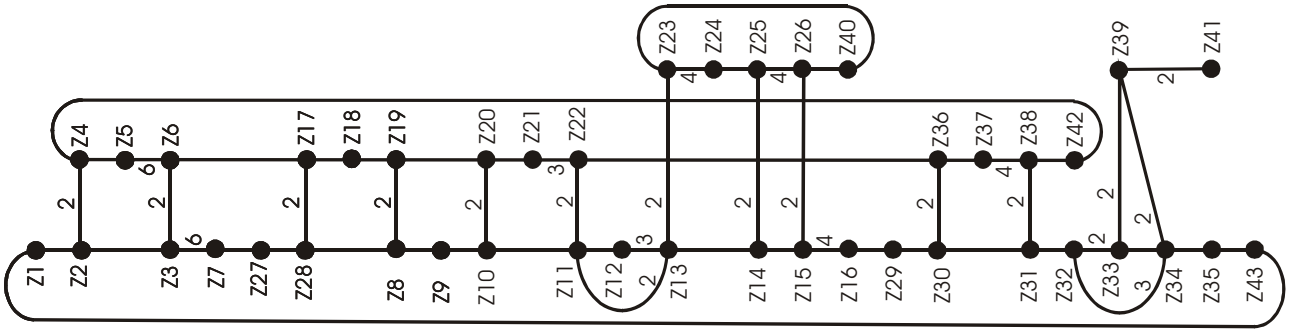


Рис. 4.23. Граф функциональных подсетей

Инвариант переходов, вычисленный программой Adriana в процессе последовательной композиции функциональных подсетей двойственной [169] сети Петри, содержит всё ненулевые компоненты, равные единице. Таким образом, модель имеет стационарно повторяющиеся последовательности срабатываний, содержащие все переходы, что является необходимым условием её живости.

Таблица 4.9 -

Инвариант позиций

P	p1.0	p1.1	p1.2	p2.0	p2.1	p2.2	p1.3	p3.0	p3.1	p3.2	p3.3
X(P)	6	6	6	6	6	1	1	6	6	6	6
P	p3.4	p3.5	p3.6	p3.7	p3.8	p3.9	p3.10	p4.0	p4.1	p4.2	p4.3
X(P)	6	6	4	2	6	6	3	6	6	6	6
P	p4.4	p4.5	p4.6	p5.0	p5.1	p5.2	p5.3	p5.4	p6.0	p6.1	p6.2
X(P)	6	4	6	5	1	4	2	5	6	6	6
P	p6.3	p6.4	p6.5	p6.6	p6.7	p7.1	p7.2	p7.3	p8.0	p8.1	p8.2
X(P)	3	2	6	4	6	6	1	1	5	1	3
P	p8.3	p10.0	p10.1	p10.2	p10.3	p11.0	p11.1	p11.2	p11.3	p11.4	p11.5
X(P)	5	1	1	1	1	1	1	1	1	1	1
P	p11.6	p11.7	p11.8	p11.9	p11.10	p11.11	p11.12	p11.13	p12.0	p12.1	p12.2
X(P)	1	1	1	1	1	1	1	1	1	1	1
P	p12.3	p12.4	p12.5	p12.6	p12.7	p20.1	p20.2	p20.3	p20.4	p20.5	p20.6
X(P)	1	1	1	1	1	1	1	1	1	1	1
P	p20.7	p20.8	p20.9	P20.10	p21.1	p21.2	p21.3	p21.4	p21.5	p21.6	p21.7
X(P)	1	1	1	1	1	1	1	1	1	1	1
P	p21.8	p21.9	p21.10	P21.11	p21.12	p21.13	p21.14	p21.15	p21.16	p22.1	p22.2
X(P)	1	1	1	1	1	1	1	1	1	1	1
P	p22.3	p22.4	p22.5	p22.6	p22.7	p22.8	p22.9	p22.10	p22.11	p22.12	p22.13
X(P)	1	1	1	1	1	1	1	1	1	1	1

Дополнительно верификация протокола выполнена с помощью графа достижимых маркировок [173], что подтверждает ограниченность и живость модели.

Таким образом, в настоящем подразделе выполнен синтез модели Петри и её верификация для транзакции Приобретение протокола электронной коммерции IOTP. Применение композиции функциональных подсетей позволило осуществить верификацию за приемлемое время. Аналогично может быть реализована верификация других транзакций протокола, а также построение и верификация моделей с уровнем детализации до XML элементов и их атрибутов, учитывающих средства обеспечения безопасности протокола.

4.6. Верификация протоколов с неограниченным числом взаимодействующих устройств

Ранее представленные методы и алгоритмы композиционного анализа сетей используют произвольные функциональные подсети (кланы), полученные в результате декомпозиции. В случаях, когда модель представляет собой регулярную структуру, являющуюся повторением базовых компонентов появляется возможность вычисления инвариантов в параметрической форме для систем произвольного размера.

Метод композиционного вычисления инвариантов в параметрической форме можно представить следующим образом:

1. Вычислить инварианты базовых компонентов.

2. Построить бесконечную систему композиции базовых компонентов.

3. Решить систему композиции в параметрической форме.

4. Получить представление инвариантов исходной модели в параметрической форме для произвольной размерности системы.

Рассмотрим особенности применения метода композиционного вычисления инвариантов в параметрической форме на примере верификации протоколов CSMA/CD.

4.6.1. Протоколы Ethernet с архитектурой общей шины

В сети Ethernet применяются протоколы множественного доступа к среде с прослушиванием несущей и определением коллизий CSMA/CD (Carrier Sense Multiply Access with Collision Detection). Исходные спецификации протоколов регламентируются стандартами IEEE 802.x.

Каждая из рабочих станций прослушивает среду передачи и, в случае отсутствия несущей, может начать передачу данных. Если несколько рабочих станций начнут передачу данных одновременно, то произойдёт наложение информации, именуемое коллизией. Рабочая станция располагает средствами обнаружения коллизий. В случае возникновения коллизии передача данных прекращается и возобновляется через некоторый случайный интервал времени.

Спецификациями протоколов регламентируются также форматы передаваемых пакетов (фреймов) данных. В настоящей работе структура пакета данных Ethernet не используется, кроме того, исследуется сеть с архитектурой общей шины. Рассматриваются начало и завершение передачи пакета, процессы распространения несущей по передающей среде во времени, а также распознавание и обработка коллизий. Появление и прекращение несущей распространяется по общей шине в течение определённого времени, поэтому рабочие станции в некоторый фиксированный момент времени располагают различной информацией о состоянии канала.

Следует отметить, что, несмотря на ограниченное использование в настоящее время Ethernet с архитектурой общей шины, технология CSMA/CD является перспективной, например, для беспроводных сетей, таких, как радио-Ethernet и другие.

4.6.2. Модель Ethernet с архитектурой общей шины

Исследуем модель Ethernet с архитектурой общей шины и произвольным количеством подключенных рабочих станций, которая представляет собой несколько видоизменённую модель, описанную в работе Маршана [66]. В указанной работе отмечено, что построенная модель является достаточно подробным и точным представлением исходных спецификаций. Но, ввиду большой размерности даже для небольшого количества рабочих станций исследование свойств модели является слишком трудоёмким. Поэтому предложено исследовать упрощённую модель.

Применение методов декомпозиции сетей Петри, позволило выполнить верификацию подробной модели, включающей произвольное количество рабочих станций. Исходная модель Маршана модифицирована таким образом, что модель отдельной рабочей станции представляет собой функциональную сеть Петри. Для этого добавлены контактные позиции. Компоновка модели локальной сети выполняется путём совмещения контактных позиций рабочих станций. Таким образом, имеется естественная декомпозиция общей модели на функциональные подсети.

Модель представлена в виде временной сети Петри [25] с ингибиторными дугами. Заметим, что использованы переходы трёх типов: срабатывающие мгновенно, изображаемые узкими прямоугольниками; с детерминированными временами срабатывания, изображаемые закрашенными прямоугольниками; со случайными равномерно распределёнными временами срабатывания, изображаемые широкими не закрашенными прямоугольниками. Ингибиторные дуги применяют для проверки маркировки позиции на ноль. Вместо стрелки такие дуги имеют окружность небольшого диаметра. Контактные позиции изображены линиями, имеющими двойную толщину. В дальнейшем используются как краткие смысловые наименования, так и номера элементов сетей Петри, изображённых на рисунках.

Опишем подробно модель рабочей станции, представленную на рис. 4.24. Для удобства описания модель можно разбить на две части с помощью пунк-

тирных линий. Верхняя часть моделирует поведение рабочей станции. Нижняя часть описывает взаимодействие рабочей станции с каналом и процесс передачи данных между соседними рабочими станциями.

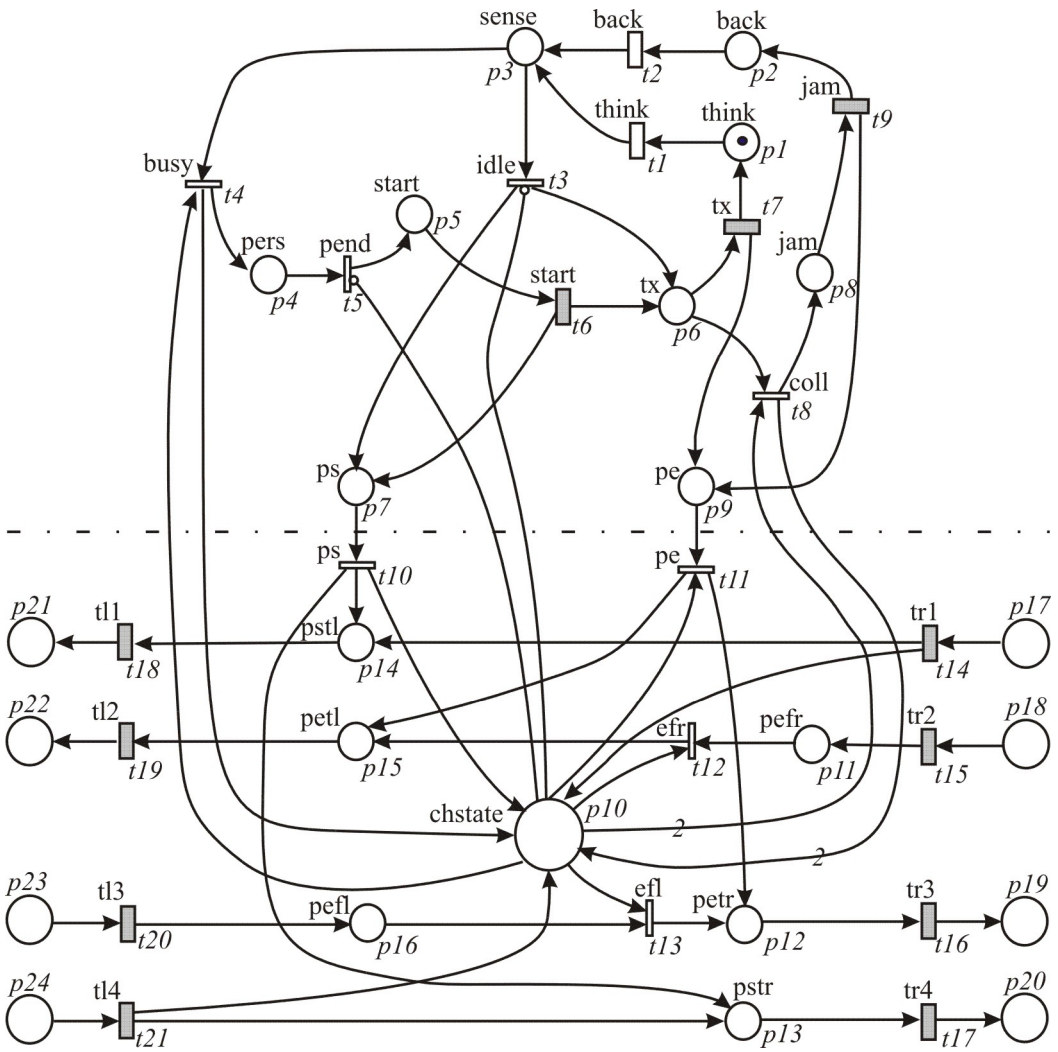


Рис. 4.24. Модель рабочей станции Ethernet

Позиция и переход *think* моделирует внутреннюю активность рабочей станции, не связанную с информационным обменом. В позиции *sense* выполняется прослушивание среды. Если в результате прослушивания установлено, что канал свободен, переход *idle* становится разрешённым. При срабатывании он помещает фишку в позицию *tx* (transmit), указывающую, что станция начинает передачу данных, а также фишку в позицию *ps* (propagation start), моделирующую начало распространения по сети несущей. Наличие фишки в позиции *tx*

возбуждает переход *tx*, моделирующий передачу данных в сети. Когда завершается срабатывание перехода *tx*, станция заканчивает передачу данных и возвращается в исходное состояние *think*. Кроме того, фишка помещается в позицию *pe* (propagation end), моделирующую распространение по сети прекращения несущей.

Кроме того, если в любой момент времени при передаче пакета позиция *chstate* (channel state) содержит более двух фишек, срабатывает мгновенный переход *coll* (collision), моделирующий распознавание коллизии в канале. Перемещение фишки с позицию *jam* и срабатывание перехода *jam* моделируют действия по прекращению передачи данных при установлении коллизии. Перемещение фишки в позицию *back* инициирует процесс повторной передачи данных после распознавания коллизии. Случайное время срабатывания перехода *back* моделирует задержку перед повторной передачей.

Если при прослушивании среды в позиции *sense* установлено, что канал занят, то срабатывает переход *busy*, помещая фишку в позицию *pers* (persist). Как только канал освобождается, возбуждается и срабатывает мгновенный переход *pend* (pending). Он перемещает фишку в позицию *start* и после детерминированной временной задержки, представленной переходом *start*, начинается передача данных.

В нижней части сети основным элементом является позиция *chstate*, представляющая состояние канала, распознаваемое рабочей станцией. Поскольку, появление и прекращение несущей представляют собой процессы, время реализации которых учитывается в модели, каждая из рабочих станций может распознавать различное состояние канала. Если позиция *chstate* пуста – канал свободен, наличие ровно одной фишки свидетельствует о занятости канала, если же позиция содержит, по крайней мере, две фишки, то имеет место коллизия.

Переходы *tl1, tl2, tl3, tl4* моделируют задержки распространения (прекращения) несущей для левой соседней рабочей станции. Аналогичную роль играют переходы *tr1, tr2, tr3, tr4* для правой соседней станции. Заметим, что, так

как предполагается компоновать общую модель путём совмещения позиций, времена срабатывания перечисленных переходов должны быть выбраны равными половине фактической задержки.

Когда станция начинает передачу данных и помещает фишку в позицию *ps*, срабатывает мгновенный переход *ps* и помещает по фишке в позиции *chastate, pstl, pstr*. Наличие фишек в позициях *pstl, pstr* моделирует распространение сигнала влево и вправо по общей шине.

Когда станция прекращает передачу данных и помещает фишку в позицию *pe*, срабатывает мгновенный переход *pe*, забирает одну фишку из позиции *chstate* и помещает по одной фишке в позиции *petl, petr*. Наличие фишек в позициях *petl* и *petr* моделирует прекращение сигнала в общей шине.

Сигнал начала передачи данных по общей шине может также поступать от соседней слева станции в позицию *pstr* и от соседней справа станции в позицию *pstl*. Прекращение передачи данных по общей шине моделируется поступлением фишки от соседней слева рабочей станции в позицию *pefl* и от соседней справа рабочей станции в позицию *pefr*.

Заметим, что при поступлении сигнала начала передачи данных слева либо справа *tl4, tr1* в позицию *chstate* поступает фишка, моделируя занятость канала, известную рабочей станции. Прекращение передачи данных рабочей станцией, расположенной слева либо справа на общей шине, приводит к срабатыванию переходов *efl, efr* соответственно. Таким образом моделируется освобождение канала.

Итак, построенная сеть имеет входные позиции $X = \{p_{17}, p_{18}, p_{23}, p_{24}\}$ и выходные позиции $Y = \{p_{19}, p_{20}, p_{21}, p_{22}\}$, и, в соответствии с определением [93, 144], является функциональной сетью Петри.

Рассмотрим правила построения модели Ethernet с архитектурой общей шины, насчитывающей k рабочих станций. Общая схема такой модели представлена на рис. 4.25.

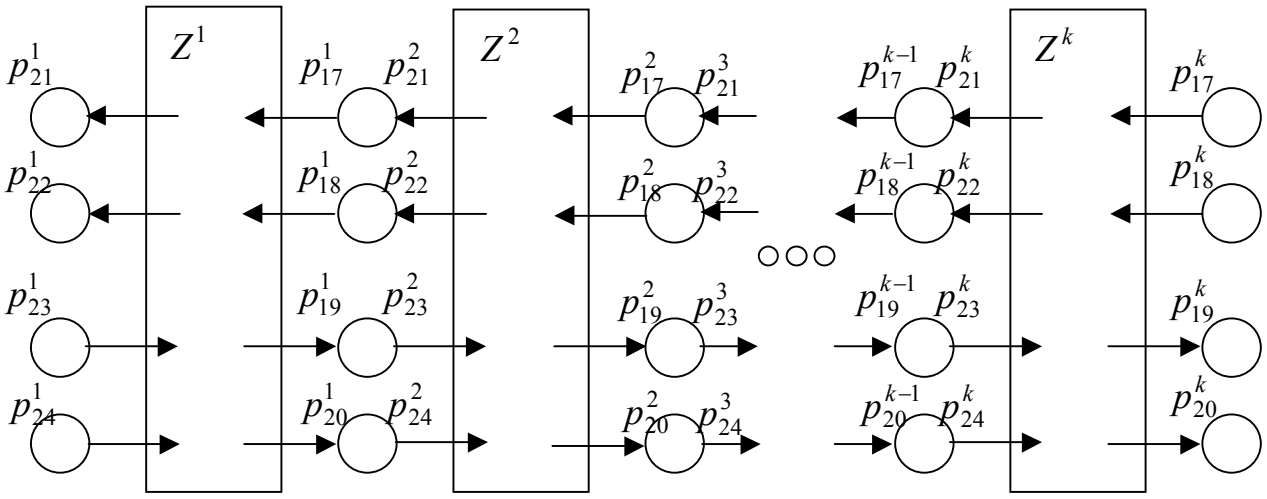


Рис. 4.25. Модель сети Ethernet с общей шиной

Модели рабочих станций Z^i объединяются путём совмещения контактных позиций. Причём каждая из станций взаимодействует ровно с двумя соседними станциями. Рассмотрим, например, подсеть Z^2. Контактные позиции $p_{21}, p_{22}, p_{23}, p_{24}$ подсети Z^2 объединяются с контактными позициями $p_{17}, p_{18}, p_{19}, p_{20}$ соответственно подсети Z^1. Кроме того, контактные позиции $p_{17}, p_{18}, p_{19}, p_{20}$ подсети Z^2 объединяются с контактными позициями $p_{21}, p_{22}, p_{23}, p_{24}$ соответственно подсети Z^3. Принадлежность элементов к той либо иной подсети будем отображать с помощью верхнего индекса, равного номеру подсети.

Построенная модель имеет естественную декомпозицию на функциональные подсети $\{Z^i\}$. Этот факт будет использован далее при анализе свойств модели.

4.6.3. Вычисление инвариантов в параметрической форме

Как отмечено в работах [9, 29, 39], идеальная модель корректного телекоммуникационного протокола должна обладать такими свойствами, как ограниченность, консервативность, живость. Для исследования перечисленных свойств применяют сетевые инварианты [169]. В этом случае модель корректного протокола должна быть инвариантной.

Заметим, что в построенной модели Ethernet все функциональные подсети имеют одинаковую структуру и различаются только верхним индексом их элементов. При вычислении инвариантов подсети Z^i ингибиторные дуги и циклы не рассматривались, так как они не изменяют маркировку сети. Кроме того, дуги (t_3, p_6), (t_6, p_6), (t_{10}, p_{13}), (t_{10}, p_{14}), (t_{11}, p_{12}), (t_{11}, p_{15}) были опущены и дуга (t_{11}, p_6) добавлена, так как исходная модель не инвариантна. Эти преобразования матрицы соответствуют запуску повторной передачи фрейма только после того, как сигнал окончания несущей поступил в шину. Таким образом, была использована следующая матрица инцидентности:

$$A = \begin{pmatrix}
-1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1
\end{pmatrix}$$

Применение системы Tina [10] даёт матрицу базисных инвариантов позиций, представленную в табл. 4.10. Строка l представленной матрицы является базисным инвариантом, который при построении общего решения домножается на неотрицательный коэффициент z_l. Столбцы матрицы соответствуют позициям сети Z^i.

Таблица 4.10 -

Базисные инварианты модели рабочей станции

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	1	1	1	1	1	1	1	1		1		1			1		1		1			1		1
2												1				1			1				1	
3											1				1			1				1		
4						1		1	1															
5														1			1				1			
6													1							1				1

В соответствии с теоремой 2.3 после вычисления инвариантов функциональных подсетей следует найти общие инварианты контактных позиций. Так как в композиции двух соседних подсетей Z^i и Z^{i+1} участвуют четыре контактных позиций $\{p^i_{17}, p^i_{18}, p^i_{19}, p^i_{20}\}$ левой подсети и четыре контактных позиций правой подсети, то система уравнений для контактных позиций имеет следующий вид:

$$\begin{cases} z^i_1 + z^i_5 = z^{i+1}_5, \\ z^i_3 = z^{i+1}_1 + z^{i+1}_3, \\ z^i_1 + z^i_2 = z^{i+1}_2, \\ z^i_6 = z^{i+1}_1 + z^{i+1}_6, \quad i = 1, k-1. \end{cases} \tag{4.4}$$

Следует отметить, что система (4.4) содержит бесконечное число уравнений. Для её решения была сгенерирована последовательность систем для $k = 2,3,...$:

$$k = 2: \begin{cases} z^1_1 + z^1_5 = z^2_5, \\ z^1_3 = z^2_1 + z^2_3, \\ z^1_1 + z^1_2 = z^2_2, \\ z^1_6 = z^2_1 + z^2_6. \end{cases}$$

$$k = 3: \begin{cases} z^1_1 + z^1_5 = z^2_5, \\ z^1_3 = z^2_1 + z^2_3, \\ z^1_1 + z^1_2 = z^2_2, \\ z^1_6 = z^2_1 + z^2_6, \\ z^2_1 + z^2_5 = z^3_5, \\ z^2_3 = z^3_1 + z^3_3, \\ z^2_1 + z^2_2 = z^3_2, \\ z^2_6 = z^3_1 + z^3_6. \end{cases}$$

…

Затем эти системы были решены с помощью системы Tina [10] (табл. 4.11):

Таблица 4.11 -

Базисные решения для различных значений параметра

k	Базисные решения
2	(z_4^1), (z_4^2), (z_2^1, z_2^2), (z_3^1, z_3^2), (z_5^1, z_5^2), (z_6^1, z_6^2), (z_3^1, z_6^1, z_1^2), (z_1^1, z_2^2, z_5^2).
3	(z_4^1), (z_4^2), (z_4^3), (z_2^1, z_2^2, z_2^3), (z_3^1, z_3^2, z_3^3), (z_5^1, z_5^2, z_5^3), (z_6^1, z_6^2, z_6^3), $(z_3^1, z_6^1, z_1^2, z_2^3, z_5^3)$, $(z_1^1, z_2^2, z_5^2, z_2^3, z_5^3)$, $(z_3^1, z_6^1, z_3^2, z_6^2, z_1^3)$.
4	(z_4^1), (z_4^2), (z_4^3), (z_4^4), $(z_2^1, z_2^2, z_2^3, z_2^4)$, $(z_3^1, z_3^2, z_3^3, z_3^4)$, $(z_5^1, z_5^2, z_5^3, z_5^4)$, $(z_6^1, z_6^2, z_6^3, z_6^4)$, $(z_3^1, z_6^1, z_1^2, z_2^3, z_5^3, z_2^4, z_5^4)$, $(z_3^1, z_6^1, z_3^2, z_6^2, z_1^3, z_2^4, z_5^4)$, $(z_3^1, z_6^1, z_3^2, z_6^2, z_3^3, z_6^3, z_1^4)$, $(z_1^1, z_2^2, z_5^2, z_2^3, z_5^3, z_2^4, z_5^4)$.
5	(z_4^1), (z_4^2), (z_4^3), (z_4^4), (z_4^5), $(z_2^1, z_2^2, z_2^3, z_2^4, z_2^5)$, $(z_3^1, z_3^2, z_3^3, z_3^4, z_3^5)$, $(z_5^1, z_5^2, z_5^3, z_5^4, z_5^5)$, $(z_6^1, z_6^2, z_6^3, z_6^4, z_6^5)$, $(z_3^1, z_6^1, z_1^2, z_2^3, z_5^3, z_2^4, z_5^4, z_2^5, z_5^5)$, $(z_3^1, z_6^1, z_3^2, z_6^2, z_1^3, z_2^4, z_5^4, z_2^5, z_5^5)$, $(z_3^1, z_6^1, z_3^2, z_6^2, z_3^3, z_6^3, z_1^4, z_2^5, z_5^5)$, $(z_3^1, z_6^1, z_3^2, z_6^2, z_3^3, z_6^3, z_3^4, z_6^4, z_1^5)$, $(z_1^1, z_2^2, z_5^2, z_2^3, z_5^3, z_2^4, z_5^4, z_2^5, z_5^5)$.

Применение индуктивных рассуждений позволяет получить параметрическую форму представления решений. Система (4.4) имеет следующие $2 \cdot k + 4$ базисных решения:

$$\begin{pmatrix} (z_4^i), \quad i = 1, k; \\ (z_2^j, \quad j = 1, k); \\ (z_3^j, \quad j = 1, k); \\ (z_5^j, \quad j = 1, k); \\ (z_6^j, \quad j = 1, k); \\ \begin{pmatrix} ((z_3^j, z_6^j), \quad j = 1, i-1), \quad (z_1^i), \\ ((z_2^j, z_5^j), \quad j = i+1, k) \end{pmatrix}, \quad i = 1, k \end{pmatrix}. \qquad (4.5)$$

Заметим, что первое и шестое решения (4.5) представляют k конкретных решений системы; кроме того, решения представлены путём указания ненулевых (равных единице) элементов.

Подставим базисные инварианты, соответствующие свободным переменным z_l в параметрические решения (4.5) системы композиции для контактных

позиций (4.4). Получим следующие параметрические решения исходной системы вида (3.2):

$$\left(\begin{array}{l}
(p_6^i, p_8^i, p_9^i), \quad i=1,k; \\
((p_{12}^j, p_{16}^j, p_{19}^j, p_{23}^{j=1}), \quad j=1,k); \\
((p_{11}^j, p_{15}^j, p_{18}^j, p_{22}^{j=1}), \quad j=1,k); \\
((p_{14}^j, p_{17}^j, p_{21}^{j=1}), \quad j=1,k); \\
((p_{13}^j, p_{20}^j, p_{24}^{j=1}), \quad j=1,k); \\
\left(\begin{array}{l}
((p_{11}^j, p_{15}^j, p_{18}^j, p_{20}^j, p_{22}^{j=1}, p_{24}^{j=1}), \\
j=1,i-1), \\
(p_1^i, p_2^i, p_3^i, p_4^i, p_5^i, p_6^i, p_7^i, p_8^i, \\
p_{10}^i, p_{12}^i, p_{15}^i, p_{17}^i, p_{19}^i, p_{22}^{i=1}, p_{24}^{i=1}), \\
((p_{12}^j, p_{14}^j, p_{16}^j, p_{17}^j, p_{19}^j, \\
p_{21}^{j=1}, p_{22}^{j=1}, p_{24}^{j=1}), j=i+1,k)
\end{array}\right), i=1,k
\end{array}\right) \qquad (4.6)$$

В представлении решений указаны условные элементы, которые присутствуют только для первой подсети, так как только она содержит позиции $p_{21}, p_{22}, p_{23}, p_{24}$; при слиянии позиций для других рабочих станций использованы их меньшие номера для левой рабочей станции $p_{17}, p_{18}, p_{19}, p_{20}$ соответственно.

Так как, например, при выборе суммы всех базисных решений получаем инвариант со всеми натуральными компонентами, то модель Ethernet с топологией общей шины и произвольным количеством *k* рабочих станций является p-инвариантной сетью Петри.

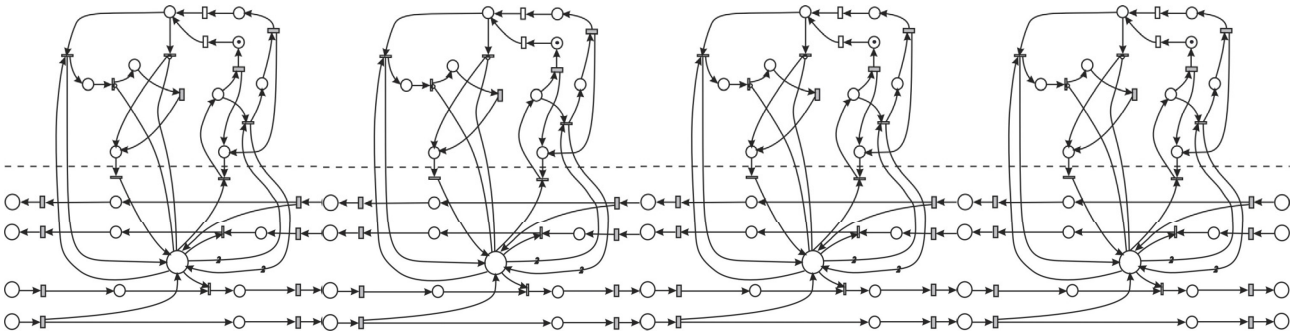


Рис. 4.26. Модель шины с четырьмя рабочими станциями

Для верификации полученного параметрического решения построены модели для различного числа рабочих станций на шине и с помощью системы Tina [10] найдены их базисные решения. Результаты совпадают. Например, для модели с четырьмя рабочими станциями (рис. 4.26) получены следующие инварианты:

$$(p_6^1, p_8^1, p_9^1);$$

$$(p_6^2, p_8^2, p_9^2);$$

$$(p_6^3, p_8^3, p_9^3);$$

$$(p_6^4, p_8^4, p_9^4);$$

$$(p_{12}^1, p_{16}^1, p_{19}^1, p_{23}^1, p_{12}^2, p_{16}^2, p_{19}^2, p_{12}^3, p_{16}^3, p_{19}^3, p_{12}^4, p_{16}^4, p_{19}^4);$$

$$(p_{11}^1, p_{15}^1, p_{18}^1, p_{22}^1, p_{11}^2, p_{15}^2, p_{18}^2, p_{11}^3, p_{15}^3, p_{18}^3, p_{11}^4, p_{15}^4, p_{18}^4);$$

$$(p_{14}^1, p_{17}^1, p_{21}^1, p_{14}^2, p_{17}^2, p_{14}^3, p_{17}^3, p_{14}^4, p_{17}^4);$$

$$(p_{13}^1, p_{20}^1, p_{24}^1, p_{13}^2, p_{20}^2, p_{13}^3, p_{20}^3, p_{13}^4, p_{20}^4);$$

$$(p_1^1, p_3^1, p_3^1, p_4^1, p_5^1, p_6^1, p_7^1, p_8^1, p_{10}^1, p_{12}^1, p_{15}^1, p_{17}^1, p_{19}^1, p_{22}^1, p_{24}^1,$$
$$p_{12}^2, p_{14}^2, p_{16}^2, p_{17}^2, p_{19}^2, p_{12}^3, p_{14}^3, p_{16}^3, p_{17}^3, p_{19}^3, p_{12}^4, p_{14}^4, p_{16}^4, p_{17}^4, p_{19}^4);$$

$$(p_{11}^1, p_{13}^1, p_{15}^1, p_{18}^1, p_{20}^1, p_{22}^1, p_{24}^1, p_1^2, p_2^2, p_3^2, p_4^2, p_5^2, p_6^2, p_7^2, p_8^2,$$
$$p_{10}^2, p_{12}^2, p_{15}^2, p_{17}^2, p_{19}^2, p_{12}^3, p_{14}^3, p_{16}^3, p_{17}^3, p_{19}^3, p_{12}^4, p_{14}^4, p_{16}^4, p_{17}^4, p_{19}^4);$$

$$(p_{11}^1, p_{13}^1, p_{15}^1, p_{18}^1, p_{20}^1, p_{22}^1, p_{24}^1, p_{11}^2, p_{13}^2, p_{15}^2, p_{18}^2, p_{20}^2, p_1^3, p_2^3, p_3^3,$$
$$p_4^3, p_5^3, p_6^3, p_7^3, p_8^3, p_{10}^3, p_{12}^3, p_{15}^3, p_{17}^3, p_{19}^3, p_{12}^4, p_{14}^4, p_{16}^4, p_{17}^4, p_{19}^4);$$

$$(p_{11}^1, p_{13}^1, p_{15}^1, p_{18}^1, p_{20}^1, p_{22}^1, p_{24}^1, p_{11}^2, p_{13}^2, p_{15}^2, p_{18}^2, p_{20}^2, p_{11}^3, p_{13}^3, p_{15}^3,$$
$$p_{18}^3, p_{20}^3, p_1^4, p_2^4, p_3^4, p_4^4, p_5^4, p_6^4, p_7^4, p_8^4, p_{10}^4, p_{12}^4, p_{15}^4, p_{17}^4, p_{19}^4).$$

Эти же инварианты могут быть сгенерированы из параметрического базиса (4.6).

Таким образом, в настоящем подразделе представлен метод композиционного вычисления инвариантов в параметрическом виде. Метод проиллюстрирован примером доказательства инвариантности модели Петри ЛВС Ethernet с топологией общей шины. Модель Ethernet представляет собой композицию

функциональных подсетей, моделирующих рабочие станции. Применение композиционного вычисления инвариантов позволило выполнить верификацию протокола для произвольного числа подключенных к сети рабочих станций. Для получения инвариантов для произвольного числа рабочих станций использована параметрическая форма представления.

Описанная параметрическая композиция может быть успешно применена для объектов с регулярной структурой, например, при проектировании коммуникационного оборудования и программного обеспечения.

Выводы к 4 разделу

1. Представлена методика построения моделей Петри различного уровня детализации по стандартным спецификациям протоколов с описанием порядка взаимодействия систем и представлением фишками сети Петри целых сообщений либо отдельных полей сообщений, предусмотренных стандартами. Построены модели Петри протоколов BGP и TCP.
2. Впервые представлены методы синтеза модели Петри с использованием промежуточного языка описаний последовательных взаимодействующих процессов Хоара. Разработаны методы синтеза сети Петри по заданной формуле ВПП, а также методы синтеза сети Петри, заданной конечным автоматом с помощью систем линейных уравнений и неравенств.
3. Выполнено построение модели Петри и верификация протокола BGP магистральной маршрутизации Internet. Верификация протокола BGP осуществлена в процессе одновременной и последовательной композиции функциональных подсетей (кланов).
4. Выполнено построение модели Петри и верификация протокола TCP – основного транспортного протокола Internet. Верификация протокола TCP осуществлена в процессе последовательной композиции функциональных под-

сетей (кланов). Получены существенные ускорения вычислений при вычислении инвариантов модели Петри протокола BGP.

5. Выполнен синтез модели Петри протокола электронной коммерции IOTP по исходным спецификациям протокола с использованием промежуточного языка последовательных взаимодействующих процессов Хоара. Осуществлена верификация протокола IOTP в процессе последовательной композиции функциональных подсетей (кланов). Получены значительные ускорения вычислений при вычислении инвариантов модели Петри протокола IOTP.

6. Впервые представлен метод композиционного вычисления инвариантов в параметрической форме, позволяющий выполнить доказательство инвариантности моделей регулярной структуры произвольной размерности.

7. С помощь метода композиционного вычисления инвариантов в параметрической форме впервые доказана инвариантность модели Маршана Ethernet с топологией общей шины для произвольного числа подключенных рабочих станций.

РАЗДЕЛ 5

МОДЕЛИ ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМ И СЕТЕЙ

Разработаны методы автоматизированного построения моделей телекоммуникационных систем в форме раскрашенных сетей Петри и оценки их функциональных характеристик. Построены модели локальных сетей Ethernet, сетей с коммутацией меток MPLS, мобильных сенсорных сетей Bluetooth; выполнены оценки времени отклика и трафика. Показана адекватность моделей реальным телекоммуникационным процессам.

5.1. Модели коммутируемых сетей Ethernet

В последнее время сеть Ethernet (IEEE 803.x) [14] стала самой распространённой ЛВС. С появлением гигабитных технологий открылся новый этап её популярности. Основным элементом сети в настоящее время являются коммутатор [78]; пассивное оборудование, такое как концентратор, служит лишь для соединения терминального и сетевого оборудования, также как и кабельная система. Терминальное оборудование ЛВС состоит из серверов и рабочих станций. Рабочие станции являются источниками запросов к серверам. Серверы выполняют запросы рабочих станций, направляя им результаты выполнения запросов. Причём взаимодействие сервера и рабочей станции может иметь более сложный алгоритм, определяемый спецификой задач, решаемых в среде ЛВС.

Задачей коммутатора является перенаправление входящих фреймов в порт, к которому подключено устройство назначение, задаваемое с помощью MAC-адреса. Заголовок фрейма содержит MAC-адреса источника и приёмника. Как правило, в настоящее время Ethernet работает в полнодуплексном режиме, обеспечивая одновременную передачу и приём фреймов. Для определения но-

мера порта используется таблица коммутации, которая может быть как статической, так и динамической.

Исходной информацией для построения модели является структурная схема сети, представляющая сетевое и терминальное оборудование ЛВС, а также способ их подключения с помощью кабельной системы. Кроме того, необходима информация о быстродействии сетевых карт, портов коммутатора, временных характеристиках процессов взаимодействия рабочей станции и сервера. На основе этой информации определяются единицы измерения модельного времени и временные характеристики элементов модели.

Выполним построение модели ЛВС для структурной схемы, представленной на рис. 5.1. ЛВС имеет архитектуру звезды и состоит из одного коммутатора (**SWI**), трёх концентраторов (**HUB1-HUB3**), а также двух серверов (**S1, S2**) и пяти рабочих станций (**WS1-WS5**). На схеме указаны также модельные значения MAC-адресов, которые могут быть получены с помощью простой нумерации реальных MAC-адресов сетевых карт.

5.1.1. Модель ЛВС

Модель ЛВС, структурная схема которой изображена на рис. 5.1, представляет собой иерархическую сеть, состоящую из пяти отдельных листов. На рис. 5.2 изображён верхний уровень иерархии модели (**LAN**), на остальных листах представлены модели коммутатора (**SWI**), сервера (**S**), рабочей станции (**WS**) и измерительной рабочей станции (**MWS**). Подмодель измерительной рабочей станции **MWS** описана в следующем подразделе; при построении модели она может быть замена типовой подмоделью рабочей станции **WS**.

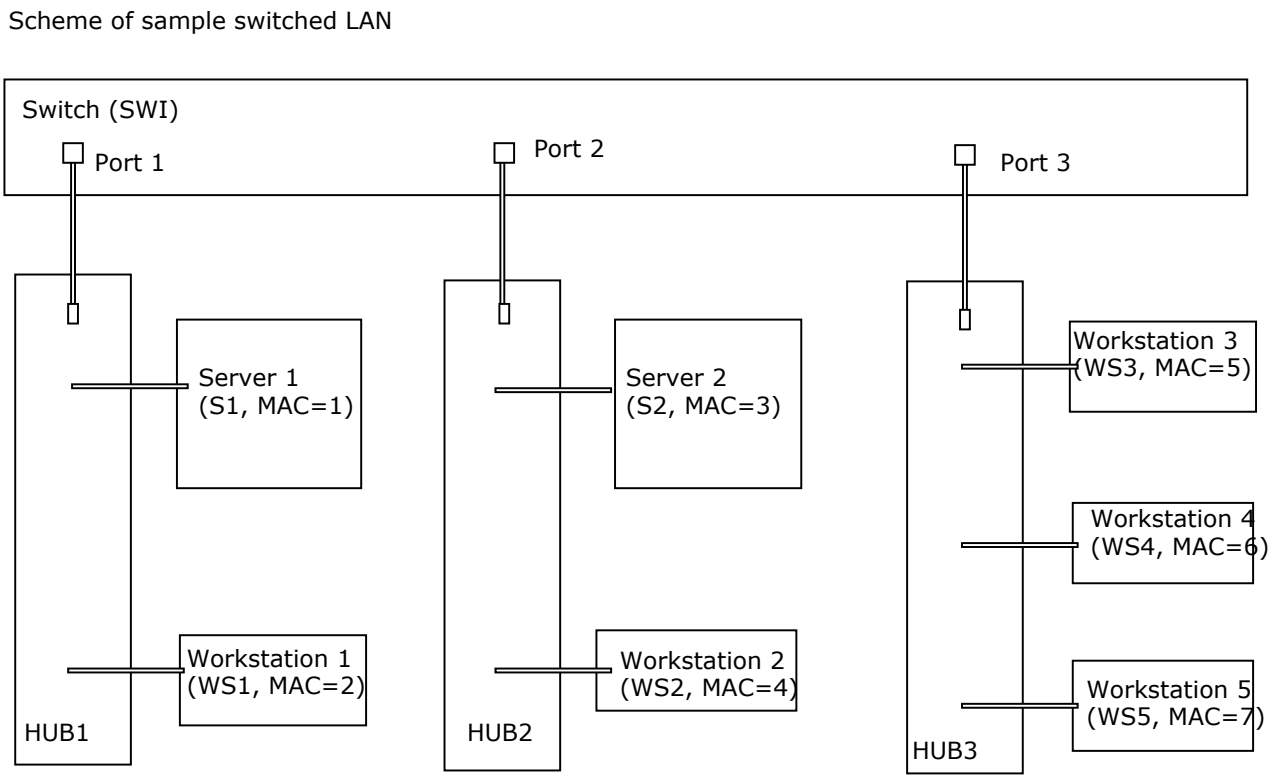


Рис. 5.1. Структурная схема ЛВС

Рабочие станции **WS1-WS4** имеют одинаковый тип **WS**, в то время как рабочая станция **WS5** имеет специальный тип **MWS** и предназначена для измерения времени отклика. Серверы **S1** и **S2** имеют одинаковый тип **S**. Коммутатор **SWI** в единственном экземпляре представлен сетью типа **SWI**.

Каждый сервер и рабочая станция имеет собственный MAC-адрес, указываемый в позициях **aS***, **aWS*** соответственно. Коммутатор содержит отдельные позиции для входящих (**p*in**) и исходящих (**p*out**) фреймов для каждого порта. Таким образом, моделируется полнодуплексный режим работы. Двунаправленные дуги используются для моделирования процедур проверки несущей. Одна из дуг проверяет состояние канала, в то время как другая осуществляет передачу фрейма.

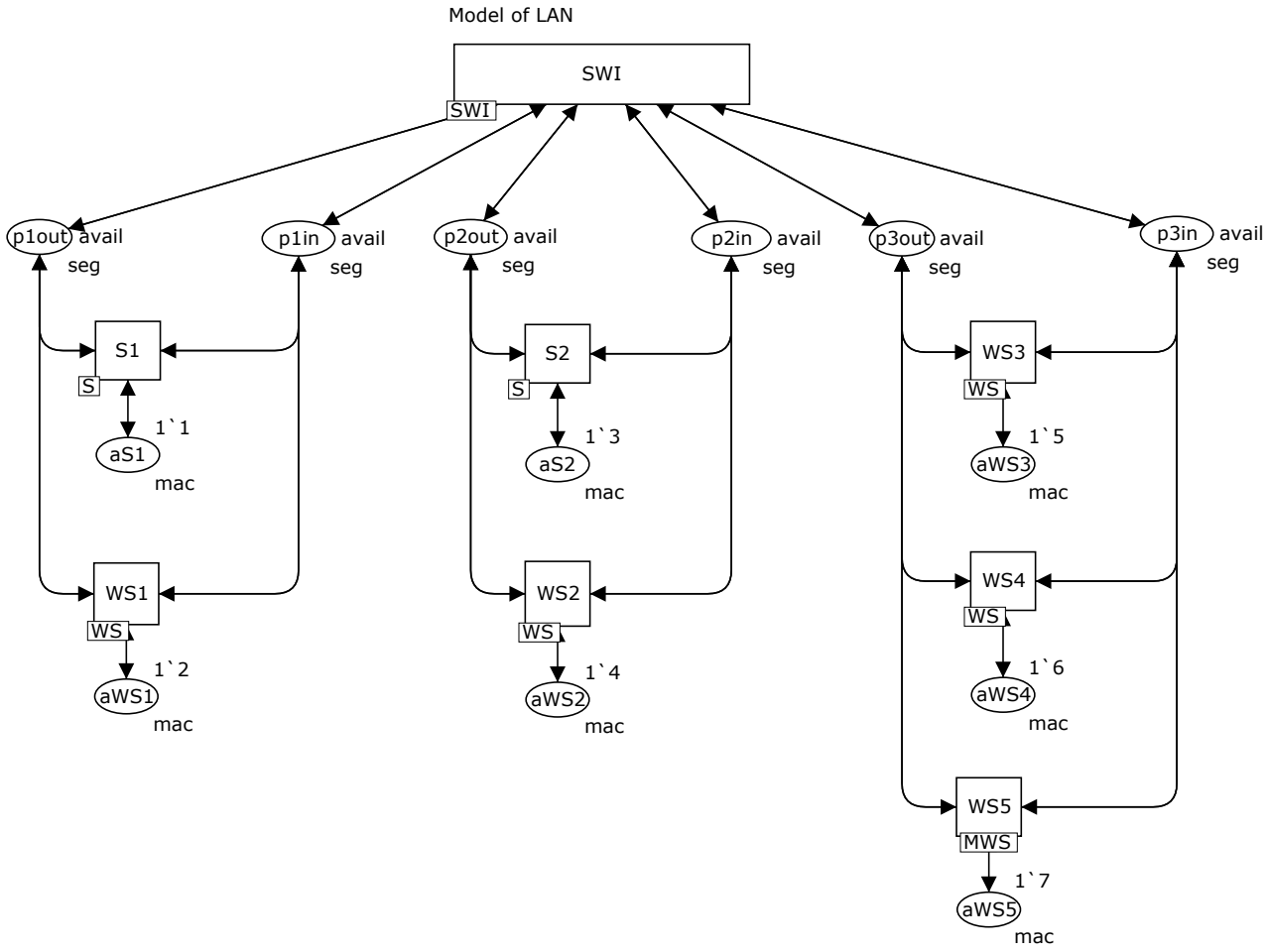


Рис. 5.2. Модель ЛВС

Все описания типов (**color**), переменных (**var**) и функций (**fun**), использованные в модели, представлены на рис. 5.3. MAC-адрес Ethernet моделируется целым числом (color **mac**). Фрейм представлен тройкой **frm**, которая содержит адреса источника (**src**) и назначения (**dst**), а также специальное поле **nfrm** для нумерации фреймов в целях организации вычислений времени отклика. Мы абстрагируемся от остальных полей фрейма, предусмотренных стандартами Ethernet. Тип **seg** представляет однонаправленный канал и может быть либо доступным для передачи (**avail**), либо занятым передачей фрейма (**f.frm**). Выбор представлен с помощью стандартного типа объединения **union**. Заметим, что описатель **timed** использован для фишек, которые задействованы во временных операциях, таких как задержки либо временные штампы.

```
color mac = INT timed;
color portnum = INT;
color nfrm = INT;
color sfrm = product nfrm * INT timed;
color frm = product mac * mac * nfrm timed;
color seg = union f:frm + avail timed;
color swi = product mac * portnum;
color swf = product mac * mac * nfrm * portnum timed;
color remsv = product mac * nfrm timed;
var src, dst, target: mac;
var port: portnum;
var nf, rnf: nfrm;
var t1, t2, s, q, r: INT;
color Delta = int with 1000..2000;
fun Delay() = Delta.ran();
color dex = int with 100..200;
fun Dexec() = dex.ran();
color dse = int with 10..20;
fun Dsend() = dse.ran();
color nse = int with 10..20;
fun Nsend() = nse.ran();
fun cT()=IntInf.toInt(!CPN'Time.model_time)
```

Рис. 5.3. Описания типов, переменных и функций

Маркировка позиций представлена в CPN Tools с помощью мультимножеств. Каждый из элементов принадлежит мультимножеству с определённой кратностью, другими словами – в нескольких экземплярах. Например, начальная маркировка позиции **aWS2** равна **1`4**. Это означает, что позиция **aWS2** содержит 1 фишку со значением 4. Объединение фишек представлено с помощью двойного знака плюс (++). Фишки временных типов имеют форму **x@t**, которая означает, что фишка **x** может быть задействована лишь после момента модельного времени **t**. Поэтому запись **@+d** используется для представления временной задержки на интервал **d**.

5.1.2. Модель коммутатора

Модель коммутатора, изученная в [100, 146], существенно модифицирована для отображения полнодуплексного режима работы и процедур CDMA. Отметим также модульный принцип построения модели. Каждый из портов

представлен стандартным фрагментом, из которых можно легко собрать модель для произвольного заданного числа портов с помощью операции совмещения позиций. Такая конструкция модели позволяет, кроме того, избежать многократных пересечений дуг [146]. Модель коммутатора (**SWI**) изображена на рис. 5.4. Таблица коммутации соответствует структурной схеме ЛВС (рис. 5.1) и является статической.

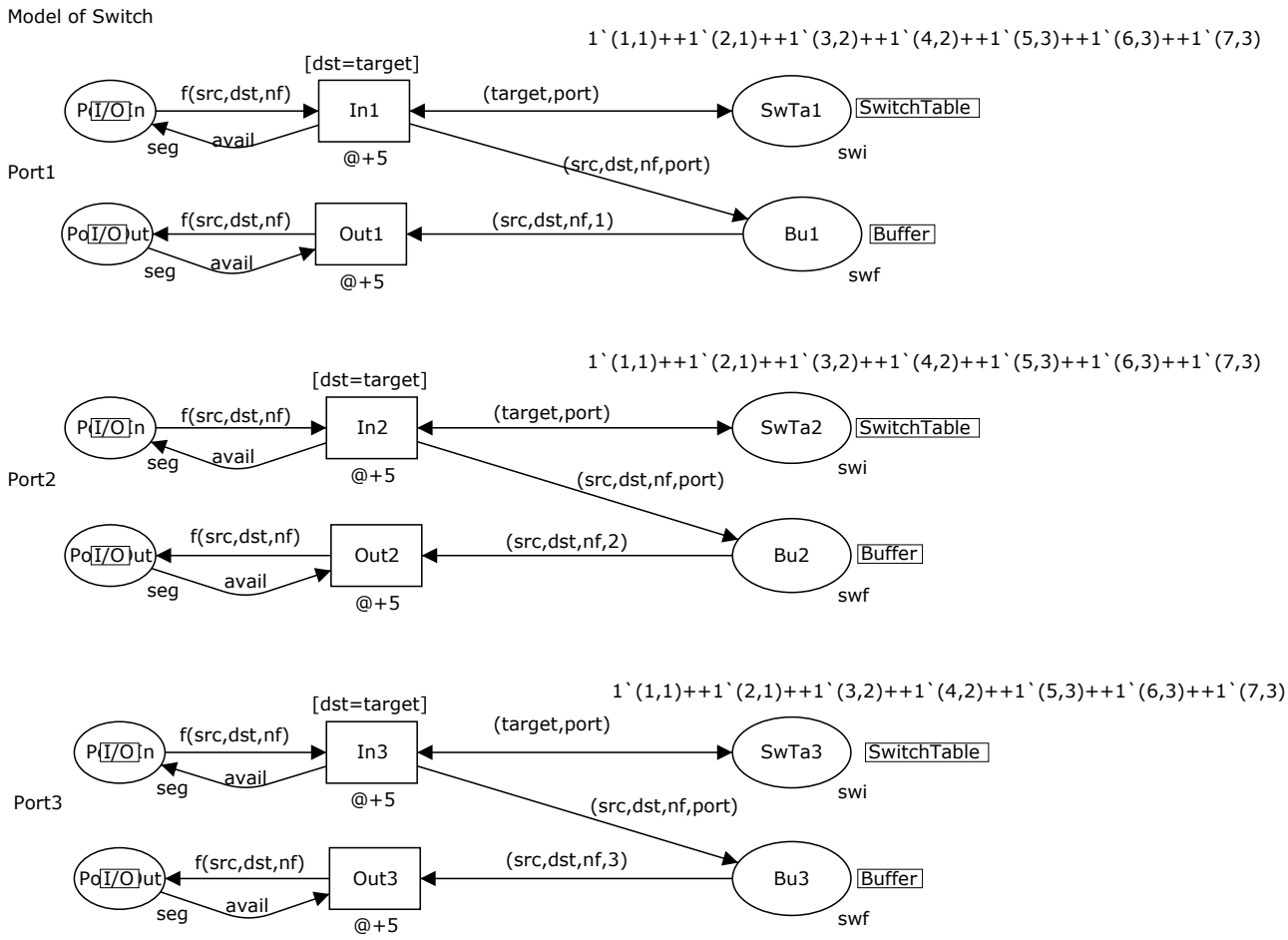


Рис. 5.4. Модель коммутатора (SWI)

Тип **swi** представляет записи таблицы коммутации, которая отображает каждый известный MAC-адрес (**mac**) на номер порта (**nport**). Тип **swf** описывает скоммутированные фреймы, ждущие выделения выходного буфера порта. Поле **portnum** хранит номер порта назначения. Позиции **Port*In** и **Port*Out** представляют входной и выходной буфер порта соответственно. Позиция

SwitchTable моделирует таблицу коммутации; каждая фишка в этой позиции представляет запись таблицы. Например, фишка **1`(4,2)** начальной маркировки означает, что компьютер с MAC-адресом 4 присоединён к порту 2. Позиция **Buffer** соответствует буферу скоммутированных фреймов. Отметим ещё раз, что каждая из позиций **SwitchTable** и **Buffer** являются совмещённой. Так, совмещённая позиция **SwitchTable** представлена на схеме позициями **SwTa1, SwTa2, SwTa3**, а совмещённая позиция **Buffer** – позициями **Bu1, Bu2, Bu3**. Такой способ построения модели позволяет наглядно изображать модели с произвольным количеством портов.

Переходы **In*** моделируют обработку входящих фреймов. Фрейм извлекается из входного буфера только в случае, если таблица коммутации содержит запись с адресом, который равен адресу назначения фрейма (**dst=target**); в процессе перемещения фрейма порт назначения (**port**) сохраняется в буфере. Переходы **Out*** моделируют перемещение скоммутированных фреймов в выходные буферы портов. Фиксированные временные задержки (@+5) присвоены операциям коммутации и записи фреймов в выходные буферы портов.

Остановимся более подробно на описаниях процедур CDMA доступа к ЛВС. Когда фрейм извлекается из входного буфера переходом **In***, он замещается пометкой **avail**. Пометка **avail** показывает, что канал свободен и доступен для передачи. Перед тем, как переход **Out*** посылает фрейм в порт, он ждёт освобождения канала, проверяя наличие фишки **avail**. Заметим, что представленная модель абстрагируется от временных процессов трансляции фрейма в сегменте и, таким образом, не допускает коллизий. Указанные процессы детально изучены в [151].

Остановимся на принципах компоновки модели ЛВС. Позиции **Port*In** и **Port*Out** модели коммутатора являются контактными. Они помечены прямоугольниками **I/O**. Контактные позиции используют для построения иерархических сетей с помощью операции подстановки перехода. Например, переход **SWI** в модели ЛВС верхнего уровня (рис. 5.2) замещается целой сетью **SWI**,

представленной на рис. 5.3. При этом позиции **Port*In** и **Port*Out** отображаются в позиции **p*in** и **p*out** соответственно.

5.1.3. Модели рабочей станции и сервера

Модели рабочей станции и сервера, изученные в [146], дополнены раздельными буферами входящих и исходящих фреймов, а также проверкой занятости канала. Рассмотрены периодически повторяющиеся запросы рабочих станций. В ответ на принятый запрос сервер посылает несколько пакетов в адрес запрашивающей рабочей станции. Количество посылаемых пакетов, периодичность и время выполнения запросов являются случайными равномерно распределёнными величинами. При необходимости могут быть изучены более сложные схемы взаимодействия в системах клиент-сервер, обусловленные спецификой решаемых задач.

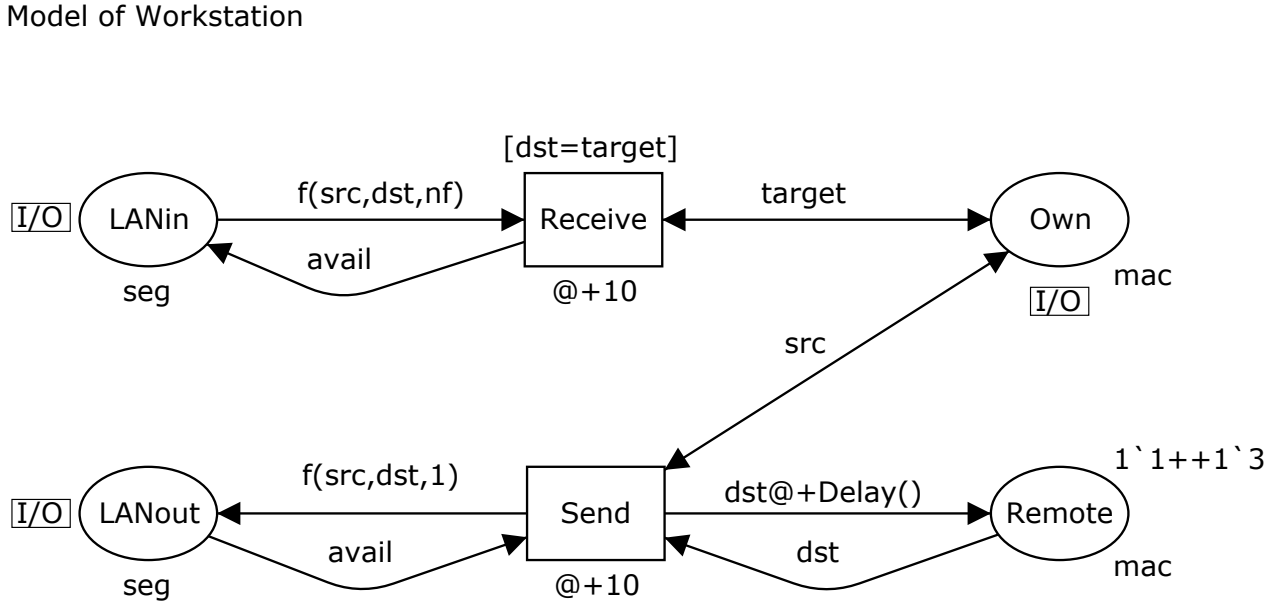


Рис. 5.5. Модель рабочей станции

Модель рабочей станции (**WS**) представлена на рис. 5.5. Позиции **LANin** и **LANout** моделируют входные и выходные каналы сегмента, работающего в полнодуплексном режиме. Рабочая станция слушает сегмент с помощью пере-

хода **Receive**, который получает фреймы с адресами назначения, равными собственному адресу рабочей станции (**dst=target**), сохранённому в позиции **Own**. Обработка фреймов представлена их поглощением. Рабочая станция посылает периодические запросы к серверу с помощью перехода **Send**. Адреса серверов хранятся в позиции **Remote**. После посылки запроса использование адреса сервера блокируется на случайный интервал времени, заданный функцией **Delay()**. Посылка фрейма выполняется лишь в том случае, если сегмент ЛВС свободен, что реализовано проверкой позиции **LANout** на наличие фишки **avail**. Рабочая станция может взаимодействовать с несколькими серверами, сохраняя их адреса в позиции **Remote**.

Заметим, что третье поле фрейма с именем **nfrm** не используется обычной рабочей станцией. Рабочая станция лишь присваивает ему значение равное единице. Это поле использовано в модели специальной измерительной рабочей станции **MWS**. Копии описанной модели **WS** представляют рабочие станции **WS1-WS4**. Для уникальной идентификации каждой рабочей станции использована контактная позиция **Own**. Эта позиция представлена в общей модели ЛВС (рис. 5.2) позициями **aWS*** и содержит MAC-адрес компьютера.

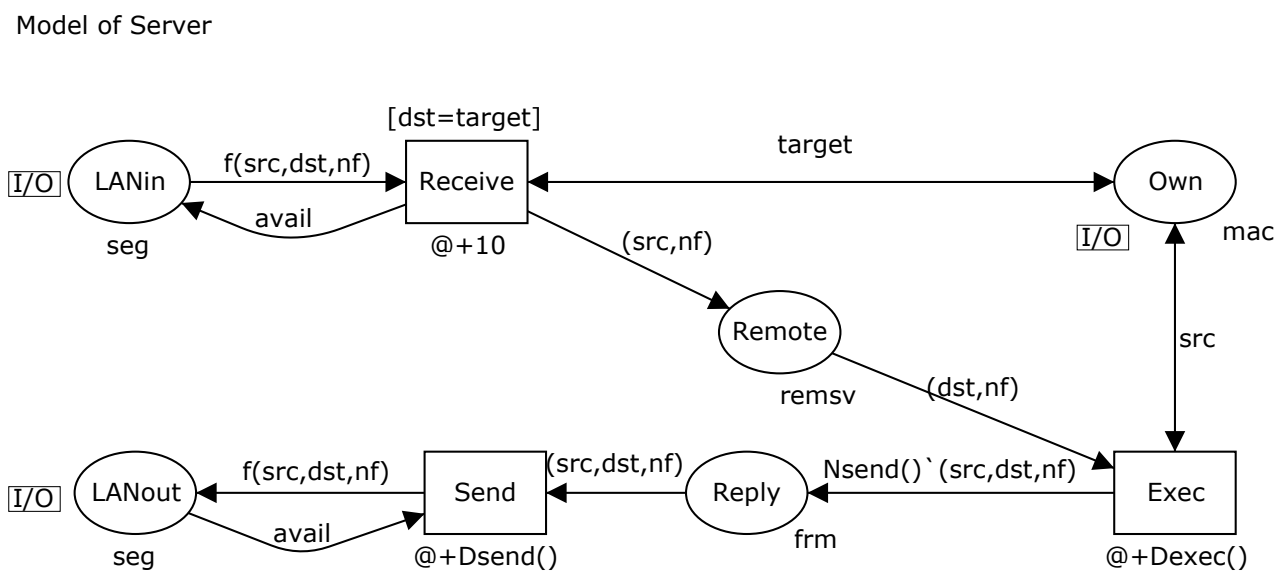


Рис. 5.6. Модель сервера

Модель сервера (**S**) изображена на рис. 5.6. Прослушивание сегмента сети аналогично модели рабочей станции, отличие состоит в том, что адрес отправителя фрейма **src** и порядковый номер запроса **nf** сохраняются в позиции **Remote**. Переход **Exec** моделирует выполнение запроса рабочей станции сервером. В результате выполнения запроса сервер генерирует случайное число **Nsend()** фреймов ответа, которые сохраняются в позиции **Reply**. Затем эти фреймы по одному передаются в сеть переходом **Send**. Заметим, что номер запроса **nf**, сохранённый в позиции **Remote**, используется для идентификации ответа тем же самым номером, что и запрос.

5.1.4. Параметры модели

Правильный выбор единицы измерения модельного времени также как и вычисление временных параметров элементов модели является ключевым вопросом для построения адекватной модели. Для этого требуется тщательный анализ характеристик реального оборудования и программного обеспечения конкретной ЛВС.

Схема, изображённая на рис. 5.1, представляет фрагмент ЛВС диспетчерского центра железной дороги, оснащённого программным обеспечением ГИД Урал-ВНИИЖТ [157]. Ядро системы состоит из пары зеркальных серверов **S1** и **S2**. Рабочие станции **WS1-WS5** расположены непосредственно на рабочих местах железнодорожных диспетчеров.

Необходимо рассмотреть производительность конкретных коммутаторов ЛВС и сетевых адаптеров для вычисления временных задержек, моделируемых переходами **In***, **Out***, **Send**, **Receive**. Кроме того, необходимо изучить особенности взаимодействия в системе клиент-сервер программного обеспечения ГИД Урал для оценки таких параметров как задержка между запросами **Delta** и время выполнения запроса **dex**. Так как единицей информации, передаваемой в сети, является Ethernet-фрейм, следует выразить длины сообщений в количестве передаваемых фреймов. Для этих целей выберем максимальную длину фрейма Ethernet, равную 1,5 Кб.

Типы используемого оборудования ЛВС указаны в табл. 5.1.

Таблица 5.1 -

Типы оборудования

Устройство	Тип
Сетевой адаптер	Intel EtherExpress 10/100
Коммутатор ЛВС	Intel SS101TX8EU
Сервер	HP Brio BA600
Рабочая станция	HP Brio BA200

В Таблице 5.2 представлены параметры описанной ранее модели. Наименьшее значение времени соответствует продолжительности операции чтения/записи фрейма для коммутатора ЛВС [73]. Но в целях дальнейшего использования более производительного оборудования выбрана единица модельного времени (EMB) равная 100 нс.

Таблица 5.2 -

Параметры модели

Параметр	Переменная /Элемент	Реальное значение	Модельное значение
Время чтения фрейма коммутатором ЛВС	In*	500 нс	5
Время записи фрейма коммутатором ЛВС	Out*	500 нс	5
Время чтения фрейма сетевым адаптером	Receive	1 мс	10
Время записи фрейма сетевым адаптером	Send	1 мс	10
Время обработки запроса сервером	Dex	10-20 мс	100-200
Интервал между запросами клиента	Delta	100-200 мс	1000-2000
Длина запроса		1.2 Кб	1
Длина ответа	Nse	15-30 Кб	10-20

Таким образом, в настоящем подразделе представлена методология композиции модели телекоммуникационной сети из подмоделей её компонентов. Построены основные компоненты для композиции моделей коммутируемой Ethernet – подмодели: рабочей станции, сервера, коммутатора.

5.2. Метод измерительных фрагментов

В процессе имитации динамики раскрашенной сети Петри, также как и в классических системах имитационного моделирования, таких, например, как GPSS [192], может быть аккумулирована статистическая информация. Но стандартные возможности моделирующей системы позволяют накапливать простейшую информацию о частотах срабатывания переходов, средней, максимальной и минимальной маркировке позиций. При исследовании телекоммуникационных систем практический интерес представляю сложные функциональные характеристики, для измерения которых на реальных телекоммуникационных сетях используют сложное оборудование и программное обеспечение для обработки результатов измерения.

Расширенные сети Петри (начиная с ингибиторных) являются универсальной алгоритмической системой [160]; они позволяют представлять арифметические, логические операции, а также управление последовательностью действий. Известно применение сетей Петри для описания параллельных алгоритмов и обработки информации, управляемой потоками данных [110, 160]. Кроме того, элементы сети можно использовать как измерители, аккумулирующие простейшие характеристики модели. Предложено использовать сети Петри для измерения и вычисления нетривиальных характеристик модели. Соответствующий метод назван методов измерительных фрагментов.

Метод измерительных фрагментов можно сформулировать следующим образом:

1. Построить исходную модель телекоммуникационной системы либо сети, содержащую элементы, моделирующие реальное оборудование и программное обеспечение.

2. Дополнить модель элементами, осуществляющими первичное измерение простейших функциональных характеристик.

3. Дополнить модель фрагментами сети Петри, реализующих вычисление сложных функциональных характеристик.

4. Выполнять имитацию динамики модели на длительных интервалах времени, обеспечивающих стационарный режим её функционирования, и табулировать результаты измерения.

Таким образом, модель дополненная измерительными фрагментами, обеспечивает получение информации, необходимой для анализа моделируемой системы и её оптимизации в процессе управляемого моделью проектирования.

Рассмотрим особенности метода на примере измерения функциональных характеристик модели коммутируемой Ethernet, построенной в предыдущем подразделе. Основной характеристикой для приложений реального времени является время отклика сети [14, 155, 190]. Для измерения времени отклика построена специальная модель измерительной рабочей станции. Модель измерительной рабочей станции (**MWS**) изображена на рис. 5.7. По существу, она представляет собой рассмотренную ранее модель рабочей станции **WS**, дополненную измерительными элементами, образующими фрагмент сети, выделенный пунктиром.

Так как расширенные сети Петри являются универсальной алгоритмической системой [160, 181], они позволяют описывать не только исследуемый объект, но также дополнительные алгоритмы, предназначенные как для управления объектом [180], так и для организации измерений функциональных характеристик. При этом алгоритм может быть сформирован из фрагментов сетей, представляющих логические, арифметические операции [181], альтернативные и параллельные процессы [110, 160], характерные для современных языков программирования. Детальные описания таких алгоритмов, полностью собранные из фрагментов сетей могут быть довольно громоздкими. CPN Tools позволяет получить разумный компромисс между единством инструментальных средств и компактностью представления модели. Встроенный язык ML даёт возможность дополнить сеть Петри операторами языка программирования, ассоциированными как с дугами, так и с переходами сети. При этом необходимые величины могут сохраняться как в позициях сети Петри, так и в переменных языка ML.

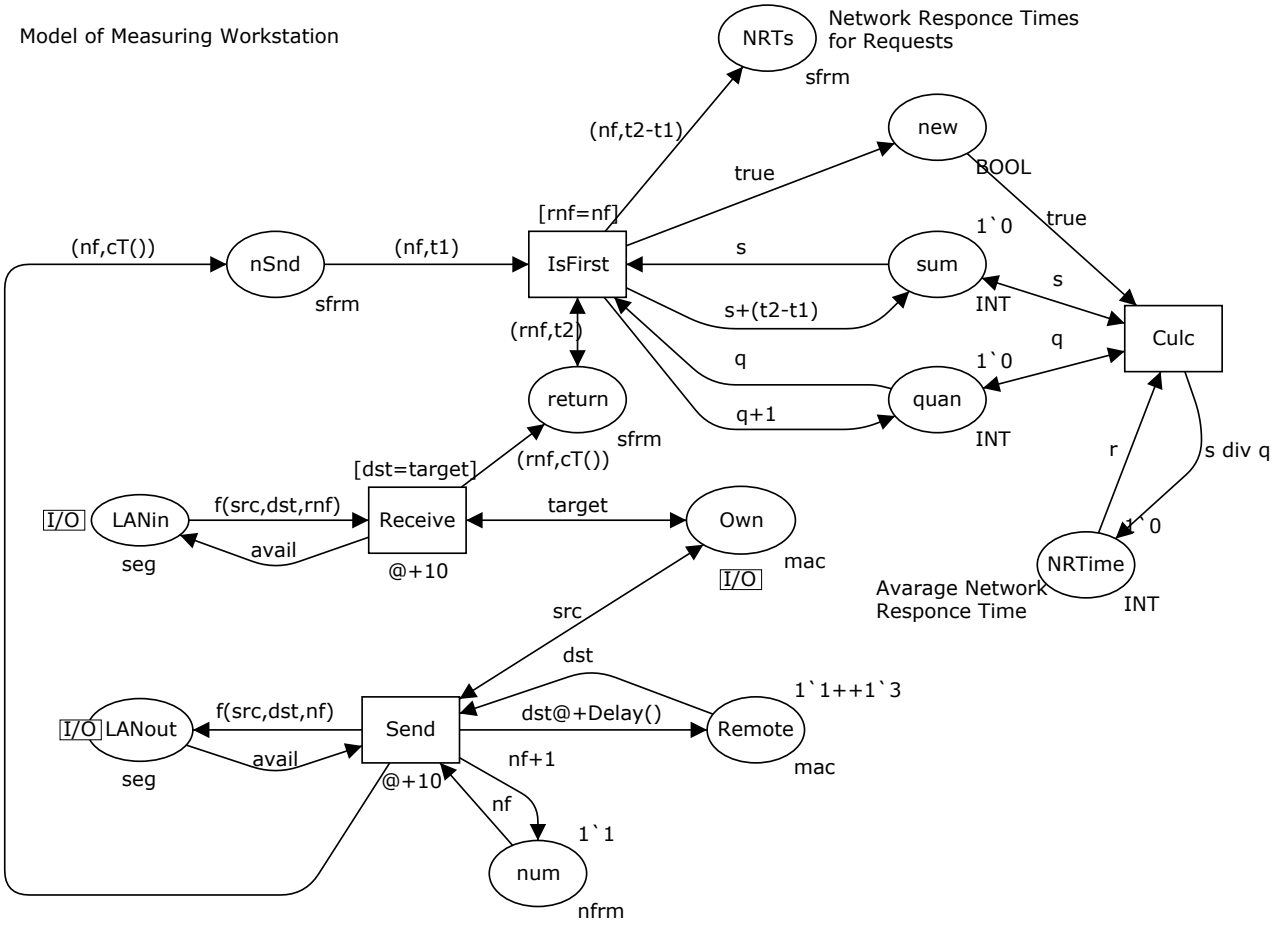


Рис. 5.7. Модель измерительной рабочей станции

Рассмотрим более детально измерительные элементы, представленные на рис. 5.7. Каждый запрос рабочей станции нумеруется уникальным числом, содержащимся в позиции **num**. Время отправления запроса сохраняется в позиции **nSnd**. Функция **cT()** определяет текущее модельное время. Позиция **nSnd** запоминает пару: номер запроса (фрейма) **nf** и время отправления запроса в сеть.

Позиция **return** запоминает временные штампы всех вернувшихся фреймов. В качестве времени отклика сети рассмотрен интервал времени между посылкой запроса и получением первого из фреймов ответа. Это значение сохраняется в позиции **NRTs** для каждого обработанного запроса. Переход **IsFirst**

распознаёт первый фрейм ответа. Описатель дуги, соединяющей переход **Is-First** с позицией **NRTs**, вычисляет время отклика (t2-t1).

Оставшаяся часть измерительного фрагмента вычисляет среднее время отклика. Позиции **sum** и **quant** накапливают сумму времён отклика и количество принятых ответов соответственно. Прибытие нового ответа распознаётся позицией **new** и инициирует перевычисление среднего времени отклика с помощью перехода **Culc**. Результат сохраняется в позиции **NRTime**.

Построенная модель отлажена и протестирована в пошаговом режиме имитации динамики сети Петри (Приложение В). Для этих целей фреймы, генерируемые рабочей станцией, трассировались через модель ЛВС по пути к серверу и обратно. Также наблюдалось поведение модели в процессе автоматической имитации с визуальным отображением динамики сети – в режиме так называемой игры фишек. Это позволяет оценить модель с помощью визуального наблюдения процессов на основной странице и страницах компонентов модели в процессе имитации.

Для более точной оценки времени отклика выбирались значительные интервалы модельного времени. При этом удобно использовать автоматический режим имитации без отображения промежуточных маркировок, целью которого является накопление статистической информации.

Образ экрана измерительной рабочей станции представлен на рис. 5.8. Пометки в прямоугольниках представляют текущую маркировку моделирующей системы. Позиция **LANin** содержит фрейм (3,6,1). Позиция **LANout** представляет доступное состояние канала **avail**. Номер следующего запроса в соответствии с маркировкой позиции **num** равен 6. Позиция **return** содержит 45 фрейма полученных ответов серверов. Позиция **NRTs** содержит времена отклика для каждого из 4 обработанных запросов. Например, время отклика сети для запроса 3 составляет 614 единиц модельного времени. Можно проверить, что среднее время отклика 578 в позиции **NRTime** равно 2315/4 в соответствии с маркировками позиций **sum** и **quant**.

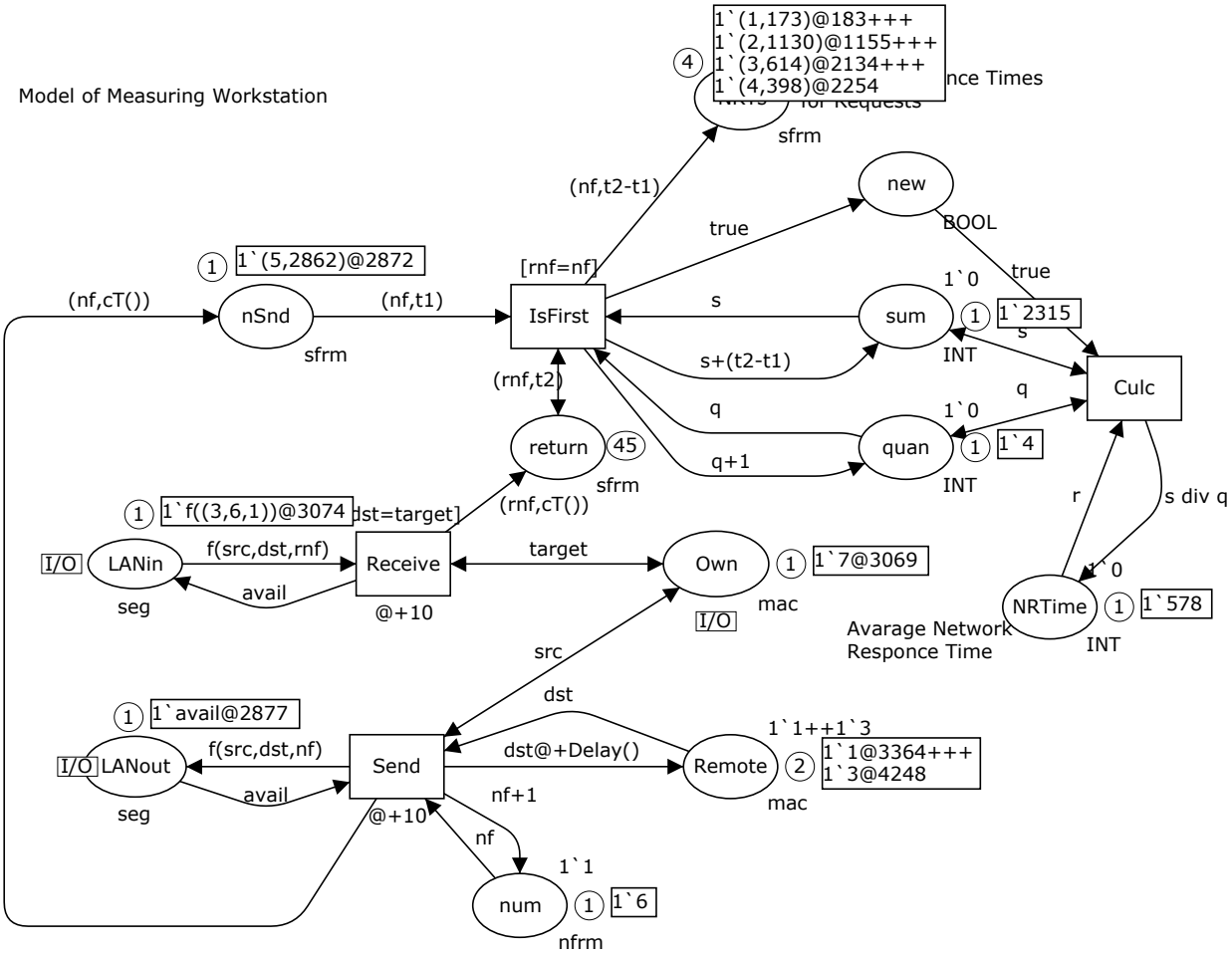


Рис. 5.8. Оценка времени отклика

При моделировании на длительных интервалах до часа реального времени полученное среднее время отклика сети равно 389 EMB или около 39 мс. Эта величина удовлетворяет требованиям оперативного управления движением поездов. Были также построены фрагменты для измерения числа коллизий в сегменте Ethernet, измерения размера буфера коммутатора и другие.

Таким образом, в настоящей подразделе представлена методология измерения характеристик моделей Петри телекоммуникационных систем с помощью специальных измерительных фрагментов сетей Петри. Особенности применения измерительных фрагментов изучены на примере определения времени отклика в модели коммутируемой локальной сети.

Представленная методология построения моделей в форме иерархических раскрашенных временных сетей Петри, а также измерения характеристик моде-

ли может быть успешно применена при исследовании и проектировании различного телекоммуникационного оборудования.

5.3. Модели сетей с коммутацией меток MPLS

Технология коммутации меток MPLS [55, 82, 83] предназначена для увеличения пропускной способности как глобальных, так и корпоративных сетей. В сетях с коммутацией пакетов, просмотр таблиц маршрутизации для каждого передаваемого пакета каждым маршрутизатором требует значительных временных затрат и зачастую ограничивает общую пропускную способность сети. Метка, назначенная в соответствии с технологией MPLS, имеет меньший размер, чем IP-адрес, и представляет собой, по существу, идентификатор виртуального канала, для обработки которого предусмотрены эффективные алгоритмы.

Построение аналитических моделей MPLS сетей затруднено ввиду сравнительно высокой сложности технологии, стандартные спецификации которой насчитывают более пяти сотен страниц. Поэтому имитационное моделирование MPLS сетей является перспективным направлением исследований. Раскрашенные сети Петри моделирующей системы CPN Tools [6] представляют собой комбинацию графа сети Петри [169] и языка программирования CPN ML, используемого для описания атрибутов элементов. Ранее система CPN Tools применена для исследования коммутируемой Ethernet [95, 100, 146]. Метод измерительных фрагментов [140] позволяет выполнить измерение нетривиальных характеристик моделируемого объекта в процессе имитации динамики сети Петри. Однако, построение моделей MPLS требует специального исследования в связи с существенным её отличием от классической технологии коммутации.

Целью настоящего подраздела является построение типовых моделей IP и MPLS маршрутизаторов в форме раскрашенных сетей Петри, а также сравни-

тельная оценка эффективности IP-маршрутизации и MPLS на примере модели фрагмента европейской магистрали Интернет.

5.3.1. Обзор технологии коммутации меток MPLS

В настоящее время спецификации стандартов MPLS технологии посвящено около двадцати документов IETF. Основными являются документы [55, 82, 83], кроме того, особенности технологии уточняются в RFC с номерами 2547, 2702, 2917, 3035, 3063, 3346, 3353, 3429, 3443, 3468, 3469, 3471, 3472, 3473, 3474, 3496, 3564.

Стандартная терминология представлена в [82], где отмечена необходимость агрегирования маршрутов сети в классы эквивалентности передачи FEC (Forwarding Equivalence Classes) и назначения каждому классу уникальной метки (label), идентифицирующей принадлежность к FEC. Метка отправляется вместе с пакетом, и после её назначения, является единственным объектом, анализируемым маршрутизаторами и используемым для доставки пакета. Далее, на следующих хопах, первоначальная метка замещается новыми метками. Таким образом, анализ заголовка пакета выполняется только первым маршрутизатором; при транспортировке пакета метка представляет собой индекс в таблицах коммутации меток, задающих следующий хоп и новую метку. Стандартом предусмотрено также создание стека меток. Для назначения меток предусмотрен протокол распределения меток LDP (Label Distribution Protocol), в соответствии с которым соседние маршрутизаторы коммутации меток LSR (Label Switching Router) обмениваются маршрутной информацией и строят свои таблицы коммутации меток.

В [83] описан формат метки и стека меток, а также процедуры обработки меток LSR. Метка занимает 32 бита и имеет формат, представленный на рис. 5.9. Собственно значение метки (Label Value) занимает 20 бит, время её жизни TTL 8 бит, бит 23 предназначен для создания стека меток, он равен единице для метки на дне стека, биты 20-22 зарезервированы для экспериментального использования.

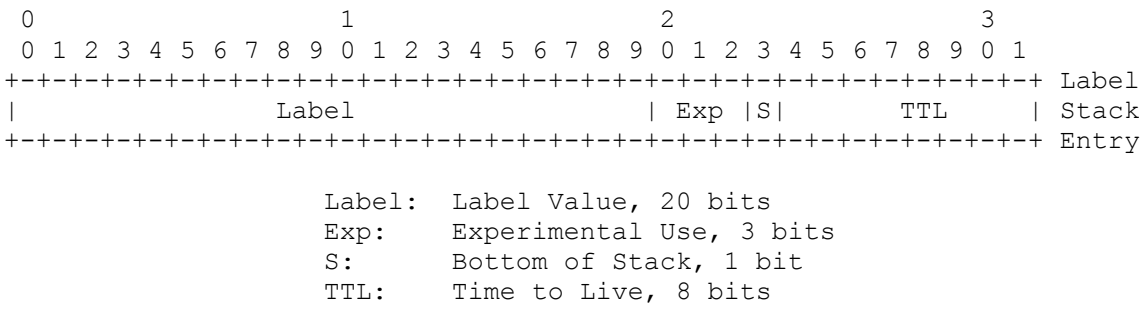


Рис. 5.9. Стандартный формат метки

Маршрутизаторы, находящиеся на границе MPLS сетей принято именовать LER (Label Edge Router); дополнительно их классифицируют во входные (ingress) и выходные (egress).

5.3.2. Модель IP-маршрутизатора

Напомним, что раскрашенная сеть Петри [50] моделирующей системы CPN Tools [6] представляет собой комбинацию графа сети Петри и языка программирования CPN ML, используемого для описания атрибутов элементов. Фишка раскрашенной сети представляет собой элемент абстрактного типа данных, традиционно называемого цветом.

В настоящей работе рассматривается статическая маршрутизация. Модель порта IP-маршрутизатора представлена на рис. 5.10. Модель конкретного маршрутизатора собирается путём клонирования модели порта в необходимом количестве экземпляров. В отличие от ранее изученных моделей Ethernet коммутаторов [95, 100, 140, 146], представленная модель маршрутизатора использует полные спецификации IP-адресов хостов и сетей. Обработка IP-адресов потребовала создания дополнительных функций для определения принадлежности IP-адреса некоторой подсети перечисленной в таблице маршрутизации.

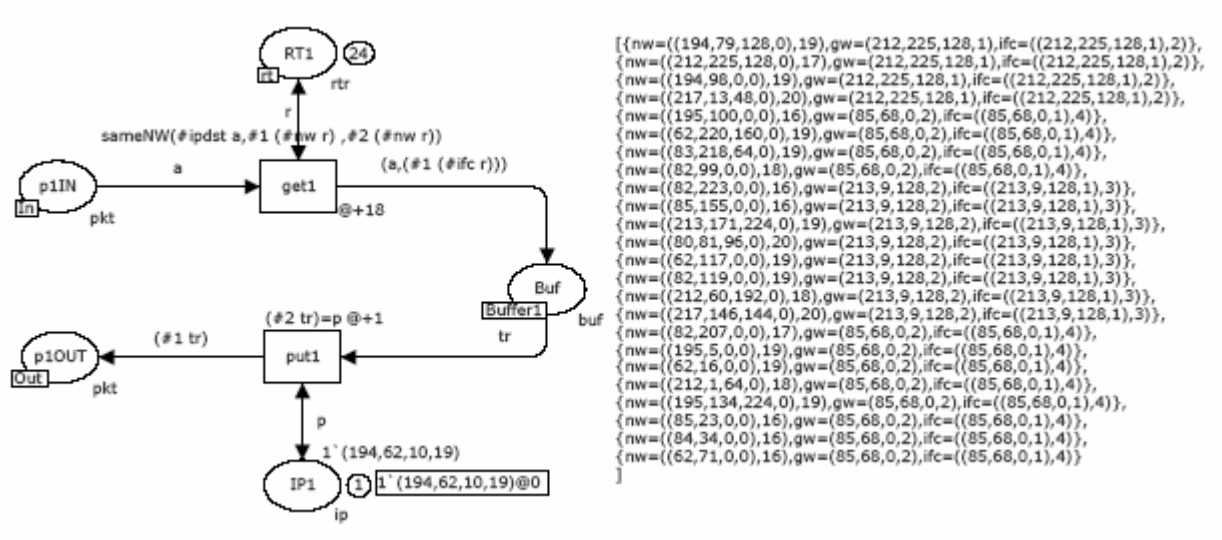


Рис. 5.10. Модель порта IP-маршрутизатора

В модели использованы четыре основных типа фишек: тип **pkt** описывает пакеты, передаваемые в сети, тип **rtr** – записи таблицы маршрутизации, тип **buf** – маршрутизированные пакеты, с указанием IP-адреса порта назначения, тип **ip** – IP-адреса. Функция **SameNW** определяет принадлежность IP-адреса сети. Описания, типов, переменных и функций представлены на рис. 5.11. Совмещённые позиции **rt** и **Buffer1** обеспечивают использование общей таблицы маршрутизации и общего выходного буфера всеми портами маршрутизатора.

```
colset ip=product INT*INT*INT*INT timed;                colset mask=INT;
colset nwt=product ip*mask timed;                       colset b=INT timed;
colset pkt=record ipsrc:ip*ipdst:ip*data:b timed;       colset nif=int;
colset interf=product ip*nif;                           colset gw=ip;
colset rtr=record nw:nwt *gw:ip*ifc:interf  timed ;     colset buf=product pkt*ip;
colset lpkt=record label:INT*ipsrc:ip*ipdst:ip*data:b timed;  colset rtrs=list rtr timed;
colset lt=record ininterf:INT*L1:INT*outinterf:INT*L2:INT;    colset aux=INT timed;
fun pow(x,0)=1 | pow(x,y)=pow(x,y-1)*x;                 colset buffer=product lpkt*INT timed;
fun sameNW(a1:ip,a2:ip,m:INT)=if (32-m) < 8 then ( if ((#1 a1) = (#1 a2)) andalso ((#2 a1) = (#2 a2)) andalso
((#3 a1) =(#3 a2)) andalso ((((#4 a1) div pow(2,(32-m)))*(pow(2,(32-m)))) = (#4 a2)) then true else false ) else if (32-
m)<16 then ( if ((#1 a1) = (#1 a2)) andalso ((#2 a1) = (#2 a2)) andalso ((((#3 a1) div pow(2,((32-m)-8))) * pow(2,((32-
m)-8))) = (#3 a2)) then true else false) else  if (32-m)<24 then  (if ((#1 a1)=(#1 a2)) andalso ((((#2 a1) div pow(2,((32-
m)-16)))*pow(2,((32-m)-16)))=(#2 a2))  then true else false) else  if ((((#1 a1) div pow(2,((32-m)-24)))*pow(2,((32-m)-
24)))=(#1 a2)) then true else false;
var i,c,k:INT;          var q:nwt;          var r:rtr;          var p,src,dst:ip;
var a:pkt ;             var tr:buf;         var v:aux;          var la:lpkt;
```

Рис. 5.11. Описания типов, переменных и функций моделей

Порты маршрутизатора работают в полнодуплексном режиме. Из входного канала порта **p1IN** пакет извлекается переходом **get1** и размещается во внутреннем буфере маршрутизатора **Buf**; при этом функция **sameNW** определяет IP-адрес интерфейса назначения в соответствии с таблицей маршрутизации, представленной позицией **RT1**. Переход **put1** извлекает пакет из буфера в соответствии с IP-адресом интерфейса, хранимым в позиции **IP1**, и размещает его в позиции **p1OUT**, моделирующей выходной канал порта.

5.3.3. Модель MPLS-маршрутизаторов

Стандарт технологии MPLS предусматривает два типа используемых маршрутизаторов: LSR-маршрутизаторы размещаются внутри MPLS сети и реализуют только коммутацию меток; LER-маршрутизаторы размещаются на границе MPLS сети и используются для стыковки с сетями других стандартов. Основное отличие состоит в том, что LER-маршрутизатор вычисляет первоначальную метку пакета на основе маршрутной информации, например, на основе таблиц IP-маршрутизации для IP-сетей. Как правило, граничные маршрутизаторы являются комбинированными LSR/LER типа. Для структурирования моделей построим типовые модели LER и LSR портов; модели конкретных маршрутизаторов будем собирать путём клонирования моделей портов требуемого типа: LSR при стыковке с MPLS сетью и LER при стыковке с IP-сетью. В настоящей работе алгоритм LDP назначения классов эквивалентности меток FEC не моделируется, рассматриваются классы эквивалентности меток, полученные в результате его применения к указанной сети.

Типовая модель LSR порта маршрутизатора изображена на рис. 5.12. Основное отличие от модели порта IP-маршрутизатора (рис. 5.10) состоит в том, что для перенаправления пакета используется таблица коммутации меток. Кроме того, моделируется замещение метки, и для идентификации интерфейсов используются их номера. Для входящего пакета по его метке определяется соответствующая запись таблицы коммутации меток **Ltable1**. Исходящая дуга перехода **get1** замещает входную метку **L1** новой меткой **L2** и присваивает но-

мер интерфейса назначения для направления пакета. Тип данных **lpkt** описывает пакет, дополненный меткой, тип **lt** – записи таблицы коммутации меток, тип **buffer** – записи внутреннего буфера пакетов. Заметим, что в настоящей работе не моделируется стек меток, предусмотренный стандартом [82, 83], а выполняется лишь замещение меток.

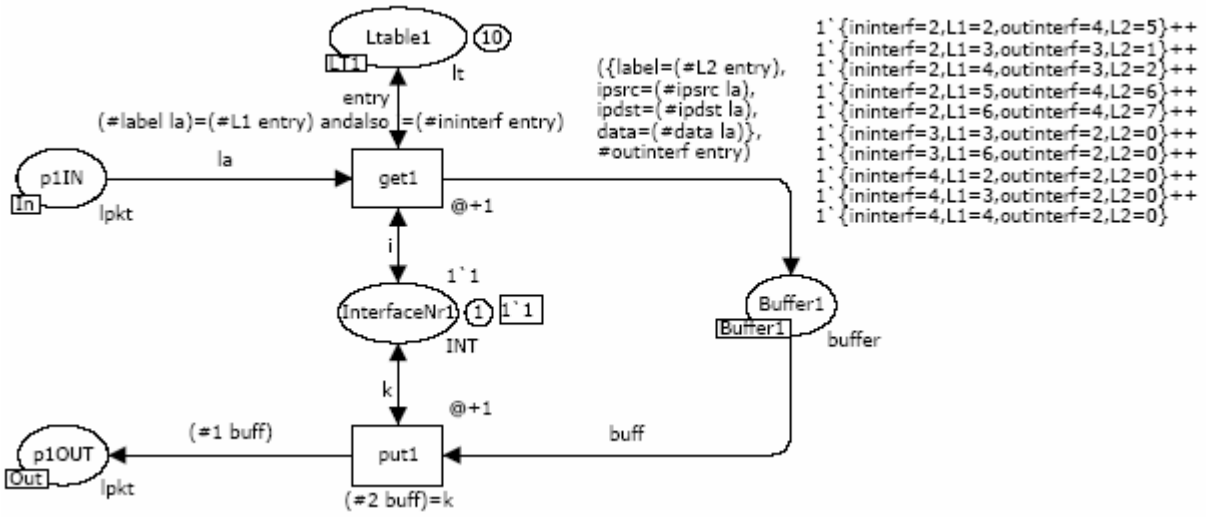


Рис. 5.12. Модель LSR порта MPLS маршрутизатора

Типовая модель LER порта маршрутизатора изображена на рис. 5.13. Порт выполняет первоначальное назначение меток в соответствии с агрегированными маршрутами сети. Внешне модель напоминает комбинацию портов IP и LSR; используется как таблица маршрутизации **RT2**, так и таблица коммутации меток **Ltable2**. Отличие таблицы маршрутизации состоит в том, что она содержит значение присваиваемой пакету метки, полученной в результате работы LDP алгоритма. Позиция **Buffer2** представляет внутренний буфер пакетов маршрутизатора; позиция **Buf1** предназначена для временного хранения пакетов после первоначального назначения меток. Заметим, что в выходной канал порта **p2OUT** выводится пакет без метки.

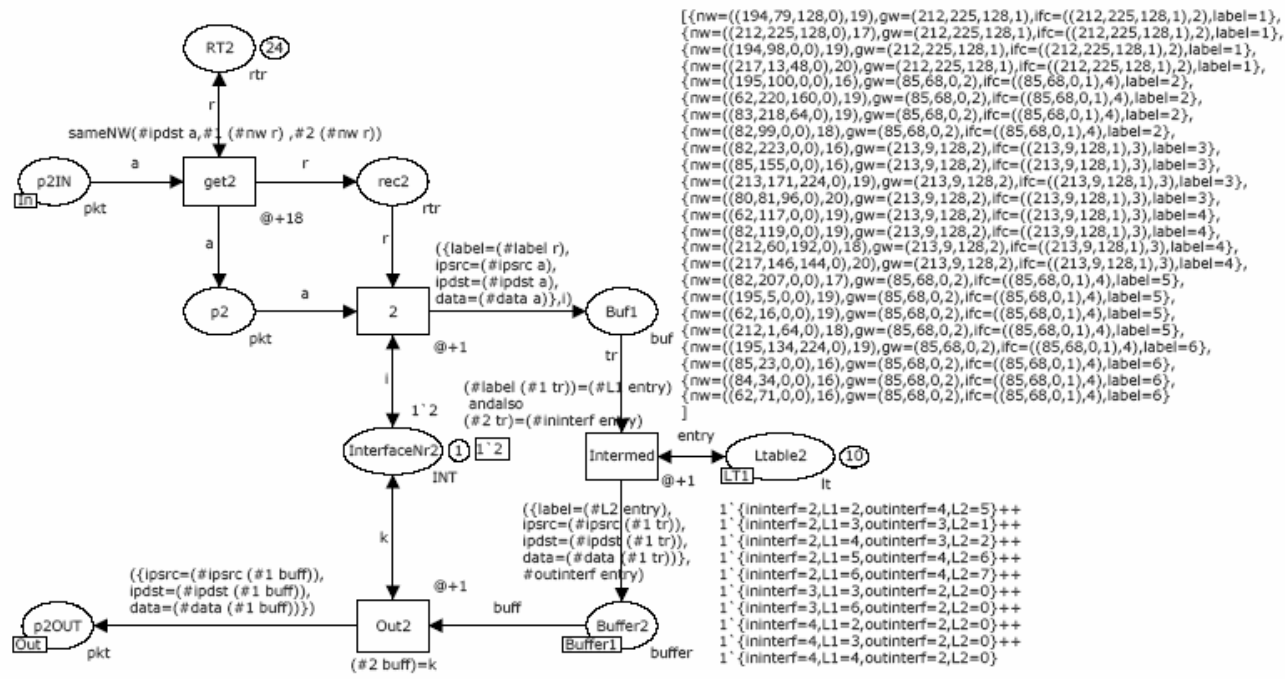


Рис. 5.13. Модель LER порта MPLS маршрутизатора

5.3.4. Модель европейской магистрали Интернет

Для оценки эффективности технологии MPLS построены модели фрагмента Европейской магистрали Интернет по структурной схеме сети, представленной на рис. 5.14. На схеме указаны 6 терминальных сетей, содержащих IP-адреса в адресном пространстве соответствующих стран; всего использовано 24 IP-сети. Магистральная сеть образована 7-ю маршрутизаторами 6 из которых (R1-R3, R5-R7) при моделировании MPLS имеют LER/LSR тип и один (R4) – LSR тип.

Главная страница модели представлена на рис. 5.15. Она построена по структурной схеме сети. Исследовалось два типа моделей: IP-маршрутизации, полученной при подстановке в качестве **Router1-Router7** подмоделей IP-маршрутизаторов; MPLS-маршрутизации, полученной при подстановке в качестве **Router1-Router7** подмоделей LER/LSR маршрутизаторов, описанных в предыдущих пунктах.

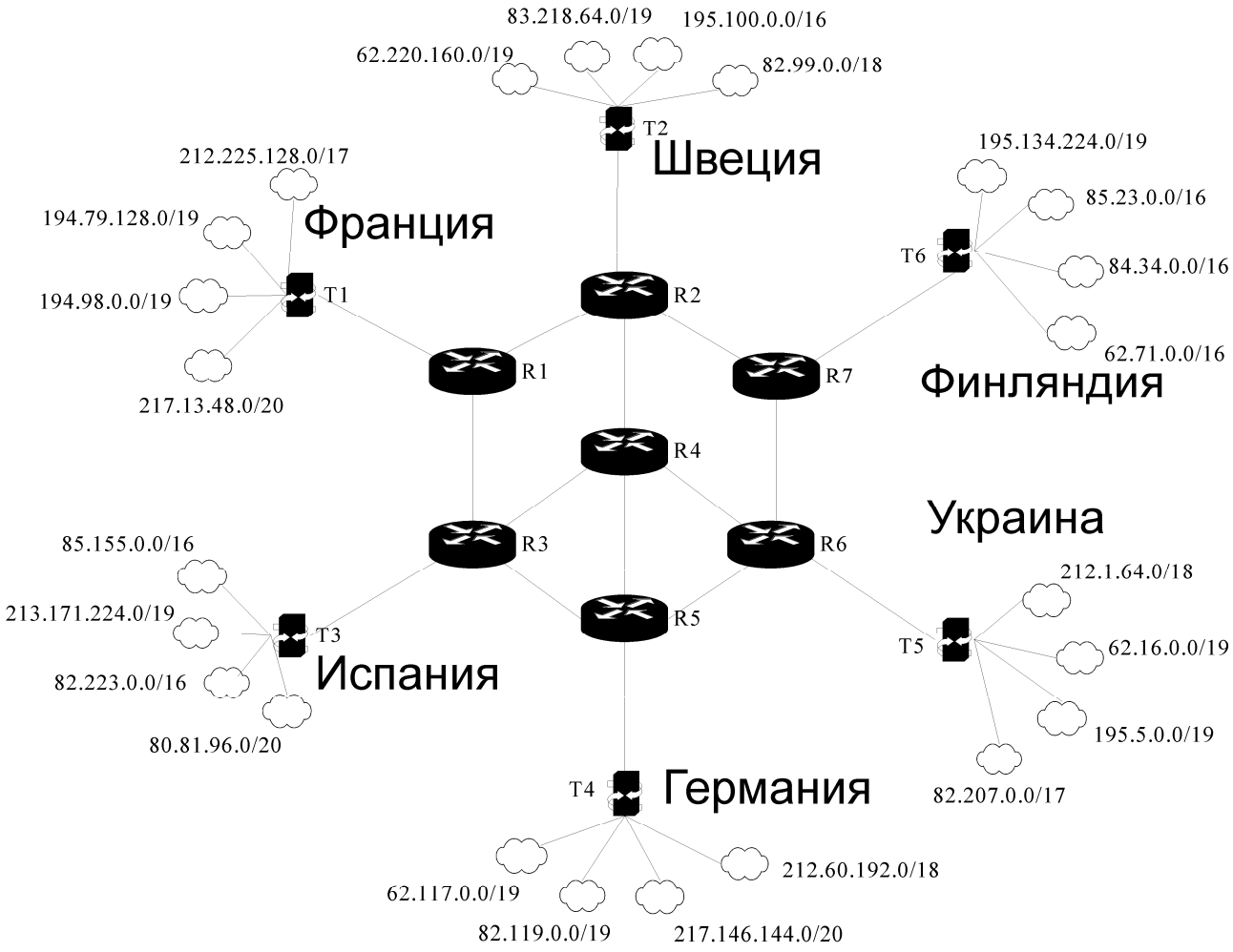


Рис. 5.14. Схема фрагмента европейской магистрали Интернет

Для моделирования реалистичного трафика построены специальные модели терминальных подсетей **Terminal1-Terminal6**, описанные в следующем пункте. Переходы **Transmit*** моделируют передачу пакетов в каналах без потери пакетов. Совмещённая позиция **Trafic** служит для измерения трафика и описана в последующих пунктах. Рассмотрим особенности построения моделей IP и MPLS маршрутизации.

Для IP-маршрутизации использованы статические таблицы маршрутизации, содержащие 24 записи для каждого маршрутизатора; заметим, что таблицы каждого из маршрутизаторов отличаются в полях адресов шлюзов и номеров интерфейсов. Пример таблицы маршрутизации **Router1** приведен на рис. 5.10.

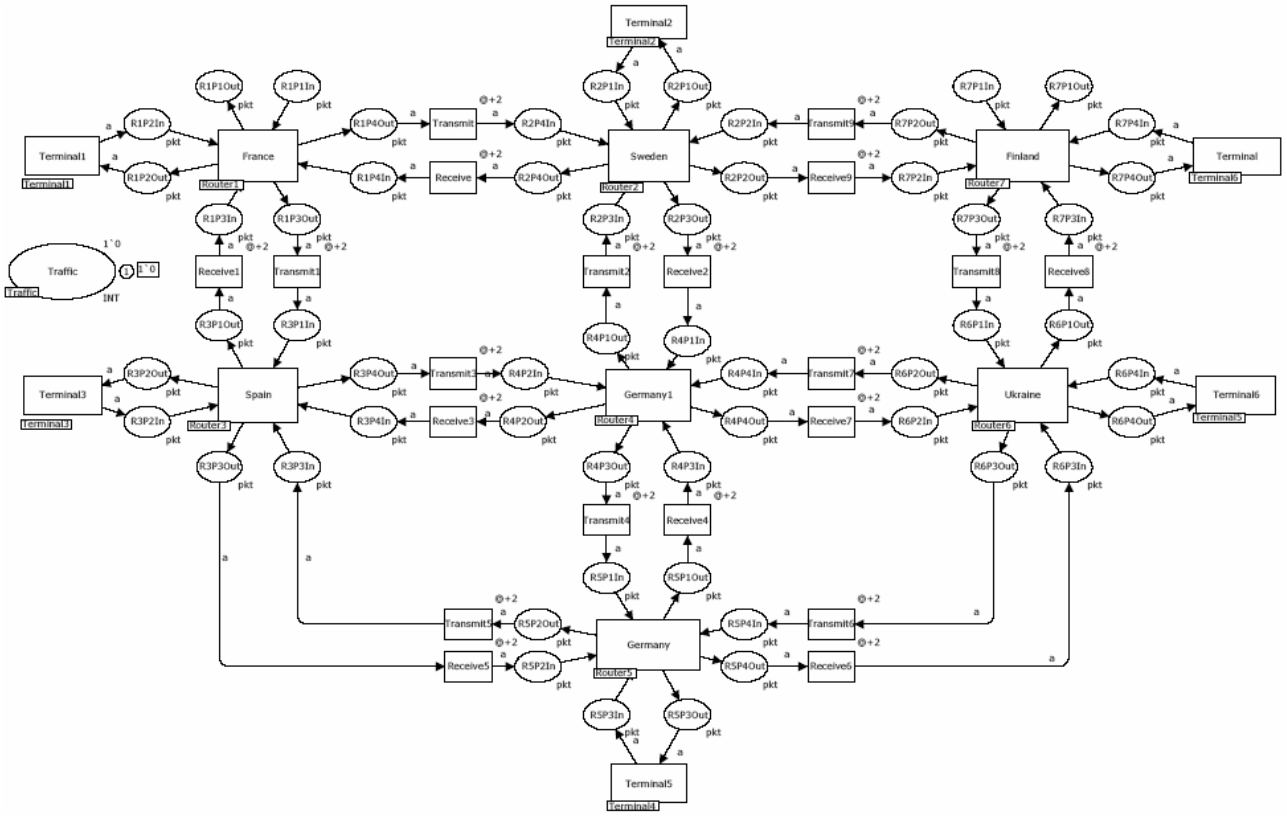


Рис. 5.15. Модель фрагмента Европейской магистрали Интернет

Таблица 5.3 -

Таблица назначения меток

Сети	T1	T2	T3	T4	T5	T6
T1	0	R1(5) -R2	R1(1) -R3	R1(2) - R3(6) -R5	R1(6) - R2(15) - R4(6) -R6	R1(7) - R2(8) -R7
T2	R2(2) -R1	0	R2(2) - R4(4) -R3	R2(3) - R4(10) -R5	R2(1) - R4(9) -R6	R2(3) -R7
T3	R3(3) -R1	R3(1) - R4(9) -R2	0	R3(3) -R5	R3(2) - R4(10) -R6	R3(3) - R4(4) - R2(6) -R7
T4	R5(5) - R3(6) -R1	R5(11) - R4(13) -R2	R5(4) -R3	0	R5(16) -R6	R5(8) - R6(10) -R7
T5	R6(7)- R4(5)- R2(3)- R1	R6(12) - R4(10) -R2	R6(11) – R4(5) -R3	R6(10) -R5	0	R6(2) -R7
T6	R7(5) - R2(4) -R1	R7(4) -R2	R7(7) - R2(8) - R4(6) -R3	R7(8) - R6(5) -R5	R7(9) -R6	0

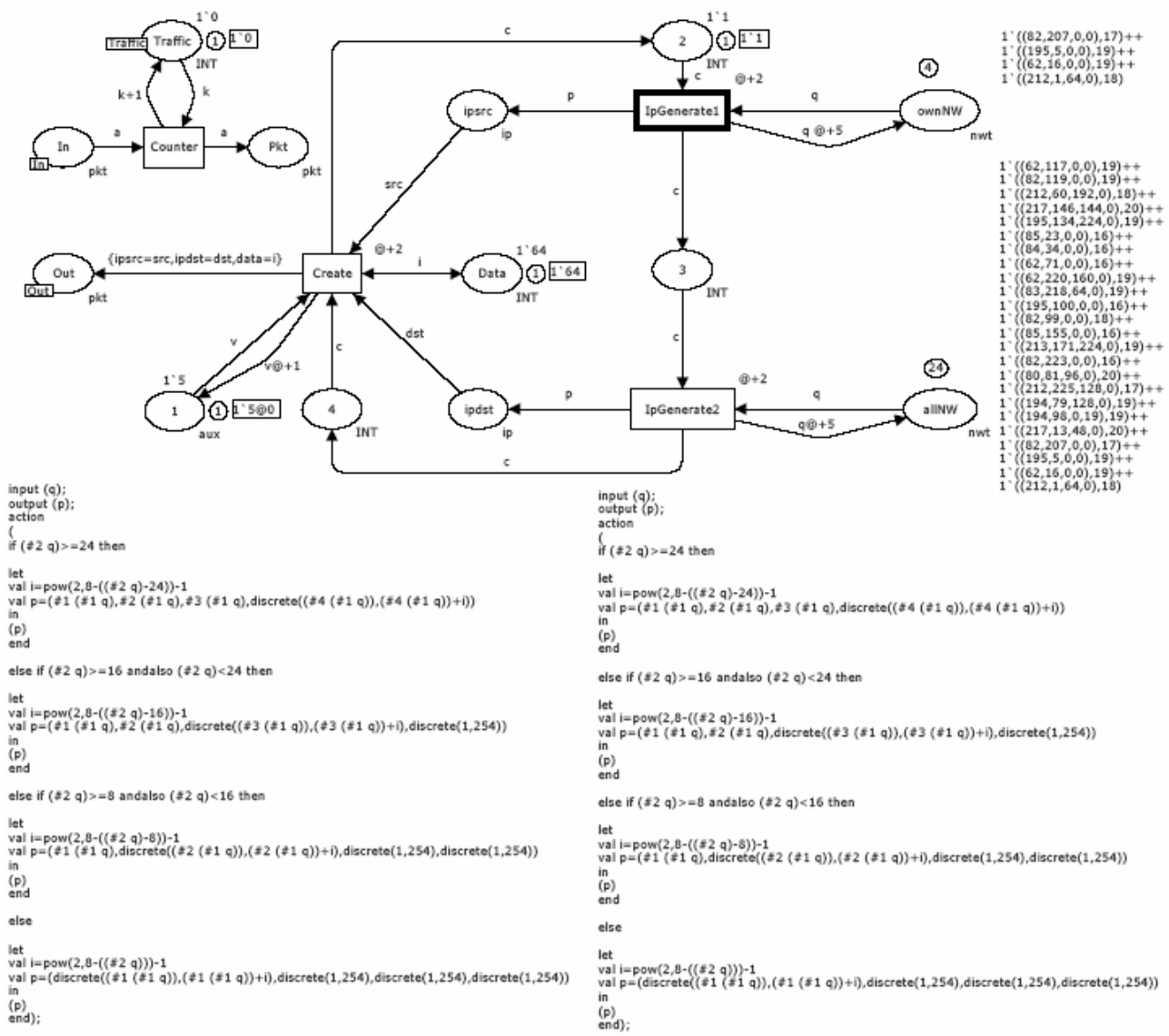
Для MPLS-маршрутизации построены агрегированные маршруты сети, которые могут быть получены как результат работы LDP алгоритма. В табл. 5.3 указаны основные маршруты доставки пакетов для каждой пары терминальных сетей и в скобках приведены значения присвоенных меток. Например, маршрут пакетов, отправленных из сети T1 в сеть T5, определяется последовательностью меток 6-15-6. По табл. 5.3 построена начальная маркировка позиций **Tlebel*** и **RT*** подмоделей маршрутизаторов, представляющих таблицы коммутации меток и таблицы назначения меток соответственно. На рис. 5.12, 5.13 представлены таблицы для маршрутизатора **Router1**.

5.3.5. Модели терминальных сетей

Модель реалистичного трафика является важной составляющей, обеспечивающей общую адекватность построенных моделей реальным процессам. Раскрашенные сети Петри предоставляют широкие возможности описания как потокового трафика с различным видом распределения случайных величин (равномерное, Пуассоновское, Эрланга), так и трафика, отображающего протоколы взаимодействия в системах клиент-сервер [140]. В настоящей работе моделировался потоковый трафик. Терминальная сеть периодически генерирует пакеты со случайными адресами источника и приёмника. Адрес источника находится в диапазоне адресов собственных сетей, а адрес приёмника в диапазоне адресов всех сетей моделируемого фрагмента магистрали Интернет.

Модель терминальной сети **Terminal5** представлена на рис. 5.16. Обработка входных пакетов представлена простым их поглощение с помощью перехода **Counter** в верхней левой части модели; при этом выполняется подсчёт количества доставленных пакетов в совмещённой позиции **Traffic**, используемой для накопления статистической информации. Генерация пакетов основана на использовании пары позиций **ownNW** и **allNW**, содержащих IP-адреса и маски собственных сетей и всех сетей моделируемого фрагмента магистрали соответственно. Адреса выбираются в позиции **ipsrc** и **ipdst**; использование выбранных

адресов блокируется на интервал времени, указанный на исходящих дугах переходов **InGenerate1**, **InGenerate2**.




```
input (q);
output (p);
action
(
if (#2 q)>=24 then

let
val i=pow(2,8-((#2 q)-24))-1
val p=(#1 (#1 q),#2 (#1 q),#3 (#1 q),discrete((#4 (#1 q)),(#4 (#1 q))+i))
in
(p)
end

else if (#2 q)>=16 andalso (#2 q)<24 then

let
val i=pow(2,8-((#2 q)-16))-1
val p=(#1 (#1 q),#2 (#1 q),discrete((#3 (#1 q)),(#3 (#1 q))+i),discrete(1,254))
in
(p)
end

else if (#2 q)>=8 andalso (#2 q)<16 then

let
val i=pow(2,8-((#2 q)-8))-1
val p=(#1 (#1 q),discrete((#2 (#1 q)),(#2 (#1 q))+i),discrete(1,254),discrete(1,254))
in
(p)
end

else

let
val i=pow(2,8-((#2 q)))-1
val p=(discrete((#1 (#1 q)),(#1 (#1 q))+i),discrete(1,254),discrete(1,254),discrete(1,254))
in
(p)
end);
```

```
input (q);
output (p);
action
(
if (#2 q)>=24 then

let
val i=pow(2,8-((#2 q)-24))-1
val p=(#1 (#1 q),#2 (#1 q),#3 (#1 q),discrete((#4 (#1 q)),(#4 (#1 q))+i))
in
(p)
end

else if (#2 q)>=16 andalso (#2 q)<24 then

let
val i=pow(2,8-((#2 q)-16))-1
val p=(#1 (#1 q),#2 (#1 q),discrete((#3 (#1 q)),(#3 (#1 q))+i),discrete(1,254))
in
(p)
end

else if (#2 q)>=8 andalso (#2 q)<16 then

let
val i=pow(2,8-((#2 q)-8))-1
val p=(#1 (#1 q),discrete((#2 (#1 q)),(#2 (#1 q))+i),discrete(1,254),discrete(1,254))
in
(p)
end

else

let
val i=pow(2,8-((#2 q)))-1
val p=(discrete((#1 (#1 q)),(#1 (#1 q))+i),discrete(1,254),discrete(1,254),discrete(1,254))
in
(p)
end);
```

Рис. 5.16. Модель терминальной сети

Наиболее сложным действием является генерация случайного IP-адреса по заданным адресу сети и маске. Для этих целей построены процедуры переходов **InGenerate1**, **InGenerate2**, представленные в нижней части рисунка. Циркуляция фишки в последовательности позиций **2**, **3**, **4** обеспечивает циклическое повторение действий. Позиция **1** служит для периодического запуска генерации пакетов; при этом временная задержка на входящей дуге позиции **1** задаёт периодичность генерации. Позиция **Data** моделирует содержимое пакетов.

Собственно формирование выходного пакета выполняется переходом **Create** путём объединения адресов источника и приёмника, а также данных.

5.3.6. Сравнительная оценка IP-маршрутизации и MPLS

Для отладки модели использован пошаговый режим имитации CPN Tools [6]; при этом выполнялась трассировка прохождения отдельных пакетов через сеть для выбранных пар терминальных сетей. Для анализа эффективности MPLS выполнялось скоростное моделирование на длительных интервалах времени; измерялся трафик в условиях пиковой нагрузки, обеспечиваемой терминальными сетями. Совмещённая позиция измерения трафика **Traffic** представлена в подмоделях терминальных сетей (рис. 5.16), а также на главной странице модели (рис. 5.15) для обеспечения удобства измерений.

Вопросы масштабирования времён изучены в [93, 140]. В настоящей работе выбрана единица модельного времени, равная 0,65 нс, обеспечивающая моделирование высокопроизводительных маршрутизаторов; время коммутации меток выбрано как 50% времени маршрутизации, что соответствует более чем двукратному уменьшению размера таблиц. Исследовано три основных типа магистральных маршрутизаторов, характеристики которых представлены в табл. 5.4.

Таблица 5.4 -

Характеристики оборудования

Маршрутизатор	Производительность (пакетов/с)	Время обработки пакета, нс	Модельное время обработки пакета
Cisco 12000	25 000 000	40	60
Juniper T320	385 000 000	2,6	4
Juniper T640	770 000 000	1,3	2

По результатам измерения трафика для указанных интервалов модельного времени получен график сравнительной эффективности IP и MPLS технологии, представленный на рис. 5.17. Кроме того, при измерении трафика на длительных интервалах времени построена диаграмма производительности сети для различных магистральных маршрутизаторов, изображённая на рис. 5.18.

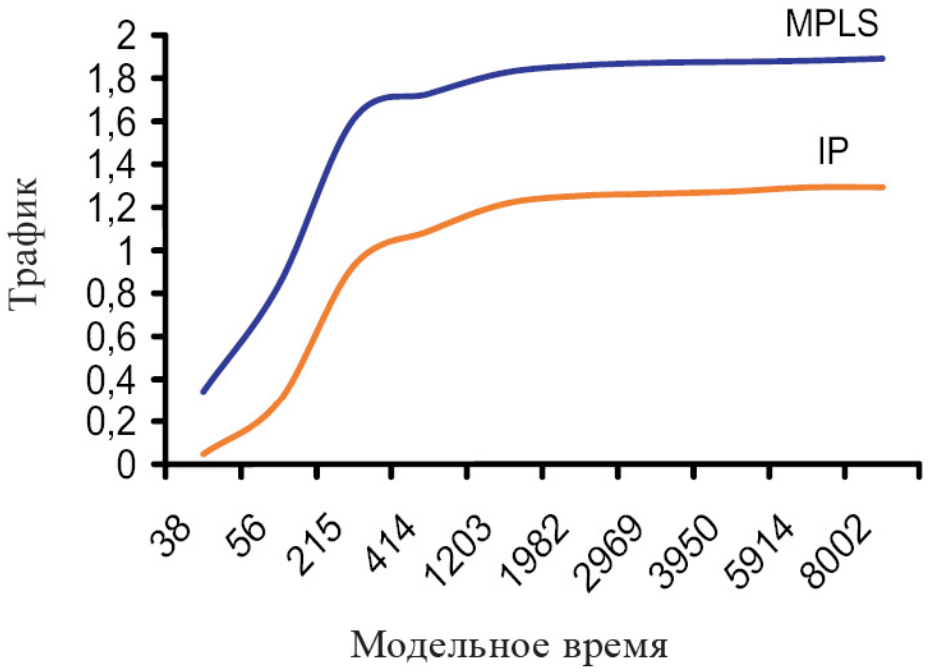


Рис. 5.17. Сравнительная оценка эффективности IP и MPLS технологий

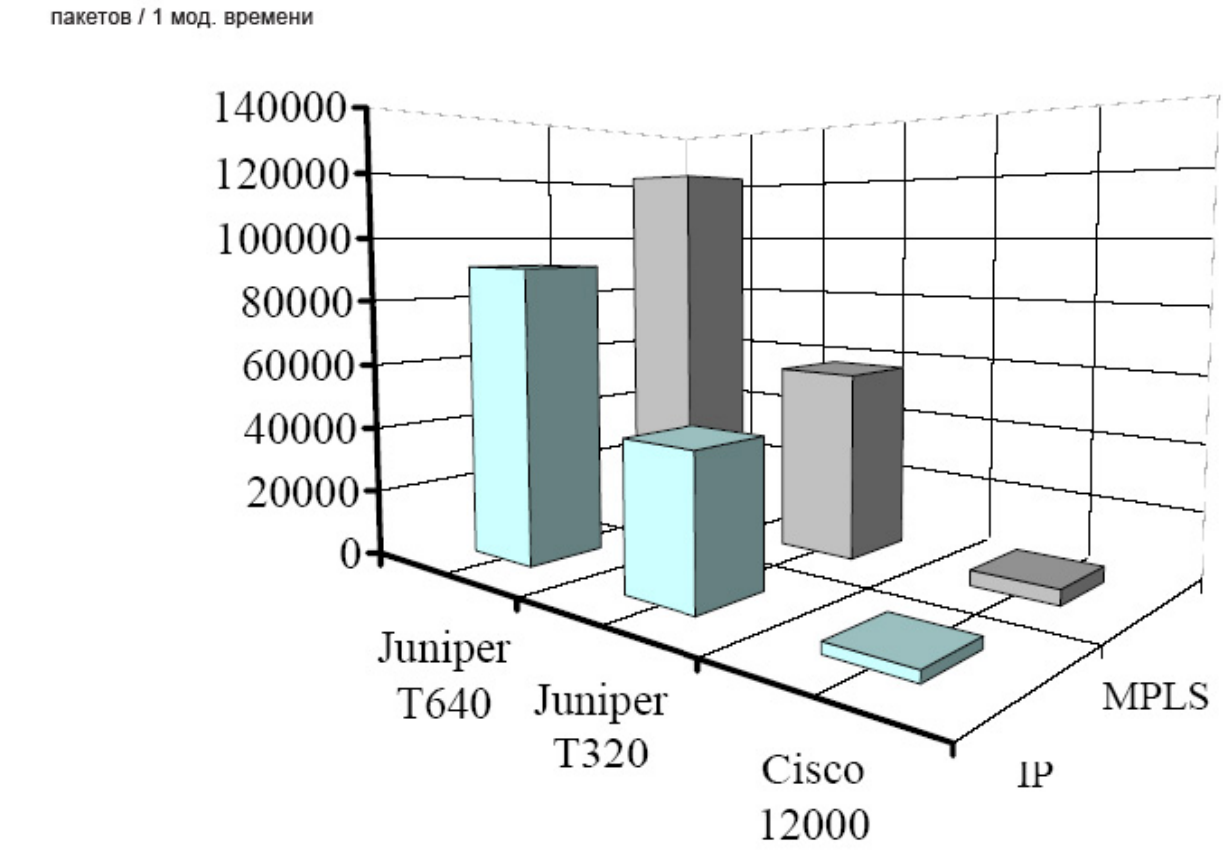


Рис. 5.18. Диаграмма производительности сети
для различных маршрутизаторов

Анализ результатов моделирования позволяет сделать вывод, что технология MPLS в среднем в 1,7 раз эффективнее классической IP-маршрутизации. Следует отметить, что результаты получены для сравнительно небольшого фрагмента сети с размером таблиц маршрутизации равным 24. Для магистралей Интернет, где маршрутные таблицы насчитывают тысячи IP-сетей, прогнозируется более существенное повышение производительности.

Таким образом, в настоящем подразделе построены модели IP и MPLS маршрутизаторов в форме раскрашенных сетей Петри в среде моделирующей системы CPN Tools. Выполнена оценка эффективности технологии коммутации меток MPLS на примере модели фрагмента европейской магистрали Интернет. Показано, что даже для сравнительно небольших сетей, насчитывающих несколько десятков узлов, получено повышение производительности сети в среднем в 1,7 раз. Построенные типовые модели маршрутизаторов могут быть применены для моделирования произвольных MPLS и IP сетей по заданной структурной схеме, характеристикам оборудования и программного обеспечения.

5.4. Модели сетей Bluetooth

На начальной стадии своего развития протокол Bluetooth [13, 53] ввиду высокой стоимости реализующих его устройств имел в основном военное и специальное применение как один из основных протоколов сетей мобильных сенсорных устройств. Он использовался для сбора информации от автономно работающих датчиков, распределённых в радиусе до одного километра, и применялся для визуального наблюдения, прослушивания, радиационного мониторинга. В [105] отмечается применение протокола Bluetooth для создания панорам возможных районов проведения военных операций.

В настоящее время в связи с удешевлением Bluetooth устройств протокол находит широкое офисное применение для создания беспроводных пикосетей. На рынке представлен широкий выбор периферийного оборудования компью-

теров, поддерживающих обмен информацией по протоколу Bluetooth: принтеры, клавиатура, манипуляторы "мышь", аккустические системы. Достаточно распространённой является Bluetooth гарнитура мобильных телефонов "свободные руки".

Так как построение аналитических моделей [63] Bluetooth сетей затруднено ввиду сравнительно высокой сложности технологии, имитационное моделирование является перспективным направлением исследований. Известно применение специализированной моделирующей системы NS для исследования режимов энергосбережения Bluetooth [105]. Однако применение специализированных систем моделирования имеет ряд недостатков, основной из которых состоит в сложности интеграции моделей при исследовании гетерогенных сетей. Раскрашенные сети Петри [50] моделирующей системы CPN Tools [6] являются универсальной алгоритмической системой и позволяют моделировать телекоммуникационные устройства и сети [128, 146]. Предложенный ранее метод измерительных фрагментов [95, 140] обеспечивает измерение нетривиальных характеристик моделируемого объекта в процессе имитации динамики сети Петри.

Целью настоящего подраздела является построение типовых моделей ведущего и ведомого Bluetooth устройств в форме раскрашенных сетей Петри, а также оценка эффективности использования адресного пространства протокола.

5.4.1. Обзор технологии Bluetooth

В настоящее время объёмы продаж Bluetooth оборудования значительно превышают объемы продаж оборудования IEEE 802.11 – известного стандарта беспроводных локальных сетей WLAN (Wireless Local Area Network). Организация IEEE создала специальную группу для разработки стандарта 802.15, во многом основанного на спецификациях Bluetooth, и названного стандартом беспроводных сетей персонального использования WPAN (Wireless Personal Area Networks), также именуемых пиконет.

Разработку и сопровождение стандартов Bluetooth выполняет специальная группа интересов SIG (Special Interest Group), образованная в мае 1998 года такими фирмами, как Intel, 3COM, Ericsson, IBM, Motorola, Nokia, Toshiba. Основные спецификации протокола представлены в документах [13]. Архитектура протокола [53] изображена на рис. 5.19.

Интерфейс протокола с эфиром представлен уровнем радио частот RF (Radio Frequency). Номинальная мощность антенны находится в диапазоне 1-100 mW, что обеспечивает радиус действия 10-100 метров. Протокол использует прыгающие частоты в нелицензируемом диапазоне от 2.402 до 2.480 GHz. Всего предусмотрено 79 уровней рабочих частот (каналов) в указанном диапазоне. Уровень радиопередачи Baseband предусматривает случайную последовательность каналов для обеспечения передачи информации и процедуру согласования последовательности. Случайная последовательность смены каналов обеспечивает совместную работу нескольких пикосетей в одном и том же диапазоне частот. Каждая пикосеть образуется одним ведущим устройством (Master) и несколькими ведомыми устройствами (Slave). Пакет передаваемой информации составляется из нескольких слотов (1, 3 или 5); передача слота занимает 625 us, каждый из слотов может передаваться по собственному каналу. Стандартная частота смены каналов 1600 каналов в секунду, что обеспечивает скорости обмена информацией до 721 Kbits/s. Каждое устройство имеет уникальный 48 битовый адрес, совместимый с IEEE 802. Для создания согласованного шаблона смены частот используется глобальный идентификатор пикосети Global ID, который ведущее устройство сообщает всем ведомым. Диаграмма состояний устройств пикосети, предусмотренная LMP (Link Management Protocol), представлена на рис. 5.20.

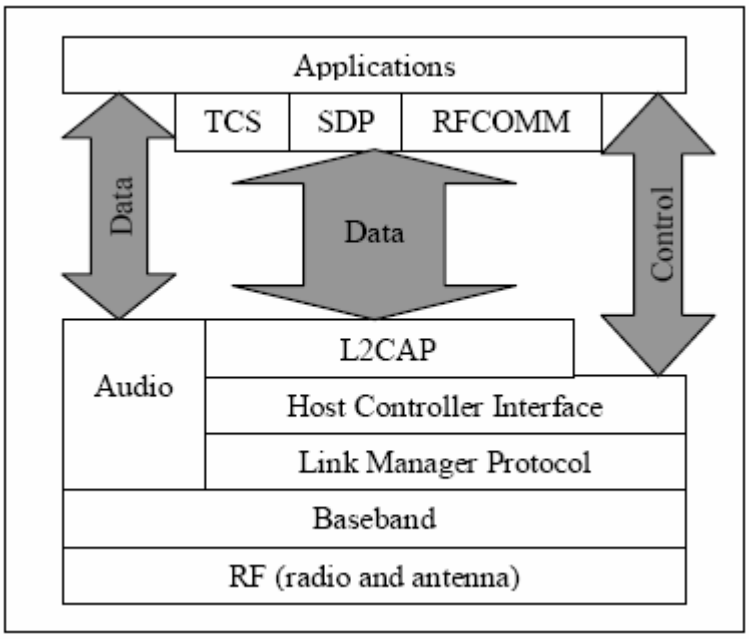


Рис. 5.19. Архитектура протокола Bluetooth

Устройство, не присоединённое к пикосети, находится в состоянии Standby. В этом состоянии устройство слушает анонсы существующих пикосетей Inquiry либо запросы на присоединение к пикосети Page. Для подключения к пикосети устройство посылает пакет Page с указанием Global ID. После подключения к пикосети устройству выделяется трёхбитовый активный адрес AMA (Active Member Address), который используется при передаче данных. Таким образом, в одной пикосети может быть до восьми одновременно активных устройств. При необходимости обмена с новыми устройствами ведущее устройство посылает пакет Park одному из ведомых устройств, принуждая его вернуть свой AMA в пул и назначает ему восьмибитовый пассивный адрес PMA (Passive Member Address). Таким образом, пиконет может содержать до 256 подключенных устройств. Предусмотрено три энергосберегающих состояния: Park, Sniff, Hold. В состоянии Hold устройство не освобождает AMA, в состоянии Sniff устройство может передавать данные через заданный интервал времени. Контроллеры Bluetooth должны поддерживать стандартный интерфейс Host Controller Interface.

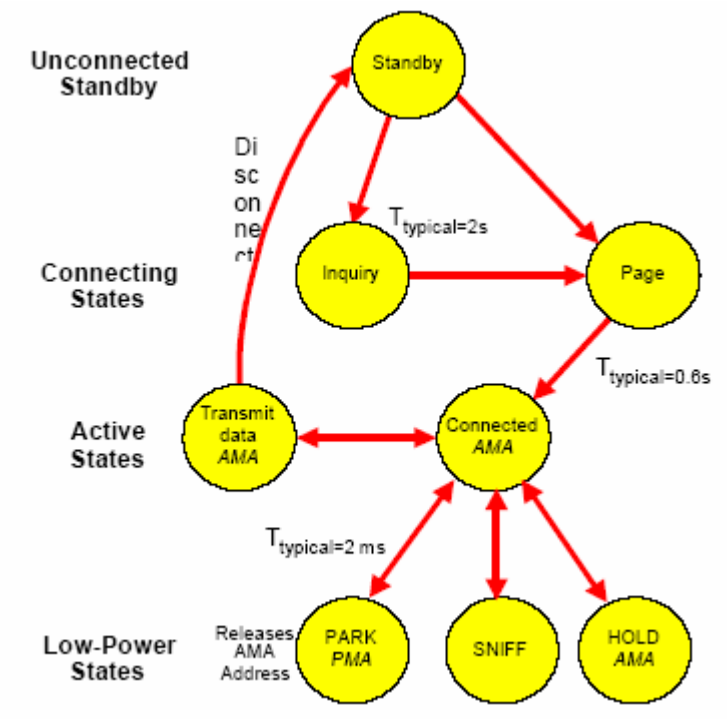


Рис. 5.20. Диаграмма состояний устройств пикосети

В соединённом состоянии Connected в соответствии с L2CAP (Logical Link Control and Adaptation Protocol) предусматривает два вида пакетов передачи данных: синхронные SCO (Synchronous Connection Oriented) и асинхронные ACL (Asynchronous Connectionless). SCO пакеты ориентированы на установление соединений и используется для передачи потоковых данных, например, голосовых; ACL пакеты применяются для передачи коротких сообщений. Для стыковки с протоколами прикладного уровня используется ряд вспомогательных протоколов, например, протокол обнаружения сервисов SDP (Service Discovery Protocol), протокол эмуляции последовательного кабеля RFCOMM и другие.

5.4.2. Модель ведомого устройства

Раскрашенная сеть Петри моделирующей сети CPN Tools [6] представляет собой комбинацию графа сети Петри [169] и языка программирования CPN ML [50], используемого для описания атрибутов элементов сети. Сети Петри, в которых фишка не является элементарной, а может иметь индивидуальные ха-

рактеристики, традиционно называют раскрашенными. В CPN Tools фишка является объектом абстрактного типа данных. Кроме того, в качестве атрибутов переходов и дуг сети указывают функции, проверяющие либо преобразующие характеристики фишек.

Модель ведомого Bluetooth устройства представлена на рис. 5.21. Существенным для понимания организации модели является описание используемых типов, переменных и функций, представленное на рис. 5.22. Основным типом (colset) фишек является **pkt**, моделирующий пакеты протокола, передаваемые между ведомым и ведущим устройствами. Заметим, что в модели представлены только заголовки пакетов, содержащие AMA отправителя **srcama**, AMA получателя **dstama**, тип сообщения **mtype** и для некоторых типов сообщений PMA ведомого устройства **pma**. Адреса и тип пакета моделируются целыми числами. Представлены следующие типы пакетов: выделение AMA (mtype=1), освобождение AMA (mtype=2), передача данных (mtype=3). Тип **rf** моделирует передачу пакетов в эфире и обеспечивает управление порядком передачи таким образом, что эфир может быть доступен для передачи ведущим устройством **availmaster**, доступен для передачи ведомым устройством **availslave**, либо занят передачей пакета **r**.

Рассмотрим организацию модели. Контактная позиция **myPMA** хранит PMA ведомого устройства. Позиция **myAMA** хранит AMA ведомого устройства; начальная маркировка этой позиции имеет значение 8, моделирующее отсутствие назначенного AMA. Контактная позиция **Air** моделирует эфир, в котором передаются пакеты. Переход **AlloAMA** посылает запрос на выделение AMA для обмена информацией; запрос отправляется только при наличии пакетов данных в выходной очереди **outbufs**. Переход **FreeAMA** посылает запрос на освобождение AMA по завершению обмена информацией, когда выходная очередь **outbufs** пуста. Переход **send** моделирует передачу пакетов данных при наличии у ведомого устройства AMA.

Рассмотрим более подробно процесс генерации пакетов данных, представленный в левой части модели. Периодичность генерации данных задаёт по-

зиция **clock**, инициирующая создание пакетов данных, количество которых задано случайной функцией **NSend()** имеющей равномерное распределение, через случайный интервал времени, заданный функцией **Delay()**. Сгенерированные пакеты поступают в промежуточную позицию **data**, из которой помещаются в выходную очередь пакетов, представленную позицией **outbufs**, списочного типа **lpkt**. Отправка данных в эфир моделируется переходом **send** и выполняется только при наличии у ведомого устройства AMA. Заметим, что переход **send** слушает эфир и осуществляет передачу только при наличии разрешающей метки **availslave**.

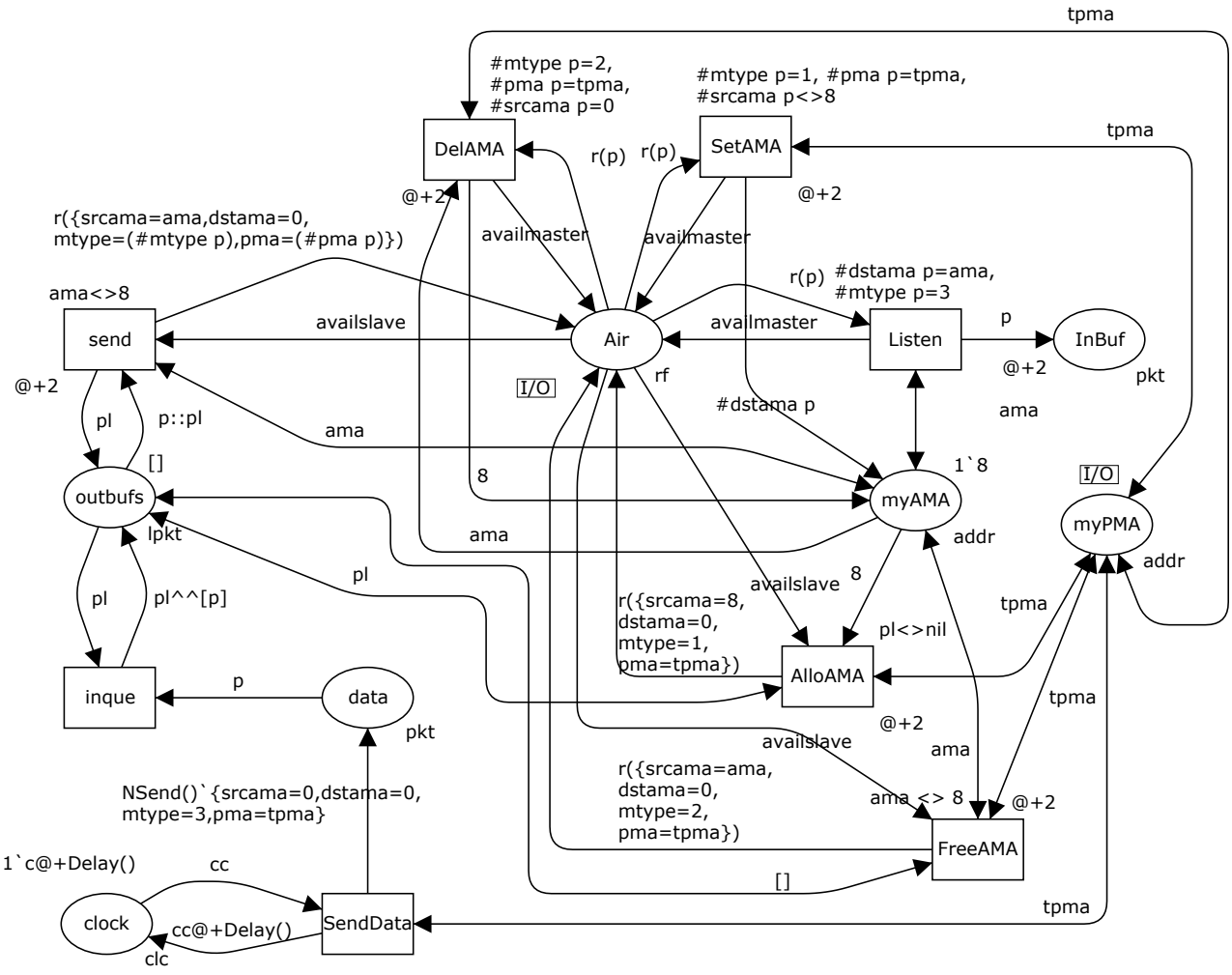


Рис. 5.21. Модель ведомого устройства (slave)

Переход **SetAMA** выполняет назначение АМА ведомого устройства при его получении от ведущего устройства; проверка включает распознавание типа сообщения (#mtype p=1), собственного РМА (#pma p=tpma) и отсутствие назначенного АМА (#srcama p<>8). Переход **DelAMA** моделирует освобождение АМА при получении соответствующего пакета(#mtype p=2). Заметим, что выполняется проверка АМА ведущего устройства (#srcama p=0) и запись специального значения 8, моделирующего отсутствие АМА, в позицию **myAMA**. Переход **Listen** моделирует получение пакетов данных и выполняет их размещение в буфере **InBuf**. Полученный пакет замещается меткой **availmaster**, обеспечивающей преимущественное использование эфира ведущим устройством.

```
colset addr=INT; colset ptype=INT;
colset pkt = record srcama:addr*dstama:addr*mtype:ptype*pma:addr;
colset rf=union r:pkt+availmaster+availslave timed;
colset lpkt=list pkt; colset clc=unit with c timed;
colset amatab=record ama:addr*pma:addr*t:INT; colset lamatab=list amatab;
colset npkt=int with 10..20; fun NSend()=npkt.ran();
colset Interval=int with 500..1000; fun Delay()=Interval.ran();
fun eqpma v (rr:amatab)=((#pma rr)=v);
fun eqt v (rr:amatab)=((#t rr)=v);
fun grec prd [] = zero | grec prd (y::z) = if prd(y) then y else grec prd z;
fun prec prd [] pm ti= [] | prec prd (y::z) pm ti=
        if prd(y) then ({ama=(#ama y),pma=pm,t=ti}::z) else y::(prec prd z pm ti);
fun cT()=IntInf.toInt(!CPN'Time.model_time);
fun mint [] = (cT()) | mint ((y::z):lamatab) = Int.min((#t y),(mint z));
var p:pkt; var pl:lpkt; var x:rf; var ama,pma,tpma:addr; var a:amatab; var al:lamatab;
var i:INT; var cc:clc; val zero={ama=8,pma=8,t=8};
```

Рис. 5.22. Описания типов, переменных и функций

Модель ведомого устройства содержит две контактных позиции **Air** и **myPMA**, помеченные тэгом I/O, используемые далее при компоновке моделей пикосетей.

5.4.3. Модель ведущего устройства

Модель ведущего устройства изображена на рис. 5.23. Ключевым элементом модели является таблица распределения АМА, представленная позицией **TabAMA**. Тип записи таблицы **amatab** сопоставляет каждому допустимому значению АМА ведомого устройства его РМА; свободные АМА помечены специальным значением РМА, равным нулю. Кроме того, запись таблицы хранит время последнего выделения АМА для принятия решения о принудительном освобождении АМА при нехватке адресов. Позиция **TabAMA** имеет списочный тип **lamatab**, обеспечивающий дальнейший выбор записи с помощью рекурсивных функций. Переход **Listen** слушает эфир и отбирает пакеты, отправляемые ведущему устройству (#dstama p=0). Полученные пакеты размещаются во временном буфере **InBuf**, а затем обрабатываются переходами **t1**, **t2**, **t3** в соответствии с типом полученного пакета.

При получении запроса на выделение АМА (#mtype p=1) при наличии свободной записи (ama=0) в таблице **TabAMA** выполняется назначение адреса с помощью перехода **AllocAMA**. Для проверки наличия свободного АМА используется функция **eqpma**, проверяющая наличие в списке записи, с **pma**, равным указанному. Изменение записи в таблице обеспечивает рекурсивная функция **prec**, корректирующая первую запись с указанными свойствами (pma=0); при этом сохраняется **pma** запросившего устройства и текущее значение модельного времени, полученное с помощью функции **cT()**. Кроме того, переход **AllocAMA** формирует пакет ответа ведомому устройству с указанием назначенного АМА. Пакет размещается в выходной очереди ведущего устройства, представленной позицией **outbuf** списочного типа **lpkt**. Рекурсивная функция **grec** обеспечивает поиск первой записи с указанными свойствами (pma=0).

При отсутствии свободного АМА в таблице **TabAMA** срабатывает переход **SwapAMA**. Переход обеспечивает принудительное освобождение наиболее длительно используемого АМА и его назначение запросившему ведомому устройству. Рекурсивная функция **mint** находит запись с минимальным временем назначения АМА. Функция **eqt** находит запись с указанным значением време-

ни, для её корректировки с помощью функции **prec**. Заметим, что в выходную очередь записывается два пакета: первый обеспечивает освобождение AMA, а второй его назначение новому ведомому устройству.

При получении запроса на освобождение AMA (#mtype p=2) срабатывает переход **RlsAMA**; **pma** соответствующей записи в таблице **TabAMA** обнуляется и формируется пакет подтверждения освобождения адреса. Обработка пакетов данных (#mtype p=3) представлена простым подсчётом их общего количества с помощью позиции **C2**.

Рассмотрим более подробно процедуру управления эфиром. Ведущее устройство осуществляет передачу пакетов из выходной очереди **outbuf** в эфир с помощью перехода **send**. Ведущее устройство выполняет передачу при наличии любой из меток доступности эфира: **availmaster**, **availslave**. Ведомое устройство, как было описано ранее, после получения пакета выставляет метку **availmaster**. Таким образом, ведущее устройство имеет преимущественное право на передачу пакетов. Метка **availslave**, разрешающая передачу пакета ведомым устройством, формируется переходом **check** только в том случае, если выходная очередь ведущего устройства **outbuf** пуста.

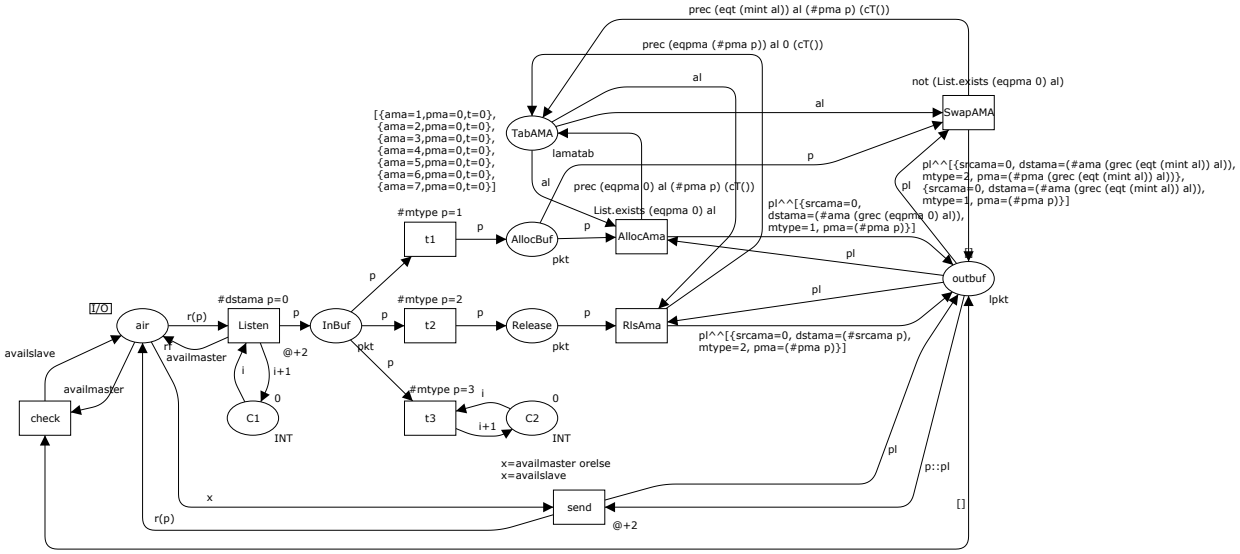


Рис. 5.23. Модель ведущего устройства (Master)

Пара позиций **C1** и **C2** используется для накопления статистической информации и выполняет подсчёт общего количества полученных пакетов и пакетов данных соответственно. Контактная позиция **Air** служит для обеспечения дальнейшей компоновки моделей пикосетей.

5.4.4. Модели пикосетей

Предложено собирать модели пикосетей из ранее построенных подмоделей ведущего (Master) и ведомых (slave) устройств для заданного конкретного количества ведомых устройств и их адресов PMA. На рис. 5.24 представлен пример модели пикосети с восьмью ведомыми устройствами. Моделирующая система CPN Tools предоставляет средства построения иерархических моделей путём подстановки переходов. Тэг с наименованием подмодели указывается возле соответствующего перехода. При подстановке перехода выполняется слияние (объединение) соответствующих контактных позиций.

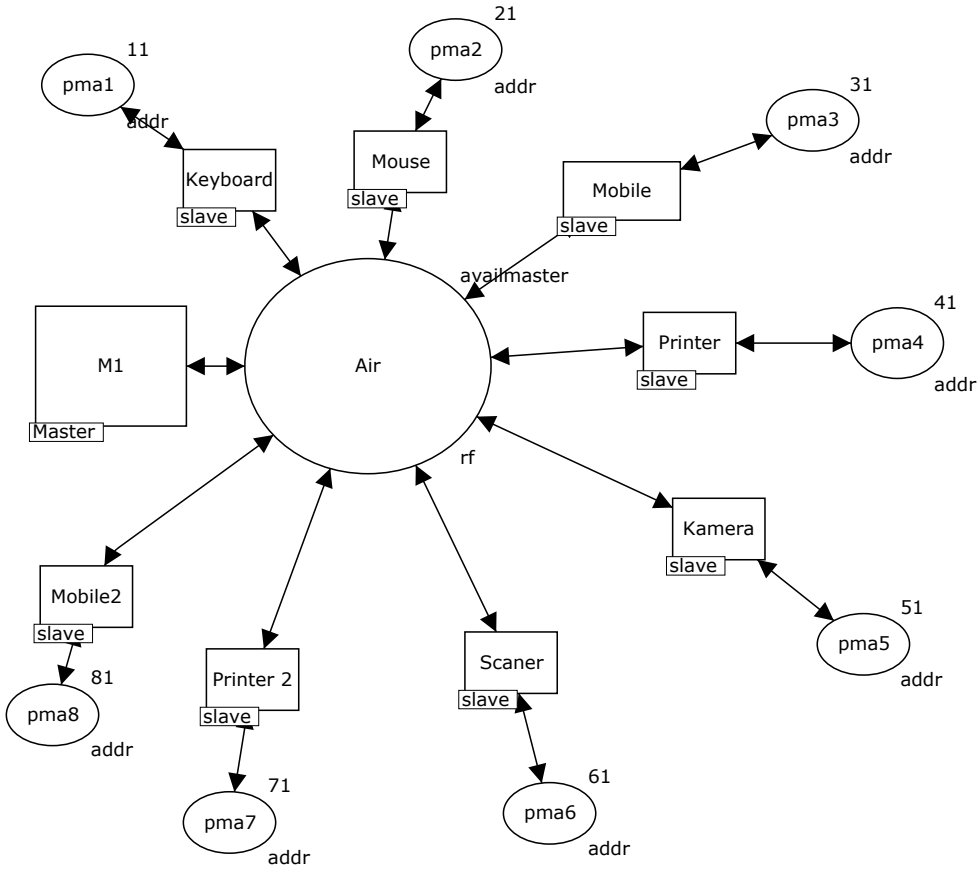


Рис. 5.24. Модель пикосети

Контактная позиция **myPMA** подмоделей ведомых устройств отображается в позиции **pma1-pma8**, задающие конкретные адреса устройств пикосети. Для отладки модели использован режим пошаговой имитации динамики сети Петри моделирующей системы CPN Tools. Выполнена трассировка прохождения всех типов пакетов между парами взаимодействующих устройств. Примеры представления процессов пошаговой имитации для моделей Ethernet рассмотрены в [135-137].

5.4.5. Оценка эффективности использования адресного пространства

Заметим, что, так как целью настоящей работы являлась оценка эффективности использования адресного пространства протокола Bluetooth, применён класс невременных сетей Петри. Применение временных сетей моделирующей системы CPN Tools и измерительных фрагментов для оценки времени отклика в Ethernet описано в [95, 140].

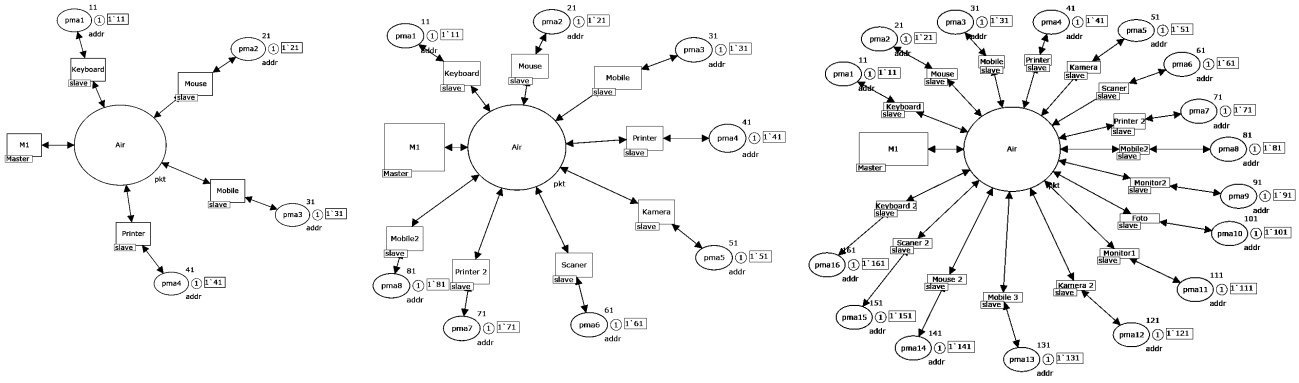


Рис. 5.25. Модели пикосетей с различным количеством ведомых устройств (4,8,16)

Напомним, что статистическая информация накапливается в позициях **C1**, **C2** подмодели ведущего устройства (Master). В качестве основной характеристики эффективности использования адресного пространства оценивалась доля полезной информации, передаваемой в сети, представленная величиной E=C2/C1 и названная коэффициентом эффективности обмена информацией.

Для измерения среднего значения коэффициента выполнялось не менее двадцати имитационных экспериментов с моделью (рис. 5.25) с количеством шагов, обеспечивающим достижение стационарного режима [135-137]. Тенденция снижения коэффициента эффективности с ростом числа ведомых устройств хорошо прослеживается на графике, представленном на рис. 5.26.

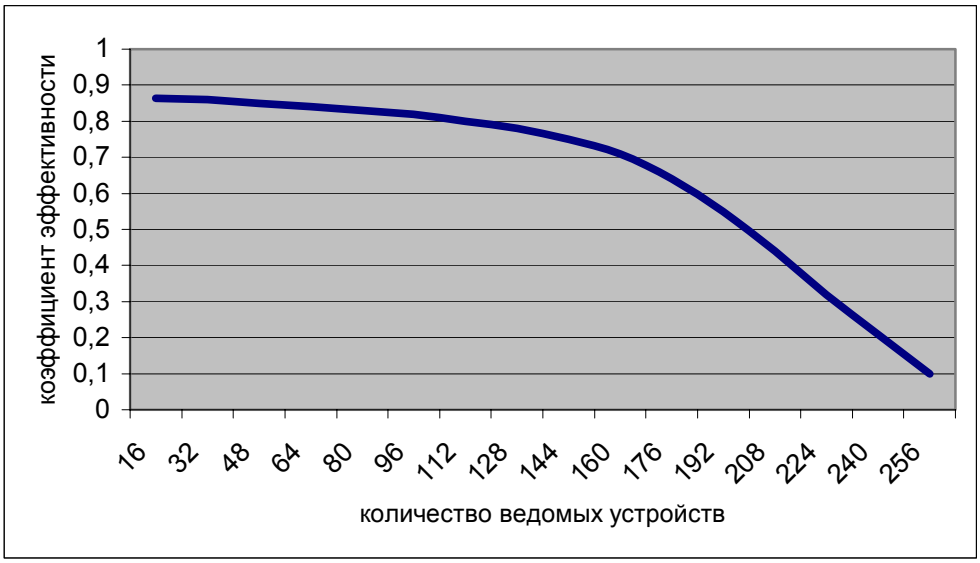


Рис. 5.26. График зависимости коэффициента эффективности от количества ведомых устройств

Полученные результаты позволяют сделать вывод, что при росте количества подключенных устройств наблюдается общее снижение эффективности информационного обмена и практическое его блокирование при количестве ведомых устройств более 200. Наблюдаемый эффект (по аналогии с терминологией операционных систем) назван пробуксовкой обмена информацией.

Таким образом, в настоящем подразделе построены типовые модели ведущего и ведомого Bluetooth устройств, предназначенные для оценки эффективности использования адресного пространства протокола. На серии моделей пикосетей для различного числа ведомых устройств выполнена оценка эффективности использования адресного пространства. Наблюдается общая тенденция снижения эффективности при росте количества ведомых устройств и про-

буксовка обмена информацией при количестве ведомых устройств близком к максимальному. Общий подход к построению моделей может быть использован при исследовании иных аспектов протокола Bluetooth и других протоколов.

5.5. Измерение характеристик реальных телекоммуникационных сетей

Адекватность построенных моделей реальным объектам является основным вопросом исследования. Для подтверждения адекватности моделей были выполнены натурные измерения функциональных характеристик телекоммуникационных сетей.

Как правило, современное оборудование телекоммуникационных сетей, такое как коммутаторы и маршрутизаторы обладает широким спектром возможностей для измерения трафика. Маршрутизаторы и коммутаторы фирмы CISCO накапливают статистическую информацию о количестве переданных, принятых пакетов и ошибках для каждого порта. Запуск сбора статистической информации может быть выполнен, например, командой IOS:

```
rmon collection stats
```

а просмотр собранной статистической информации командами:

```
show controllers ethernet-controller
show interface interface-name
```

Однако, статистическая информация, аккумулируемая оборудованием, связана с потоковым представлением трафика и не отображает особенности взаимодействия в системах клиент-сервер. Время отклика сети, которое иссле-

довалось для коммутируемой Ethernet в подразделах 5.1, 5.2 требует применения более сложной технологии измерения.

Измерение времени отклика в среде реальной сети диспетчерского центра железной дороги выполнено с помощью анализатора пакетов WinDump, который представляет собой MS Windows версию известного анализатора трафика TCPDump операционной системы Unix. Результаты измерений были также подтверждены с помощью программного обеспечения SoftPerfect Network Protocol Analyzer.

WinDump представляет собой программное обеспечение с интерфейсом командной строки, которое обеспечивает запись передаваемых фреймов Ethernet, дополненных временными штампами, в файл. Затем содержимое файла можно просматривать и анализировать. Следующая командная строка обеспечивает запись фреймов в файл SavedFrames:

```
WinDump -w SavedFrames
```

Для анализа процессов передачи фреймов и вычисления времени отклика была использована следующая командная строка:

```
WinDump -ttt -r SavedFrames
```

Опция –ttt использован для автоматического вычисления временного интервала между поступлением фреймов; опция –r обеспечивает чтение ранее сохранённой информации из файла SavedFrames. Пример полученного листинга представлен на рис. 5.27.

```
000252 IP 192.168.0.158.1172 > 192.168.0.130.139: P 854:917(63) ack 840 win 64957
000854 IP 192.168.0.130.139 > 192.168.0.158.1172: . 840:2300(1460) ack 917 win 64502
000141 IP 192.168.0.130.139 > 192.168.0.158.1172: . 2300:3760(1460) ack 917 win 64502
000029 IP 192.168.0.158.1172 > 192.168.0.130.139: . ack 3760 win 65535
000107 IP 192.168.0.130.139 > 192.168.0.158.1172: . 3760:5220(1460) ack 917 win 64502
000138 IP 192.168.0.130.139 > 192.168.0.158.1172: . 5220:6680(1460) ack 917 win 64502
000024 IP 192.168.0.158.1172 > 192.168.0.130.139: . ack 6680 win 65535
000114 IP 192.168.0.130.139 > 192.168.0.158.1172: . 6680:8140(1460) ack 917 win 64502
000086 IP 192.168.0.130.139 > 192.168.0.158.1172: P 8140:9095(955) ack 917 win 64502
000287 IP 192.168.0.158.1172 > 192.168.0.130.139: . ack 9095 win 65535
000606 IP 192.168.0.158.1172 > 192.168.0.130.139: P 917:980(63) ack 9095 win 65535
000729 IP 192.168.0.130.139 > 192.168.0.158.1172: . 9095:10555(1460) ack 980 win 64439
```

Рис. 5.27. Листинг фреймов

Рассмотрим информацию листинга фреймов. Первая колонка содержит интервалы между временами поступления фреймов в миллисекундах, затем следуют IP-адрес и номера портов отправителя и получателя. Поле двоеточия указаны поля заголовка пакета такие как начальный и конечный номера передаваемых байтов, длина пакета в скобках, номер подтверждённого байта и длина окна. В рассматриваемом примере 192.168.0.158 является IP-адресом рабочей станции, а 192.168.0.130 IP-адресом сервера. Номер порта 139 соответствует сервису MS NetBIOS TCP, номер порта 1172 является случайно выбранным номером порта клиентского программного обеспечения. Из листинга получаем время отклика для первого запроса 954 мс, для второго – 107 мс, для третьего – 114 мс.

WinDump был запущен на действующей рабочей станции диспетчерского центра железной дороги, оснащённой системой ГИД Урал ВНИИЖТ. Измерялось среднее время отклика среди отдельных запросов, аккумулированных за период работы равный одной рабочей смене (около 12 часов). Полученное среднее время отклика равняется 380 нс. Таким образом, ошибка измерения времени отклика с помощью модели (подразделы 5.1, 5.2) составляет не более 5%, что является достаточно хорошим результатом и подтверждает адекватность построенных моделей.

Выводы к 5 разделу

1. Предложено выполнять построение моделей телекоммуникационных систем путём композиции подмоделей компонентов, представляющих собой функциональные подсети либо более общий класс подсетей с контактными позициями.

2. Впервые предложен метод измерительных фрагментов для измерения нетривиальных функциональных характеристик модели. Раскрашенная сеть Петри, являясь универсальной алгоритмической системой, позволяет предста-

вить процессы измерения и алгоритмы вычисления функциональных характеристик.

3. Построены типовые компоненты для композиции моделей коммутируемой Ethernet: коммутатора, рабочей станции, сервера. Построены модели сети Ethernet диспетчерского центра железной дороги и фрагменты модели для измерения времени отклика сети. Выполнено измерение времени отклика в процессе имитации динамики модели.

4. Построены типовые компоненты для композиции моделей IP и MPLS сетей: IP-маршрутизатора, MPLS-маршрутизатора, терминальной сети. Построены модели фрагмента Европейской магистрали Internet. Выполнена сравнительная оценка эффективности классической маршрутизации и технологии MPLS.

5. Построены типовые компоненты для композиции моделей WPAN на основе технологии Bluetooth: ведущего устройства, ведомого устройства. Построены модели сетей Bluetooth с различным числом ведомых устройств. Выполнена оценка эффективности использования адресного пространства протокола Bluetooth.

6. Выполнены измерения характеристик функционирования реальных телекоммуникационных сетей, которые подтверждают адекватность построенных моделей.

ВЫВОДЫ

Для решения научной проблемы доказательства корректности и оценки эффективности телекоммуникационных систем, в диссертационной работе представлены методы синтеза моделей Петри телекоммуникационных протоколов по стандартным спецификациям, методы композиции моделей Петри телекоммуникационных систем и сетей, методы оценки функциональных характеристик построенных моделей, а также методы композиционного анализа моделей Петри большой размерности. Кроме того, разработаны эффективные алгоритмы композиционного анализа сетей Петри, представлена их программная реализация. Выполнена верификация протоколов ECMA, BGP, TCP, IOTP; построены и исследованы модели сетей Ethernet, MPLS, Bluetooth.

1. Решена задача верификации сложных телекоммуникационных протоколов.

Модели Петри реальных телекоммуникационных протоколов имеют большую размерность. Все известные ранее алгебраические методы анализа сетей Петри обладают экспоненциальной вычислительной сложностью, что делает анализ моделей протоколов телекоммуникационных систем в ряде случаев практически неосуществимым.

Для решения указанной задачи в диссертационной работе построены основы теории функциональных сетей Петри. Впервые введен класс функциональных сетей Петри, исследованы свойства функциональных подсетей. Разработаны методы декомпозиции заданной сети Петри на функциональные подсети.

Введены и исследованы слабые типы эквивалентности сетей Петри. Получено представление передаточной функции сети Петри, позволяющее выполнить эквивалентные преобразования сетей на основе алгебраических преобразований формул, описывающих передаточную функцию. Эквива-

лентные преобразования являются основой для редукции временных сетей Петри – последовательном уменьшении размерности сети с сохранением заданных свойств, что ускоряет процессы верификации протоколов телекоммуникационных систем. В уравнении состояний временной сети Петри и формулах её передаточной функции использованы операции многозначной (непрерывной) логики. Впервые простроены методы синтеза функции непрерывной логики, заданной таблично.

Выполнено обобщение функциональных сетей Петри для произвольных систем линейных алгебраических уравнений над кольцом со знаком. Соответствующие структуры названы кланами линейных систем. Разработаны методы решения СЛАУ в процессе одновременной и последовательной композиции их кланов, обеспечивающие существенное ускорение вычислений.

Задача последовательной композиции кланов сформулирована в терминах теории графов и названа оптимальным коллапсом взвешенного графа. Получены оценки верхней и нижней границ ширины коллапса заданного взвешенного графа, которые могут быть использованы при решении задачи методом ветвей и границ. Представлен и статистически обоснован эвристический алгоритм оптимального коллапса взвешенного графа.

Построены эффективные алгоритмы декомпозиции на функциональные подсети (кланы), композиционного решения СЛАУ. Выполнена их программная реализация, обеспечивающая переносимость разработанного программного обеспечения на различные платформы и его встраивание в известные моделирующие системы. Программное обеспечение применено для верификации протоколов ECMA, BGP, TCP, IOTP.

Областью применения композиционных алгоритмов является анализ свойств моделей Петри большой размерности, в частности, верификация телекоммуникационных протоколов, представленных детализированной моделью Петри.

2. Решена задача синтеза моделей Петри по стандартным спецификациям протоколов.

В условиях динамичного роста количества применяемых в телекоммуникациях протоколов и усложнения их стандартных спецификаций всё более трудоёмким становится построение детализированной моделей Петри по исходным спецификациям протоколов.

Для решения указанной задачи в диссертационной работе предложено выполнять синтез моделей Петри протоколов с использованием промежуточного языка взаимодействующих последовательных процессов Хоара. Вербальное описание протокола естественным образом трансформируется в формулы ВПП. Разработаны методы синтеза сети Петри по заданной формуле ВПП. В качестве вспомогательной задачи исследован синтез сети Петри заданной конечным автоматом. Предложено новое решение задачи с помощью линейных систем уравнений и неравенств специального вида. Выполнен синтез модели Петри протокола электронной коммерции IOTP.

Предложено использовать сеть Петри в качестве универсального языка спецификации протоколов. Представлено два подхода к управлению уровнем абстракции спецификаций: описание фишкой сети Петри сообщения протокола и описание фишкой сети Петри полей заголовка протокола. Построена модель протокола BGP на основе первого подхода и протокола TCP – на основе второго.

Областью применения разработанных методов синтеза является автоматизированное построение детализированных моделей Петри сложных телекоммуникационных протоколов при решении задач верификации протоколов.

3. Решена задача построения модели Петри по заданной структурной схеме телекоммуникационной системы либо сети и экспресс оценки её функциональных характеристик.

В управляемом моделью проектировании телекоммуникационных сетей и систем требуется обеспечить динамичную перекомпоновку модели для выбора оптимальных структур, а также анализ свойств модели за приемлемое время.

В диссертационной работе предложено выполнять компоновку модели Петри по структурной схеме системы либо сети в процессе композиции компонентов, являющихся функциональными подсетями либо подсетями с контактными позициями. Библиотека стандартных компонентов обеспечивает перекомпоновку модели в кратчайшие сроки. Построены стандартные компоненты для построения моделей: 1) коммутируемой Ethernet: коммутатор, рабочая станция, сервер; 2) сетей с коммутацией меток MPLS: IP-маршрутизатор, LSR-маршрутизатор, LSR/LER-маршрутизатор, терминальная сеть; 3) беспроводных сетей Bluetooth: ведущее устройство, ведомое устройство. Выполнено построение моделей сетей Ethernet, MPLS, Bluetooth по заданным структурным схемам.

Для экспресс оценки функциональных характеристик построенных моделей предложен метод измерительных фрагментов. Первичное измерение простых характеристик и алгоритмы вычисления сложных характеристик реализуются элементами сетей Петри, составляющими измерительные фрагменты. Модель, дополненная измерительными фрагментами, позволяет выполнять измерение и вычисление функциональных характеристик проектируемых систем и сетей в кратчайшие сроки в процессе имитации динамики сети Петри. При этом часы реального времени функционирования систем могут быть представлены секундами вычислительных экспериментов с моделью.

Разработаны измерительные фрагменты для оценки времени отклика коммутируемой Ethernet, трафика MPLS сетей, эффективности использования адресного пространства протокола Bluetooth. Измерительные фрагменты применены для проектирования ЛВС диспетчерских центров железной дороги, магистральных транспортных сетей Internet, мобильных сенсорных сетей

на основе технологии Bluetooth. Адекватность моделей доказана с помощью измерения функциональных характеристик реальных сетей.

Областью применения разработанных методов композиции моделей Петри и измерительных фрагментов является управляемое моделью проектирование телекоммуникационных систем и сетей.

4. Решена задача верификации протоколов с неограниченным числом компонентов.

Телекоммуникационные протоколы содержат, как правило, описание парного взаимодействия систем (BGP, TCP), однако более сложные протоколы, например, IOTP, содержат множество ролей, исполняемых взаимодействующими системами. Кроме того, ряд протоколов, например, беспроводных сетей, предусматривает одновременное взаимодействие неограниченного числа устройств. Соответствующая модель Петри имеет бесконечную размерность, что затрудняет её анализ традиционными методами.

Для решения указанной задачи в диссертационной работе предложен метод композиционного вычисления инвариантов сетей Петри в параметрической форме. Метод может быть применён для бесконечных моделей регулярной структуры, полученных в результате композиции функциональных подсетей компонентов. Суть метода заключается в построении системы композиции и её решении в параметрической форме с последующим получением параметрического представления инвариантов бесконечной модели.

Метод композиционного вычисления инвариантов в параметрической форме применен для верификации протоколов Ethernet с архитектурой общей шины. Доказана инвариантность модели для произвольного числа подключенных рабочих станций.

Областью применения метода композиционного вычисления инвариантов в параметрической форме является анализ свойств моделей Петри, содержащих неограниченное число однотипных компонентов, в частности, моделей телекоммуникационных систем и сетей.

В диссертационной работе также решён ряд вспомогательных задач, таких как теоретическое обоснование и оценка вычислительной сложности метода Тудика, применяемого для вычисления инвариантов сетей Петри; эффективная реализация алгоритмов декомпозиции на функциональные подсети (кланы) и композиционного решения СЛАУ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Adreolini M., Lancellotti R. Analysis of peer-to-peer systems: workload characterisation and effects on traffic cacheability // Proc. of 12th Annual Meeting of the IEEE / ACM International Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, October 5-7, 2004 (MASCOTS 2004). – Volendam (Netherlands). – 2004. – P. 95-104.

2. Albert K., Jensen K., Shapiro R. Design/CPN: A Tool Package Supporting the Use of Colored Nets // Petri Net Newsletter. – April 1989. – P. 22-35.

3. Alur R., Dill D.L. A theory of timed automata // Theor. Comp. Sci. – 1994. – Vol. 126. – P. 183-235.

4. Baader F., Ziekmann J. Unification theory. Handbook of logic in artificial intelligence and logic programming. – Oxford: Univ. Press, 1994. – P. 1-85.

5. Berge C. The theory of graphs. – N.Y.: Mineola, Dover, 2001. – 247 p.

6. Beaudouin-Lafon M., Mackay W.E., Jensen M. et al. CPN Tools: A Tool for Editing and Simulating Coloured Petri Nets // LNCS: Tools and Algorithms for the Construction and Analysis of Systems. – 2001. – Vol. 2031. – P. 574-580.

7. Born1 M., Schieferdecker I. et al. Model-Driven Development and Testing – A Case Study. – Berlin: Fraunhofer FOKUS, 2005. – 8 p.

8. Berthelot G. Transformation and decomposition of nets // LNSC. – 1987. – Vol. 254. – P. 359-376.

9. Berthelot G., Terrat R. Petri Nets Theory for the Correctness of Protocols // IEEE Trans. on Communications. – 1982. – Vol. 30, № 12. – P. 2497-2505.

10. Berthomieu B., Ribet O.-P., Vernadat F. The tool TINA - construction of abstract state space for Petri nets and Time Petri nets // International Journal of Production Research. – 2004. – Vol.42, №14. – P. 2741-2756.

11. Best E., Cherkasova L., Desel J. Compositional generation of home states in free choice systems // Formal aspects of computing. – 1992. – №. 4. – P. 572-581.

12. Bidgoli H. Electronic Commerce. – Academic Press, 2002. – 487 p.

13. Bluetooth Special Interest Group: Specification of the Bluetooth System. Version 2.0. – Bluetooth Special Interest Group. – 2004. – 1230 p.

14. Breyer R., Riley S. Switched, fast, and gigabit Ethernet. – MacMillan Technical Pub., 1999. – 618 p.

15. Burdett D. Internet Open Trading Protocol – IOTP. Version 1.0E. – FRC 2801, April 2000. – 290 p.

16. Calafate C.T., Manzoni P., Malumbres M.P. Assessing the effectiveness of IEEE 802.11e in multi-hop mobile network environments // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 205-212.

17. Chandra P., Kshemkalyani A.D. Causal Multicast in Mobile Networks // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 213-220.

18. Chen J., Gupta D. et al. Routing in Internet-Scale Network Emulator // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 275-283.

19. Christensen S., Petrucci L. Modular analysis of Petri nets // The Computer Journal. – 2000. – Vol. 43, № 3. – P. 224-242.

20. Chumchu P., Boreli R., Seneviratne A. Performance Analysis of Reliable Multicast Transport Protocols for GEO Satelite networks // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 318-326.

21. Clarkey E., Luz Y., Veithy H., Wangz D. EPSL: Executable Protocol Specification Language. – USA: Carnegie Mellon University, 2005. – 2 p.

22. Colom J.M., Silva M. Improving the linearly based characterization of P/T nets // LNCS 483, ATPN. – 1990. – P. 248-257.

23. Contejan E., Ajili F. Avoiding slack variables in solving linear Diophantine equations and inequations // Theor. Comp. Sci. – 1997. – Vol. 173. – P. 183-208.

24. Cortadella J., Kishinevsky M., Kondratyev A., Lavagno L., Yakovlev A. Logic synthesis of asynchronous controllers and interfaces. – Springer-Verlag, 2002. – 273 p.

25. Chretienne P. Execution controles des resaux de Petri syncronyses // Techn. et sci. Inform. – 1984. – Vol. 3, № 1. – P. 23-31.

26. Darondeau P., Badouel E., Caillaud B. Distributing Finite Automata Through Petri Net Synthesis // Formal Aspects of Computing. – 2002. – Vol. 13. – P. 447-470.

27. Desel J., Esparza J.: Free choice Petri nets. – Cambridge university press, 1995. – 244 p.

28. Desel J., Reisig W. The synthesis problem of Petri nets // Acta Informatica. – Vol. 33, 1996. – 297-315.

29. Diaz M. Modelling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Model // Computer Networks. – 1982, № 6. – P. 419-441.

30. Diaz M. (dir.) Les résaux de Petri. – Hermés, Traité IC2, Paris, 2001. – 386 p.

31. Dimopoulos P., Zeephongsekul P., Tari Z. Modeling the burstiness of TCP // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 175-183.

32. Dimitropoloulos X.A., Riley G.F. Large-Scale Simulation of BGP // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 287-294.

33. Ehrenfeucht A., Rozenberg G. Partial (Set) 2-Structure // Acta Informatica. – 1990. – Vol. 27. – 315-368.

34. Esparza J., Silva M. Compositional synthesis of live and bounded free choice Petri nets // LNCS. – 1991. – Vol. 527. – P. 172-187.

35. Fei Y., Zhong L., Jha N.K. An Energy-aware Framework for Coordinated Dynamic Software Management in Mobile Computers // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004.– P. 306-317.

36. Franklin M.A., Joshi V. SimplePipe: A Simulation Tool for Task Allocation and Design of Processor Pipelines with Application to Network Processors // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 59-66.

37. Ichikawa A., Yokoyama K., Kurogi S. Reachability and Control of Discrete Event Systems Represented by Conflict-free Petri Nets // Proc.of ISCAS'85. – 1985. – P. 487-490.

38. Gilly K., Alcaraz S. at al Comparison of predictive techniques in Cluster-Based Network Servers with Resource Allocation // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 545-552.

39. Girault C., Valk R. Petri nets for systems engineering. – Springer-Verlag, 2003. – 607 p.

40. Gu Y., Fujimoto R. A Flexible Architecture for Remote Server-Based Emulation // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 447-454.

41. Gulpmar N., Harrison P., Rustem B. An Optimisation for a Two-Node Router Network // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 147-156.

42. Hack M. Extended State-Machine Allocatable Nets, an Extension of Free Choice Petri Nets Results. – Cambridge, Massachussets, MIT Project MAC, CSG-Memo 78-1, 1974. – 124 p.

43. Haddad S., Moreaux P. Asynchronous composition of high-level Petri nets: a quantative approach // Proc. of 17th ATPN Osaka, Japan, June 24-28, 1996. – LNCS. – 1996. – Vol. 1091. – P. 192-211.

44. Haddad S., Klai K., Ilie J.-M. An incremental verification technique using decomposition of Petri nets // Proc. of the second IEEE International Conf. on Systems, Man and Cybernetics (SMC'02), October 6-9, 2002. – Hammamet (Tunisia). – 2002. –Vol. 2. – P. 381-386.

45. Haddad S., Moreaux P. Approximate analysis of non Markovian stochastic systems with multiple time scale delay // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 23-30.

46. He Xudong, Lee John A. N. A Methodology for Constructing Predicate Transition Net Specifications // Software-Practice and Experience. – 1991. – Vol. 21, № 8 (Aug.). – P. 845-875.

47. Heiner M., Koch I. Petri net based model validation in systems biology // Proc. of the 25-th International conf. on application and theory of Petri nets, June 21-25, 2004. – Bologna, Italy. – 2004. – P. 216-237.

48. Hunt R. Evolving Technologies for New Internet Applications // IEEE Internet Computing. – 1999. – № 5. – P. 16-26.

49. Operations research: methods, models, and applications / Ed.: Jay E. – Aronson and Stanley Zionts. Westport, Conn. Quorum, 1998. – 369 p.

50. Jensen K. Colored Petri Nets - Basic Concepts, Analysis Methods and Practical Use. – Springer-Verlag, 1997. – Vol. 1-3. – 673 p.

51. Juan E.Y.T., Tsai J.J.P., Murata T. Compositional verification of concurrent systems using Petri net based condensation rules // ACM Trans. on Programming Languages and Systems. – 1998. – Vol. 20, № 5. – P. 917-979.

52. Juhas G., Lorenz R., Neumair C. Synthesis of Controlled Behaviour with Models of Signal Nets // Proc. of the 25-th International conf. on application and theory of Petri nets, Bologna, Italy, June 21-25, 2004. – LNCS. – 2004. – Vol. 3099. – P. 238-257.

53. Kardach J. Bluetooth Architecture Overview. – Mobile Computing Group, Intel Corporation, 2002. – 7 p.

54. Latvala T., Makela M. LTL Model Checking for Modular Petri Nets // Proc. of the 25-th International conf. on application and theory of Petri nets, Bologna, Italy, June 21-25, 2004. – LNCS. – 2004. – Vol. 3099. – P. 298-311.

55. Le Faucheur F., L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, J. Heinanen. Multi-Protocol Label Switching (MPLS) Support of Differentiated Services. – RFC 3270, May 2002. – 64 p.

56. Lee J.-K. Decomposition of Petri nets using the transitive matrix // Proc. of the second IEEE International Conf. on Systems, Man and Cybernetics (SMC'02), October 6-9, 2002. – Hammamet (Tunisia). – 2002. – Vol. 2. – P. 648-655.

57. Lee W.J., Kim H.N., Cha S.D., Kwon Y.R. A slicing-based approach to enhance Petri net reachability analysis // Journal of Research Practices and Information Technology. – 2000. – Vol. 32, № 2. – P. 131-143.

58. Levin R., DesJardins R. Theory of games and strategies. – Prentice-Hall, 1970. – 132 p.

59. Li X., Cuthbert L. Stable Node-Disjoint Multipath Routing with Low Overhead in Mobile Ad hoc Networks // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 184-191.

60. Li Y., Williamson C. A Hysteresis Model for Web/TCP Transfer Latency // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 167-174.

61. Lloyd J. Foundation of logic programming. – Berlin: Springer-Verlag, 1987. – 216 p.

62. Loogheed K., Rekhter Y., Watson T.J. A Border Gateway Protocol (BGP). – RFC 1105, 1989. – 17 p.

63. Lui Ming T., Elsaadany A., Singhal M. Performance study of buffering within switches in local area networks // Proc. of 4th International Conf. on Computer Communications and Networks. – 1995. – P. 451-460.

64. Mandalia V. Protocol Specification Language. – India: Indian Institute of Science, 2005. – 41p.

65. Margi C.B., Obraczka K. Instrumenting Metwork Simulators for Evaluating Energy Consumption in Power-Aware Ad-Hoc Network Protocols // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 337- 346.

66. Marsan A.M., Chiola G., Fumagalli A. An Accurate Performance Model of CSMA/CD Bus LAN // Advances in Petri Nets, LNCS. – 1987. – Vol. 266. – P. 146-161.

67. Martinez J., Silva M. A simple and fast algorithm to obtain all invariants of a generalized Petri net // Application and theory of Petri nets. – 1982. – P. 301-310.

68. Model-driven development of mobile phones. – Nokia Research Center, Materials of tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, October 8-11, 2004. – Aarhus: Denmark. – 2004. – 48 p.

69. Moses J., Illikal R. et al. ASPEN: Towards Effective Simulation of Threads & Engines in Evolving Platforms // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 51-58.

70. Neglia G., Falleta V., Bianchi G. Is TCP Packet Reordering Always Harmful? // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 87-94.

71. Orfanos G., Habetha J., Liu L. MC-CDMA Based IEEE 802.11 Wireless LAN // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 400-405.

72. Ouyang C., Kristensen L.M., Billington J. A Formal and Executable Specification of the Internet Open Trading Protocol // Proc. 3rd International Conference, EC-Web. – Lecture Notes in Comput. Sci. – 2002. – Vol. 2455. – P. 377-387.

73. Parhomenko S., Samohin S. Testing of Fast Ethernet adapters // ComputerPress. – 2001, № 8. – P. 85-93.

74. Peterke G., Murata T. Proof procedure and answer extraction in Petri net model of logic programs // IEEE Trans. Soft. Eng. – 1989. – Vol. 15, № 2. – P. 209-217.

75. Pomello L., Bernardinello L. Formal Tools For Modular Systems Development // Proc. of the 25-th International conf. on application and theory of Petri nets, Bologna, Italy, June 21-25, 2004. – LNCS. – Vol. 3099. – P. 77-96.

76. Postel, J. (ed.) Transmission Control Protocol, STD 7. – RFC 793, September 1981. – 85 p.

77. Pottie L. Minimal Solutions of linear Diophantine systems: bounds and algorithms // Proc. of the Fourth Intern. Conf. on Rewriting Techn. and Appl. – Como (Italy). – 1991. – P. 162-173.

78. Rahul V. LAN Switching. – OHIO, 2002. – 20 p.

79. Reisig W. Petri nets: an introduction. – Springer-Verlag, 1982. – 161 p.

80. Rekhter Y., Watson T.J., Li T. A Border Gateway Protocol 4 (BGP-4). – RFC 1771, 1995. – 57 p.

81. Riley G.F. Simulating Internet Worms // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 268-274.

82. Rosen E., A. Viswanathan, R. Callon. Multiprotocol Label Switching Architecture. – FRC 3031, January 2001. – 61 p.

83. Rosen E, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, A. Conta. MPLS Label Stack Encoding. – RFC 3032, January 2001. – 23 p.

84. Russell T. Telecommunications Protocols, 2nd Edition. – McGraw-Hill, 2004. – 686 p.

85. Sobeih A., Yurcil W., Hou J.C. VRing: A Case for Building Application-Layer Multicast Rings // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 437- 446.

86. Souissi I., Younes A. On Liveness Preservation by Composition of Nets via a Set of Places // Proc. of the 11th International Conf. on Application and Theory of Petri Nets, 1990, Paris, France. – LNCS. – 1990. – Vol.524. – P. 277-295.

87. The *ns* Manual / Ed.: Kevin Fall, Kannan Varadhan. – The VINT Project, 2006. – 414 p.

88. Toudic J.M. Linear Algebra Algorithms for the Structural Analysis of Petri Nets // Rev. Tech. Thomson CSF. – 1982. – Vol. 14, № 1. – P. 136-156.

89. Valmari A. Compositional State Space Generation // Proc. of the 11th International Conf. on Application and Theory of Petri Nets, 1990, Paris, France. – LNCS: Advances in Petri Nets. – 1993. – Vol. 674. – P. 427-457.

90. Verbeek E., Van der Torn R. Transit case study // Proc. of the 25-th International conf. on application and theory of Petri nets, June 21-25, 2004. – Bologna (Italy). – 2004. – P. 392-410.

91. Von Oheimb D. The High-Level Protocol Specification Language HLPSL developed in the EU project AVISPA. – Munich: Siemens Corporate Technology, 2005. – 17p.

92. Wang Y., Kaeli D. Execution-driven Simulation of Network Storage Systems // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 604-611.

93. Zaitsev D.A. Functional Petri Nets. – Universite Paris-Dauphine, Cahier du Lamsade 224, Avril 2005. – 62 p.

94. Zaitsev D.A. Verification of Protocol BGP via Decomposition of Petri Net Model into Functional Subnets // Proc. of the Design, Analysis, and Simulation of Distributed Systems Symposium, April 2-8, 2005. – San Diego (USA). – 2005. – P. 72-78.

95. Zaitsev D.A. An Evaluation of Network Response Time using a Coloured Petri Net Model of Switched LAN // Proc. of Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, October 8-11, 2004. – Aarhus (Denmark). – 2004. – P. 157-167.

96. Zaitsev D.A. Verification of protocol TCP via decomposition of Petri net model into functional subnets // Proc. of the Poster session of 12th Annual Meeting of the IEEE / ACM International Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, October 5-7, 2004. – Volendam (Netherlands). – 2004. – P. 73-75.

97. Zaitsev D.A. Solving the fundamental equation of Petri net using the decomposition into functional subnets // 11th Workshop on Algorithms and Tools for Petri Nets, September 30 - October 1, 2004. – University of Paderborn (Germany). – 2004. – P. 75-81.

98. Zaitsev D.A. Verification of Protocol ECMA with Decomposition of Petri Net Model // Proc. of The International Conf. on Cybernetics and Information Technologies, Systems and Applications, July 21-25, 2004. – Orlando (USA). – 2004. – P. 287-295.

99. Zaitsev D.A. Decomposition-based calculation of Petri net invariants // Proc. of Workshop on Token based computing of the 25-th International conf. on application and theory of Petri nets, June 21-25, 2004. – Bologna (Italy). – 2004. – P. 79-83.

100. Zaitsev D.A. Switched LAN Simulation by Colored Petri Nets // Mathematics and Computers in Simulation. – 2004. – Vol. 65, № 3. – P. 245-249.

101. Zaitsev D.A. Switched LAN simulation by colored Petri nets // Proc. of European Simulation and Modelling Conf., October 27-29, 2003. – Naples (Italy). – 2003. – P. 485-489.

102. Zaitsev D.A. Formal grounding of Toudic method // Proc. of the 10th Workshop "Algorithms and Tools for Petri Nets", September 26-27, 2003. – Eichstaett (Germany). – 2003. – P. 184-190.

103. Zaitsev D.A. Subnets with input and output places // Petri Net Newsletter. – April 2003, Vol. 64. – P. 3-6. (Cover Picture Story).

104. Zeng X., Lung C.-H., Huang C. A Bandwidth-efficient Scheduler for MPLS DiffServ Networks // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 251-258.

105. Zhang X., Riley G.F. Bluetooth Simulations for Wireless Sensor Networks using GTNetS // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 375-382.

106. Zhang Z., Hasegawa G., Murata M. Performance Analysis and Improvement of HighSpeed TCP with TailDrop/RED Routers // Proc. of MASCOTS 2004. – Volendam (Netherlands). – 2004. – P. 505-512.

107. Адельсон-Вельский Г.М., Диниц Е.А., Карзанов А.В. Потоковые алгоритмы. – М.: Наука, 1975. – 120 с.

108. Анисимов Н.А., Голенков Е.А., Харитонов Д.И. Композиционный подход сетей Петри к разработке параллельных и распределённых систем // Программирование и программное обеспечение. – 2001. – Т. 26, № 6. – С. 309-319.

109. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Т.1: Синтаксический анализ. – М.: Мир, 1978. – 488с.

110. Ачасова С.М., Бандман О.Л. Корректность параллельных вычислительных процессов. – Н.: Наука, 1990. – 253 с.

111. Бандман М.К., Бандман О.Л., Есикова Т.Н. Территориально-производственные комплексы: Прогнозирование процесса формирования с использованием сетей Петри. – Новосибирск: Наука, 1990. – 303 с.

112. Бандман О.Л. Поведенческие свойства сетей Петри // Известия АН СССР. Техническая кибернетика. – 1987. – № 5. – С. 134-150.

113. Батищев Д.И. Методы оптимального проектирования. – М.: Радио и связь, 1984. – 248 с.

114. Буч Г., Рамбо Г., Якобсон И. UML. Руководство пользователя. – М.: ДМК, 2000. – 358 с.

115. Ван дер Варден Б.Л. Алгебра. – М: Наука, 1979. – 624 с.

116. Васильев В.В., Кузьмук В.В. Сети Петри, параллельные алгоритмы и модели мультипроцессорных систем. – Киев: Наукова думка, 1990. – 216 с.

117. Вегенша Ш. Качество обслуживания в сетях IP. – М.: “Вильямс”, 2003. – 383 с.

118. Вишневский В.М. Теоретические основы проектирования компьютерных сетей. – М.: Техносфера, 2003. – 512 с.

119. Волгин Л.И., Левин В.И. Непрерывная логика. Теория и применение. – Таллин, 1991. – 210 с.

120. Вотяков А.А., Фрумкин М.А. Алгоритм нахождения общего целочисленного решения системы линейных уравнений // Исследования по дискретной оптимизации. – М.: Наука, 1976. – С. 128-140.

121. Гинзбург С.А. Математическая непрерывная логика и изображение функций. – М.: Энергия, 1968. – 136 с.

122. Глушков В.М. Синтез цифровых автоматов. – М.: Физматгиз, 1962. – 476с.

123. Годунов С.К. Решение систем линейных уравнений. – Н.: Наука, 1980. – 177 с.

124. Гольдштейн Б. Протоколы сети доступа. – М.: Радио и связь, 1999. – 246 с.

125. Зайцев Д.А., Чорногала Е.Я. Синтез модели Петри и верификация протокола электронной коммерции IOTP // Радиотехника: Всеукр. межведомств. науч.-техн. сб. 2006. – Вып. 144. – С. 28-35.

126. Зайцев Д.А. Передаточная функция сети Петри // Искусственный интеллект. – 2006. – №1. – С. 23-30.

127. Зайцев Д.А., Березнюк М.В. Исследование эффективности использования адресного пространства протокола Bluetooth // Радиоэлектроника. Информатика. Управление. – 2006. – №1. – С. 57-63.

128. Зайцев Д.А., Сакун А.Л. Исследование эффективности технологии MPLS с помощью раскрашенных сетей Петри // Зв'язок. – 2006. – Т. 65, №5. – С. 49-55.

129. Зайцев Д.А. Синтез моделей Петри телекоммуникационных протоколов // Труды Одесской национальной академии связи им. А.С.Попова. – 2005. – №2. – С. 36-42.

130. Зайцев Д.А. Верификация протокола TCP в процессе последовательной композиции модели Петри // Зв'язок. – 2006. – Т. 64, №4. – С. 49-58.

131. Зайцев Д.А. Последовательная композиция моделей Петри телекоммуникационных протоколов // Зв'язок. – 2006. – Т. 61, №1. – С. 45-50.

132. Зайцев Д.А. Последовательная композиция кланов линейных систем // Системні дослідження та інформаційні технології. – 2006. – №2. – С. 121-137.

133. Зайцев Д.А. Композиционный анализ сетей Петри // Кибернетика и системный анализ. – 2006. – № 1. – С. 143-154.

134. Зайцев Д.А. О реализации композиционных алгоритмов решения систем линейных уравнений // Управляющие системы и машины. – 2006. – №3.– С. 32-41.

135. Зайцев Д.А., Шмелёва Т.Р. Основы построения параметрических моделей Петри коммутируемых сетей // Моделирование и компьютерная графика: Материалы 1-й международной научно-технической конференции. – Донецк: ДонНТУ. – 2005. – С. 207-215.

136. Зайцев Д.А., Шмелёва Т.Р. Измерение характеристик одноуровневой коммутируемой сети с помощью параметрической модели Петри // Радиотехника: Всеукр. межведомств. науч.-техн. сб. – 2005. – Вып. 142. – С. 40-47.

137. Зайцев Д.А., Шмелёва Т.Р. Параметрическая модель Петри одноуровневой коммутируемой сети // Труды Одесской национальной академии связи им. А.С.Попова. – 2005. – № 1. – С. 33-40.

138. Зайцев Д.А. Программное обеспечение для декомпозиции двудольных орграфов // Научные труды Донецкого государственного технического университета, серия "Информатика, кибернетика и вычислительная техника". – 2005. – Вып. 93. – С. 60-70.

139. Зайцев Д.А. Решение линейных систем с помощью декомпозиции // Системні дослідження та інформаційні технології. – 2005. – №2. – С. 131-143.

140. Зайцев Д.А. Измерительные фрагменты в моделях Петри телекоммуникационных сетей // Зв'язок. – 2005. – Т. 54, №2. – С. 65-71.

141. Зайцев Д.А. Верификация телекоммуникационных протоколов с помощью декомпозиции сетей Петри // Зв'язок. – 2005. – Т. 53, №1. – С. 41-47.

142. Зайцев Д.А. Решение фундаментального уравнения сетей Петри в процессе композиции функциональных подсетей // Искусственный интеллект. – 2005. – № 1. – С. 59-68.

143. Зайцев Д.А. Последовательная композиция функциональных подсетей // Труды Одесской национальной академии связи им. А.С.Попова. – 2004. – № 3. – С. 33-40.

144. Зайцев Д.А. Декомпозиция сетей Петри // Кибернетика и системный анализ. – 2004. – №5. – С. 131-140.

145. Зайцев Д.А. Инварианты временных сетей Петри // Кибернетика и системный анализ. – 2004. – Т. 40, № 2. – С. 92-106.

146. Зайцев Д.А., Шмелёва Т.Р. Моделирование коммутируемой локальной сети раскрашенными сетями Петри // Зв'язок. – 2004. – Т. 46, № 2. - С. 56-60.

147. Зайцев Д.А. Инвариантность модели Петри протокола TCP // Труды Одесской национальной академии связи им. А.С.Попова. – 2004. – № 2. – С. 19-27.

148. Зайцев Д.А. К вопросу о вычислительной сложности метода Тудика // Искусственный интеллект. – 2004. – № 1. – С. 29-37.

149. Зайцев Д.А. Теоретическое обоснование метода Тудика // Научные труды Донецкого государственного технического университета, серия "Информатика, кибернетика и вычислительная техника". – 2004. – Вып. 74. – С. 286-293.

150. Зайцев Д.А. Декомпозиция протокола ECMA // Радиотехника: Всеукр. межведомств. науч.-техн. сб. – 2004. – Вып. 138. – С. 75-82.

151. Зайцев Д.А. Верификация протоколов Ethernet // Труды Одесской национальной академии связи им. А.С.Попова. – 2004. – № 1. – С. 42-48.

152. Зайцев Д.А. Инварианты функциональных подсетей // Труды Одесской национальной академии связи им. А.С.Попова. – 2003. – № 4. – С. 57-63.

153. Зайцев Д.А., Сарбей В.Г., Слепцов А.И. Синтез функций непрерывной логики заданных таблично // Кибернетика и системный анализ. – 1998. – № 2. – С. 47-56.

154. Зайцев Д.А., Слепцов А.И. Уравнение состояний и эквивалентные преобразования временных сетей Петри // Кибернетика и системный анализ. – 1997. – № 5. – С. 59-76.

155. Зайченко Ю.П., Гонта Ю.В. Структурная оптимизация сетей ЭВМ. – К.: Техника, 1986. – 168 с.

156. Золотова Т.М., Кербников Ф.И., Розенблат М.А. Резервирование аналоговых устройств автоматики. – М.: Энергия, 1975. – 127 с.

157. Зябиров Х.Ш., Кузнецов Г.А., Шевелев Ф.А., Слободенюк Н.Ф., Крашенинников С.В., Крайсвитний В.П., Ведищев А.Н. Автоматизированная система оперативного управления эксплуатационной работой ГИД "Урал-ВНИИЖТ" // Железнодорожный транспорт. – 2003. – №2. – С. 36-45.

158. Компьютерная алгебра (Символьные и алгебраические вычисления) / Под. Ред. Б.Бухбергера, Дж.Коллинза, Р.Лооса. – М.: Мир, 1986. – 386 с.

159. Косачёв А.С., Пономаренко В.Н. Анализ подходов к верификации функций безопасности и мобильности. – М: Институт системного программирования РАН, 2004. – 101с.

160. Котов В.Е. Сети Петри. – М. Наука, 1984. – 160 с.

161. Крон Г. Тензорный анализ сетей. – М.: Сов. Радио, 1978. – 719 с.

162. Крывый С.Л. О некоторых методах решения и критериях совместимости систем линейных диофантовых уравнений в области натуральных чисел // Кибернетика и системный анализ. – 1999. – № 4. – С. 12-36.

163. Крывый С.Л. Об алгоритмах решения систем линейных диофантовых констрейнтов в области {0,1} // Кибернетика и системный анализ. – 2003. – №5. – С. 58-69.

164. Кривый С.Л., Матвеева Л.Е. Формальные методы анализа свойств систем // Кибернетика и системный аналіз. – 2003. – № 2. – С. 15-36.

165. Крывый С.Л., Чугаенко А.В., Богак Н.А., Бура В.В. О реализации алгоритмов проверки совместимости систем линейных диофантовых уравнений в области натуральных чисел // Управляющие системы и машины. – 1999. – № 3. – С. 26-32.

166. Крывый С.Л. Совместимость систем линейных неравенств на множестве натуральных чисел // Кибернетика и системный анализ. – 2002. – №1. – С. 17-27.

167. Мак-Нотон Р. Теорема о бесконечнозначной логике высказываний. Кибернетический сборник. – М.: ИИЛ, 1961, Вып.3. – С.59-78.

168. Мулен Э. Теория игр. – М.: Наука, 1985. – 200 с.

169. Мурата Т. Сети Петри: Свойства, анализ, приложения // ТИИЭР. – 1989. – Т. 77, №4. – С. 41-85.

170. Назаров А.Н., Симонов М.В. АТМ: технология высокоскоростных сетей. - М.: ЭКО-ТРЭНДЗ, 1997. – 232 с.

171. Невдяев Л.М. Мобильная связь 3-го поколения. – М.: Связь и бизнес, 2000. – 208 с.

172. Ортега Дж. Введение в параллельные и векторные методы решения линейных систем. – М.: Мир, 1991. – 367 с.

173. Питерсон Дж. Теория сетей Петри и моделирование систем. – М. Мир, 1984. – 264 с.

174. Прокис Дж. Цифровая связь. – М.: Радио и связь, 2000. – 800 с.

175. Ратынский М.В. Основы сотовой связи. – М.: Радио и связь, 2000. – 248с.

176. Сейдж Э.П., Мелс Дж. Л. Теория оценивания и её применение в связи и управлении. – М.: Связь, 1976. – 496 с.

177. Сергиенко И.В. Математические модели и методы решения задач дискретной оптимизации. – Киев: Наукова думка, 1988. – 472 с.

178. Сингх М., Титли А. Системы: декомпозиция, оптимизация и управление. – М.: Машиностроение, 1986. – 496 с.

179. Скляр Б. Цифровая связь: Теоретические основы и практическое применение. – М.: “Вильямс”, 2004. – 1104 с.

180. Слепцов А.И., Юрасов А.А. Автоматизация проектирования управляющих систем гибких автоматизированных производств / Под ред. Б.Н.Малиновского. – К. Техніка, 1986. – 160 с.

181. Слепцов А.И. Уравнения состояний и эквивалентные преобразования нагруженных сетей Петри (алгебраический подход) // Формальные модели параллельных вычислений: Докл. и сообщ. Всесоюзн. конф.– Новосибирск. – 1988. - С. 151-158.

182. Столлингс В. Беспроводные линии связи и сети.– М: Вильямс, 2003.– 640 с.

183. Схрейвер А. Теория линейного и целочисленного программирования. В 2-х т. – М.: Мир, 1991. – 726 с.

184. Татт У. Теория графов. – М.: Мир, 1988. – 424 с.

185. Труб И.И. Объектно-ориентированное моделирование на языке С++. – С.-П.Б.: Питер, 2005. – 411 с.

186. Форд Л., Фалкерсон Д. Потоки в сетях. – М.: Мир, 1966. – 276 с.

187. Фрумкин М.А. Алгоритм решения систем линейных уравнений в целых числах // Исследования по дискретной оптимизации. – М.: Наука, 1976. – С. 97-127.

188. Харари Ф. Теория графов. – М.: Мир, 1973. – 300 с.

189. Хоар Ч. Взаимодействующие последовательные процессы. – М.: Мир, 1989. – 264 с.

190. Шварц М. Сети связи: протоколы, моделирование и анализ: В 2-х ч. – М: Наука, 1992. – 336 с.

191. Шимбирев П.Н. Гибридные непрерывно-логические устройства. – М.: Энергоатомиздат, 1990. – 174 с.

192. Шрайбер Т. Дж. Моделирование на GPSS. – М.: Машиностроение, 1980. – 592 с.

ПРИЛОЖЕНИЯ

Приложение А. Верификация протокола ECMA

Известный протокол ECMA Европейской ассоциации производителей компьютеров (European Computer Manufacturer Association) является транспортным протоколом, располагающимся между сетевым и сессионным уровнями модели ISO. Далее будет использована модель протокола, представленная в работе [9]. С одной стороны модель является достаточно упрощённой и содержит приемлемое для изложения материала количество элементов, с другой стороны её исследование позволяет продемонстрировать особенности применения предлагаемых методов.

Напомним, что четыре класса протоколов ECMA обеспечивают устойчивое взаимодействие систем в условиях возникновения ошибок возрастающей серьёзности. Каждый из классов состоит из трёх фаз: установление транспортного соединения, передача данных и завершение транспортного соединения (разъединение). Рассматриваемая далее модель представляет фазы соединения-разъединения, и абстрагируется от конкретных механизмов передачи данных.

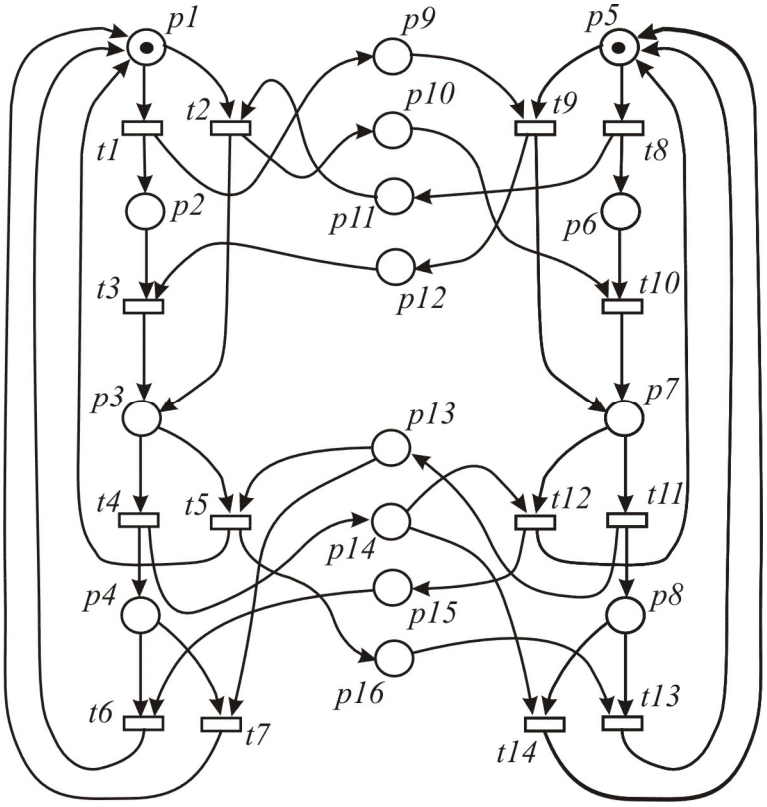


Рис. А.1. Модель протокола ECMA

Модель протокола ECMA в форме сети Петри изображена на рис. А.1. В модели, представленной на рис. А.1, можно выделить три основных части: левая взаимодействующая система – позиции $p_1 - p_4$, переходы $t_1 - t_7$; правая взаимодействующая система – позиции $p_5 - p_8$, переходы $t_8 - t_{14}$; коммуникационная подсистема – позиции $p_9 - p_{16}$. Смысловое описание элементов модели приведено в табл. А.1.

Таблица А.1

Описание элементов модели

Позиция	Описание	Переход	Описание
p_1, p_5	Начальное состояние систем	t_1, t_8	Послать запрос на соединение
p_2, p_6	Ожидание соединения	t_2, t_9	Принять запрос на соединение
p_3, p_7	Передача данных	t_3, t_{10}	Принять подтверждение соединения
p_4, p_8	Ожидание разъединения	t_4, t_{11}	Послать запрос на разъединение
p_9, p_{11}	Запрос на соединение	t_5, t_{12}	Принять запрос на разъединение
p_{10}, p_{12}	Подтверждение соединения	t_6, t_{13}	Принять подтверждение разъединения
p_{13}, p_{14}	Запрос на разъединение	t_7, t_{14}	Принять встречный запрос разъединения
p_{15}, p_{16}	Подтверждение разъединения		

Отметим, что разрыв соединения представлен более сложным фрагментом сети, поскольку учитывается ситуация, в которой запрос на разъединение при ожидании подтверждения разъединения интерпретируется как подтверждение (переходы t_7, t_{14}). В этом состоит особенность протоколов ECMA, позволяющая нормально функционировать в условиях коллизии запросов на разъединение.

Декомпозиция протокола. Выполним декомпозицию исходной модели протокола ECMA, представленной на рис. А.1, на минимальные функциональные подсети в соответствии с алгоритмом 2.1.

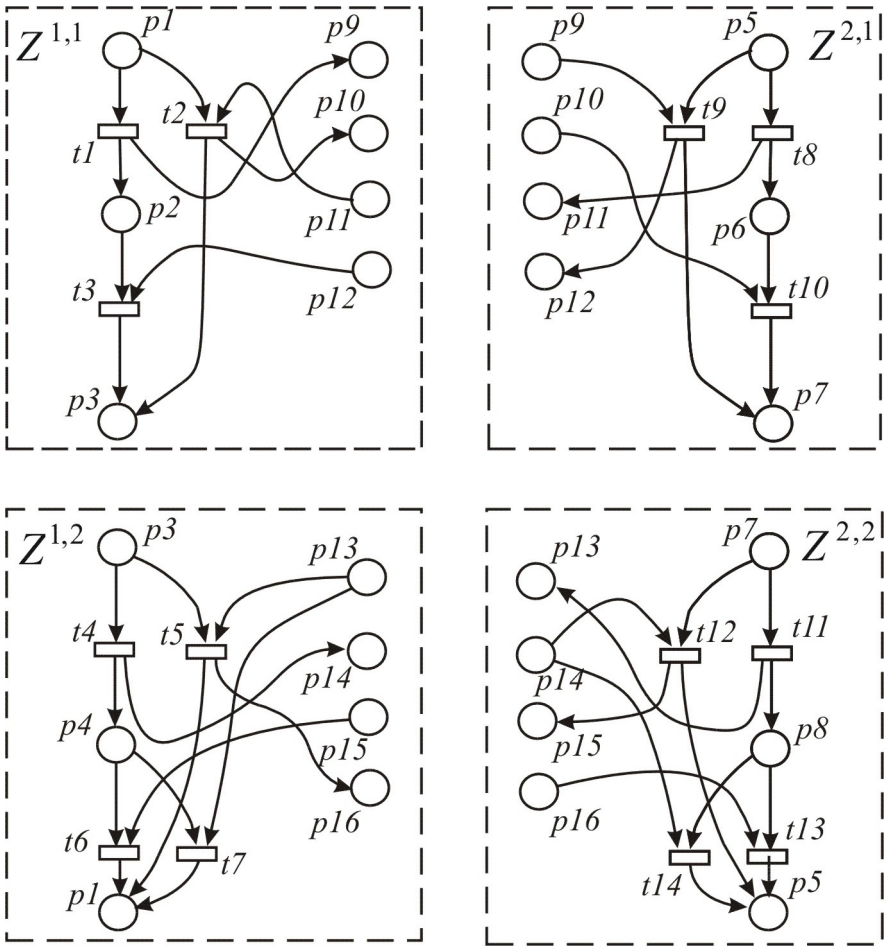


Рис. А.2. Декомпозиция протокола ECMA

Применение алгоритма 2.1 к модели протокола ECMA (рис. А.1) приводит к получению множества $\{Z^{1,1}, Z^{1,2}, Z^{2,1}, Z^{2,2}\}$, состоящего из четырёх минимальных функциональных подсетей, представленных на рис. А.2. Граф функциональных подсетей [93, 144, 150] изображён на рис. А.3. Заметим, что в силу симметрии процессов взаимодействия систем, пары подсетей $Z^{1,1}$ и $Z^{2,1}$, а также $Z^{2,1}$ и $Z^{2,2}$ являются изоморфными. Поэтому в дальнейшем необходимо исследовать свойства лишь двух из перечисленных четырёх подсетей.

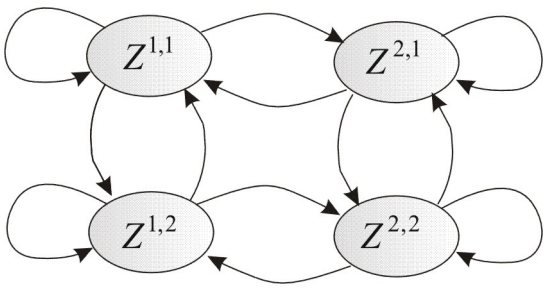


Рис. А.3. Граф функциональных подсетей

Различные способы композиции минимальных функциональных подсетей позволяет получить декомпозицию исходной модели на левую и правую взаимодействующие системы Z^1 и Z^2, а также декомпозицию на сеть, устанавливающую соединение и сеть, выполняющую разъединение Z'^1 и Z'^2, где $Z^1 = Z^{1,1} + Z^{1,2}$, $Z^2 = Z^{2,1} + Z^{2,2}$, $Z'^1 = Z^{1,1} + Z^{2,1}$, $Z'^2 = Z^{2,1} + Z^{2,2}$.

Таким образом, выполнена декомпозиция модели протокола ECMA на минимальные функциональные подсети; левую и правую взаимодействующие системы; подсети установления соединения и разъединения.

Инвариантность протокола. Инварианты [169] являются мощным инструментом исследования структурных свойств сетей Петри. Они позволяют определять ограниченность, консервативность, необходимые условия живости и отсутствия тупиков. Эти свойства являются существенными для анализа поведения реальных объектов, в особенности, коммуникационных протоколов [9, 29, 39]. В тех случаях, когда модель обладает внутренней симметрией, вследствие чего некоторые минимальные функциональные подсети являются изоморфными, описанный процесс целесообразно выполнять последовательно.

Используем изоморфность подсетей Z^1 и Z^2. Вначале вычислим инварианты подсети Z^1. Затем построим инвариант изоморфной сети Z^2. И, наконец, вычислим инвариант всей заданной сети Петри.

Инварианты подсетей $Z^{1,1}$ и $Z^{2,1}$ представим как

$$(x_1, x_2, x_3, x_9, x_{10}, x_{11}, x_{12}) = (z_1^1, z_2^1, z_3^1, z_4^1, z_5^1) \cdot R^{1,1},$$
$$(x_1, x_3, x_4, x_{13}, x_{14}, x_{15}, x_{16}) = (z_1^2, z_2^2, z_3^2) \cdot R^{1,2},$$

где матрицы $R^{1,1}$ и $R^{1,2}$ имеют вид:

$$R^{1,1} = \begin{vmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{vmatrix}, \quad R^{1,2} = \begin{vmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{vmatrix}.$$

Заметим, что компоненты вектора $\bar{x}$, соответствующие подсетям $Z^{1,1}$ и $Z^{2,1}$, выписаны в явном виде; они задают индексацию столбцов построенных матриц. Индексы строк соответствуют компонентам векторов $\bar{z}^1 = (z_1^1, z_2^1, z_3^1, z_4^1, z_5^1)$ и $\bar{z}^1 = (z_1^2, z_2^2, z_3^2)$.

Построим систему уравнений вида (3.13) для контактных позиций:

$$\begin{cases} z_1^1 + z_3^1 + z_4^1 - z_2^2 - z_3^2 = 0, \\ z_1^1 + z_2^1 + z_4^1 - z_1^2 - z_3^2 = 0. \end{cases}$$

Отметим, что в композиции подсетей $R^{1,1}$ и $R^{1,2}$ контактными являются позиции p_1 и p_3. Общее решение системы имеет вид

$$(z_1^1, z_2^1, z_3^1, z_4^1, z_5^1, z_1^2, z_2^2, z_3^2) = \bar{y} \cdot G^1,\ G^1 = \begin{vmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{vmatrix}.$$

Для вычисления базисных инвариантов сети Z^1 в соответствии с (3.16) построим из инвариантов подсетей $R^{1,1}$ и $R^{2,1}$ объединённую матрицу R^1:

$$R^1 = \begin{vmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix} \text{ либо } R^1 = \begin{vmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}.$$

Отметим, что разница между матрицами заключена в столбцах, соответствующих контактным позициям (p_1 и p_3). В первом случае инварианты контактных позиций вычисляются в соответствии с матрицей $R^{1,1}$, а во втором случае – в соответствии с $R^{2,1}$. Индексация столбцов соответствует вектору $(x_1, x_2, x_3, x_4, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16})$.

Матрица базисных решений имеет вид

$$H^1 = \begin{vmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}.$$

Отметим, что после вычисления произведения $G \cdot R$ в соответствии с (3.16) из матрицы удалены линейно-зависимые строки.

Далее аналогичным образом построим инварианты всей сети, являющейся композицией подсетей Z^1 и Z^2. Система уравнений для контактных позиций имеет вид:

$$\begin{cases} p_9: \ z_2^1 + z_4^1 + z_6^1 - z_5^2 - z_7^2 = 0, \\ p_{10}: \ z_4^1 + z_7^1 - z_2^2 - z_5^2 - z_6^2 = 0, \\ p_{11}: \ z_5^1 + z_7^1 - z_2^2 - z_4^2 - z_6^2 = 0, \\ p_{12}: \ z_2^1 + z_5^1 + z_6^1 - z_4^2 - z_7^2 = 0, \\ p_{13}: \ z_1^1 + z_2^1 + z_4^1 - z_1^2 - z_2^2 - z_5^2 = 0, \\ p_{14}: \ z_1^1 + z_2^1 + z_5^1 - z_1^2 - z_2^2 - z_4^2 = 0, \\ p_{15}: \ z_1^1 + z_2^1 + z_4^1 - z_1^2 - z_2^2 - z_5^2 = 0, \\ p_{16}: \ z_1^1 + z_2^1 + z_5^1 - z_1^2 - z_2^2 - z_4^2 = 0. \end{cases}$$

Решим систему уравнений, вычислим произведение $G \cdot R$ и удалим линейно-зависимые строки. Получим следующие базисные инварианты сети Петри:

$$H = \begin{vmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}.$$

Полученный результат совпадает с инвариантами, вычисленными обычными методами для всей сети, а также с инвариантами, полученными путём одновременной композиции всех четырёх минимальных функциональных подсетей.

Сеть Петри инвариантна, так как, например, инвариант

$$\bar{x}^* = (2 \ 1 \ 2 \ 1 \ 2 \ 1 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1),$$

являющийся суммой базисных инвариантов с номерами 1, 3 и 9, содержит все натуральные компоненты. Таким образом, модель протокола ECMA консервативна и ограничена.

Следует отметить, что хотя сеть является также и t-инвариантной, она содержит тупик, в котором маркированы позиции p_9 и p_{11}. Сеть приходит к тупиковой маркировке в результате срабатываний последовательности переходов $t_1 t_5$ либо $t_5 t_1$.

Оценка ускорения вычислений. Оценим полученные ускорения вычислений в предположении экспоненциальной сложности алгоритмов [162-166] решения систем линейных диофантовых уравнений в целых неотрицательных числах равной 2^q, где q – количество вершин сети.

Исходная сеть содержит 16 позиций, поэтому прямое вычисление инвариантов требует решения системы с 16 неизвестными. Композиция четырёх минимальных подсетей требует решения систем порядка 7 для нахождения инвариантов минимальных подсетей и решения системы порядка 12 для нахождения инвариантов контактных позиций. Последовательная композиция предполагает решение систем порядка 7 для нахождения инвариантов минимальных подсетей, решения системы порядка 5 для нахождения инвариантов контактных позиций первой композиции и решения системы порядка 8 для нахождения инвариантов контактных позиций второй композиции. Заметим, что при экспоненциальном росте функций сложность умножения матриц, представленная полиномом третьей степени, является несущественной и поэтому не учитывается в оценках.

Сложности вычислений для каждого из перечисленных способов нахождения инвариантов могут быть оценены с помощью следующих выражений:

$$S^I = 2^{16} \approx 65000,$$
$$S^{II} = 2^7 + 2^{12} \approx 4300,$$
$$S^{II} = 2^7 + 2^5 + 2^8 \approx 500.$$

Таким образом, использование декомпозиции позволило ускорить вычисления более чем в десять раз по сравнению с традиционными методами, а применение последовательной композиции позволило получить дополнительное почти десятикратное ускорение.

Следует отметить, что ускорения были вычислены для сети, насчитывающей менее двух десятков вершин. При исследовании сетей большой размерности ускорения могут быть весьма значительными, поскольку они оцениваются экспоненциальными функциями.

Таким образом, выполнена декомпозиция исходной модели Петри протокола ECMA на минимальные функциональные подсети, левую и правую взаимодействующие системы, подсети установления соединения и разъединения. Инвариантность исходной модели доказана на основе установленной инвариантности функциональных подсетей. Изоморфность некоторых подсетей позволила вычислить инварианты в процессе последовательной композиции сети. Выполнена оценка ускорения вычислений при использовании композиционных методов нахождения инвариантов.

Приложение Б. Теоретическое обоснование и оценки сложности метода Тудика

Известен метод, предложенный Тудиком [88] и позволяющий находить целые неотрицательные решения системы линейных диофантовых уравнений посредством преобразований матриц. Метод представлен без формального обоснования и поэтому характеризуется в литературе [110-112] как эвристический. В начале мы имеем единичную матрицу, которая при завершении вычислений содержит базисные решения. В настоящее время, не смотря на асимптотически экспоненциальную сложность, метод находит широкое применение в промышленных моделирующих системах [10, 112].

Б.1. Решение одного уравнения

Начнём с построения решений для одного уравнения. Итак, имеем уравнение

$$\bar{a} \cdot \bar{x} - \bar{b} \cdot \bar{y} = 0, \qquad (Б.1)$$

где $\bar{a}, \bar{x}, \bar{b}, \bar{y}$ это целые неотрицательные векторы, причём размерность $\bar{a}, \bar{x}$ равна m, а размерность $\bar{b}, \bar{y}$ равна n; $\bar{a}, \bar{b}$ векторы известных коэффициентов, а $\bar{x}, \bar{y}$ векторы неизвестных.

Теорема Б.1. Общее решение уравнения (Б.1) в неотрицательных целых числах имеет вид $\frac{\bar{z} \cdot G}{k}$, где k - общий делитель компонент вектора $\bar{z} \cdot G$, $\bar{z} = (z_1^1, z_2^1, ..., z_n^1, z_1^2, z_2^2, ..., z_n^2, ..., z_1^m, z_2^m, ..., z_n^m)$ - произвольный вектор неотрицательных целых, а матрица G имеет форму:

$$G = \begin{vmatrix}
b_1 & 0 & .. & 0 & a_1 & 0 & .. & 0 \\
b_2 & 0 & .. & 0 & 0 & a_1 & .. & 0 \\
: & : & : & : & : & : & : & : \\
b_n & 0 & .. & 0 & 0 & 0 & .. & a_1 \\
0 & b_1 & .. & 0 & a_2 & 0 & .. & 0 \\
0 & b_2 & : & 0 & 0 & a_2 & .. & 0 \\
: & : & : & : & : & : & : & ; \\
0 & b_n & .. & 0 & 0 & 0 & .. & a_2 \\
 & . & & & & . & & \\
 & . & & & & . & & \\
 & . & & & & . & & \\
0 & 0 & .. & b_1 & a_m & 0 & .. & 0 \\
0 & 0 & .. & b_2 & 0 & a_m & .. & 0 \\
: & : & : & : & : & : & : & 0 \\
0 & 0 & .. & b_n & 0 & 0 & .. & a_m
\end{vmatrix}$$

Иными словами, строка матрицы G является базисным решением для пары (b_i, a_j). Краткое описание структуры матрицы G можно представить следующим образом. Пусть $\bar{g}^l$ обозначает i-ю строку матрицы G. Тогда $\bar{g}^l, l = \overline{1, m \times n}$ имеет два ненулевых компонента:

$$g_{l,((l-1)\ div\ n)+1} = b_{((l-1)\ \bmod\ n)+1},$$
$$g_{l,m+((l-1)\ \bmod\ n)+1} = a_{((l-1)\ div\ n)+1}$$

Доказательство. Итак $(\bar{x}, \bar{y}) = \dfrac{\bar{z} \cdot G}{k}$. Пусть $(\bar{c}, \bar{d})$ произвольное решение уравнения (Б.1). Покажем, что существуют такие $k, \bar{z}$, что

$$k \cdot (\bar{c}, \bar{d}) = \bar{z} \cdot G. \qquad \text{(Б.2)}$$

Запишем более детальное покомпонентное представление (Б.2), выделив суммы, соответствующие переменным $\bar{c}, \bar{d}$:

$$\begin{cases} \sum_i b_i \cdot z_i^j - c_j \cdot k = 0, \quad j = \overline{1, m} \\ \sum_j a_j \cdot z_i^j - d_i \cdot k = 0, \quad i = \overline{1, n} \end{cases} \qquad \text{(Б.3)}$$

Выполним доказательство конструктивно. Укажем конкретное решение системы (Б.3):

$$\begin{cases} k = \bar{a} \cdot \bar{c} \quad (or \quad k = \bar{b} \cdot \bar{d}) \\ z_i^j = c_j \cdot d_i, \quad j = \overline{1, m}, i = \overline{1, n} \end{cases} \qquad \text{(Б.4)}$$

Действительно, значения $k, \bar{z}$ могут быть эффективно вычислены в соответствии с (Б.4) для произвольного решения $(\bar{c}, \bar{d})$ уравнения (Б.1).

Теперь покажем, что значения (Б.4) являются решением уравнения (Б.3):

$$\sum_i b_i \cdot c_j \cdot d_i - c_j \cdot \bar{a} \cdot \bar{c} = c_j \cdot \bar{b} \cdot \bar{d} - c_j \cdot \bar{a} \cdot \bar{c} = c_j \cdot (\bar{b} \cdot \bar{d} - \bar{a} \cdot \bar{c}) = 0, \quad j = \overline{1, m}$$
$$\sum_j a_j \cdot c_j \cdot d_i - d_i \cdot \bar{a} \cdot \bar{c} = d_i \cdot \bar{a} \cdot \bar{c} - d_i \cdot \bar{a} \cdot \bar{c} = 0, \quad i = \overline{1, n}$$

Терема доказана в силу произвольности выбора решения $(\bar{c}, \bar{d})$. □

Удобно ввести специальную операцию над векторами с целыми неотрицательными компонентами, сокращающую значения на общий делитель компонентов. Для обозначения операции выберем угловые скобки. Итак, $\bar{y} = <\bar{x}>$ тогда и только тогда когда существует натуральное k, такое,

что $k \cdot \bar{y} = \bar{x}$. Следовательно, результатом выполнения операции сокращения в общем случае является множество векторов. Заметим, что это множество всегда конечно.

Тогда решение уравнения (Б.1) в соответствии с теоремой Б.1 можно представить в форме

$$\bar{x} = < \bar{z} \cdot G >$$

Теорема Б.2. Базис решений уравнения (Б.1), представленный в теореме Б.1, является минимальным.

Доказательство. Выберем в матрице G произвольное базисное решение

$$\bar{g}_i^j = (0,...,b_i,...,0,0,...,a_j,...,0) \qquad (Б.5)$$

Покажем, что решение (Б.5) не может быть получено из оставшихся базисных решений посредством линейной комбинации с целыми неотрицательными коэффициентами и сокращения на общий делитель.

Действительно, любое решение, которое содержит ненулевую компоненту j, содержит также некоторую ненулевую компоненту l, причём $l \neq j$, в силу построения матрицы G. Таким образом, мы не можем получить нулевое значение компоненты l с помощью операций сложения, вычитания, деления на множестве неотрицательных целых. □

Таким образом, в настоящем получен формальный метод решения в целых неотрицательных числах одного уравнения. Минимальный базис состоит из $m \times n$ векторов, представленных матрицей G.

Б.2. Решение системы уравнений

Рассмотрим систему уравнений

$$\bar{x} \cdot A = 0, \qquad (Б.6)$$

где A заданная матрица целых размерности $m \times n$, $\bar{x}$ целый неотрицательный вектор неизвестных. Заметим, что уравнениям системы соответствуют столбцы матрицы A.

В отличие от (Б.1), каждое из уравнений системы (Б.6) может содержать неизвестные с нулевыми коэффициентами. Значения таких неизвестных при решении отдельного уравнения могут быть выбраны произвольно из множества целых неотрицательных чисел. То есть такой переменной x_i соответствует отдельное базисное решение, имеющее единичное значение компоненты i; остальные компоненты базисного решения равны нулю.

Введём следующие преобразования матриц. Будем получать матрицу D из матрицы A с помощью двух элементарных преобразований:

I) Переписать строку y матрицы A в строку v матрицы D:

$\bar{l}^v \leftarrow \bar{l}^y$

II) Записать сумму строк y, z, умноженных на константы c_y, c_z, матрицы A в строку v матрицы D:

$\bar{l}^v \leftarrow c_y \cdot \bar{l}^y + c_z \cdot \bar{l}^z$

В таком случае матрица D может быть получена из матрицы A путём домножения слева на матрицу преобразований R. При этом матрица R имеет следующие ненулевые компоненты:

а) Для каждого преобразования I:

$r_{v,y} = 1$

б) Для каждого преобразования II:

$r_{v,y} = c_y, \ \ r_{v,z} = c_z$

Это может быть легко доказано путём детального рассмотрения представления матрицы R:

$$D = R \cdot A, \ \ d_{i,j} = \sum_k r_{i,k} \cdot a_{k,j}$$

Заметим, что матрица G базисных решений уравнения (Б.1) представляет собой матрицу преобразований II. Если же переменная входит в уравнение с нулевым коэффициентом, то её значение может быть выбрано произвольно. Этот случай соответствует преобразованиям I, а матрица преобразований переменных является единичной.

Используемый далее метод решения системы имеет некоторое сходство с методом Гаусса. Он основан на решении отдельного уравнения и подстановке полученного общего решения в оставшуюся часть системы. Процесс продолжается до тех пор, пока все уравнения не будут решены.

В начале рассмотрим решение одного, для определённости первого, уравнения системы (Б.6). То есть, имеем уравнение:

$$\bar{x} \cdot \bar{a}^1 = 0$$

Построим множества индексов переменных I^+, I^0, I^- такие что

$$I^+ = \{i \mid a_{i,1} > 0\}, \ \ I^0 = \{i \mid a_{i,1} = 0\}, \ \ I^- = \{i \mid a_{i,1} < 0\}$$

Далее, в соответствии с теоремой Б.1 построим матрицу решений, соответствующую преобразованиям I, II:

$$R = \left| \begin{array}{c} (I^0)^I \\ \hline (I^+ \times I^-)^{II} \end{array} \right| = \left| \begin{array}{c} E \\ \hline G \end{array} \right|$$

Тогда

$$\bar{x} =< \bar{z} \cdot R > \text{ или } k \cdot \bar{x} = \bar{z} \cdot R . \qquad (Б.7)$$

Домножим систему (Б.6) на натуральное число k и, подставив в неё выражение (Б.7), получим:

$$\bar{z} \cdot R \cdot A = 0$$

или

$$\bar{z} \cdot D = 0, \qquad (Б.8)$$

где $D = R \cdot A$. Следует отметить, что в результате подстановки в систему решений уравнения мы перешли от неизвестных $\bar{x}$ к новым неизвестным $\bar{z}$.

Теорема Б.3. Системы (Б.7), (Б.8) эквивалентны исходной системе (Б.6).
Доказательство. Описанные выше преобразования доказывают необходимость условия теоремы. Докажем его достаточность. Заменим в (Б.8) $\bar{z} \cdot R$ в соответствии с (Б.7) и получим

$$k \cdot \bar{x} \cdot A = 0$$

Так как k натуральное число, разделим полученное уравнение на k и получим (Б.6). □

Рассмотрим процесс последовательного решения уравнений системы (Б.6):

$$\bar{x} =< \bar{z}^1 \cdot R^1 >$$
$$\bar{z}^1 \cdot R^1 \cdot A = 0$$

Продолжая, таким образом, мы получим

$$\bar{z}^n \cdot R^n \cdot R^{n-1} \cdots R^1 \cdot A = 0 .$$

Обозначим

$$\bar{z} := \bar{z}^n , \; R := R^n \cdot R^{n-1} \cdots R^1$$

и получим

$$\bar{z} \cdot R \cdot A = 0, \; R \cdot A = 0$$

Так как на каждом шаге мы решали одно уравнение системы, обнуляя соответствующий столбец матрицы. Таким образом, мы имеем

$$\bar{z} \cdot 0 = 0$$

Решением полученной системы является произвольный вектор $\bar{z}$ целых неотрицательных чисел. Тогда решение исходной системы (Б.6) можно представить в виде

$$\bar{x} = < \bar{z} \cdot R > \text{ или } k \cdot \bar{x} = \bar{z} \cdot R . \qquad \text{(Б.9)}$$

Так как на каждом шаге в соответствии с теоремой Б.3 использованы эквивалентные преобразования, описанные выкладки доказывают следующую теорему.

Теорема Б.4. Выражение (Б.9) представляет собой общее решение системы (Б.6) в целых неотрицательных числах.

Таким образом, для решения системы необходимо построить матрицу R. Матрица R содержит базисные решения системы. Заметим, что для генерации частных решений необходимо использовать как линейную комбинацию базисных решений с целыми неотрицательными коэффициентами, так и дополнительную операцию сокращения на общий делитель компонент вектора.

Далее на примере будет показано, что существуют решения, которые не могут быть получены в результате использования только линейной комбинации.

Б.3. Описание алгоритма

Выражение (Б.9) можно представить в форме

$$\bar{x} = < \bar{z} \cdot R \cdot E > ,$$

где E единичная матрица размерности $m \times m$. Рассмотрим процесс формирования матрицы R:

$$R = R^n \cdot R^{n-1} \cdots R^1 \cdot E$$

Таким образом, для того, чтобы вычислить матрицу R необходимо повторить все преобразования над единичной матрицей E. Преобразования на каждом шаге удобно выполнять сразу над двумя матрицами, не сохраняя, таким образом, промежуточные матрицы преобразований. В результате описанных рассуждений мы приходим к алгоритму, представленному Тудиком в работе [88]:

Алгоритм Б.1 (Тудика):
Шаг 0. Положим $D := A$, $R := E$.

Шаг 1. Если $D=0$, то *Останов*. Матрица R является искомой матрицей базисных решений системы.

Шаг 2. Если все столбцы матрицы D содержат ненулевые коэффициенты одного знака, то *Останов*. Система имеет только тривиальное решение.

Шаг 3. Выберем столбец j матрицы D с минимальным значением произведения $\left|I^+\right|\times\left|I^-\right|$, где $I^+=\{i\,|\,a_{i,j}>0\}$, $I^0=\{i\,|\,a_{i,j}=0\}$, $I^-=\{i\,|\,a_{i,j}<0\}$.

Шаг 4. Построим матрицу D' следующим образом: скопируем в матрицу D' строки I^0 матрицы D, затем допишем дополнительные строки для каждой комбинации $(k,r), k\in I^+, r\in I^-$, построенные как $\left|a_{r,j}\right|\cdot\bar{l}^k+\left|a_{k,j}\right|\cdot\bar{l}^r$.

Шаг 5. Аналогичные преобразования выполним над матрицей R для получения матрицы R'.

Шаг 6. Положим $D:=D', R:=R'$ и *перейдём к Шагу 1*.

Отметим, что на Шаге 3 выбор столбца обеспечивает минимальное количество новых решений соответствующего уравнения. Кроме того, в соответствии с результатами раздела 3 строки матриц D и R могут быть сокращены совместно на общий делитель компонентов вектора на любом проходе алгоритма. Возможно также сокращение значений столбцов матрицы D.

Отличительной особенностью полученных результатов является способ использования матрицы R для генерации произвольного частного решения. Линейная комбинация с целыми неотрицательными коэффициентами расширена дополнительной операцией сокращения на общий делитель компонент вектора. Так как операция деления на натуральное число не производит новые нулевые или ненулевые компоненты, то дополнительная операция сокращения на общий делитель не влияет на суппорты полученных инвариантов. Напомним, что суппортом инварианта [169] является множество ненулевых компонентов решения.

Б.4. Пример вычисления инвариантов

Для иллюстрации полученных результатов решим следующую систему уравнений:

$$\begin{cases}5\cdot x_1+5\cdot x_2-2\cdot x_3-2\cdot x_4=0\\2\cdot x_1-5\cdot x_2-5\cdot x_3-5\cdot x_5=0\\5\cdot x_1+2\cdot x_2+2\cdot x_3-2\cdot x_4-5\cdot x_5=0\end{cases} \qquad (Б.10)$$

В матричной форме (Б.6) имеем следующие компоненты:

$$\bar{x} = (x_1, x_2, x_3, x_4, x_5),\ A = \begin{pmatrix} 5 & 5 & -2 & -2 & 0 \\ 2 & -5 & -5 & 0 & -5 \\ 5 & 2 & 2 & -2 & -5 \end{pmatrix}^T.$$

Запишем пару матриц (D, R):

$$\left|\begin{array}{ccc|ccccc} 5 & 2 & 5 & 1 & 0 & 0 & 0 & 0 \\ 5 & -5 & 2 & 0 & 1 & 0 & 0 & 0 \\ -2 & -5 & 2 & 0 & 0 & 1 & 0 & 0 \\ -2 & 0 & -2 & 0 & 0 & 0 & 1 & 0 \\ 0 & -5 & -5 & 0 & 0 & 0 & 0 & 1 \end{array}\right|$$

Минимальное значение произведения $\left|I^+\right| \times \left|I^-\right| = 3$ получаем для второго столбца; он будет выбран в качестве столбца j. Таким образом $j = 2$. Вычислим новые значения матриц:

$$\left|\begin{array}{ccc|ccccc} -2 & 0 & -2 & 0 & 0 & 0 & 1 & 0 \\ 35 & 0 & 29 & 5 & 2 & 0 & 0 & 0 \\ 21 & 0 & 29 & 5 & 0 & 2 & 0 & 0 \\ 25 & 0 & 15 & 5 & 0 & 0 & 0 & 2 \end{array}\right|$$

Выберем столбец $j = 1$ и получим:

$$\left|\begin{array}{ccc|ccccc} 0 & 0 & -12 & 10 & 4 & 0 & 35 & 0 \\ 0 & 0 & 16 & 10 & 0 & 4 & 21 & 0 \\ 0 & 0 & -20 & 10 & 0 & 0 & 25 & 4 \end{array}\right|$$

После обработки столбца $j = 3$ получим:

$$\left|\begin{array}{ccc|ccccc} 0 & 0 & 0 & 280 & 64 & 48 & 203 & 0 \\ 0 & 0 & 0 & 360 & 0 & 80 & 820 & 64 \end{array}\right|$$

И, наконец, сократив строки на 4, получим базисные решения системы:

$$R = \begin{pmatrix} 70 & 16 & 12 & 203 & 0 \\ 90 & 0 & 20 & 205 & 16 \end{pmatrix}.$$

Укажем вектор

$$\bar{b} = (20 \quad 2 \quad 4 \quad 51 \quad 2),$$

который является решением системы (Б.10), но не может быть представлен как линейная комбинация базисных решений с целыми неотрицательными коэффициентами. С использованием операции сокращения его можно представить как:

$$\bar{b} = (\bar{r}^1 + \bar{r}^2)/8.$$

Аналогичным образом формируются следующие решения:

$$(150,24,28,407,8) = (3 \cdot \bar{r}^1 + \bar{r}^2)/2,$$
$$(170,8,36,409,24) = (\bar{r}^1 + 3 \cdot \bar{r}^2)/2,$$
$$(130,4,28,307,20) = (4 \cdot \bar{r}^1 + 20 \cdot \bar{r}^2)/16.$$

Б.5. Оценка вычислительной сложности метода Тудика

Построим точные оценки вычислительной сложности метода Тудика и определим условия, при которых метод может быть эффективно применён для анализа свойств сетей Петри.

1) Условия полиномиальной сложности

Следует отметить, что в системе (Б.6) столбцы матрицы A представляют уравнения, а строки соответствуют переменным. На каждом проходе алгоритма обнуляется один из столбцов, при этом образуются новые переменные. В заключение мы приходим к нулевой матрице D и новые переменные, таким образом, становятся свободными.

Следовательно, количество проходов алгоритма не превышает n. Основными источниками роста вычислительной сложности выступают, во-первых, увеличения числа строк матрицы, а во-вторых, рост абсолютного значения элементов.

Обозначим q максимальное количество переходов либо позиций сети $q = \max(|P|,|T|)$, а $m(i)$ – количество строк матрицы D на i-м проходе алгоритма. Тогда имеем следующее рекуррентное соотношение:

$$\begin{cases} m(i+1) = m(i) - (|I^-| + |I^+|) + |I^-| \cdot |I^+|, \\ m(0) = q. \end{cases} \qquad (Б.11)$$

Действительно, на очередном проходе $|I^-| + |I^+|$ строк матрицы заменяются новыми $|I^-| \cdot |I^+|$ строками. Знак выражения $h(i) = |I^-| \cdot |I^+| - |I^-| + |I^+|$

определяет, увеличивается, либо уменьшается количество строк матрицы. Выполним оценку сложности алгоритма в лучшем и в худшем случаях.

В лучшем случае $h(i) \le 0$. То есть:

$$\left|I^-\right| + \left|I^+\right| \ge \left|I^-\right| \cdot \left|I^+\right| \qquad \text{(Б.12)}$$

Тогда сложность алгоритма полиномиальна и не превышает $o(q^3)$, так как количество строк матрицы не возрастает и на каждом проходе выполняется около q^2 операций.

Рассмотрим сочетание значений $\left|I^-\right|$ и $\left|I^+\right|$, обеспечивающее выполнение неравенства (Б.12). Возможно два варианта:

1. $\left|I^-\right| = 1$ либо $\left|I^+\right| = 1$.
2. $\left|I^-\right| = \left|I^+\right| = 2$.

Таким образом, каждый из переходов имеет не более четырёх инцидентных дуг, либо имеет ровно одну входящую (одну исходящую) дугу. Причём эти ограничения должны оставаться справедливыми в процессе выполнения алгоритма.

2) Сложность в худшем случае

В худшем случае $\left|I^+\right| = \left|I^-\right| = \dfrac{m(i)}{2}$, так как при таком соотношении размеров множеств достигается максимум функции $h(i)$ на текущем проходе алгоритма, ведь $\left|I^+\right| + \left|I^-\right| \le m(i)$. Таким образом, из (Б.11) получаем:

$$\begin{cases} m(i+1) = \left(\dfrac{m(i)}{2}\right)^2, \\ m(0) = n. \end{cases} \qquad \text{(Б.13)}$$

Например: $m(1) = \dfrac{n^2}{2^2}$, $m(2) = \dfrac{n^4}{2^6}$, $m(3) = \dfrac{n^8}{2^{14}}, \ldots$

Лемма Б.1. Рекуррентное выражение (Б.13) можно представить в явной форме записи как:

$$m(i) = \frac{n^{2^i}}{2^{2^{i+1}-2}} \qquad \text{(Б.14)}$$

Доказательство. Построим доказательство на основе индукции. Предположим, что выражение (Б.14) представляет рекуррентное соотношение (Б.13) для $i = 1,2,...,j$. Пусть

$$m(j) = \frac{n^{2^j}}{2^{2^{j+1}-2}}.$$

Докажем справедливость утверждения леммы для шага $j+1$. В соответствии с (Б.13):

$$m(j+1) = \left(\frac{m(j)}{2}\right)^2 = \left(\frac{\frac{n^{2^j}}{2^{2^{j+1}-2}}}{2}\right)^2 = \frac{\left(n^{2^j}\right)^2}{\left(2^{2^{j+1}-2}\right)^2 \cdot 2^2} = \frac{n^{2^j \cdot 2}}{2^{(2^{j+1}-2)2+2}} = \frac{n^{2^{j+1}.}}{2^{2^{j+2}-2}}.$$

□

Тогда сложность алгоритма как количество выполняемых операций может быть оценена следующим выражением:

$$Y_T(q) = \sum_{i=i,n} n \cdot \frac{n^{2^i}}{2^{2^{i+1}-2}} \sim q^2 \cdot \frac{q^{2^q}}{2^{2^{q+1}}} \sim \frac{q^{2^q}}{2^{2^{q+1}}}.$$

Представим

$$q^{2^q} = \left(2^{\log_2 q}\right)^{2^q} = 2^{2^q \cdot \log_2 q} = \left(2^{2^q}\right)^{\log_2 q}.$$

А также

$$2^{2^{q+1}} = 2^{2^q \cdot 2} = \left(2^{2^q}\right)^2.$$

Тогда

$$Y_T(q) = \left(2^{2^q}\right)^{\log_2 q} \Big/ \left(2^{2^q}\right)^2 = \left(2^{2^q}\right)^{\log_2 q - 2} = 2^{2^q \cdot (\log_2 q - 2)} \qquad \text{(Б.15)}$$

Полученный результат можно представить в виде следующей теоремы.

Теорема Б.5. Временная сложность метода Тудика не превышает величины $Y_T(q)$, заданной выражением (Б.15), где q - количество вершин сети.

Таким образом, метод Тудика имеет асимптотически экспоненциальную временную сложность. Представленное в (Б.15) значение $Y_T(q)$ действительно растёт довольно быстро с ростом q. Например: $Y_T(5) \approx 10^9$, $Y_T(10) \approx 10^{300}$. Таким образом, даже для небольших сетей вычисления могут потребовать тысячелетий.

Оценим емкостную сложность метода. Пусть a – максимальное по модулю значение числа в матрице A: $a = \max_{i,j} \left| a_{i,j} \right|$. Оценим значение максимального по модулю числа в матрицах D, R на i-м проходе алгоритма, обозначенное как $v(i)$. В соответствии с описанием шага 4 алгоритма максимальное значение числа на предыдущем и последующем проходах алгоритма связаны рекуррентным следующим соотношением:

$$\begin{cases} v(i+1) = 2 \cdot (v(i))^2, \\ v(0) = a. \end{cases} \qquad \text{(Б.16)}$$

Лемма Б.2. Рекуррентное выражение (Б.16) можно представить в явной форме записи как:

$$v(i) = 2^{2^{i+1}-1} \cdot l^{2^i} \qquad \text{(Б.17)}$$

Также как в лемме Б.1 доказательство может быть выполнено индуктивно. Обозначим $k = \log_2 a$. Тогда (Б.17) представимо в виде:

$$v(i) = 2^{2^{i+1}-1} \cdot \left(2^k\right)^{2^i} = 2^{2^{i+1}-1} \cdot 2^{k \cdot 2^i} = 2^{2^{i+1}+k \cdot 2^i - 1} = 2^{(k+2)\cdot 2^i - 1}.$$

Так как $v(i)$ возрастает с ростом i, наибольшее значение достигается на проходе с номером n:

$$v(n) = 2^{(k+2)\cdot 2^n - 1}.$$

Тогда для хранения числа потребуется не менее $b(n)$ бит, где

$$b(n) = \log_2 v(n) = (k+2) \cdot 2^n - 1 \sim (\log_2 a + 2) \cdot 2^n.$$

Количество строк матриц на заключительном проходе алгоритма представлено выражением (Б.15). Тогда емкостную сложность метода можно оценить следующим образом:

$$Z_T(q) = b(q) \cdot Y_T(q) \cdot n \approx 2^{2^q \cdot (\log_2 q - 2)} \cdot (\log_2 a + 2) \cdot 2^q = (\log_2 a + 2) \cdot 2^{2^q \cdot (\log_2 q - 2) + q} \qquad \text{(Б.18)}$$

Полученный результат можно представить в виде следующей теоремы.

Теорема Б.6. Емкостная сложность метода Тудика не превышает величины $Z_T(q)$, заданной выражением (Б.18), где q – количество вершин сети.

Таким образом, получены оценки временной и емкостной сложности метода Тудика в худшем случае. Оценки являются экспоненциальными, что затрудняет практическое применение метода при исследовании сетей большой размерности.

3) Оценка сложности для разрежённых матриц

Сложность в худшем случае оценивалась для матрицы A, большинство значений которой являются ненулевыми. В сетях Петри, моделирующих реальные объекты, связи вершин локализованы. Количество инцидентных

вершин, как правило не превышает некоторой заранее заданной константы, значение которой во много раз меньше размерности сети. Таким образом, исходная матрица является разрежённой.

Можно предположить, что оценки сложности метода Тудика для разрежённых матриц будут гораздо более оптимистичными, чем полученные в предыдущем разделе. Тем более, что в литературе имеются сообщения об успешном практическом применении метода для сетей имеющих несколько сот вершин [22, 67, 112].

Итак, пусть, каждая из вершин сети Петри имеет не более k входящих и не более k исходящих дуг. Таким образом, каждый столбец и каждая строка матрицы A содержит не более $2 \cdot k$ ненулевых элементов, причём k из них отрицательны, а остальные k положительны. Рассмотрим каким образом будет изменятся заполненность матриц D, R ненулевыми элементами, так как именно она существенно влияет на оценки сложности алгоритма.

С одной стороны, на каждом проходе обнуляется столбец матрицы D; таким образом, количество гарантированно нулевых элементов в строках возрастает. С другой стороны, при образовании новой строки матрицы D на шаге 4 суммируются значения двух строк. При суммировании строк количество ненулевых элементов $u(i)$ в худшем случае удваивается:

$$\begin{cases} u(i+1) = 2 \cdot v(i), \\ u(0) = 2 \cdot k. \end{cases}$$

Тогда

$$u(i) = 2^{i+1} \cdot k \qquad \text{(Б.19)}$$

Темпы заполнения строки ненулевыми элементами, представленные экспоненциальной функцией (Б.19), значительно опережают темпы обнуления столбцов, характеризуемые линейной функцией $f(i) = i$.

Таким образом, уже на проходе r новые строки матрицы будут заполнены ненулевыми элементами, где значение r задаётся неравенством

$$u(r) \geq n - r .$$

Для простоты оценки, рассмотрим заполнение всей строки: $2^{r+1} \cdot k \geq n$; тогда $2^r \geq \dfrac{n}{2 \cdot k}$ или:

$$r \geq \log_2 \frac{n}{2 \cdot k} .$$

При этом на проходе r появляется не менее k^2 заполненных строк и дальнейшие оценки сложности соответствуют оценкам (Б.15), (Б.18), выполненным для худшего случая.

Так, например при $n = 1000$ и $k = 5$ получаем $r \geq 7$. То есть после седьмого прохода новые строки матрицы D будут заполнены ненулевыми элементами.

Итак, обнаружен эффект, состоящий в экспансии ненулевых элементов в процессе применения метода Тудика к разрежённой матрице и названный эффектом заливки матрицы. Полученные результаты позволяют сделать вывод, что реальная вычислительная сложность метода Тудика незначительно отличается от оценок, выполненных для худшего случая и является экспоненциальной.

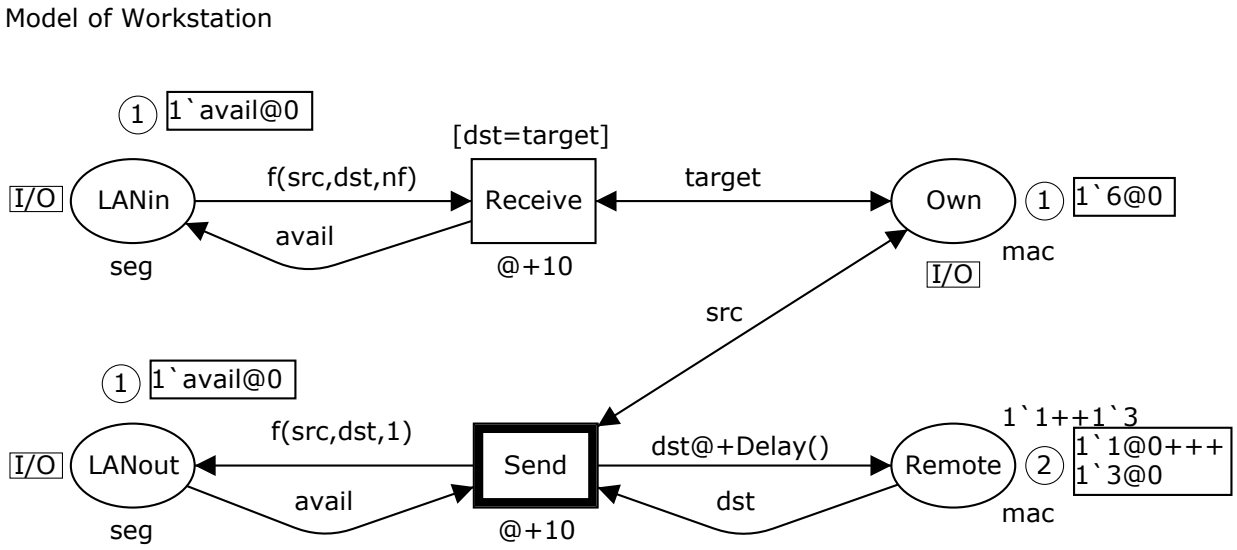
Таким образом, представлено теоретическое обоснование метода Тудика. Метод известен как эвристический и предназначен для решения систем линейных однородных диофантовых уравнений в целых неотрицательных числах. Он широко используется в промышленных системах автоматизированного анализа свойств сетей Петри при вычислении сетевых инвариантов. Показано, что для генерации всех частных решений системы необходимо расширить линейную комбинацию базисных решений дополнительной операцией сокращения на общий делитель компонент вектора.

Получены оценки временной и емкостной сложности метода Тудика, предназначенного для решения линейных однородных систем диофантовых уравнений в целых неотрицательных числах. Определены условия полиномиальной сложности метода. Показано, что в общем случае метод Тудика имеет асимптотически экспоненциальную сложность, что затрудняет его практическое применение. Кроме того, оценена сложность метода для разрежённых матриц. Показано, что имеет место эффект заливки матрицы, в результате которого разрежённая матрица на первых нескольких шагах алгоритма заполняется ненулевыми значениями и дальнейшие оценки сложности соответствуют оценкам, выполненным для худшего случая.

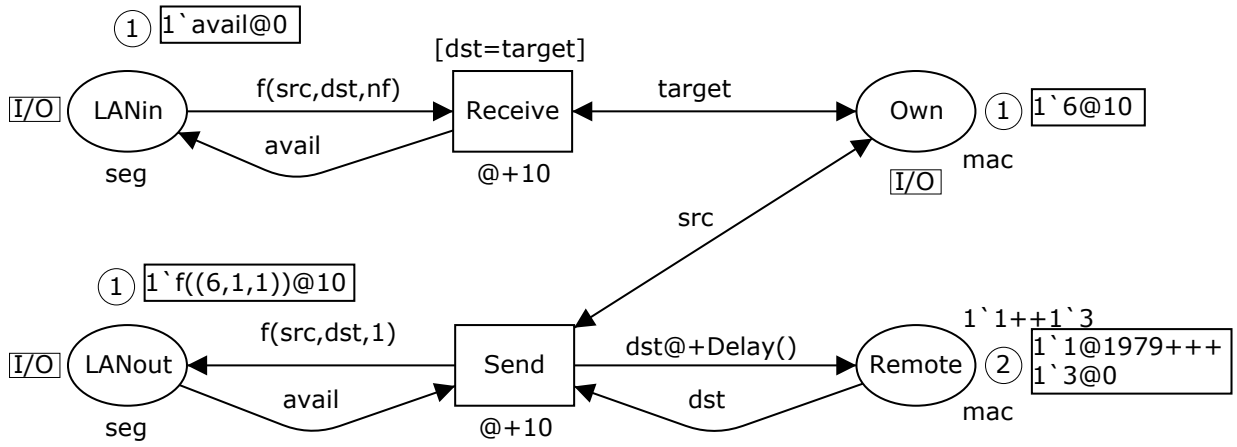
Приложение В. Трассировка прохождения фреймов в модели коммутируемой Ethernet

Для отладки модели выполнена трассировка процесса взаимодействия клиент-сервер. Далее представлена трасса прохождения фрейма запроса рабочей станции WS4 к серверу S1 и обратного прохождения первого из фреймов ответа сервера S1.

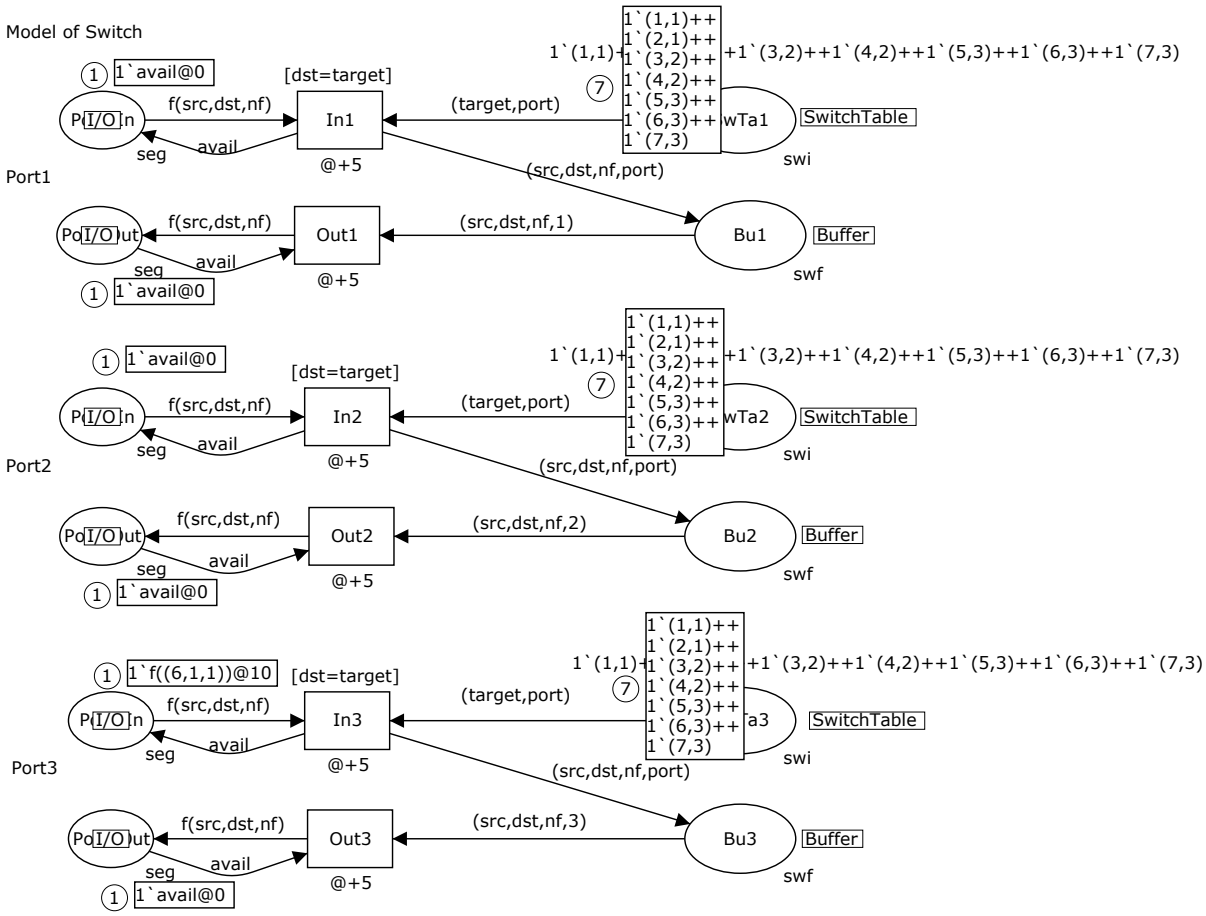
1) Исходное состояние рабочей станции WS4:



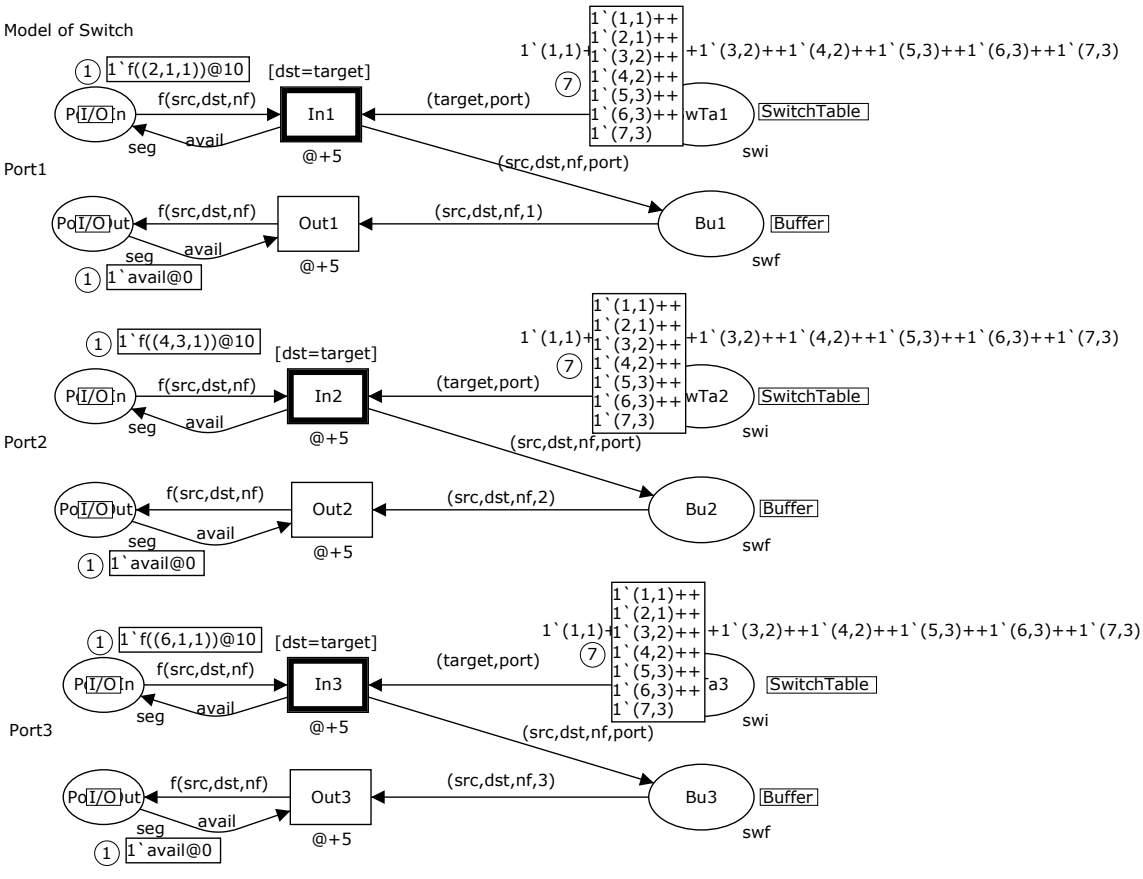
2) Фрейм запроса сгенерирован рабочей станцией WS4 и передаётся в сегменте 3:



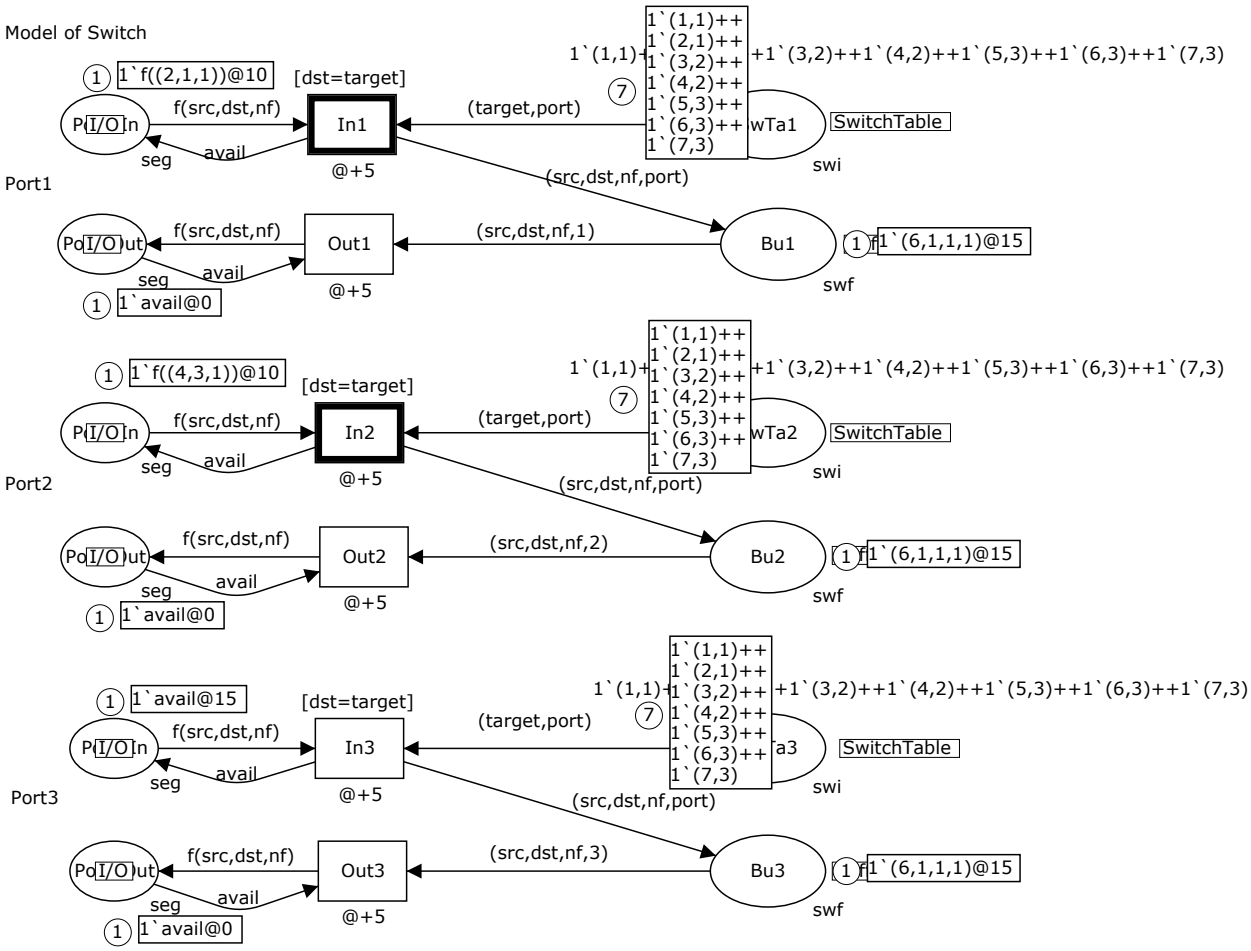
3) Фрейм запроса рабочей станции WS4 получен коммутатором:



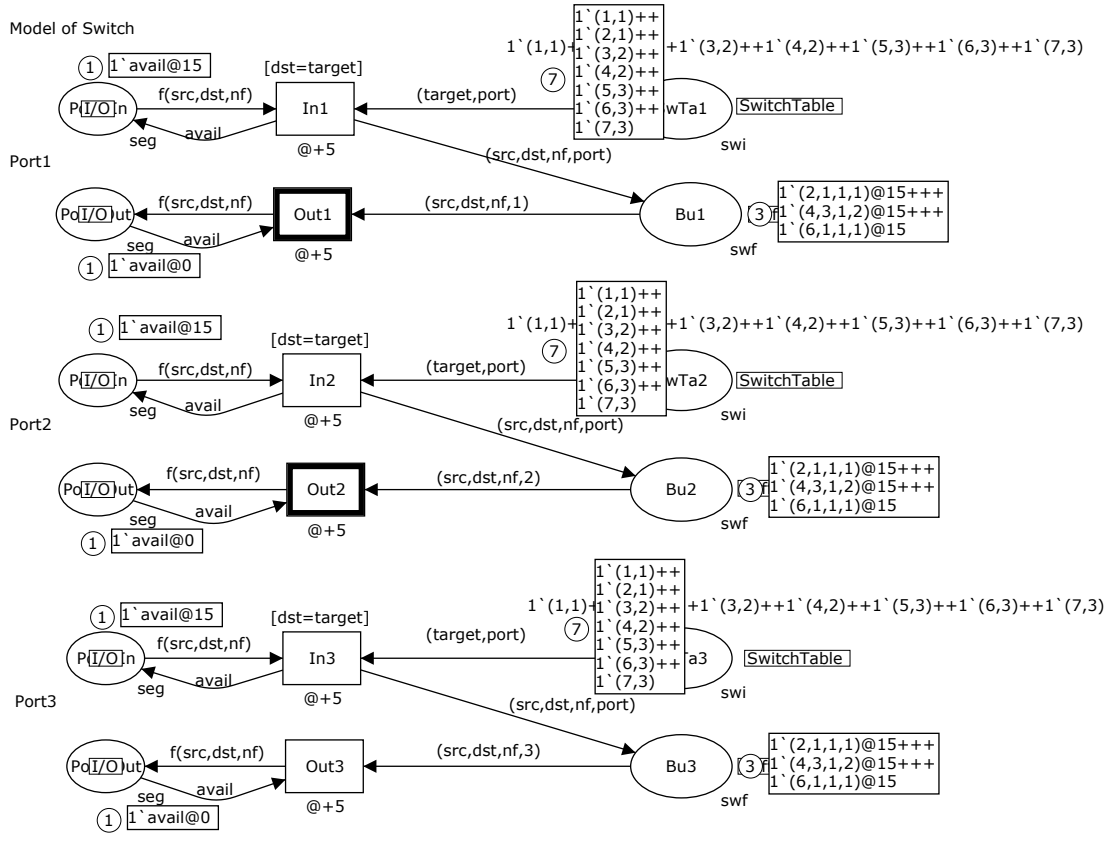
4) Коммутатором получены фреймы запросов других рабочих станций:



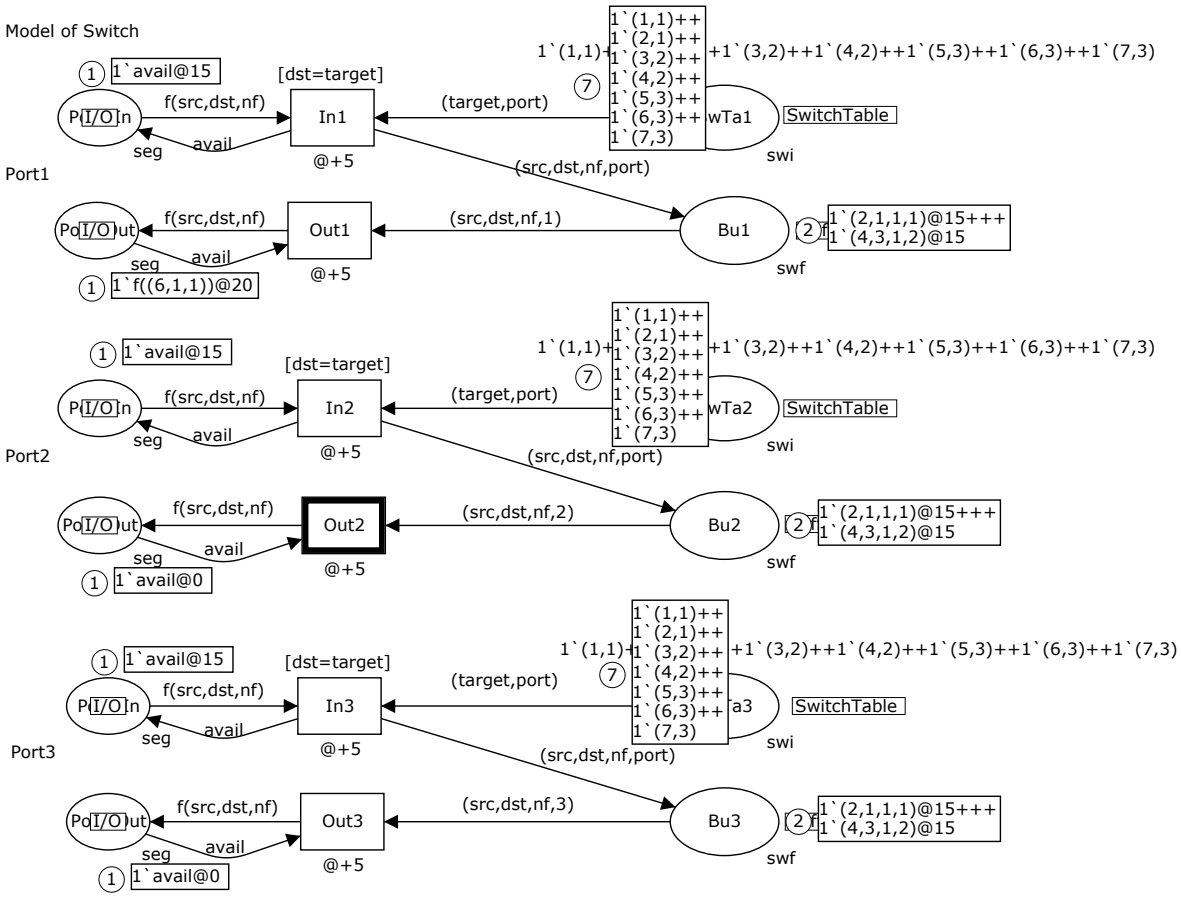
5) Фрейм запроса рабочей станции WS4 размещён во внутреннем буфере коммутатора:



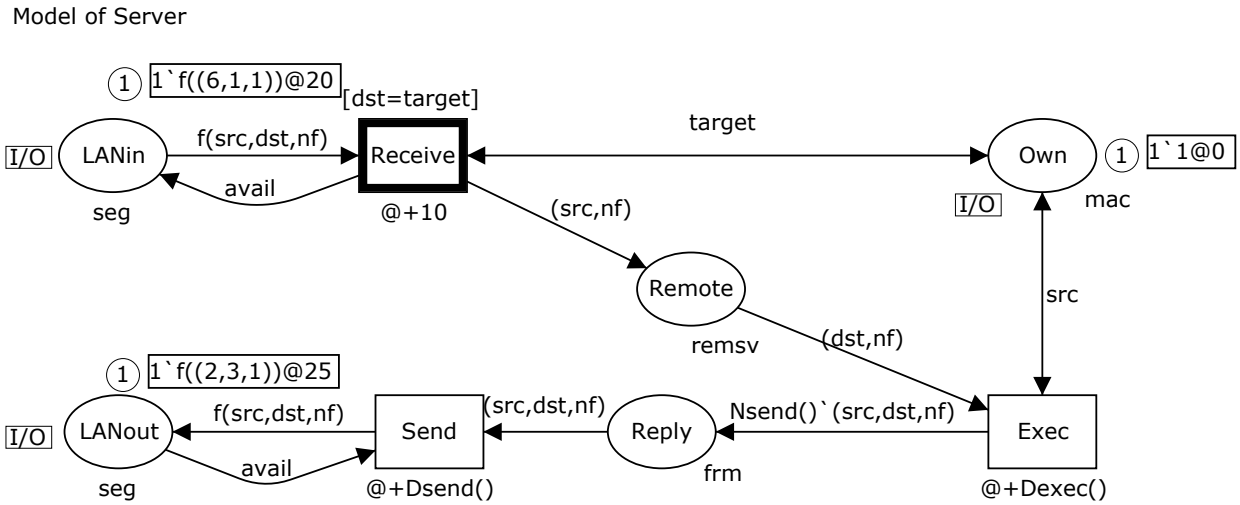
6) Фреймы запросов других рабочих станций размещены во внутреннем буфере коммутатора:



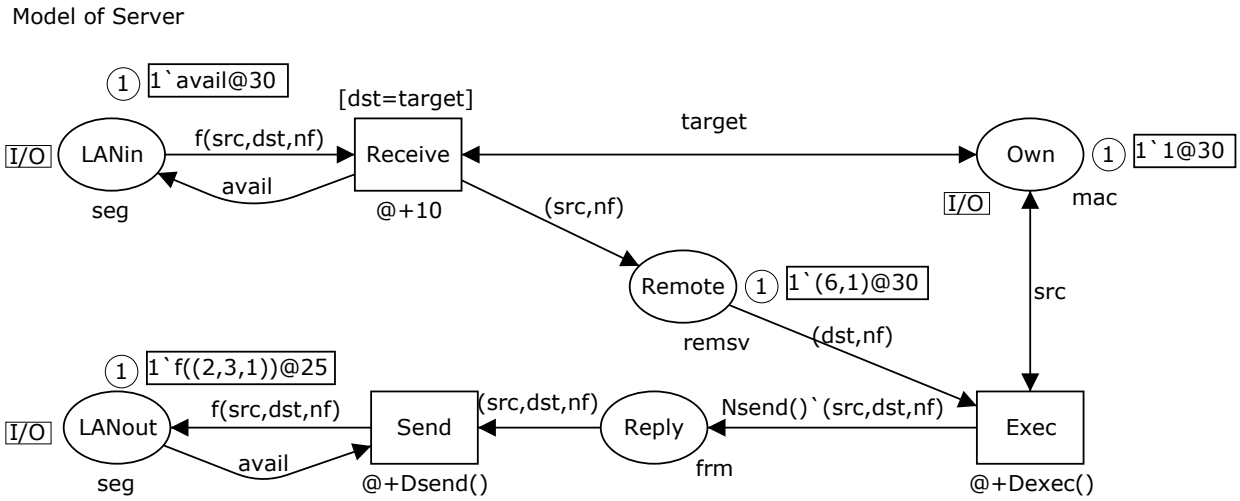
7) Фрейм запроса рабочей станции WS4 скоммутирован в первый порт:



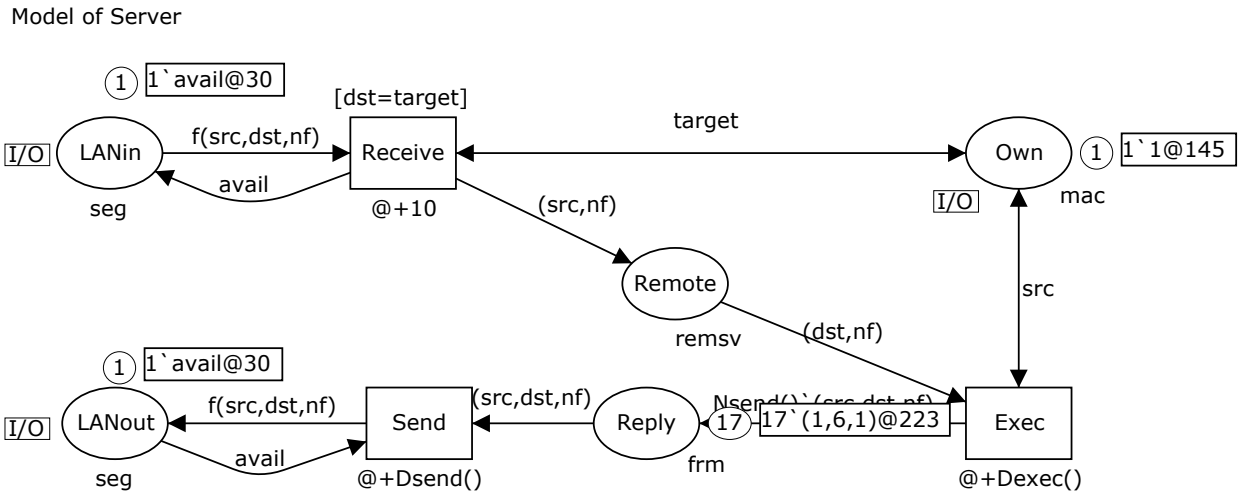
8) Фрейм запроса рабочей станции WS4 поступил во входной порт сервера S1:



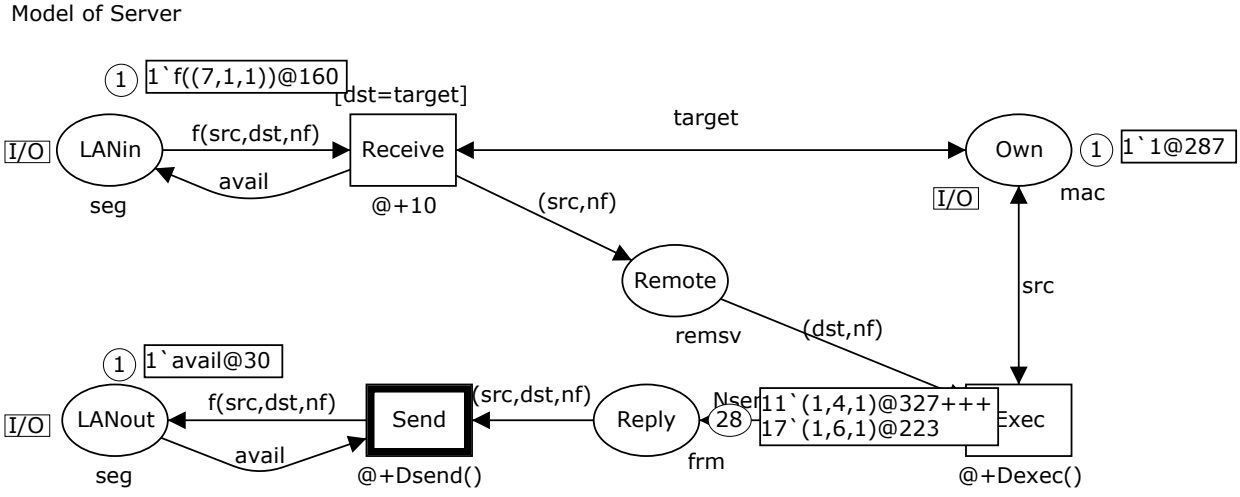
9) Фрейм запроса рабочей станции WS4 размещён во внутреннем буфере сервера S1:



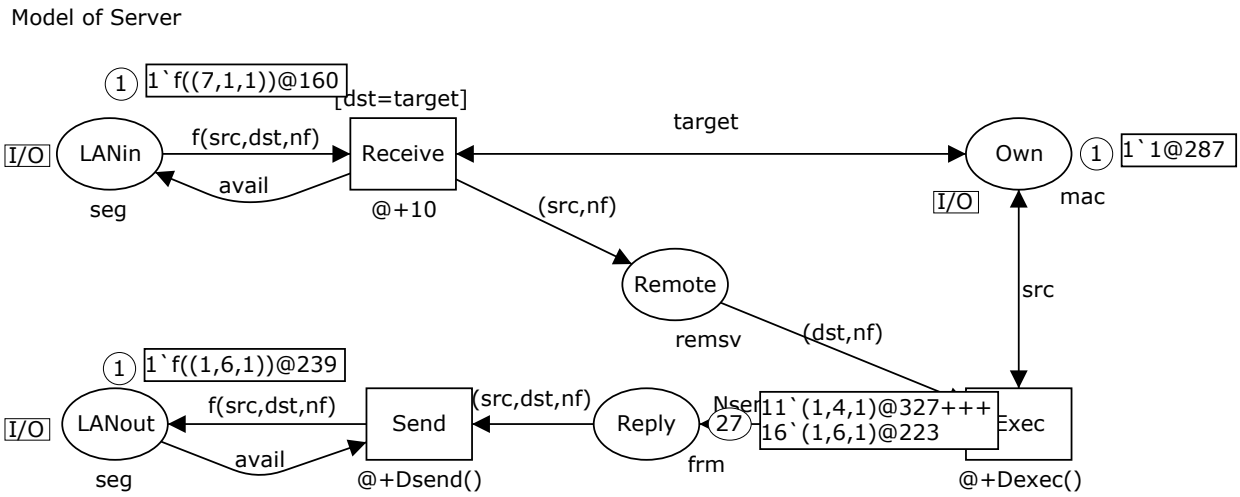
10) Запрос рабочей станции WS4 выполнен сервером S1; результат размещён в выходном буфере сервера:



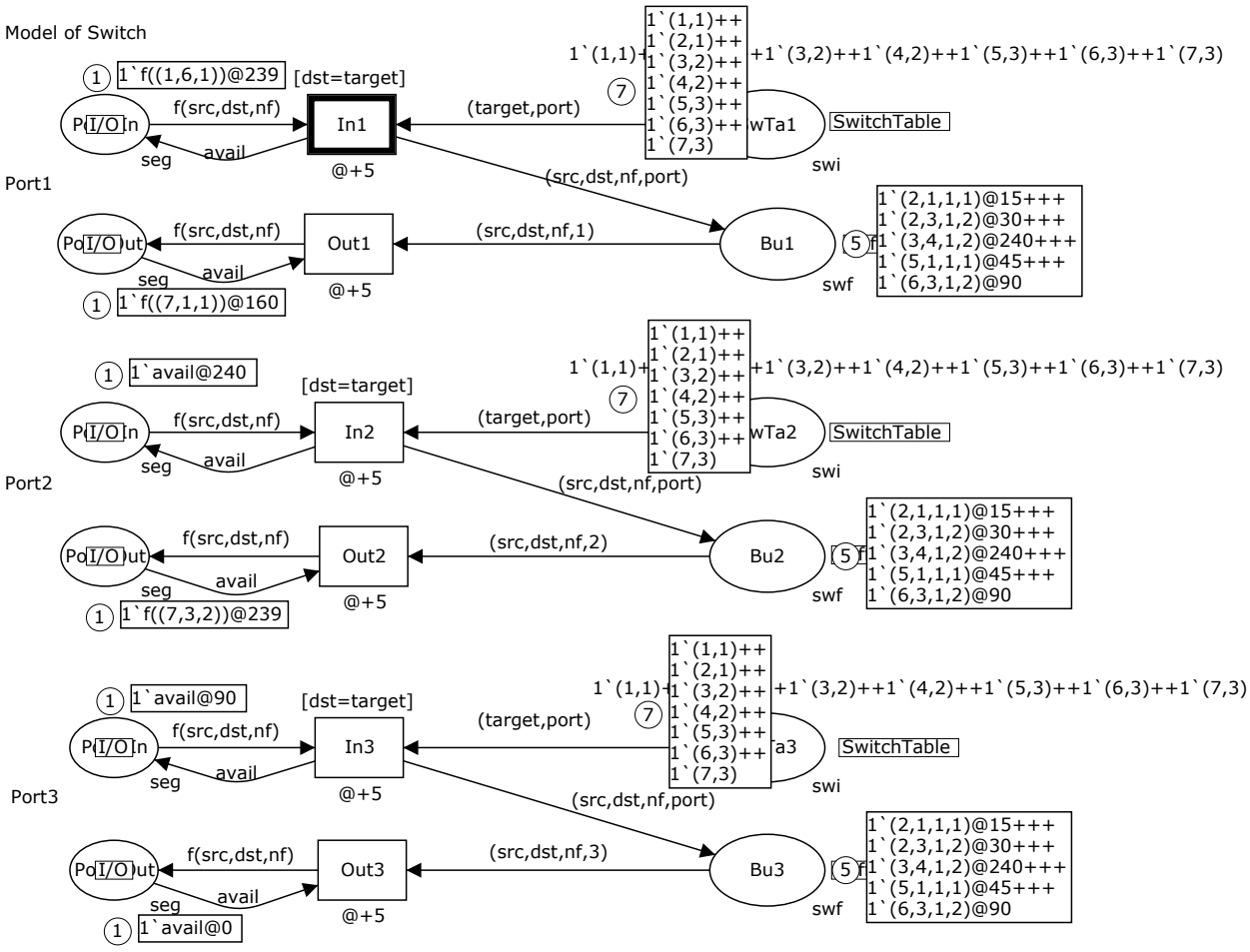
11) Сервером S1 выполнены запросы других рабочих станций:



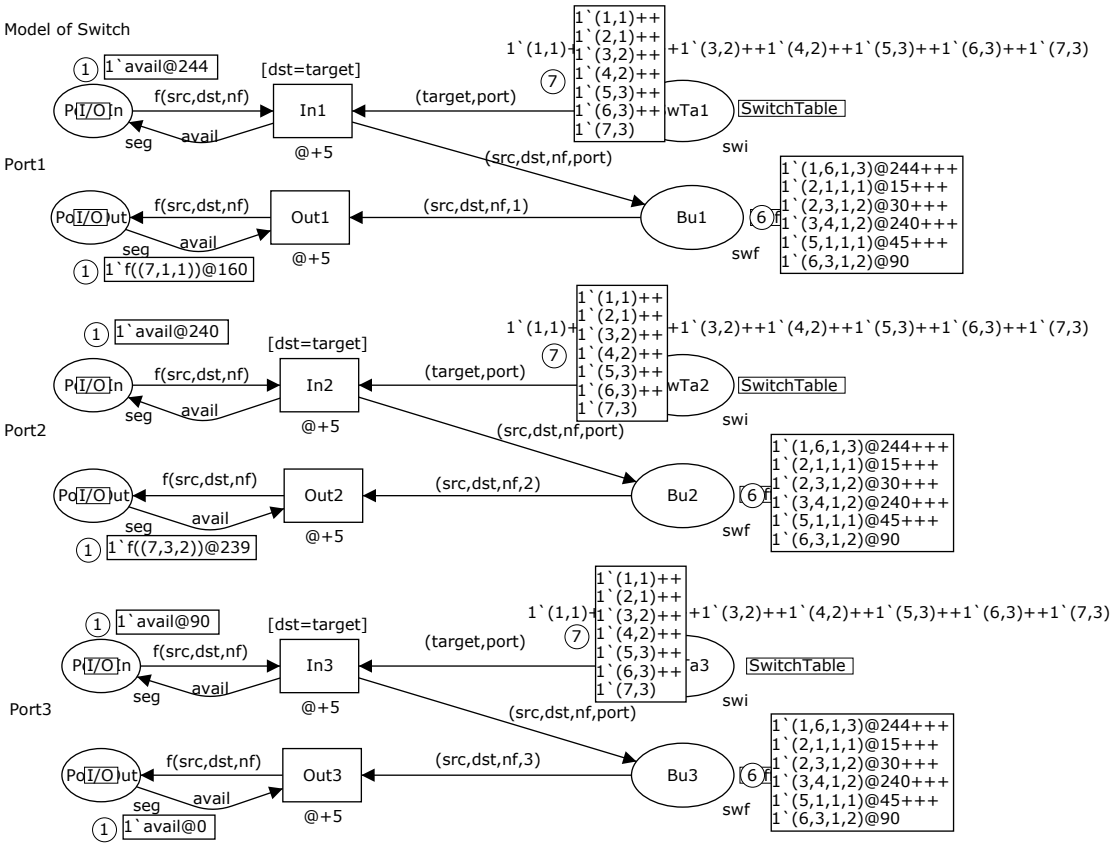
12) Первый фрем ответа на запрос рабочей станции WS4 передаётся в сегменте 1:



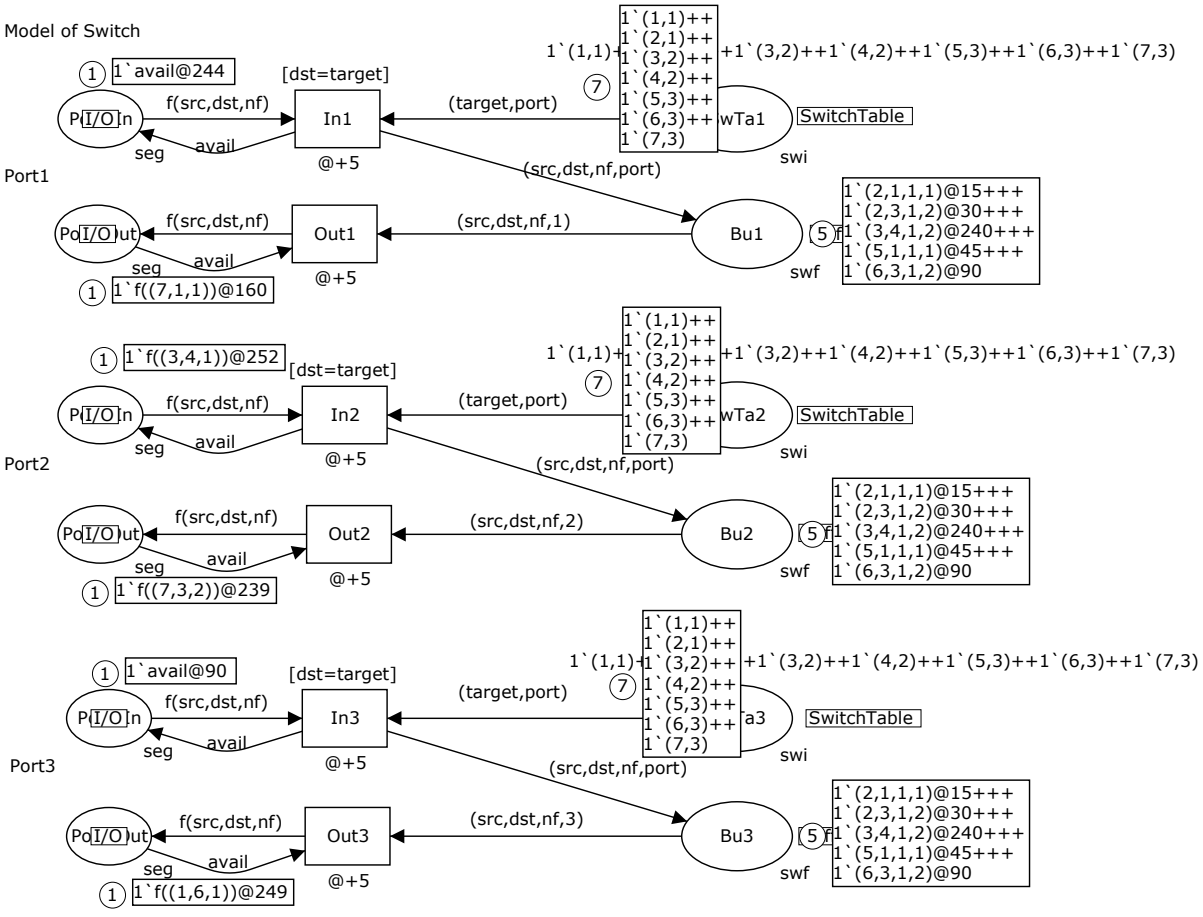
13) Первый фрейм ответа сервера S1 на запрос рабочей станции WS4 поступил во входной порт коммутатора:



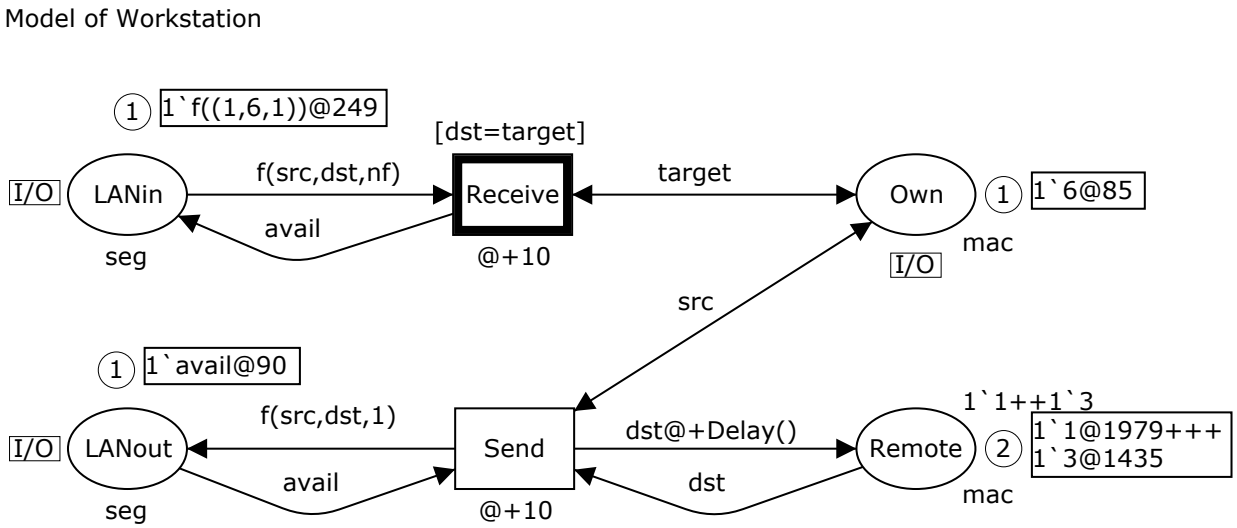
14) Первый фрейм ответа сервера S1 на запрос рабочей станции WS4 размещён во внутреннем буфере коммутатора:



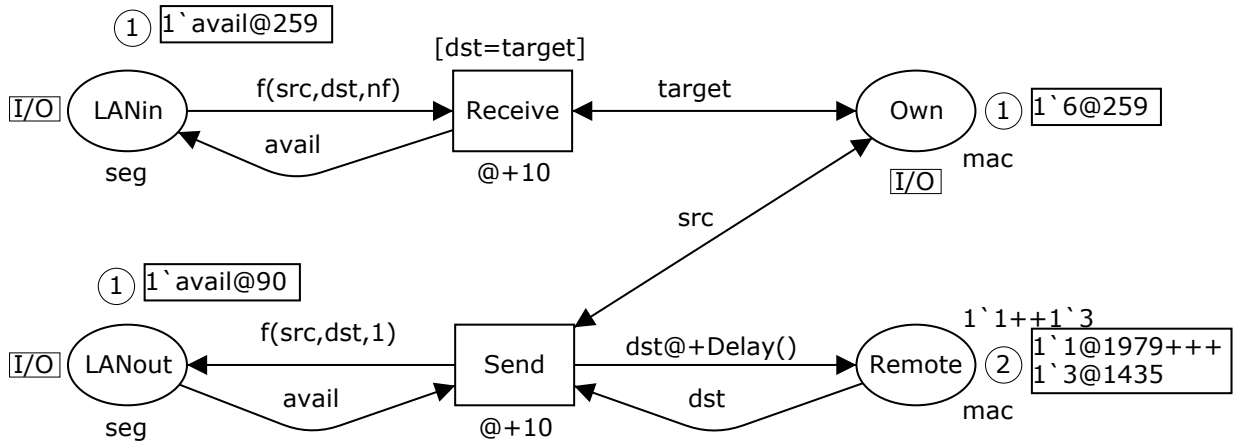
15) Первый фрейм ответа сервера S1 на запрос рабочей станции WS4 передаётся в сегменте 3:



16) Первый фрейм ответа сервера S1 на запрос рабочей станции WS4 получен рабочей станцией WS4:



17) Первый фрейм ответа сервера S1 на запрос рабочей станции WS4 обработан рабочей станцией WS4:



Таким образом, выполнена трассировка процесса взаимодействия клиент-сервер в построенной модели коммутируемой Ethernet.

Приложение Д. Adriana: программное обеспечение для композиционного вычисления инвариантов сетей Петри; Deborah: программное обеспечение для декомпозиции сети Петри на функциональные подсети

Тексты программ написаны на ANSI C и могут быть скомпилированы на платформах Windows, Unix, QNX и других.

```
/* Adriana.c */
/* PN invariants calculation in the process of Sequential composition of its
minimal FSN */

#include <stdio.h>
#include <string.h>
#include <string.h>
#include "uti.h"
#include "readnet.h"
#include "writeinv.h"
#include "chi.h"
#include "ana.h"

#define FILENAMELEN 256
#define MAXSTRLEN 1024

static char Help[] =
"Adriana version 1.0.1alpha -- 06/25/05 -- ZSoftUA for LAAS/CNRS\n\n"
"action: calculates linear invariants of Petri net in the process of\n"
"        composition of its minimal functional subnes\n"
"file formats: .ndr, .net (www.laas.fr/tina)\n"
"usage: Adriana [-h | -help]\n"
"               [-v | -q]\n"
"               [-P | -T]\n"
"               [-c] [-s]\n"
"               [infile] [outfile]\n"
"FLAGS          WHAT                                    DEFAULT\n"
"-h | -help     this mode\n"
"-c             simultaneous composition\n"
"-s             sequential composition\n"
"-P | -T        place or transition invariants      -P\n"
"-v | -q        textual output (full | digest)      -v\n"
"infile         input file (stdin if -)             stdin\n"
"outfile        output file (stdout if -)           stdout\n\n"
"@ Dmitry Zaitsev: zsoftua@yahoo.com, www.geocities.com/zsoftua\n";

int main( int argc, char *argv[] )
{
  char fname1[ FILENAMELEN+1 ];
  char fname2[ FILENAMELEN+1 ];
  char fname3[ FILENAMELEN+1 ];
  char * NetFileName;
  char * InvFileName;
  char * SPMFileName;
  char WorkFileName[ FILENAMELEN+1 ];
  FILE * out;
  int i,numf, snumf, inv;
  char *s;
  int place=1;
```

```
  int transition=0;
  int mode=0;
  int work_file=0;
  int debug_level=0;
  int save_SPM=0;
  int verbose=1;

  /* parse command line */
 numf=0;
 for( i=1; i<argc; i++ )
 {
    if( strcmp( argv[i], "-h" )==0 || strcmp( argv[i], "-help")==0 )
    {
      printf( "%s", Help ); return( 0);
    }
    else if( strcmp( argv[i], "-q" )==0 ) verbose=0;
    else if( strcmp( argv[i], "-v" )==0 ) verbose=1;
    else if( strcmp( argv[i], "-P" )==0 ) { transition=0; place=1; }
    else if( strcmp( argv[i], "-T" )==0 ) { place=0; transition=1; }
    else if( strcmp( argv[i], "-c" )==0 ) mode=1;
    else if( strcmp( argv[i], "-s" )==0 ) mode=2;
    else if( strcmp( argv[i], "-w" )==0 ) { work_file=1; snumf=numf; numf=-1;
}
    else if( numf==-1 ) { strcpy(WorkFileName,argv[ i ]); numf=snumf; }
    else if( strcmp( argv[i], "-d" )==0 ) { snumf=numf; numf=-2; }
    else if( numf==-2 ) { debug_level = atoi( argv[ i ] ); numf=snumf; }
    else if( strcmp( argv[i], "-o" )==0 ) { save_SPM=1; snumf=numf; numf=-3;
}
    else if( numf==-3 ) { SPMFileName = argv[ i ]; numf=snumf; }
    else if( numf==0 ) { NetFileName=argv[i]; numf++; }
    else if( numf==1 ) { InvFileName=argv[i]; numf++; }
    else
      { printf( "*** unknown option: %s\n", argv[i] ); return(4); }

 } /* for */

 if( numf==0 ) NetFileName = "-";
 if( numf<=1 ) InvFileName = "-";

  if( strcmp( InvFileName, "-" )==0 ) out = stdout;
  else out = fopen( InvFileName, "w" );
  if( out == NULL ) {printf( "*** error open file %s\n", InvFileName
);exit(2);}
  fprintf( out, "Adriana version 1.0.1alpha --- 06/25/05 --- ZSoftUA for
LAAS/CNRS\n\n" );
  if( out != stdout ) fclose( out );

  if( ! work_file )  sprintf( WorkFileName, "__%s", NetFileName );

  ReadNDRNET( NetFileName, WorkFileName, InvFileName, verbose );

  if( place )
  {
    sprintf( fname1, "%s.spm", WorkFileName );
    sprintf( fname2, "%s.spm.inv", WorkFileName );
    Adriana( fname1, fname2, debug_level, mode );

    inv = CheckInv( fname1, fname2 );

    if( strcmp( InvFileName, "-" )==0 ) out = stdout;
    else out = fopen( InvFileName, "a" );
    if( out == NULL ) {printf( "*** error open file %s\n", InvFileName
);exit(2);}
```

```
    fprintf( out, "P-SEMI-FLOWS GENERATING SET ------------------------------
--\n\n" );
    s = (inv)? "invariant" : "not invariant";
    fprintf( out, "%s\n", s );
    if( out != stdout ) fclose( out );

    sprintf( fname3, "%s.nmp", WorkFileName );
    WriteInv( fname3, fname2, InvFileName, verbose );

    sprintf( fname1, "%s.spm.inv", WorkFileName );
    if( save_SPM )
      rename( fname1, SPMFileName );
    else
      if( debug_level == 0 ) remove( fname1 );
    if( debug_level == 0 )
    {
      sprintf( fname1, "%s.spm", WorkFileName );
      remove( fname1 );
      sprintf( fname1, "%s.nmp", WorkFileName );
      remove( fname1 );
      sprintf( fname1, "%s.nmt", WorkFileName );
      remove( fname1 );
    }
  }

  if( transition )
  {
    sprintf( fname1, "%s.spm", WorkFileName );
    sprintf( fname2, "%s.spmt", WorkFileName );
    TransposeMatrix( fname1, fname2 );
    sprintf( fname3, "%s.spmt.inv", WorkFileName );
    Adriana( fname2, fname3, debug_level, mode );

    inv = CheckInv( fname2, fname3 );

    if( strcmp( InvFileName, "-" )==0 ) out = stdout;
    else out = fopen( InvFileName, "a" );
    if( out == NULL ) {printf( "*** error open file %s\n", InvFileName
);exit(2);}
    fprintf( out, "T-SEMI-FLOWS GENERATING SET ------------------------------
--\n\n" );
    s = (inv)? "consistent" : "not consistent";
    fprintf( out, "%s\n", s );
    if( out != stdout ) fclose( out );

    sprintf( fname1, "%s.nmt", WorkFileName );
    WriteInv( fname1, fname3, InvFileName, verbose );

    sprintf( fname1, "%s.spmt.inv", WorkFileName );
    if( save_SPM )
      rename( fname1, SPMFileName );
    else
      if( debug_level == 0 ) remove( fname1 );
    if( debug_level == 0 )
    {
      sprintf( fname1, "%s.spm", WorkFileName );
      remove( fname1 );
      sprintf( fname1, "%s.spmt", WorkFileName );
      remove( fname1 );
      sprintf( fname1, "%s.nmp", WorkFileName );
      remove( fname1 );
      sprintf( fname1, "%s.nmt", WorkFileName );
      remove( fname1 );
    }
```

```
  }

  if( strcmp( InvFileName, "-" )==0 ) out = stdout;
  else out = fopen( InvFileName, "a" );
  if( out == NULL ) {printf( "*** error open file %s\n", InvFileName
);exit(2);}
  fprintf( out, "ANALYSIS COMPLETED ----------------------------------------
\n" );
  if( out != stdout ) fclose( out );

} /* main */

/* Adriana project: ana.h                                        */
/* Solution of system via composition (sequential) of FSN    */

int Adriana( char * NetFileName, char * InvFileName, int debug, int ty );

/* INPUT: NetFileName - system in SPM format */
/*        debug - level of debug info; debug = 0, 1, 2 */
/*        ty - type of solution; defs for ty */
#define DIRECT 0
/* without composition */
#define SIMULT 1
/* simultaneous composition */
#define SEQUEN 2
/* sequential composition */

/* OUTPUT: InvFileName - basis invariants */

/* end ana.h */

/* ana.c */

#include <stdio.h>
#include <string.h>
#include "toy2.h"
#include "deborah.h"
#include "fic.h"
#include "coy.h"
#include "fui.h"
#include "mui.h"
#include "smmul2.h"
#include "smb2.h"
#include "uti.h"
#include "ana.h"

#define FILENAMELEN 256
#define MAXSTRLEN 1024

int CleanWorkFiles( int nz, char * NetFileName );
int ReadFusionSet( char * TraceFileName, int nline, char * FusionSet );
int NextFSN( char * FusionSet, int nitem );

int Adriana( char * NetFileName, char * InvFileName, int debug, int ty )
{
  int nz, z, nf, f, i, j;

  char fname1[ FILENAMELEN+1 ];
  char fname2[ FILENAMELEN+1 ];
  char fname3[ FILENAMELEN+1 ];

  char FusionSet[ MAXSTRLEN +1 ];

  if( ty == DIRECT )
```

```
{
  return( Toudic( NetFileName, InvFileName ) );
}

nz = DecomposeIntoFSN( NetFileName ); /* deborah */

/* solve systems for FSN */
for( z=1; z<=nz; z++ )
{
  sprintf( fname1, "%s.z%d", NetFileName, z );
  sprintf( fname2, "%s.i%d", NetFileName, z );

  Toudic( fname1, fname2 ); /* toy2 */

  if( ty == SEQUEN )
  {
    sprintf( fname1, "%s.z%d", NetFileName, z );
    sprintf( fname2, "%s.y%d", NetFileName, z );
    rename( fname1, fname2 ); /* .z# -> .y# */

    sprintf( fname1, "%s.i%d", NetFileName, z );
    sprintf( fname2, "%s.x%d", NetFileName, z );
    rename( fname1, fname2 ); /* .i# -> .x# */
  }
}

if( nz == 1 )
{
  if( ty == SEQUEN )
    sprintf( fname1, "%s.x1", NetFileName );
  else
    sprintf( fname1, "%s.i1", NetFileName );
  rename( fname1, InvFileName );
  if( debug < 2 ) CleanWorkFiles( nz, NetFileName );
  return(1);
}

if( ty == SEQUEN )
{
  sprintf( fname1, "%s.z", NetFileName );
  sprintf( fname2, "%s.y", NetFileName );
  rename( fname1, fname2 ); /* .z -> .y */
}

if( ty == SIMULT )
{
  /* composition */
  CreateCompositionSystem( NetFileName ); /* coy */
  if( debug > 0 )
  {
    sprintf( fname1, "%s.s", NetFileName );
    sprintf( fname2, "%s-s1", NetFileName );
    CopyFile( fname1, fname2, "w" );
  }

  sprintf( fname1, "%s.s", NetFileName );
  sprintf( fname2, "%s.r0", NetFileName );
  Toudic( fname1, fname2 ); /* toy2 */

  sprintf( fname1, "%s.r0", NetFileName );
  sprintf( fname2, "%s.u", NetFileName );
  sprintf( fname3, "%s.r", NetFileName );
  AddUnitSolutions( fname1, fname2, fname3 ); /* fui */
```

```
    ComposeJointMatrix( NetFileName ); /* mui */

    sprintf( fname1, "%s.r", NetFileName );
    sprintf( fname2, "%s.g", NetFileName );
    sprintf( fname3, "%s.h0", NetFileName );
    MultiplySPM( fname1, fname2, fname3 ); /* smmul2 */

    sprintf( fname1, "%s.h0", NetFileName );
    sprintf( fname2, "%s.h", NetFileName );
    MinimizeBasis( fname1, fname2 ); /* smb2 */
    /* end of composition */

    /* save solution */
    sprintf( fname1, "%s.h", NetFileName, nf );
    rename( fname1, InvFileName );

    if( debug < 2 ) CleanWorkFiles( nz, NetFileName );
    return(0);
  } /* if ty == SIMULT */

  sprintf( fname1, "%s.p", NetFileName );
  sprintf( fname2, "%s.t", NetFileName );
  nf = CollapseOfGraph( 'm', fname1, fname2 ); /* cor */

  for( f=1; f<=nf; f++ )
  {
    z = FilterContactPlaces( NetFileName, f ); /* fic */
/*printf( "fic z=%d\n", z );*/

    /* prepare FSN files */
    sprintf( fname1, "%s.t", NetFileName );
    ReadFusionSet( fname1, f, FusionSet );
/*printf( "fusion set: %s\n", FusionSet );*/
    j=1;
    while( i = NextFSN( FusionSet, j ) )
    {
/*printf( "next FSN: %d\n", i );*/
      sprintf( fname1, "%s.y%d", NetFileName, i );
      sprintf( fname2, "%s.z%d", NetFileName, j );
      rename( fname1, fname2 );

      sprintf( fname1, "%s.x%d", NetFileName, i );
      sprintf( fname2, "%s.i%d", NetFileName, j );
      rename( fname1, fname2 );

      j++;
    }

    /* composition */
    CreateCompositionSystem( NetFileName ); /* coy */
    if( debug > 0 )
    {
      sprintf( fname1, "%s.s", NetFileName );
      sprintf( fname2, "%s-s%d", NetFileName, f );
      CopyFile( fname1, fname2, "w" );
    }

    sprintf( fname1, "%s.s", NetFileName );
    sprintf( fname2, "%s.r0", NetFileName );
    Toudic( fname1, fname2 ); /* toy2 */

    sprintf( fname1, "%s.r0", NetFileName );
    sprintf( fname2, "%s.u", NetFileName );
    sprintf( fname3, "%s.r", NetFileName );
```

```
    AddUnitSolutions( fname1, fname2, fname3 ); /* fui */

    ComposeJointMatrix( NetFileName ); /* mui */

    sprintf( fname1, "%s.r", NetFileName );
    sprintf( fname2, "%s.g", NetFileName );
    sprintf( fname3, "%s.h0", NetFileName );
    MultiplySPM( fname1, fname2, fname3 ); /* smmul2 */

    sprintf( fname1, "%s.h0", NetFileName );
    sprintf( fname2, "%s.h", NetFileName );
    MinimizeBasis( fname1, fname2 ); /* smb2 */
    /* end of composition */

    if( debug > 0 ) /* store Inv */
    {
      sprintf( fname1, "%s.h", NetFileName );
      sprintf( fname2, "%s-i%d", NetFileName, f );
      CopyFile( fname1, fname2, "w" ); /* .h -> -i# */
    }

    sprintf( fname1, "%s.h", NetFileName );
    sprintf( fname2, "%s.x%d", NetFileName, z );
    CopyFile( fname1, fname2, "w" ); /* .h -> .x# */

    /* merge FSN */
    sprintf( fname1, "%s.y%d", NetFileName, z );
    remove( fname1 );
    j=1;
    while( i = NextFSN( FusionSet, j ) )
    {
      sprintf( fname2, "%s.z%d", NetFileName, j );
      CopyFile( fname2, fname1, "a" );

      if( i != z )
      {
        sprintf( fname3, "%s.y%d", NetFileName, i );
        remove( fname3 );
      }

      j++;
    }

    if( debug > 0 ) /* store FSN */
    {
      sprintf( fname2, "%s-z%d", NetFileName, f );
      CopyFile( fname1, fname2, "w" );
    }

  } /* sequential composition (for) */

  /* save solution */
  sprintf( fname1, "%s.h", NetFileName, nf );
  rename( fname1, InvFileName );

  if( debug < 2 ) CleanWorkFiles( nz, NetFileName );

} /* Adriana */

int CleanWorkFiles( int z, char * NetFileName )
{
  char fname[ FILENAMELEN+1 ];
  int i;
```

```
  sprintf( fname, "%s.z", NetFileName );
  remove( fname );
  sprintf( fname, "%s.y", NetFileName );
  remove( fname );
  sprintf( fname, "%s.s", NetFileName );
  remove( fname );
  sprintf( fname, "%s.g", NetFileName );
  remove( fname );
  sprintf( fname, "%s.r", NetFileName );
  remove( fname );
  sprintf( fname, "%s.r0", NetFileName );
  remove( fname );
  sprintf( fname, "%s.u", NetFileName );
  remove( fname );
  sprintf( fname, "%s.h0", NetFileName );
  remove( fname );
  sprintf( fname, "%s.p", NetFileName );
  remove( fname );
  sprintf( fname, "%s.t", NetFileName );
  remove( fname );
  sprintf( fname, "%s.x", NetFileName );
  remove( fname );

  for( i=1; i<=z; i++ )
  {
    sprintf( fname, "%s.x%d", NetFileName, i );
    remove( fname );
  }

  for( i=1; i<=z; i++ )
  {
    sprintf( fname, "%s.y%d", NetFileName, i );
    remove( fname );
  }

  for( i=1; i<=z; i++ )
  {
    sprintf( fname, "%s.z%d", NetFileName, i );
    remove( fname );
  }

  for( i=1; i<=z; i++ )
  {
    sprintf( fname, "%s.i%d", NetFileName, i );
    remove( fname );
  }

} /* CleanWorkFiles */

int ReadFusionSet( char * TraceFileName, int nline, char * FusionSet )
{
  FILE * ftrace;
  char str[ MAXSTRLEN+1 ];
  int l=0;

  ftrace = fopen( TraceFileName, "r" );
  if( ftrace == NULL ) {printf( "*** error open file %s\n", TraceFileName
);exit(2);}

 /* read solutions */
 while( ! feof( ftrace ) )
 {
   fgets( str, MAXSTRLEN, ftrace );
```

```
   if( feof( ftrace ) ) break;

   l++;

   if( l == nline ) { strcpy( FusionSet, str ); break; }
 }

 fclose( ftrace );
 return(0);

} /* ReadFusionSet */

int NextFSN( char * str, int nitem )
{
  int i, len, item=0;

  len=strlen( str ); i=0;

  while( i<len )
  {
    SwallowSpace( str, &i );
    item++;
    if( item == nitem ) { return( atoi( str+i ) ); }

    while( ! IsSpace(str,i) && i<len )i++;

  }

  return(0);

} /* NextFSN */

/* Adriana project; chi.h                                        */
/* Check invariance on PN and basis invariants in SPM format */

int CheckInv( char * NetFileName, char * InvFileName );

/* end chi.h */

/* Check invariance of PN */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "uti.h"

#define LINESIZE 256

#define initROWS 1024
#define deltaROWS 1024

int CheckInv( char * NetFileName, char * InvFileName )
{
  char line[ LINESIZE+1 ];
  FILE * NetFile, * InvFile;
  int mc, maxrows;
  int i, p, t, v, l;
  int inv, found;
  int * rows, * inds;

  NetFile = fopen( NetFileName, "r" );
  if( NetFile == NULL ) {printf( "*** error open file %s\n", NetFileName
);exit(2);}
  InvFile = fopen( InvFileName, "r" );
```

```
  if( InvFile == NULL ) {printf( "*** error open file %s\n", InvFileName
);exit(2);}

  maxrows = initROWS; rows = malloc( maxrows*sizeof(int) );
  if( rows==NULL ) {printf("*** not enough memory for rows\n");exit(1);};

  /* collect numbers of places */
  mc=0;
  while ( ! feof( NetFile ) )
  {
    fgets( line, LINESIZE, NetFile );

    if( feof( NetFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
    sscanf( line, "%d %d %d", &p, &t, &v );

    found=0;
    for( i=1; i<=mc; i++ ) if( rows[i]==p ) { found=1; break; }
    if( ! found )
    {
      if( mc >= maxrows )
      {
        maxrows+=deltaROWS;
      rows = realloc( rows, maxrows*sizeof(int) );
      if( rows == NULL ) {printf("*** not enough memory for
rows\n");exit(1);};
      }
      rows[++mc]=p;
    }
  } /* feof NetFile */

  fclose( NetFile );
  rows = realloc( rows, (mc+1)*sizeof(int) );
  inds = calloc( (mc+1), sizeof(int) );
  if( rows==NULL || inds==NULL ) {printf("*** not enough memory for rows,
inds\n");exit(1);}

  /* check invariants */
  while ( ! feof( InvFile ) )
  {
    fgets( line, LINESIZE, InvFile );

    if( feof( InvFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
    sscanf( line, "%d %d %d", &l, &p, &v );

    if( v > 0 )
    {
      found=0;
      for( i=1; i<=mc; i++ ) if( rows[i]==p ) { found=1; break; }
      if( found )
      {
        inds[i]=1;
      }
    }
  } /* feof InvFile */

  fclose( InvFile );
```

```
  inv=1;
  for( i=1; i<=mc; i++ ) {inv = inv && inds[i];}

  free(rows); free(inds);

  return( inv );

} /* CheckInv */

#ifdef __MAIN__
int main( int argc, char * argv[] )
{
  int i;

  i = CheckInv( argv[1], argv[2] );
  printf( "%d\n", i );
}
#endif

/* Adriana project; cor.h                        */
/* Collapse of weighted graph                    */

int CollapseOfGraph( char ty, char * GraphFileName, char * TraceFileName );

/* INPUT: ty - type of collapse: 'f' - first max; 'm' - random max; 'r' -
random; 'i' - last min */
/*        GraphFileName - weighted graph given by edges: <node1> <node2>
<weight> */
/* OUTPUT: TraceFileName - sequence (trace) of collapse */
/* FUNCTION: collapse weighted graph acoording to a given rule of edge choice
*/

/* end cor.h */

/* co */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define LINESIZE 256

#define min(a,b) (((a)<(b))?(a):(b))
#define max(a,b) (((a)>(b))?(a):(b))

struct edge {
  int v1;
  int v2;
  int w;
};

int maxwe( struct edge *e, int n, int b )
{
  int mw, mi, i;

  mw = e[b].w;
  mi = b;

  for( i=b+1; i<n; i++)
  {
    if( e[i].w > mw )
    {
        mw = e[i].w;
```

```
        mi = i;
      }
    }

    return mi;

} /* maxwe */

void sorte( struct edge *e, int n)
{
 int i, m, v1, v2, w;

    for( i=0; i<n-1; i++)
    {
      m = maxwe( e, n, i );

      v1 = e[i].v1;
      v2 = e[i].v2;
      w = e[i].w;

      e[i].v1 = e[m].v1;
      e[i].v2 = e[m].v2;
      e[i].w = e[m].w;

      e[m].v1 = v1;
      e[m].v2 = v2;
      e[m].w = w;

    }

}/* sorte */

void shiftr( struct edge *e, int *n, int d )
{
    int i;

    (*n)--;
    for( i=d; i < (*n); i++)
    {
      e[i].v1 = e[i+1].v1;
      e[i].v2 = e[i+1].v2;
      e[i].w = e[i+1].w;
    }
} /* shiftr */

void merge( struct edge * e, int *n, int m )
{
    int i, j;
    int v, v1, v2, v1c, v2c;
    int ii;

    v =  e[m].v1;
    v1 =  e[m].v1;
    v2 =  e[m].v2;

    /* merge edge */
    shiftr( e, n, m );

  /* renumerate nodes */
  for( i=0; i<(*n); i++)
  {
     v1c = e[i].v1;
     v2c = e[i].v2;
```

```
   if( v1c == v2 ) { e[i].v1 = min( v, v2c); e[i].v2 = max( v, v2c); }

   if( v2c == v2 ) { e[i].v1 = min( v, v1c); e[i].v2 = max( v, v1c); }
 }

/* merge parell edges */
i=0;
while( i < (*n) )
{

  j = i+1;
  while( j < (*n) )
  {
    if( (e[i].v1 == e[j].v1) && (e[i].v2 == e[j].v2))
    {
      e[i].w+=e[j].w;
      shiftr( e, n, j );
    }
    else
      j++;
  }

  i++;

}

 } /* merge */

int CollapseOfGraph( char ty, char * GraphFileName, char * TraceFileName )
{
  char line[ LINESIZE+1 ];
  FILE * InputFile, * OutputFile;
  struct edge *e;
  int n, i, j;
  int b, f, w;
  int maxw, m;
  int ii, c;
  int ri, maxc, maxn;
  double r;
  int sw;
  double per;
  int k;

  InputFile = fopen( GraphFileName, "r" );
  if( InputFile == NULL ) {printf( "*** error open file %s\n", GraphFileName
);exit(2);}
  OutputFile = fopen( TraceFileName, "w" );
  if( OutputFile == NULL ) {printf( "*** error open file %s\n", TraceFileName
);exit(2);}

  n=0;
  while ( ! feof( InputFile ) )
  {
    fgets( line, LINESIZE, InputFile );

    if( line[0] == ';' || feof(InputFile) ) continue;

    n++;
  } /* feof InputFile */

  e = calloc( n, sizeof(struct edge) );
  if( e == NULL )
  {
    printf( "*** not enough memory for e\n" );
```

```
    exit( 1 );
  }
  rewind ( InputFile );

  i = 0; sw=0;
  while ( ! feof( InputFile ) )
  {
    fgets( line, LINESIZE, InputFile );

    if( line[0] == ';' || feof(InputFile ) ) continue;

    w=1;
    sscanf( line, "%d %d %d", &b, &f, &w );

    e[i].v1 = min( b, f );
    e[i].v2 = max( b,f );
    e[i].w = w;

    sw+=w;

    if( b == f ) {printf("*** error input file: loop %d %d\n", b, f ); return
2;}
    for( j=0; j<i; j++)
      if( (e[j].v1==e[i].v1) && (e[j].v2==e[i].v2) )  {printf("*** error
input file: duplicated arcs %d %d\n", b, f ); return 2;}

    i++;

  } /* feof InputFile */

  fclose( InputFile );

  maxw=0;
  k=0;
  while( n > 0 )
  {
    sorte( e,n );

    /*fprintf( OutputFile, "----------------------------------------\n" );
    for( i=0; i<n; i++ )
      { fprintf( OutputFile, " %d %d %d\n", e[i].v1, e[i].v2, e[i].w ); }*/


    switch( ty )
    {
      case 'f': m = 0; break;

      case 'm':
        /* random choice of max */
        maxc = e[0].w;
        maxn = 1;
        while( e[maxn].w==maxc && maxn<n ) maxn++;

      r = rand();
        ri=r*maxn/RAND_MAX;

        m=ri-1; if( m < 0 ) m=0; if( m>maxn-1 ) m=maxn-1;
      break;

      case 'r':
        r = rand();
        ri=r*n/RAND_MAX;

        m=ri-1; if( m < 0 ) m=0; if( m>n-1 ) m=n-1;
```

```
        break;

      case 'i': m = n-1; break;

    default: m=0;
    }

    /*fprintf( OutputFile, "mergenum %d of %d\n",m, n );*/

    maxw = max( maxw, e[m].w );

    /*fprintf( OutputFile, "merge %d %d %d\n", e[m].v1, e[m].v2, e[m].w );*/

    fprintf( OutputFile, "%d %d\n", e[m].v1, e[m].v2 );

    merge(  e, &n, m );
    k++;

  } /* while */

  /*per = maxw*100.0/sw;
  fprintf( OutputFile, "sumw: %d       maxw: %d        per: %e\n", sw, maxw,
per );*/

  free( e );
  fclose( OutputFile );

  return( k );

} /* main */

#ifdef __MAIN__
int main( int argc, char * argv[] )
{
  int c, k;

  if (argc < 4 )
  {
    printf( "Routine COR (c) 2005 by Dmitry Zaitsev\n");
    printf( "USAGE: COR s InFile OutFile\n" );
    printf( "s: f - first max, m - random max, r - random, i - last min\n" );
    return 3;
  }
  c = argv[1][0];
  if( strchr( "fmri", c ) == NULL ) { printf( "*** error: wrong switch %c\n",
argv[1][0] ); return 2; }

  k=CollapseOfGraph( c, argv[2], argv[3] );
  printf( "%d\n", k );
}
#endif

/* Adriana project; coy.h                        */
/* Creates composition system                    */

void CreateCompositionSystem( char * NetFileName );

/* INPUT: NetFileName.z - set of contact places */
/*        NetFileName.i#i - invariants of contact places */
/* OUTPUT: NetFileName.s - composition system */
/*         NetFileName.u - unit solutions for unused variables */
/* FUNCTION: create composition system and set of unit solutions */

/* end coy.h */
```

```
/* creates system for contact places */

#include <stdio.h>
#include <malloc.h>
#include "uti.h"

#define FILENAMELEN 256
#define MAXSTRLEN 256

struct contact {
  int p;
  int x;
  int y;
};

typedef struct contact coty;

static int is=0;

#define vnum( j ) (is+(j))

void CreateCompositionSystem( char * NetFileName )
{
 char FileName[ FILENAMELEN+1 ];
 char str[ MAXSTRLEN+1 ];
 FILE * zFile, * sFile, * uFile;
 int p, x, y, v, i, k, z;
 int nz, nc, ni, ic, ii, nl;
 int *in;

 coty * c;

 sprintf( FileName, "%s.z", NetFileName );
 zFile = fopen( FileName, "r" );
 if( zFile == NULL ) {printf( "*** error open file %s\n", FileName
);exit(2);}
 sprintf( FileName, "%s.s", NetFileName );
 sFile = fopen( FileName, "w" );
 if( sFile == NULL ) {printf( "*** error open file %s\n", FileName
);exit(2);}

 /* get size */
 nc=0;
 fgets( str, MAXSTRLEN, zFile);
 sscanf( str, "%d", &nz );
 while( ! feof( zFile ) )
 {
   fgets( str, MAXSTRLEN, zFile);

   if( feof( zFile ) ) break;

   if( str[0]=='#' || isempty(str) ){ continue; }

   sscanf( str, "%d %d %d", &p, &x, &y );

   nc++;
 }

 c = calloc( nc, sizeof( coty ) );
 if( c==NULL ) {printf("*** not enough memory for c\n");exit(1);};

 rewind( zFile );
```

```
 /* read contact info */
 ic=0;
 fgets( str, MAXSTRLEN, zFile);
 sscanf( str, "%d", &nz );
 while( ! feof( zFile ) )
 {
   fgets( str, MAXSTRLEN, zFile);

   if( feof( zFile ) ) break;

   if( str[0]=='#' || isempty(str) ){ continue; }

   sscanf( str, "%d %d %d", &p, &x, &y );
   c[ic].p=p; c[ic].x=x; c[ic].y=y;

   ic++;
 }

 fclose( zFile );

 for( z=1; z<=nz; z++ )
 {
   sprintf( FileName, "%s.i%d", NetFileName, z );
   zFile = fopen( FileName, "r" );
   if( zFile == NULL ) {printf( "*** error open file %s\n", FileName
);exit(2);}

   /* read inv of subnet */
   ni=0;
   nl=0;
   while( ! feof( zFile ) )
   {
     fgets( str, MAXSTRLEN, zFile);

     if( feof( zFile ) ) break;

     if( str[0]==';' || isempty(str) ){ continue; }

     sscanf( str, "%d %d %d", &i, &p, &v );

     for( ic=0; ic<nc; ic++ )
     {
       if( c[ic].p==p && c[ic].x==z )
         fprintf( sFile, "%d %d %d\n", vnum(i), p, -v );

       if( c[ic].p==p && c[ic].y==z )
         fprintf( sFile, "%d %d %d\n", vnum(i), p, v );
     }

     nl=max(nl,i);
     ni++;
   }

   is+=nl;

   fclose( zFile );

 } /* on subnets */

 fclose( sFile );

 /* compose additional solutions for internal invariants */
 in=calloc((is+1), sizeof(int));
 if( in==NULL ) {printf("*** not enough memory for in\n");exit(1);};
```

```
 sprintf( FileName, "%s.s", NetFileName );
 sFile = fopen( FileName, "r" );
 if( sFile == NULL ) {printf( "*** error open file %s\n", FileName
);exit(2);}

 while( ! feof(sFile) )
 {
     fgets( str, MAXSTRLEN, sFile);

     if( feof( sFile ) ) break;

     if( str[0]==';' || isempty(str) ){ continue; }

     sscanf( str, "%d %d %d", &i, &p, &v );

     in[i]=1;

 }/* sFile */

 fclose(sFile);

 /* write unit solutions */
 sprintf( FileName, "%s.u", NetFileName );
 uFile = fopen( FileName, "w" );
 if( uFile == NULL ) {printf( "*** error open file %s\n", FileName
);exit(2);}

 k=1;
 for( i=1; i<=is; i++)
   if( in[i]==0 )
     fprintf( uFile, "%d %d %d\n", k++, i, 1 );

 free(c); free(in);

 fclose( uFile );

} /* CreateCompositionSystem */

#ifdef __MAIN__
main( int argc, char * argv[] )
{
  if( argc < 2 ) return;
  CreateCompositionSystem( argv[1] );
}
#endif

/* Adriana project; deborah.h                    */
/* Deconposition into FSN for PN in SPM format */

int DecomposeIntoFSN( char * NetFileName );

/* INPUT: NetFileName - PN in SPM format */
/* OUTPUT: NetFileName.z - contact places info: <place> <input_FSN>
<outputFSN> */
/*         NetFileName.p - graph of decomposition */
/*         NetFileName.z#i - FSN in SPM format */
/* FUNCTION: decomposes PN into its minimal FSN */

/* end deborah.h */

/* Petri Net Decomposition (Deborah.c)*/
/* version for decomposition-based solution of equations + sequential */
```

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include "uti.h"

#define MAXINPSTRLEN 256
#define FILENAMELEN 256
#define initPL 10000
#define initTR 10000

struct netarc {
  int p;
  int t;
  int v;
};

typedef struct netarc arcty;

int DecomposeIntoFSN( char * NetFileName )
{
 int debug = 0;

 char SubNetFileName[ FILENAMELEN + 1];
 char zFileName[ FILENAMELEN + 1];
 FILE * NetFile, * SubNetFile, * zFile;
 char str[ MAXINPSTRLEN + 1 ]; /* buffers */

 int n, m, l; /* net size */
 int *tx, *ty, *at, fat; /* trs */
 int *px, *py, *ap, fap; /* pos */

 int p, t, i, j, jp, jt, u, v, z; /* indexes */

 int *SubNet, iSubNet; /* SubNets: subnet num of trs */
 int *R, *S, *X, *Y, *Q;
 int nR, nS, nX, nY, nQ; /* SubNet */

 int pinp, pout, belong; /* flags */

 int *SX, *SY; /* subnet num of input/output pos */
 int *GA, nGA; /* arcs of subnet graph */

 int *pl, *tr;
 int maxpl, maxtr, found, k;

 arcty *a;

 /* open files */
 NetFile = fopen( NetFileName, "r" );
 if( NetFile == NULL ) {printf( "*** error open file %s\n", NetFileName
);exit(2);}
 sprintf( SubNetFileName, "%s", "/dev/null" );
 SubNetFile = fopen( SubNetFileName, "w" );

 maxpl=initPL; pl=malloc( maxpl * sizeof( int ) );
 if( pl==NULL ) {printf("*** not enough memory for pl\n");exit(1);};
 maxtr=initTR; tr=malloc( maxtr * sizeof( int ) );
 if( tr==NULL ) {printf("*** not enough memory for tr\n");exit(1);};

 /* scan input to obtaine m, n, l */
 n=0; m=0; l=0;
 while( ! feof( NetFile ) )
 {
   fgets( str, MAXINPSTRLEN, NetFile);
```

```
   if( feof( NetFile) ) break;

   if( str[0]=='#' || isempty(str) ){ continue; }

   v=1;
   sscanf( str, "%d %d %d", &p, &t, &v );

   l++;

   found = 0;
   for( i=1; i<=m; i++ ) { if( pl[i]==p ){ found=1; break; } }
   if( ! found ) { pl[ ++m ]=p; }

   found = 0;
   for( j=1; j<=n; j++ ) { if( tr[j]==t ){ found=1; break; } }
   if( ! found ) { tr[ ++n ]=t; }

   /*m=max(m,p);
   n=max(n,t);  */
 }

/*printf("estimated sizes\n");*/

 /* allocate arrays */

 a = calloc( l, sizeof( arcty ) );
 if( a==NULL ) {printf("*** not enough memory for a\n");exit(1);};

/* read net */
 rewind( NetFile );
 k=0;
 while( ! feof( NetFile ) )
 {
   fgets( str, MAXINPSTRLEN, NetFile); if( str[0]==';' ) continue;

   if( feof( NetFile ) ) break;

   if( str[0]=='#' || isempty(str) ){ continue; }

   v=1;
   sscanf( str, "%d %d %d", &p, &t, &v );

   found = 0;
   for( i=1; i<=m; i++ ) { if( pl[i]==p ){ found=1; break; } }
   if( ! found ) { printf( "*** error net read\n" ); exit(3); }

   found = 0;
   for( j=1; j<=n; j++ ) { if( tr[j]==t ){ found=1; break; } }
   if( ! found ) { printf( "*** error net read\n" ); exit(3); }

   a[k].p=i; a[k].t=j; a[k].v=v;
   k++;
 }

/*printf("input matr\n");*/

 tx = calloc( n+2, sizeof(int) );
 ty = calloc( n+2, sizeof(int) );
 at = calloc( l+1, sizeof(int) ); fat=0;
 if( tx==NULL || ty==NULL || at==NULL ) {printf("*** not enough memory for
tx,ty,at\n");exit(1);};

 /* create tx, ty, at */
```

```
 for( t=1; t<=n; t++)
 {
   tx[t]=fat;
   for( i=0; i<l; i++ )
     if( a[i].t==t && a[i].v<0 ) at[fat++]=a[i].p;

   ty[t]=fat;
   for( i=0; i<l; i++ )
     if( a[i].t==t && a[i].v>0 ) at[fat++]=a[i].p;

 }
 tx[n+1]=fat; /* fictive trs */

 px = calloc( m+2, sizeof(int) );
 py = calloc( m+2, sizeof(int) );
 ap = calloc( l+1, sizeof(int) ); fap=0;
 if( px==NULL || py==NULL || ap==NULL ) {printf("*** not enough memory for
px,py,ap\n");exit(1);};

 SubNet = calloc( n+1, sizeof(int) );
 if( SubNet==NULL ) {printf("*** not enough memory for SubNet\n");exit(1);};
 for( t=1; t<=n; t++) SubNet[t]=0;
 iSubNet=1;

 R = calloc( n, sizeof(int) ); nR=0;
 S = calloc( n, sizeof(int) ); nS=0;
 X = calloc( m, sizeof(int) ); nX=0;
 Y = calloc( m, sizeof(int) ); nY=0;
 Q = calloc( m, sizeof(int) ); nQ=0;
 if( R==NULL || S==NULL || X==NULL || Y==NULL || Q==NULL )
   {printf("*** not enough memory for R,S,X,Y,Q\n");exit(1);};

 SX = calloc( m+1, sizeof(int) );
 SY = calloc( m+1, sizeof(int) );
 if( SX==NULL || SY==NULL ) {printf("*** not enough memory for
SX,SY\n");exit(1);};
 for( p=1; p<=m; p++) { SX[p]=0; SY[p]=0; }

 /* create px, py, ap */
 for( p=1; p<=m; p++)
 {
   px[p]=fap;
   for( t=1; t<=n; t++ )
     for( i=ty[t]; i<tx[t+1]; i++ )
       if( at[i]==p ) ap[fap++]=t;

   py[p]=fap;
   for( t=1; t<=n; t++ )
     for( i=tx[t]; i<ty[t]; i++ )
       if( at[i]==p ) ap[fap++]=t;

 }
 px[m+1]=fap; /* fictive pos */

if( debug ) {
 /* print input net */
 fprintf( SubNetFile, "n: %d m: %d l: %d\n", n, m, l );
 /* print trs */
 fprintf( SubNetFile, "at:\n" );
 for( t=1; t<=n; t++ )
 {
   for( i=tx[t]; i<ty[t]; i++ )
     fprintf( SubNetFile, "%d ", -at[i] );
```

```
    for( i=ty[t]; i<tx[t+1]; i++ )
      fprintf( SubNetFile, "%d ", at[i] );

    fprintf( SubNetFile, "\n" );
  }
  /* print pos */
  fprintf( SubNetFile, "ap:\n" );
  for( p=1; p<=m; p++ )
  {
    for( i=px[p]; i<py[p]; i++ )
      fprintf( SubNetFile, "%d ", ap[i] );

    for( i=py[p]; i<px[p+1]; i++ )
      fprintf( SubNetFile, "%d ", -ap[i] );

    fprintf( SubNetFile, "\n" );
  }
 } /* if( debug ) */

  /* assign initial R set */
  t=1; R[0]=t; nR=1; SubNet[t]=iSubNet;

  loop:
  /* create R borned net B(R)=(X,Q,Y) */
  nX=0; nY=0; nQ=0;
  for( p=1; p<=m; p++)
  {
    pinp=0; pout=0;
    for( jt=0; jt<nR; ++jt )
    {
      t=R[jt];
      for( i=tx[t]; i<ty[t]; i++ )
        if( at[i]==p ) pinp=1;
      for( i=ty[t]; i<tx[t+1]; i++ )
        if( at[i]==p ) pout=1;
    }
    if( pinp && pout ) { Q[nQ++]=p; SX[p]=0; SY[p]=0; }
      else if( pinp ) { X[nX++]=p; SX[p]=iSubNet; }
        else if ( pout ) { Y[nY++]=p; SY[p]=iSubNet; }
  }

  /* check is subnet B(R) closed: create new trs set S */
  nS=0;
  /* check output trs of X */
  for( jp=0; jp<nX; jp++)
  {
    p=X[jp]; /* input pos */
    for( i=py[p]; i<px[p+1]; i++)
    {
      t=ap[i]; /* it's output trs */
      belong=0; /* to R */
      for( jt=0; jt<nR; jt++)
        if( t==R[jt] ){ belong=1; break; }
      if( ! belong )
      {
        /* add new trs to S if absent */
        belong=0; /* to S */
        for( jt=0; jt<nS; jt++ )
        if( t==S[jt] ){ belong=1; break; }
        if( ! belong ) S[nS++]=t;
      }
    }
  }
  /* check input trs of Y */
```

```
for( jp=0; jp<nY; jp++)
{
  p=Y[jp]; /* output pos */
  for( i=px[p]; i<py[p]; i++)
  {
    t=ap[i]; /* it's input trs */
    belong=0; /* to R */
    for( jt=0; jt<nR; jt++)
      if( t==R[jt] ){ belong=1; break; }
    if( ! belong )
    {
      /* add new trs to S if absent */
      belong=0; /* to S */
      for( jt=0; jt<nS; jt++ )
      if( t==S[jt] ){ belong=1; break; }
      if( ! belong ) S[nS++]=t;
    }
  }
}
/* check input & otput trs of Q */
for( jp=0; jp<nQ; jp++)
{
  p=Q[jp]; /* internal pos */
  for( i=px[p]; i<px[p+1]; i++)
  {
    t=ap[i]; /* it's input or output trs */
    belong=0; /* to R */
    for( jt=0; jt<nR; jt++)
      if( t==R[jt] ){ belong=1; break; }
    if( ! belong )
    {
      /* add new trs to S if absent */
      belong=0; /* to S */
      for( jt=0; jt<nS; jt++ )
      if( t==S[jt] ){ belong=1; break; }
      if( ! belong ) S[nS++]=t;
    }
  }
}

/* check ready condition: is subnet N(R) closed (empty S) */
if( nS==0 ) goto final;

/* expand subnet N(R): add S to R */
for( jt=0; jt<nS; jt++)
{
  t=S[jt];
  R[nR++]=t;
  SubNet[t]=iSubNet;
}
goto loop;

final:
/* print R, X, Q, Y */
if( debug)
{
  fprintf( SubNetFile, "N%d:\n", iSubNet );
  fprintf( SubNetFile, "R: " );
  for( jt=0; jt<nR; jt++ )
    fprintf( SubNetFile, "%d ", R[jt] );
  fprintf( SubNetFile, "\nX: " );
  for( jp=0; jp<nX; jp++ )
    fprintf( SubNetFile, "%d ", X[jp] );
  fprintf( SubNetFile, "\nQ: " );
```

```
  for( jp=0; jp<nQ; jp++ )
    fprintf( SubNetFile, "%d ", Q[jp] );
  fprintf( SubNetFile, "\nY: " );
  for( jp=0; jp<nY; jp++ )
    fprintf( SubNetFile, "%d ", Y[jp] );
  fprintf( SubNetFile, "\n" );
 }

 /* check are all trs processed */
 for( t=1; t<=n; t++ )
  if( SubNet[t]==0 ) /* non processed trs */
  {
    /* new subnet */
    R[0]=t; nR=1; SubNet[t]=++iSubNet;
    goto loop;
  }

 /* print subnet */
 if( debug )
 {
  fprintf( SubNetFile, "TS:\n" );
  for( t=1; t<=n; t++ )
    fprintf( SubNetFile, "%d ", SubNet[t] );
  fprintf( SubNetFile, "\nSX:\n" );
  for( p=1; p<=m; p++ )
    fprintf( SubNetFile, "%d ", SX[p] );
  fprintf( SubNetFile, "\nSY:\n" );
  for( p=1; p<=m; p++ )
    fprintf( SubNetFile, "%d ", SY[p] );
  fprintf( SubNetFile, "\n" );
 }

 /* create & write graph of subnets */
 sprintf( zFileName, "%s.p", NetFileName );
 zFile = fopen( zFileName, "w" );
 if( zFile == NULL ) {printf( "*** error open file %s\n", zFileName
);exit(2);}
 GA = calloc( iSubNet+1, sizeof(int) );
 if( GA==NULL ) {printf("*** not enough memory for GA\n");exit(1);};
 for( u=1; u<=iSubNet; u++ ) /* subnets */
 {
  for( v=1; v<=iSubNet; v++ )
    GA[v]=0;

  /* input arcs */
  for( jp=1; jp<=m; jp++ )
    if( SX[jp]==u ) /* p is input pos of subnet u */
    {
      v=SY[jp]; /* v is input node of u */
      if( v <= u ) continue;
      (GA[v])++;
    }

  /* output arcs */
  for( jp=1; jp<=m; jp++ )
    if( SY[jp]==u ) /* p is output pos of subnet u */
    {
      v=SX[jp]; /* v is output node of u */
      if( v <= u ) continue;
      (GA[v])++;
    }

  for( v=1; v<=iSubNet; v++ )
    if( GA[v] != 0 )
```

```
       fprintf( zFile, "%d %d %d\n", u, v, GA[v] );

 }
 free( GA );
 fclose( zFile );

 /* write contact place info */
 sprintf( zFileName, "%s.z", NetFileName );
 zFile = fopen( zFileName, "w" );
 if( zFile == NULL ) {printf( "*** error open file %s\n", zFileName
);exit(2);}
 fprintf( zFile, "%d\n", iSubNet );
 for( p=1; p<=m; p++ )
 {
   if( SX[p]>0 && SY[p]>0 )
     fprintf( zFile, "%d %d %d\n", pl[ p ], SX[p], SY[p] );
 }
 fclose( zFile );

 /* write files of subnets */
 for( z=1; z<=iSubNet; z++ )
 {
   sprintf( zFileName, "%s.z%d", NetFileName, z );
   zFile = fopen( zFileName, "w" );
   if( zFile == NULL ) {printf( "*** error open file %s\n", zFileName
);exit(2);}
   for( i=0; i<l; i++ )
     if( SubNet[ a[i].t ]==z )
       fprintf( zFile, "%d %d %d\n", pl[ a[i].p ], tr[ a[i].t ], a[i].v );
   fclose( zFile );
 }


free(pl);free(tr);free(a);free(tx);free(ty);free(at);free(px);free(py);free(a
p);
 free(SubNet);free(R);free(S);free(X);free(Y);free(Q);free(SX);free(SY);

 fclose( NetFile );
 fclose( SubNetFile );

 return( iSubNet );

}/*--- DecomposeIntoFSN ---*/

#ifdef __MAIN__
int main( int argc, char *argv[] )
{
  char * NetFileName =  argv[1];
  int nz;

  if( argc < 2 ) return;

  nz = DecomposeIntoFSN( NetFileName );
  printf("%d\n", nz);
}
#endif

/* Adriana project; fic.h                        */
/* Filters FSN contact places for a given fusion of FSN */

int FilterContactPlaces( char * NetFileName, int l );

/* INPUT: NetFileName.y - set of contact places */
/*        NetFileName.t - trace of collapse */
```

```
/*        l - number of collapse step */
/* OUTPUT: NetFileName.z - set of contact places for step of collapse */
/*         NetFileName.y - new set of contact places after fusion */
/* FUNCTION: create set of contact places for a given step of collapse and
filter contact places */

/* end fic.h */

/* filters contact places for fusion */

#include <stdio.h>
#include <malloc.h>
#include "uti.h"

#define FILENAMELEN 256
#define MAXSTRLEN 256

struct contact {
  int p;
  int x;
  int y;
};

typedef struct contact coty;

int InFusion( int z, int *fu, int nfu, int *i )
{
  int j, found=0;

  for( j=0; j<nfu; j++ )
  {
    if( fu[j] == z ) { (*i)=j; found=1; break; }
  }
  return( found );

} /* InFusion */

int FilterContactPlaces( char * NetFileName, int l )
{
 char FileName[ FILENAMELEN+1 ];
 char str[ MAXSTRLEN+1 ];
 FILE * zFile, * yFile;
 int p, x, y, v, i, j, k, z, ll;
 int nz, nc, ni, ic, ii, nl;
 int ifu, nfu;
 int *in;

 coty * c;
 int * fu;

 sprintf( FileName, "%s.y", NetFileName );
 zFile = fopen( FileName, "r" );
 if( zFile == NULL ) {printf( "*** error open file %s\n", FileName
);exit(2);}

 /* get size */
 nc=0;
 fgets( str, MAXSTRLEN, zFile);
 sscanf( str, "%d", &nz );
 while( ! feof( zFile ) )
 {
   fgets( str, MAXSTRLEN, zFile);

   if( feof( zFile ) ) break;
```

```
    if( str[0]=='#' || isempty(str) ){ continue; }

    sscanf( str, "%d %d %d", &p, &x, &y );

    nc++;
  }

  c = calloc( nc, sizeof( coty ) );
  if( c==NULL ) {printf("*** not enough memory for c\n");exit(1);};

  rewind( zFile );

  /* read contact info */
  ic=0;
  fgets( str, MAXSTRLEN, zFile);
  sscanf( str, "%d", &nz );
  while( ! feof( zFile ) )
  {
    fgets( str, MAXSTRLEN, zFile);

    if( feof( zFile ) ) break;

    if( str[0]=='#' || isempty(str) ){ continue; }

    sscanf( str, "%d %d %d", &p, &x, &y );
    c[ic].p=p; c[ic].x=x; c[ic].y=y;

    ic++;
  }

  fclose( zFile );

  /* read file of sequence */
  sprintf( FileName, "%s.t", NetFileName );
  zFile = fopen( FileName, "r" );
  if( zFile == NULL ) {printf( "*** error open file %s\n", FileName
);exit(2);}

/*printf( "l=%d\n", l );*/

  /* get size */
  ll=l;
  while( --ll && ! feof( zFile ) ) fgets( str, MAXSTRLEN, zFile);

  nfu=0;
  if( ! feof( zFile ) )
  {
    fgets( str, MAXSTRLEN, zFile);
    i=0;
    while( ! isempty( str+i ) )
    {
      SwallowSpace( str, &i );
      while( ! IsSpace( str, i ) ) i++;
      nfu++;
    }
  }
  else { printf( "*** unexpected end of sequence file\n" ); exit(2);}

/*printf( "nfu=%d\n", nfu );*/

  fu = calloc( nfu, sizeof( int ) );
  if( fu==NULL ) {printf("*** not enough memory for fu\n");exit(1);};
```

```
 rewind( zFile );

 /* read file */
 ll=l;
 while( --ll && ! feof( zFile ) ) fgets( str, MAXSTRLEN, zFile);

 ifu=0;
 if( ! feof( zFile ) )
 {
   fgets( str, MAXSTRLEN, zFile);
   i=0;
   while( ! isempty( str+i ) )
   {
     SwallowSpace( str, &i );
     fu[ifu]=atoi( str+i );
     while( ! IsSpace( str, i ) ) i++;
     ifu++;
   }
 }
 else { printf( "*** unexpected end of sequence file\n" ); exit(2);}

/*for( i=0; i<nfu; i++ )
  printf( "%d ", fu[i] );
printf( "\n" );*/

 fclose( zFile );

 /* write file of fusion on step .z and new .y */
 sprintf( FileName, "%s.z", NetFileName );
 zFile = fopen( FileName, "w" );
 if( zFile == NULL ) {printf( "*** error open file %s\n", FileName
);exit(2);}
 fprintf( zFile, "%d\n", nfu );

 sprintf( FileName, "%s.y", NetFileName );
 yFile = fopen( FileName, "w" );
 if( yFile == NULL ) {printf( "*** error open file %s\n", FileName
);exit(2);}
 fprintf( yFile, "%d\n", nz-nfu );

 for( ic=0; ic<nc; ic++ )
 {
   if( InFusion( c[ic].x, fu, nfu, &i ) && InFusion( c[ic].y, fu, nfu, &j ) )
   {
     fprintf( zFile, "%d %d %d\n", c[ic].p, i+1, j+1 );
   }
   else
   {
     if( InFusion( c[ic].x, fu, nfu, &i ) ) c[ic].x=fu[0];
     else if( InFusion( c[ic].y, fu, nfu, &j ) ) c[ic].y=fu[0];

     fprintf( yFile, "%d %d %d\n", c[ic].p, c[ic].x, c[ic].y );
   }

 }

 z=fu[0];
 free( c ); free( fu );

 fclose( zFile );
 fclose( yFile );

 return( z );
```

```
} /* FilterContactPlaces */

#ifdef __MAIN__
main( int argc, char * argv[] )
{
  int l, z;

  if( argc < 3 ) return;
  l = atoi( argv[2]);

  z = FilterContactPlaces( arfv[1], l );

  printf( "%d\n", z );
}
#endif

/* Adriana project; fui.h                                    */
/* Adds unit solution to solutions of composition system */

void AddUnitSolutions( char * r0FileName, char * uFileName, char * rFileName
);

/* INPUT: r0FileName - solutions of composition system */
/*        uFileName  - unit solutions of composition system */
/* OUTPUT: rFileName - all the solutions of composition system */
/* FUNCTION: add unit solutions to solutions of composition system */

/* end fui.h */

/* fusion of composition system's solution */

#include <stdio.h>
#include "uti.h"

#define MAXSTRLEN 256

void AddUnitSolutions( char * aFileName, char * bFileName, char * cFileName )
{
 FILE * aFile, * bFile, * cFile;
 char str[ MAXSTRLEN+1 ];
 int k, i, z, v;

 aFile = fopen( aFileName, "r" );
 if( aFile == NULL ) {printf( "*** error open file %s\n", aFileName
);exit(2);}
 bFile = fopen( bFileName, "r" );
 if( bFile == NULL ) {printf( "*** error open file %s\n", bFileName
);exit(2);}
 cFile = fopen( cFileName, "w" );
 if( cFile == NULL ) {printf( "*** error open file %s\n", cFileName
);exit(2);}

 k=0;
 /* read solutions */
 while( ! feof( aFile ) )
 {
   fgets( str, MAXSTRLEN, aFile);

   if( feof( aFile ) ) break;

   if( str[0]=='#' || isempty(str) ){ continue; }

   sscanf( str, "%d %d %d", &i, &z, &v );
   fprintf( cFile, "%d %d %d\n", i, z, v );
```

```
    k=max(k,i);
  }
  fclose(aFile);

  /* read internal solutions */
  while( ! feof( bFile ) )
  {
    fgets( str, MAXSTRLEN, bFile);

    if( feof( bFile ) ) break;

    if( str[0]=='#' || isempty(str) ){ continue; }

    sscanf( str, "%d %d %d", &i, &z, &v );

    fprintf( cFile, "%d %d %d\n", ++k, z, v );
  }
  fclose( bFile );
  fclose( cFile );

} /* AddUnitSolutions */

#ifdef __MAIN__
main( int argc, char * argv[] )
{
  if( argc < 3 ) return;
  AddUnitSolutions( argv[1], argv[2], argv[3] );
}
#endif

/* Adriana project: mui.h                                  */
/* Composes joint matrix of FSN basis invariants           */

void ComposeJointMatrix( char * NetFileName );

/* INPUT: FileName.z - contact places of decomposition */
/*        FileName.i#i  - basis invariants of FSN */
/* OUTPUT: FileName.g - joint matrix of FSN basis invariants */
/* FUNCTION: add unit solutions to solutions of composition system */

/* end mui.h */

/* creates united matrix of invariants */

#include <stdio.h>
#include <malloc.h>
#include "uti.h"

#define FILENAMELEN 256
#define MAXSTRLEN 256

struct contact {
  int p;
  int x;
  int y;
};

typedef struct contact coty;

static int is=0;

#define vnum( j ) (is+(j))
```

```
void ComposeJointMatrix( char * NetFileName )
{
 char zFileName[ FILENAMELEN+1 ];
 char gFileName[ FILENAMELEN+1 ];
 char str[ MAXSTRLEN+1 ];
 FILE * zFile, * gFile;
 int p, x, y, v, i, z;
 int nz, nc, ni, ic, ii, nl;

 coty * c;
 int yes;

 sprintf( zFileName, "%s.z", NetFileName );
 zFile = fopen( zFileName, "r" );
 if( zFile == NULL ) {printf( "*** error open file %s\n", zFileName
);exit(2);}
 sprintf( gFileName, "%s.g", NetFileName );
 gFile = fopen( gFileName, "w" );
 if( gFile == NULL ) {printf( "*** error open file %s\n", gFileName
);exit(2);}

 /* get size */
 nc=0;
 fgets( str, MAXSTRLEN, zFile);
 sscanf( str, "%d", &nz );
 while( ! feof( zFile ) )
 {
   fgets( str, MAXSTRLEN, zFile);

   if( feof( zFile ) ) break;

   if( str[0]=='#' || isempty(str) ){ continue; }

   sscanf( str, "%d %d %d", &p, &x, &y );

   nc++;
 }

 c = calloc( nc, sizeof( coty ) );
 if( c==NULL ) {printf("*** not enough memory for c\n");exit(1);};

 rewind( zFile );

 /* read contact info */
 ic=0;
 fgets( str, MAXSTRLEN, zFile);
 sscanf( str, "%d", &nz );
 while( ! feof( zFile ) )
 {
   fgets( str, MAXSTRLEN, zFile);

   if( feof( zFile ) ) break;

   if( str[0]=='#' || isempty(str) ){ continue; }

   sscanf( str, "%d %d %d", &p, &x, &y );
   c[ic].p=p; c[ic].x=x; c[ic].y=y;

   ic++;
 }

 fclose( zFile );

 for( z=1; z<=nz; z++ )
```

```
 {
   sprintf( zFileName, "%s.i%d", NetFileName, z );
   zFile = fopen( zFileName, "r" );
   if( zFile == NULL ) {printf( "*** error open file %s\n", zFileName
);exit(2);}

   /* read inv of subnet */
   ni=0;
   nl=0;
   while( ! feof( zFile ) )
   {
     fgets( str, MAXSTRLEN, zFile);

     if( feof( zFile ) ) break;

     if( str[0]=='#' || isempty(str) ){ continue; }

     sscanf( str, "%d %d %d", &i, &p, &v );

     yes=1;
     for( ic=0; ic<nc; ic++ )
       if( c[ic].p==p && ( c[ic].y==z && c[ic].x>0 ) ) { yes=0; break; }

     if(yes) fprintf( gFile, "%d %d %d\n", vnum(i), p, v );

     nl=max(nl,i);
     ni++;
   }

   is+=nl;

   fclose( zFile );

 } /* on subnets */

 free(c);

 fclose(gFile);

} /* ComposeJointMatrix */

#ifdef __MAIN__
main( int argc, char * argv[] )
{
  if( argc < 2 ) return;
  ComposeJointMatrix( argv[1] );
} /* main */
#endif

/* Adriana project; readnet.h                          */
/* read net in .ndr/.net format and transform it in SPM format */

int ReadNDRNET( char * NetFileName, char * SPMFileName, char * TxtFileName,
int verbose );

/* INPUT: NetFileName - PN in .ndr/.net format */
/* OUTPUT: NetFileName.spm - nat in SPM format */
/*         NetFileName.nmp - table of places names */
/*         NetFileName.nmt - table of transitions names */
/* FUNCTION: transforms PN into SPM format and store tables of names */

/* end readnet.h */
```

```
/* READNET - read .net .ndr files and write 'm in sparse matrix and names'
tables */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>
#include "uti.h"

#define MAXINPSTRLEN 1024
#define MAXFILENAME 256

#define nINIT 1024
#define mINIT 1024
#define atpINIT 2048
#define aptINIT 2048
#define namesINIT 16384

#define nDELTA 1024
#define mDELTA 1024
#define atpDELTA 2048
#define aptDELTA 2048
#define namesDELTA 16384

#define NDR 1
#define NET 2

static char str[ MAXINPSTRLEN + 1 ]; /* line buffer */

static int n, m, maxn, maxm; /* net size: trs, pls, arcs */
static int *tn, fat; /* trs */
static int *pn, fap; /* pls */

static char *names; /* all the names */
static int fnames, maxnames;
static int netname=-1;

static int *atpp, *atpt, *atpw; /* arcs t->p */
static int *aptp, *aptt, *aptw; /* arcs p->t */
static int fatp, fapt, maxatp, maxapt;

void GetName( int *i, int *j )
{
 int state;

 if( str[(*i)]=='{' )
 {
   state=1;
   while( str[(*i)]!='\0' && state && str[(*i)]!=0xa && str[(*i)]!=0xd )
   {
    names[ (*j)++ ]=str[ (*i)++ ];
    if( str[(*i)-1]=='}' && state==1 ) state=0; else
      if( str[(*i)-1]=='\\' && state==2 ) state=1; else
        if( str[(*i)-1]=='\\' && state==1 ) state=2; else
        if( str[(*i)-1]=='}' && state==2 ) state=1; else
          if( state==2 ) state=1;
   }
   names[ (*j)++ ]='\0';
 }
 else
 {
   while( str[(*i)]!=' ' && str[(*i)]!='\0' && str[(*i)]!=0xa &&
str[(*i)]!=0xd && str[(*i)]!=0x9 && str[(*i)]!='*' )
    names[ (*j)++ ]=str[ (*i)++ ];
```

```
   names[ (*j)++ ]='\0';
 }

} /* GetName */

int RecognizeFile( FILE * f )
{
  int i, len, format;

  while( ! feof( f ) )
  {
    fgets( str, MAXINPSTRLEN, f );
    if( ferror( f ) ) {printf("*** wrong input file\n"); exit(2);}
    if( str[0]=='#' ) continue; /* comment line */

    len=strlen(str); i=0;
    SwallowSpace( str, &i );
    if( i==len ) continue; /*empty line */

    names[0]=str[i]; names[1]=str[i+1]; names[2]='\0';

    if( strcmp( names, "p ")==0 || strcmp( names, "t ")==0 || strcmp( names,
"h ")==0 )
      format=NDR; else format=NET;

    break;
  }

  return( format );

} /* RecognizeInputFile */

void ExpandNames()
{
  char * newnames;

  if( fnames+MAXINPSTRLEN > maxnames )
  {
    maxnames+=namesDELTA;
    newnames = (char*) realloc( names, maxnames );
    if( newnames==NULL ) { printf( "*** not enough memory\n" ); exit(3); }
    else names=newnames;
  }

} /* ExpandNames */

void ExpandP()
{
  int *newpx, *newpy, *newpn;

  if( m >= maxm - 2 )
  {
    maxm+=mDELTA;
    newpn = (int*) realloc( pn, maxm * sizeof(int) );
    if( newpx==NULL || newpy==NULL || newpn==NULL )
      { printf( "*** not enough memory\n" ); exit(3); }
    else
      { pn=newpn; }
  }

} /* ExpandP */

void ExpandT()
{
```

```
  int *newtx, *newty, *newtn;

  if( n >= maxn - 2 )
  {
    maxn+=nDELTA;
    newtn = (int*) realloc( tn, maxn * sizeof(int) );
    if( newtx==NULL || newty==NULL || newtn==NULL )
      { printf( "*** not enough memory\n" ); exit(3); }
    else
      { tn=newtn; }
  }

} /* ExpandT */

void ExpandAtp()
{
  int *newatpp, *newatpt;

  if( fatp>=maxatp )
  {
    maxatp+=atpDELTA;
    newatpp = (int*) realloc( atpp, maxatp * sizeof(int) );
    newatpt = (int*) realloc( atpt, maxatp * sizeof(int) );
    if( newatpp==NULL || newatpt==NULL )
      { printf( "*** not enough memory\n" ); exit(3); }
    else
      { atpp=newatpp; atpt=newatpt; }
  }

} /* ExpandAtp */

void ExpandApt()
{
  int *newaptp, *newaptt;

  if( fapt>=maxapt )
  {
    maxapt+=aptDELTA;
    newaptp = (int*) realloc( aptp, maxapt * sizeof(int) );
    newaptt = (int*) realloc( aptt, maxapt * sizeof(int) );
    if( newaptp==NULL || newaptt==NULL )
      { printf( "*** not enough memory\n" ); exit(3); }
    else
      { aptp=newaptp; aptt=newaptt; }
  }

} /* ExpandApt */

void ReadNDR( FILE * f )
{
 int i, found, p, t, inames, len, w;
 char *name1, *name2;

 m=0; n=0;
 while( ! feof( f ) )
 {
   ExpandNames();

   fgets( str, MAXINPSTRLEN, f );
   if( feof(f) ) break;
   if( str[0]=='#' ) continue; /* comment line */

   len=strlen(str); i=0;
   SwallowSpace( str, &i );
```

```
   if( i==len ) continue; /*empty line */

   switch( str[i++] )
   {
     case 'p':
      SwallowSpace( str, &i );
      while( ! IsSpace( str,i) && i<len )i++; /* x */
      SwallowSpace( str, &i );
      while( ! IsSpace( str,i) && i<len )i++; /* y */
      SwallowSpace( str, &i );
      ExpandP();
      pn[ ++m ] = fnames;
      GetName( &i, &fnames );
      found=0;
      for( p=1; p<m; p++ )
        if( strcmp( names+pn[p], names+pn[m] )==0 ) { found=1; break; }
      if( ! found )
      {
        for( t=1; t<=n; t++ )
            if( strcmp( names+tn[t], names+pn[m] )==0 ) { found=1; break; }
      }
      if( found ) { printf( "*** duplicate name: %s\n", names+pn[m] );
exit(2); }
      break;

     case 't':
            SwallowSpace( str, &i );
      while( ! IsSpace( str,i) && i<len )i++; /* x */
      SwallowSpace( str, &i );
      while( ! IsSpace( str,i) && i<len )i++; /* y */
      SwallowSpace( str, &i );
      ExpandT();
      tn[ ++n ] = fnames;
      GetName( &i, &fnames );
      found=0;
      for( p=1; p<=m; p++ )
        if( strcmp( names+pn[p], names+tn[n] )==0 ) { found=1; break; }
      if( ! found )
      {
        for( t=1; t<n; t++ )
            if( strcmp( names+tn[t], names+tn[n] )==0 ) { found=1; break; }
      }
      if( found ) { printf( "*** duplicate name: %s\n", names+tn[n] );
exit(2); }
      break;

     case 'e':
      SwallowSpace( str, &i );
      inames=fnames;
      name1=names+inames;
      GetName( &i, &inames );
      SwallowSpace( str, &i );
      if( isdigit(str[i])) while( ! IsSpace( str,i) && i<len )i++; /* rad */
      SwallowSpace( str, &i );
      if( isdigit(str[i])) while( ! IsSpace( str,i) && i<len )i++; /* ang */
      SwallowSpace( str, &i );
      name2=names+inames;
      GetName( &i, &inames );
      /* start from end */
      i=strlen( str )-1;
      while( IsSpace( str,i) && i>0 )i--;
      while( ! IsSpace( str,i) && i>0 )i--; /* anchor */
      while( IsSpace( str,i) && i>0 )i--;
      while( ! IsSpace( str,i) && i>0 )i--; /* weight */
```

```
      w=atoi( str+i+1 ); /* multiplicity */

      /* recognize arc */

      if( fapt>=maxapt || fatp>=maxatp ) { ExpandAtp(); ExpandApt(); }

      found=0;
      for( p=1; p<=m; p++ )
        if( strcmp( names+pn[p], name1 )==0 ) { ExpandApt(); aptp[fapt]=p;
found=1; break; }
      if( found )
      {
        found=0;
        for( t=1; t<=n; t++ )
          if( strcmp( names+tn[t], name2 )==0 ) { aptw[fapt]=w;
aptt[fapt++]=t; found=1; break; }
        if( ! found ) { printf( "*** unknown arc: %s -> %s\n", name1, name2 );
exit(2); }
      }
      else
      {
        found=0;
        for( t=1; t<=n; t++ )
          if( strcmp( names+tn[t], name1 )==0 ) { ExpandAtp(); atpt[fatp]=t;
found=1; break; }
        if( ! found ) { printf( "*** unknown arc: %s -> %s\n", name1, name2 );
exit(2); }
        found=0;
        for( p=1; p<=m; p++ )
          if( strcmp( names+pn[p], name2 )==0 ) { atpw[fatp]=w;
atpp[fatp++]=p; found=1; break; }
        if( ! found ) { printf( "*** unknown arc: %s -> %s\n", name1, name2 );
exit(2); }
      }
      break;

     case 'h':
       SwallowSpace( str, &i );
       netname = fnames;
       GetName( &i, &fnames );
       break;

   } /* switch */
 } /* while */
}/* ReadNDR */

void ReadNET( FILE * f )
{
 int i, j, found, p, t, inames, len, w;
 char *name1, *name2;
 char arrow[3];

 while( ! feof( f ) )
 {
   ExpandNames();
   fgets( str, MAXINPSTRLEN, f );
   if( feof(f) ) break;
   if( str[0]=='#' ) continue; /* comment line */

   len=strlen(str); i=0;

   SwallowSpace( str, &i );
   if( i==len ) continue; /*empty line */
```

```
   inames=fnames;
   GetName( &i, &inames ); /* command */

/* pl */
   if( strcmp( names+fnames, "pl" )==0 )
   {
      SwallowSpace( str, &i );
      inames=fnames;
      ExpandP();
      pn[ ++m ] = fnames;
      GetName( &i, &inames ); /* pl name */
      if( strcmp( names+fnames,"->")==0 ){ printf( "*** empty place name:
%s\n", str ); exit(2); }
      found=0;
      for( p=1; p<m; p++ )
        if( strcmp( names+pn[p], names+pn[m] )==0 ) { found=1; break; }
      if( found )
      {
        --m;
      }
      else
      {
        p=m;
        found=0;
        for( t=1; t<=n; t++ )
            if( strcmp( names+tn[t], names+pn[p] )==0 ) { found=1; break; }

        if( found ) { printf( "*** duplicate name: %s\n", names+pn[p] );
exit(2); }
          else { fnames=inames; }
      }

      SwallowSpace( str, &i );
      if( str[i]=='(' ) while( ! IsSpace( str, i ) ) i++; /* marking */
      SwallowSpace( str, &i );

      /* input arcs of pl */
      if( i<len-2) { arrow[0]=str[i]; arrow[1]=str[i+1]; arrow[2]='\0'; } else
arrow[0]='\0';
      while( strcmp( arrow, "->" )!=0 && i<len )
      {
        inames=fnames;
        name1=names+inames;
        GetName( &i, &inames ); /* input tr name */

        found=0;
        for( t=1; t<=n; t++ )
          if( strcmp( names+tn[t], name1 )==0 ) { found=1; break; }
        if( ! found ) { ExpandT(); t=++n; tn[t]=fnames; fnames=inames; }

        /* create pl input arc */
          found=0;
        for( j=0; j<fatp; j++ )
          if( atpt[j]==t && atpp[j]==p ) { found=1; break; }
        if( ! found ) { ExpandAtp(); atpt[fatp]=t; atpw[fatp]=1;
atpp[fatp++]=p; }

        /* multiplicity */
        if( str[i]=='*' )
        {
          i++;
          w=atoi( str+i );
          while( ! IsSpace( str, i ) ) i++;
```

```
          atpw[fatp-1]=w;
        }

        SwallowSpace( str, &i );
        if( i<len-2) { arrow[0]=str[i]; arrow[1]=str[i+1]; arrow[2]='\0'; }
else arrow[0]='\0';
      } /* while */

      i+=2;
      SwallowSpace( str, &i );

      /* output arcs of pl */
      while( i<len )
      {
        inames=fnames;
        name1=names+inames;
        GetName( &i, &inames ); /* output tr name */

        found=0;
        for( t=1; t<=n; t++ )
          if( strcmp( names+tn[t], name1 )==0 ) { found=1; break; }
        if( ! found ) { ExpandT(); t=++n; tn[t]=fnames; fnames=inames; }

        /* create pl output arc */
          found=0;
        for( j=0; j<fapt; j++ )
          if( aptt[j]==t && aptp[j]==p ) { found=1; break; }
        if( ! found ) { ExpandApt(); aptt[fapt]=t; aptw[fapt]=1;
aptp[fapt++]=p; }

        /* multiplicity */
        if( str[i]=='*' )
        {
          i++;
          w=atoi( str+i );
          while( ! IsSpace( str, i ) ) i++;

          aptw[fapt-1]=w;
        }

          SwallowSpace( str, &i );

      } /* while */

    } else

/* tr */
    if( strcmp( names+fnames, "tr" )==0 )
    {
      SwallowSpace( str, &i );
      inames=fnames;
      ExpandT();
      tn[ ++n ] = fnames;
      GetName( &i, &inames ); /* tr name */
      if( strcmp( names+fnames,"->")==0 ){ printf( "*** empty transition name:
%s\n", str ); exit(2); }
      found=0;
      for( t=1; t<n; t++ )
        if( strcmp( names+tn[t], names+tn[n] )==0 ) { found=1; break; }
      if( found )
      {
        --n;
      }
      else
```

```
      {
        t=n;
        found=0;
        for( p=1; p<=m; p++ )
            if( strcmp( names+pn[p], names+tn[t] )==0 ) { found=1; break; }

        if( found ) { printf( "*** duplicate name: %s\n", names+tn[t] );
exit(2); }
          else { fnames=inames; }
      }

      SwallowSpace( str, &i );
      if( str[i]=='[' || str[i]==']' ) { while( ! IsSpace( str, i ) ) i++; }
/* interval */
      SwallowSpace( str, &i );

      /* input arcs of tr */
      if( i<len-2) { arrow[0]=str[i]; arrow[1]=str[i+1]; arrow[2]='\0'; } else
arrow[0]='\0';

      while( strcmp( arrow, "->" )!=0 && i<len )
      {
        inames=fnames;
        name1=names+inames;
        GetName( &i, &inames ); /* input pl name */

        found=0;
        for( p=1; p<=m; p++ )
          if( strcmp( names+pn[p], name1 )==0 ) { found=1; break; }
        if( ! found ) { ExpandP(); p=++m; pn[p]=fnames; fnames=inames; }

        /* create tr input arc */
          found=0;
        for( j=0; j<fapt; j++ )
          if( aptt[j]==t && aptp[j]==p ) { found=1; break; }
        if( ! found ) { ExpandApt(); aptt[fapt]=t; aptw[fapt]=1;
aptp[fapt++]=p; }

        /* multiplicity */
        if( str[i]=='*' )
        {
          i++;
          w=atoi( str+i );
          while( ! IsSpace( str, i ) )i++;

          aptw[fapt-1]=w;
        }

        SwallowSpace( str, &i );
        if( i<len-2) { arrow[0]=str[i]; arrow[1]=str[i+1]; arrow[2]='\0'; }
else arrow[0]='\0';
      } /* while */

      i+=2;
      SwallowSpace( str, &i );

      /* output arcs of tr */
      while( i<len )
      {
        inames=fnames;
        name1=names+inames;
        GetName( &i, &inames ); /* output pl name */

        found=0;
```

```
        for( p=1; p<=m; p++ )
          if( strcmp( names+pn[p], name1 )==0 ) { found=1; break; }
        if( ! found ) { ExpandP(); p=++m; pn[p]=fnames; fnames=inames; }

        /* create tr output arc */
          found=0;
        for( j=0; j<fatp; j++ )
          if( atpt[j]==t && atpp[j]==p ) { found=1; break; }
        if( ! found ) { ExpandAtp(); atpt[fatp]=t; atpw[fatp]=1;
atpp[fatp++]=p; }

        /* multiplicity */
        if( str[i]=='*' )
        {
          i++;
          w=atoi( str+i );
          while( ! IsSpace( str, i ) )i++;

          atpw[fatp-1]=w;
        }

        SwallowSpace( str, &i );

      } /* while */


    } else

/* lb */
    if( strcmp( names+fnames, "lb" )==0 ) ;
    else

/* net */
    if( strcmp( names+fnames, "net" )==0 )
    {
       SwallowSpace( str, &i );
       netname = fnames;
       GetName( &i, &fnames ); /* net name */
    }
    else
      { printf( "*** unknown command: %s\n", names+fnames ); exit(2); }

  } /* while */

}/* ReadNET */

void WriteSPM( FILE * f )
{
  int i;

  for( i=0; i<fapt; i++ )
    fprintf( f, "%d %d %d\n", aptp[i], aptt[i], -aptw[i] );

  for( i=0; i<fatp; i++ )
    fprintf( f, "%d %d %d\n", atpp[i], atpt[i], atpw[i] );

}/* WriteSPM */

void WriteNMP( FILE * f )
{
  int p;

  for( p=1; p<=m; p++ )
  {
```

```
    fprintf( f, "%d %s\n", p, names + pn[p]  );
  }

}/* WriteNMP */

void WriteNMT( FILE * f )
{
  int t;

  for( t=1; t<=n; t++ )
  {
    fprintf( f, "%d %s\n", t, names + tn[t] );
  }

}/* WriteNMT */

void WriteNet( FILE * f )
{
  int t, i;

  fprintf( f, "net %s\n", (netname==-1)? "noname": names+netname );
  for( t=1; t<=n; t++ )
  {
    fprintf( f, "tr %s", names + tn[t] );
    for( i=0; i<fapt; i++ )
    {
      if( aptt[i] == t )
      {
        fprintf( f, " %s", names + pn[aptp[i]] );
        if( aptw[i] > 1 )
          fprintf( f, "*%d", aptw[i] );
      }
    }

    fprintf( f, " ->" );

    for( i=0; i<fatp; i++ )
    {
      if( atpt[i] == t )
      {
        fprintf( f, " %s", names + pn[atpp[i]] );
        if( atpw[i] > 1 )
          fprintf( f, "*%d", atpw[i] );
      }
    }

    fprintf( f, "\n" );
  }

  fprintf( f, "\n" );

}/* WriteNet */

int ReadNDRNET( char * NetFileName, char * SPMFileName, char * OutFileName,
int verbose )
{
 char nFileName[ MAXFILENAME+1 ];
 char MatrFileName[ MAXFILENAME+1 ];
 FILE * NetFile, * MatrFile, * nFile, * OutFile;
 int format;
 int z;

 int p, t, l, i, jp, jt; /* indexes */
```

```
 int pinp, pout, belong; /* flags */

 sprintf( MatrFileName, "%s.spm", SPMFileName );

 /* open files */
 if( strcmp( NetFileName, "-" )==0 ) NetFile = stdin;
   else NetFile = fopen( NetFileName, "r" );
 if( NetFile == NULL ) {printf( "*** error open file %s\n", NetFileName
);exit(2);}
 MatrFile = fopen( MatrFileName, "w" );
 if( MatrFile == NULL ) {printf( "*** error open file %s\n", MatrFileName
);exit(2);}

 /* init net size  */
 maxn=nINIT;
 maxm=mINIT;
 maxnames=namesINIT;
 maxatp=atpINIT;
 maxapt=aptINIT;

 /* allocate arrays */
 tn = (int*) calloc( maxn, sizeof(int) ); n=0;

 pn = (int*) calloc( maxm, sizeof(int) ); m=0;

 names = (char*) calloc( maxnames, sizeof(char) ); fnames=0;
 aptp = (int*) calloc( maxapt, sizeof(int) );
 aptw = (int*) calloc( maxapt, sizeof(int) );
 aptt = (int*) calloc( maxapt, sizeof(int) ); fapt=0;
 atpp = (int*) calloc( maxatp, sizeof(int) );
 atpw = (int*) calloc( maxatp, sizeof(int) );
 atpt = (int*) calloc( maxatp, sizeof(int) ); fatp=0;

 if( tn==NULL ||
     pn==NULL ||
     names==NULL ||
     aptp==NULL || aptt==NULL || aptw==NULL ||
     atpp==NULL || atpt==NULL || atpw==NULL )
   { printf( "*** not enough memory for net\n" ); return(3); }

 /* recognize input file */
 format=RecognizeFile( NetFile );

 rewind( NetFile );

 if( format==NDR )
 {
   ReadNDR( NetFile );
 }
 else
 {
   ReadNET( NetFile );
 }
 if( NetFile != stdin ) fclose( NetFile );

 WriteSPM( MatrFile );
 fclose( MatrFile );

 sprintf( nFileName, "%s.nmp", SPMFileName );
 nFile = fopen( nFileName, "w" );
 if( nFile == NULL ) {printf( "*** error open file %s\n", nFileName
);exit(2);}
 WriteNMP( nFile );
 fclose( nFile );
```

```
 sprintf( nFileName, "%s.nmt", SPMFileName );
 nFile = fopen( nFileName, "w" );
 if( nFile == NULL ) {printf( "*** error open file %s\n", nFileName
);exit(2);}
 WriteNMT( nFile );
 fclose( nFile );

 if( strcmp( OutFileName, "-" )==0 ) OutFile = stdout;
 else OutFile = fopen( OutFileName, "a" );
 if( OutFile == NULL ) {printf( "*** error open file %s\n", OutFileName
);exit(2);}
 fprintf( OutFile, "parsed net %s\n\n", (netname==-1)? "noname":
names+netname );
 fprintf( OutFile, "%d places, %d transitions\n\n", m, n );
 if( verbose ) WriteNet( OutFile );
 if( OutFile != stdout ) fclose( OutFile );

 free( tn ); free(atpp); free(atpw); free(atpt);

 free( pn ); free(aptp); free(aptw); free(aptt);

 free( names );

}/* ReadNDRNET */

#ifdef __MAIN__
int main( int argc, char *argv[] )
{
  if( argc < 2 ) return;
  ReadNDRNET( argv[1] );
}
#endif

/* Adriana project: smb2.h                                  */
/* Minimizes basuis in SPM format                          */

void MinimizeBasis( char * AFileName, char * BFileName );

/* INPUT: AFileName - source basis A */
/* OUTPUT: BFileName - minimized basis B */
/* FUNCTION: minimize basis in SPM format */

/* end smb2.h */

/* minimizes basis with dynamic allocation of matrices */

#include <stdio.h>
#include <stdlib.h>
#include "uti.h"

/* sizes of matrices */
static int ma, na;

#define ea( a, i, j ) (*((a)+(((i)-1)*na)+((j)-1)))

#define initROWS 1024
#define deltaROWS 1024
#define initCOLS 1024
#define deltaCOLS 1024

#define LINESIZE 256

void MinimizeBasis( char * AFileName, char * BFileName )
```

```
{
 FILE * AFile, * BFile;
 char   line[ LINESIZE+1 ];
 int * A, * B, * rowsa, * colsa;
 int maxrowsa, maxcolsa;
 int i, j, k, v, p, t, found;
 int i1, i2, i1lei2;
 int s;

 /* matrix A */
  AFile = fopen( AFileName, "r" );
  if( AFile == NULL ) {printf("*** error open file %s\n", AFileName );
exit(0);}

  maxrowsa = initROWS; rowsa = malloc( maxrowsa*sizeof(int) );
  maxcolsa = initCOLS; colsa = malloc( maxcolsa*sizeof(int) );
  if( rowsa==NULL || colsa==NULL ) {printf("*** not enough memory for
rowsa,colsa\n");exit(1);};

  /* estimate matrix's A size */
  ma=na=0;
  while ( ! feof( AFile ) )
  {
    fgets( line, LINESIZE, AFile );

    if( feof( AFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
    sscanf( line, "%d %d %d", &p, &t, &v );

    found=0;
    for( i=1; i<=ma; i++ ) if( rowsa[i]==p ) { found=1; break; }
    if( ! found )
    {
      if( ma >= maxrowsa )
      {
        maxrowsa+=deltaROWS;
      rowsa = realloc( rowsa, maxrowsa*sizeof(int) );
      if( rowsa == NULL ) {printf("*** not enough memory for
rowsa\n");exit(1);};
      }
      rowsa[++ma]=p;
    }
    found=0;
    for( j=1; j<=na; j++ ) if( colsa[j]==t ) { found=1; break; }
    if( ! found )
    {
      if( na >= maxcolsa )
      {
        maxcolsa+=deltaCOLS;
      colsa = realloc( colsa, maxcolsa*sizeof(int) );
      if( colsa == NULL ) {printf("*** not enough memory for
colsa\n");exit(1);};
      }
      colsa[++na]=t;
    }

  } /* feof AFile */

/*printf("rows (mc=%d): ",mc);
for(i=1;i<=mc;i++)
  printf("%d ", rows[i] );
```

```
printf("\ncols (nc=%d): ",nc);
for(i=1;i<=nc;i++)
  printf("%d ", cols[i] );
printf("\n");*/

  rewind( AFile );
  rowsa = realloc( rowsa, (ma+1)*sizeof(int) );
  colsa = realloc( colsa, (na+1)*sizeof(int) );
  A = calloc( ma*na, sizeof(int) );
  if( rowsa==NULL || colsa==NULL || A==NULL ) {printf("*** not enough memory
for rowsa,colsa,A\n");exit(1);}

  /* read matrix A */
  while ( ! feof( AFile ) )
  {
    fgets( line, LINESIZE, AFile );

    if( feof( AFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
    sscanf( line, "%d %d %d", &p, &t, &v );

    found=0;
    for( i=1; i<=ma; i++ ) if( rowsa[i]==p ) { found=1; break; }
    if( ! found ) { printf("***inconsistent input"); exit(4); }
    found=0;
    for( j=1; j<=na; j++ ) if( colsa[j]==t ) { found=1; break; }
    if( ! found ) { printf("***inconsistent input"); exit(4); }

    ea( A, i, j )+=v;

  } /* feof AFile */

  fclose( AFile );

 B = calloc( ma+1, sizeof(int) );
 if( B==NULL ) {printf("*** not enough memory for B\n");exit(1);}

 for( i1=1; i1<=ma; ++i1 )
 {
   if( B[i1] > 0 ) continue;
   for( i2=1; i2<=ma; ++i2 )
   {
     if( B[i2] > 0 ) continue;
     if( i1 == i2 ) continue;
     i1lei2=1;
     for( j=1; j<=na; ++j )
     {
        if( ea( A, i1, j ) > ea( A, i2, j ) ) { i1lei2=0; break; }
     }
     B[i2]=i1lei2;
   }
 }

 BFile = fopen( BFileName, "w" );
 if( BFile == NULL ) {printf( "*** error open file %s\n", BFileName
);exit(2);}

 k=0;
 for( i=1; i<=ma; ++i )
 {
   if( B[i] > 0 ) continue;
```

```
   k++;
   for( j=1; j<=na; ++j )
   {
     if( ea( A, i, j )!=0 )
       fprintf( BFile, "%d %d %d\n", k, colsa[j], ea( A, i, j ) );
   }

 }

 free(rowsa); free(colsa); free(A);
 free(B);

 fclose( BFile );

}/* MinimizeBasis */

#ifdef __MAIN__
int main( int argc, char *argv[] )
{
  if( argc < 3 )
  {
    printf("mb: minimization of basis\n");
    printf("MB matrAfile matrBfile\n");
    exit(0);
  }
  MinimizeBasis( argv[1], argv[2] );
}/* main */
#endif

/* Adriana project: smmul2.h                                    */
/* Multiplies matrices in SPM format                            */

void MultiplySPM( char * AFileNeme, char * BFileName, char * CFileName );

/* INPUT: AFileName - first matrix A */
/*        BFileName - second matrix B */
/* OUTPUT: CFileName - result matrix C = A x B */
/* FUNCTION: multilply matrices in SPM format */

/* end smmul2.h */

/* multiplies sparse matrices with dynamic allocation of matrices*/

#include <stdio.h>
#include <stdlib.h>
#include "uti.h"

static int ma, na, mb, nb;

#define ea( a, i, j ) (*((a)+(((i)-1)*na)+((j)-1)))
#define eb( b, i, j ) (*((b)+(((i)-1)*nb)+((j)-1)))

#define initROWS 1024
#define deltaROWS 1024
#define initCOLS 1024
#define deltaCOLS 1024

#define LINESIZE 256

void MultiplySPM( char * AFileName, char * BFileName, char * CFileName )
{
 FILE * AFile, * BFile, * CFile;
 int * A, * B, * rowsa, *rowsb, * colsa, * colsb, * pairs;
 int maxrowsa, maxrowsb, maxcolsa, maxcolsb;
```

```
 int i, j, k, l, v, s, p, t;
 char line[ LINESIZE+1 ];
 int first, found;

  /* matrix A */
  AFile = fopen( AFileName, "r" );
  if( AFile == NULL ) {printf("*** error open file %s\n", AFileName );
exit(0);}

  maxrowsa = initROWS; rowsa = malloc( maxrowsa*sizeof(int) );
  maxcolsa = initCOLS; colsa = malloc( maxcolsa*sizeof(int) );
  if( rowsa==NULL || colsa==NULL ) {printf("*** not enough memory for rowsa,
colsa\n");exit(1);};

  /* estimate matrix's A size */
  ma=na=0;
  while ( ! feof( AFile ) )
  {
    fgets( line, LINESIZE, AFile );

    if( feof( AFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
    sscanf( line, "%d %d %d", &p, &t, &v );

    found=0;
    for( i=1; i<=ma; i++ ) if( rowsa[i]==p ) { found=1; break; }
    if( ! found )
    {
      if( ma >= maxrowsa )
      {
        maxrowsa+=deltaROWS;
      rowsa = realloc( rowsa, maxrowsa*sizeof(int) );
      if( rowsa == NULL ) {printf("*** not enough memory for
rowsa\n");exit(1);};
      }
      rowsa[++ma]=p;
    }
    found=0;
    for( j=1; j<=na; j++ ) if( colsa[j]==t ) { found=1; break; }
    if( ! found )
    {
      if( na >= maxcolsa )
      {
        maxcolsa+=deltaCOLS;
      colsa = realloc( colsa, maxcolsa*sizeof(int) );
      if( colsa == NULL ) {printf("*** not enough memory for
colsa\n");exit(1);};
      }
      colsa[++na]=t;
    }

  } /* feof AFile */

  rewind( AFile );
  rowsa = realloc( rowsa, (ma+1)*sizeof(int) );
  colsa = realloc( colsa, (na+1)*sizeof(int) );
  A = calloc( ma*na, sizeof(int) );
  if( rowsa==NULL || colsa==NULL || A==NULL ) {printf("*** not enough memory
for rowsa, colsa, A\n");exit(1);}

  /* read matrix A */
```

```
  while ( ! feof( AFile ) )
  {
    fgets( line, LINESIZE, AFile );

    if( feof( AFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
    sscanf( line, "%d %d %d", &p, &t, &v );

    found=0;
    for( i=1; i<=ma; i++ ) if( rowsa[i]==p ) { found=1; break; }
    if( ! found ) { printf("***inconsistent input"); exit(4); }
    found=0;
    for( j=1; j<=na; j++ ) if( colsa[j]==t ) { found=1; break; }
    if( ! found ) { printf("***inconsistent input"); exit(4); }

    ea( A, i, j )+=v;

  } /* feof AFile */

  fclose( AFile );

  BFile = fopen( BFileName, "r" );
  if( BFile == NULL ) {printf("*** error open file %s\n", BFileName );
exit(0);}

  /* matrix B */
  maxrowsb = initROWS; rowsb = malloc( maxrowsb*sizeof(int) );
  maxcolsb = initCOLS; colsb = malloc( maxcolsb*sizeof(int) );
  if( rowsb==NULL || colsb==NULL ) {printf("*** not enough memory for rowsb,
colsb\n");exit(1);};

  /* estimbte mbtrix's B size */
  mb=nb=0;
  while ( ! feof( BFile ) )
  {
    fgets( line, LINESIZE, BFile );

    if( feof( BFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
    sscanf( line, "%d %d %d", &p, &t, &v );

    found=0;
    for( i=1; i<=mb; i++ ) if( rowsb[i]==p ) { found=1; break; }
    if( ! found )
    {
      if( mb >= maxrowsb )
      {
        maxrowsb+=deltaROWS;
      rowsb = realloc( rowsb, maxrowsb*sizeof(int) );
      if( rowsb == NULL ) {printf("*** not enough memory for
rowsb\n");exit(1);};
      }
      rowsb[++mb]=p;
    }
    found=0;
    for( j=1; j<=nb; j++ ) if( colsb[j]==t ) { found=1; break; }
    if( ! found )
    {
```

```
      if( nb >= maxcolsb )
      {
        maxcolsb+=deltaCOLS;
      colsb = realloc( colsb, maxcolsb*sizeof(int) );
      if( colsb == NULL ) {printf("*** not enough memory for
colsb\n");exit(1);};
      }
      colsb[++nb]=t;
    }

  } /* feof BFile */

  rewind( BFile );
  rowsb = realloc( rowsb, (mb+1)*sizeof(int) );
  colsb = realloc( colsb, (nb+1)*sizeof(int) );
  B = calloc( mb*nb, sizeof(int) );
  if( rowsb==NULL || colsb==NULL || B==NULL ) {printf("*** not enough memory
for rowsb, colsb, B\n");exit(1);}

  /* rebd mbtrix B */
  while ( ! feof( BFile ) )
  {
    fgets( line, LINESIZE, BFile );

    if( feof( BFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
    sscanf( line, "%d %d %d", &p, &t, &v );

    found=0;
    for( i=1; i<=mb; i++ ) if( rowsb[i]==p ) { found=1; break; }
    if( ! found ) { printf("***inconsistent input"); exit(4); }
    found=0;
    for( j=1; j<=nb; j++ ) if( colsb[j]==t ) { found=1; break; }
    if( ! found ) { printf("***inconsistent input"); exit(4); }

    eb( B, i, j )+=v;

  } /* feof BFile */

  fclose( BFile );

 /* multiplication */

 pairs = calloc( na+1, sizeof(int) );
 if( pairs==NULL ) {printf("*** not enough memory for pairs\n");exit(1);};
 for( k=1; k<=na; k++)
   for( l=1; l<=mb; l++ )
   {
     if( colsa[k] != rowsb[l] ) continue;
     pairs[k]=l;
   }

 CFile = fopen( CFileName, "w" );
 if( CFile == NULL ) {printf("*** error open file %s\n", CFileName );
exit(0);}
 for( i=1; i<=ma; i++)
 {
   for( j=1; j<=nb; j++)
   {
     s=0;
```

```
     for( k=1; k<=na; k++)
     {
         if( pairs[k] == 0 ) continue;
       l=pairs[k];
         s+=ea( A, i, k ) * eb( B, l, j );
     }

     if( s != 0 )
       fprintf( CFile, "%d %d %d\n", rowsa[i], colsb[j], s );

   }
 }

 free(rowsa); free(colsa); free(A);
 free(rowsb); free(colsb); free(B);
 free(pairs);

 fclose( CFile );

}/* MultiplySPM */

#ifdef __MAIN__
int main( int argc, char *argv[] )
{
  if( argc < 3 )
  {
    printf("smmul: multiplication of sparse matrixes\n");
    printf("SMMUL matrAfile matrBfile matrCfile\n");

    exit(0);
  }
  MultiplySPM( argv[1], argv[2], argv[3] );
}/* main */
#endif

/* Adriana project; toy2.h                    */
/* Toudic method for SPM matrix format        */

int Toudic( char * SystemFileName, char * SolutionsFileName );

/* end toy2.h */

/* Toudic for zero & dynamic allocation of matrixes & fast on fly filtering
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "uti.h"

#define LINESIZE 256

/* sizes of matrices */
static int mc, nc, mx, nx;

#define ec( c, i, j ) (*((c)+(((i)-1)*nc)+((j)-1)))
#define ex( x, i, j ) (*((x)+(((i)-1)*nx)+((j)-1)))

#define lc( c, i ) ((c)+(((i)-1)*nc))
#define lx( x, i ) ((x)+(((i)-1)*nx))

#define sizelc (nc*sizeof(int))
#define sizelx (nx*sizeof(int))
```

```
#define initROWS 1024
#define deltaROWS 1024
#define initCOLS 1024
#define deltaCOLS 1024
#define deltaSOL 64

void onex( int * X )
{
 int i;

 for( i=1; i<=mx; ++i )
   ex( X, i, i )=1;
} /* onex */

int findzero( int * C, int mm, int *nIz, int * Iz )
{
 int i, j, k, kk;
 int mi, pl;
 int * wIm, * wIp;
 int found;

 wIm = malloc( mm*sizeof(int) );
 wIp = malloc( mm*sizeof(int) );
 if( wIm==NULL || wIp==NULL ) {printf("*** not enough memory for
wIm,wIp\n");exit(1);};
 (*nIz)=0;

 for( j=1; j<=nc; ++j )
 {
   pl=mi=0;
   for( i=1; i<=mm; ++i )
   {
       if( ec( C, i, j ) < 0 ) {wIm[mi]=i; ++mi;}
       else if ( ec( C, i, j ) > 0 ) {wIp[pl]=i; ++pl;}
   }

   if( (pl==0) && (mi!=0) )
   {
     for( k=0; k<mi; ++k )
     {
       i=wIm[k];
       found=0;
       for( kk=0; kk<(*nIz); ++kk )
         if( Iz[kk]==i ) {found=1; break;}
       if( !found ) { Iz[(*nIz)++]=i; }
     }
   }

   if( (pl!=0) && (mi==0) )
   {
     for( k=0; k<pl; ++k )
     {
       i=wIp[k];
       found=0;
       for( kk=0; kk<(*nIz); ++kk )
         if( Iz[kk]==i ) {found=1; break;}
       if( !found ) { Iz[(*nIz)++]=i; }
     }
   }

 }

 free( wIp );
 free( wIm );
```

```
 return( (*nIz) );

} /* findzero */

void copyrowc( int * C, int i1, int i2 )
{
  int j;

  for( j=1; j<=nc; ++j ) ec( C, i1, j )=ec( C, i2, j );

} /* copyrowc */

void copyrowx( int * X, int i1, int i2 )
{
  int j;

  for( j=1; j<=nx; ++j ) ex( X, i1, j )=ex( X, i2, j );

} /* copyrowx */

void erasezero( int * C, int * X, int * mm, int nIz, int * Iz )
{
  int i1, i2, k;
  int found;

  i1=i2=1;

  while( i2<=(*mm) )
  {
    found=0;
    for( k=0; k<nIz; ++k )
      if( Iz[k]==i2 ) { found=1; break; }
    if     ( i1==i2 && !found ) { i1++; i2++; }
    else if( i1==i2 && found )  { i2++; }
    else if( i1<i2  && !found ) { copyrowc( C, i1, i2 ); copyrowx( X, i1++,
i2++ ); }
    else if( i1<i2  && found )  { i2++; }
  }

  (*mm)-=nIz;

} /* erasezero */

int findcol( int * C, int mm, int *col, int *nIp, int * Ip, int *nIm, int *
Im )
{
 int i, j, k;
 int mi, pl;
 int * wIm, * wIp;
 int success=1;

 wIm = malloc( mm*sizeof(int) );
 wIp = malloc( mm*sizeof(int) );
 if( wIm==NULL || wIp==NULL ) {printf("*** not enough memory for
wIm,wIp\n");exit(1);};
 (*nIm)=(*nIp)=mm;
 (*col)=0;

/*printf("findcol: nc=%d\n", nc );*/
 for( j=1; j<=nc; ++j )
 {
/*printf("findcol1: j=%d\n",j );fflush(stdout);*/
   pl=mi=0;
   for( i=1; i<=mm; ++i )
```

```
    {
        if( ec( C, i, j ) < 0 ) {wIm[mi]=i; ++mi;}
        else if( ec( C, i, j ) > 0 ) {wIp[pl]=i; ++pl;}
    }

/*printf("j=%d pl=%d mi=%d\n", j, pl, mi );*/

    if( (pl==0) && (mi==0) ) continue;
    if( (pl==0) || (mi==0) ) { success=0; (*col)=j; break; }
    if( pl*mi < (*nIp) * (*nIm) )
    {
      (*col)=j; (*nIm)=mi; (*nIp)=pl;
      for( k=0; k<mm; ++k)
        { Im[k]=wIm[k]; Ip[k]=wIp[k];}
    }
 }

 if( (*col) == 0 ) success = 0;
 free( wIp );
 free( wIm );
 return( success );

} /* findcol */

int nod( int y, int z )
{
 int a, b, c;

 a=max(y,z);
 b=min(y,z);

 while( a != b )
 {
   c=a-b;
   a=max(b,c);
   b=min(b,c);
 }

 return( a );

} /* nod */

int nodv( int * a, int n )
{
 int i,d;
 int yes=0;

 for( i=0; i<n; ++i) if(a[i]!=0) { yes=1; break;}

 if(!yes) return 0;

 d=abs(a[i]);

 for( i=0; i<n; ++i) if(a[i]!=0) { d=nod(d,abs(a[i]));}

 return d;

} /* nodm */

void divv( int * a, int n, int d )
{
 int i;

 for( i=0; i<n; ++i) if(a[i]!=0) {a[i]=a[i]/d;}
```

```
} /* nodm */

void shiftmatrc( int * C, int mm, int col )
{
  int i, j;

  for( i=col; i<mm; i++ )
    for( j=1; j<=nc; j++ )
      ec( C, i, j ) = ec( C, i+1, j );

} /* shiftmatrc */

void shiftmatrx( int * X, int mm, int col )
{
  int i, j;

  for( i=col; i<mm; i++ )
    for( j=1; j<=nx; j++ )
      ex( X, i, j ) = ex( X, i+1, j );

} /* shiftmatrx */

void delzero( int * C, int * X, int *mm )
{
  int i, ii, j;
  int zero, dbl;

  /* erase zero solutions */
  i=1;
  while( i<=(*mm) )
  {
    zero=1;
    for(j=1; j<=nx; j++ )
      if( ex( X, i, j ) != 0 ) { zero=0; break; }

    if( zero ) { shiftmatrc( C, (*mm), i );  shiftmatrx( X, (*mm), i );
(*mm)--; } else i++;
  }

} /* delzero */

void deldbl( int * C, int * X, int *mm )
{
  int i, ii, j;
  int zero, dbl;

  /* erase double solutions */
  for( i=1; i<=(*mm); i++ )
    for( ii=i+1; ii<=(*mm); ii++)
    {
      dbl=1;
      for(j=1; j<=nx; j++ )
        if( ex( X, i, j ) != ex( X, ii, j ) ) { dbl=0; break; }

      if( dbl ) {  shiftmatrc( C, (*mm), ii );  shiftmatrx( X, (*mm), ii );
(*mm)--; }
    }

} /* deldbl */

void FilterNewSolution( int * C, int * X, int *f )
{
  int i, ii, j;
```

```
  int lo_f, hi_f, upper;

  /* compare with last */
  upper=0;
  for( i=1; i<(*f); i++ )
  {
    lo_f=1;
    hi_f=1;

    for( j=1; j<=nx; j++ )
    {
      if( ex( X, *f, j ) > ex( X, i, j ) ) lo_f=0;
      if( ex( X, *f, j ) < ex( X, i, j ) ) hi_f=0;
      if( lo_f==0 && hi_f==0 ) break;
    }

    /* erase X(i) */
    if( lo_f && !hi_f ) { for( j=1; j<=nx; j++) ex( X, i, j ) = 0; }

    upper = upper || hi_f;

  }

  /* erase upper */
  if( upper ) { for( j=1; j<=nx; j++) ex( X, *f, j ) = 0; }

  delzero( C, X, f );

} /* FilterNewSolution */

int Toudic( char * InputFileName, char * OutputFileName )
{
  char line[ LINESIZE+1 ];
  FILE * InputFile, * OutputFile;
  int * C, * C1;
  int * X, * X1;
  int mc1, mx1;
  int col, nIp, nIz, nIm;
  int ilp, ilm;
  int * Im, * Ip, * Iz;
  int i, j, k;
  int v, f, found;
  int lm, lz, lp;
  int cm, cp, d, d1, d2;
  int in, p, t;
  int * rows, * cols;
  int nr, ret=0;
  int maxrows, maxcols;

  InputFile = fopen( InputFileName, "r" );
  if( InputFile == NULL ) {printf( "*** error open file %s\n", InputFileName
);exit(2);}
  OutputFile = fopen( OutputFileName, "w" );
  if( OutputFile == NULL ) {printf( "*** error open file %s\n",
OutputFileName );exit(2);}

  maxrows = initROWS; rows = malloc( maxrows*sizeof(int) );
  maxcols = initCOLS; cols = malloc( maxcols*sizeof(int) );
  if( rows==NULL || cols==NULL ) {printf("*** not enough memory for
rows,cols\n");exit(1);};

  /* estimate matrix's size */
  mc=nc=0;
  while ( ! feof( InputFile ) )
```

```
  {
    fgets( line, LINESIZE, InputFile );

    if( feof( InputFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
    sscanf( line, "%d %d %d", &p, &t, &v );

    found=0;
    for( i=1; i<=mc; i++ ) if( rows[i]==p ) { found=1; break; }
    if( ! found )
    {
      if( mc >= maxrows )
      {
        maxrows+=deltaROWS;
      rows = realloc( rows, maxrows*sizeof(int) );
      if( rows == NULL ) {printf("*** not enough memory for
rows\n");exit(1);};
      }
      rows[++mc]=p;
    }
    found=0;
    for( j=1; j<=nc; j++ ) if( cols[j]==t ) { found=1; break; }
    if( ! found )
    {
      if( nc >= maxcols )
      {
        maxcols+=deltaCOLS;
      cols = realloc( cols, maxcols*sizeof(int) );
      if( cols == NULL ) {printf("*** not enough memory for
cols\n");exit(1);};
      }
      cols[++nc]=t;
    }

  } /* feof InputFile */

/*printf("rows (mc=%d): ",mc);
for(i=1;i<=mc;i++)
  printf("%d ", rows[i] );
printf("\ncols (nc=%d): ",nc);
for(i=1;i<=nc;i++)
  printf("%d ", cols[i] );
printf("\n");*/

  rewind( InputFile );
  rows = realloc( rows, (mc+1)*sizeof(int) );
  cols = realloc( cols, (nc+1)*sizeof(int) );
  C = calloc( mc*nc, sizeof(int) );
  if( rows==NULL || cols==NULL || C==NULL ) {printf("*** not enough memory
for rows,cols,C\n");exit(1);}

  /* read matrix */
  while ( ! feof( InputFile ) )
  {
    fgets( line, LINESIZE, InputFile );

    if( feof( InputFile ) ) break;

    if( line[0] == '#' || isempty(line) ) continue;

    v=1;
```

```
    sscanf( line, "%d %d %d", &p, &t, &v );

    found=0;
    for( i=1; i<=mc; i++ ) if( rows[i]==p ) { found=1; break; }
    if( ! found ) { printf("***inconsistent input"); exit(4); }
    found=0;
    for( j=1; j<=nc; j++ ) if( cols[j]==t ) { found=1; break; }
    if( ! found ) { printf("***inconsistent input"); exit(4); }

    ec( C, i, j )+=v;

  } /* feof InputFile */
  in=mc;

  fclose( InputFile );

  mx=mc;
  nx=mc;
  X = calloc( mx*nx, sizeof(int) );
  if( X==NULL ) {printf("*** not enough memory for X\n");exit(1);};
  onex( X );

  /* clear columns */
  while( 1 )
  {
/*printf( "--- in=%d\n", in );
printf("-------------------------------------------------------------------
------------------\n");
for( i=1; i<=mc; ++i )
{
    for( j=1; j<=nc; ++j )
      printf( "%2d ", ec( C, i, j ) );
    printf("\n");
}*/

    Iz=malloc( in*sizeof(int) );
    if( Iz==NULL ) { printf("*** not enough memory for Iz\n");exit(1);};

    while( findzero( C, in, &nIz, Iz ) > 0 )
    {
/*printf("in=%d nIz=%d\n",in,nIz);*/
      erasezero( C, X, &in, nIz, Iz ) ;
    }

/*printf("erasezero in=%d nc=%d\n", in, nc);*/

    free( Iz );
    Ip=malloc( in*sizeof(int) );
    Im=malloc( in*sizeof(int) );
    if( Ip==NULL || Im==NULL ) {printf("*** not enough memory for
Ip,Im\n");exit(1);};

    if( ! findcol( C, in, &col, &nIp, Ip, &nIm, Im ) ) { break;}

/*printf("findcol col=%d nIp=%d nIm=%d\n", col,nIp,nIm);*/

    mc1=mx1=in-nIp-nIm + deltaSOL;
    C1 = calloc( mc1*nc, sizeof(int) );
    X1 = calloc( mx1*nx, sizeof(int) );
    if( C1==NULL || X1==NULL ) {printf("*** not enough memory for
C1,X1\n");exit(1);};

    /* copy raws for zero */
    f=1;
```

```
    for( i=1; i<=in; ++i )
    {
      if( ec( C, i, col ) == 0 )
      {
/*printf("copy row i=%d\n", i);*/
      for( k=1; k<=nc; ++k )
        { ec( C1, f, k ) = ec( C, i, k ); }
      for( k=1; k<=nx; ++k )
        { ex( X1, f, k ) = ex( X, i, k ); }
      f++;
      }
    }

/*printf("rows copied f=%d\n",f);*/

    /* combine raws for each plus-minus pair */
    for( i=0; i<nIp; ++i )
      for( j=0; j<nIm; ++j )
      {
      /* realloc */
      if( f > mx1 )
      {
        mc1+=deltaSOL;
        mx1+=deltaSOL;
          C1 = realloc( C1, mc1*nc * sizeof(int) );
          X1 = realloc( X1, mx1*nx * sizeof(int) );
          if( C1==NULL || X1==NULL ) {printf("*** not enough memory for
C1,X1\n");exit(1);};
      }

      ilp=Ip[i];
      ilm=Im[j];
/*printf("combine rows ilp=%d ilm=%d\n", ilp,ilm);*/
      cp=abs( ec( C, ilp, col ) );
      cm=abs( ec( C, ilm, col ) );
      d=nod( cp, cm );
      if( d > 1 )
      {
        cp=cp/d;
        cm=cm/d;
      }

      for( k=1; k<=nc; ++k )
        ec( C1, f, k ) = ec( C, ilp, k )*cm + ec( C, ilm, k )*cp;
      for( k=1; k<=nx; ++k )
        ex( X1, f, k ) = ex( X, ilp, k )*cm + ex( X, ilm, k )*cp;

      /* Filter New Solution */
      d1=nodv( lc( C1, f ), nc ); d2=nodv( lx( X1, f ), nx );
        if( (d1>1 || d1==0) && (d2>1) )
      {
        if( d1==0 ) d=d2; else d=nod(d1,d2);
        divv( lc( C1, f ), nc, d );
        divv( lx( X1, f ), nx, d );
      }
      FilterNewSolution( C1, X1, &f );

      f++;
      }

    in=--f;
    free(Ip); free(Im);

/*printf("rows combined f=%d\n",f); */
```

```
    /* divide by common divisor */
    /*for( i=1; i<=in; ++i )
    {
      d=nodv( lx( X1, i ), nx );
      if(d>1) { divv( lc( C1, i ), nc, d ); divv( lx( X1, i ), nx, d ); }
    }*/

/*printf("rows devided\n");*/

    free( C );
/*printf("free C\n");*/
    C = C1; mc=mc1;
/*printf("assigned C=C1\n");*/
    free( X );
/*printf("free X\n");*/
    X = X1; mx=mx1;

/*printf("matrices swapped\n");*/

    /*delzero( C, X, &in );

    deldbl( C, X, &in );

    latfilter( C, X, &in ); */
/*if(ret++==4)break;*/
  } /* while */

/*printf("write nc=%d\n", nc);*/

  if( col == 0 )
  {
    for( i=1; i<=in; ++i )
    {
      for( j=1; j<=nx; ++j )
      if( ex( X, i, j ) != 0 ) fprintf( OutputFile, "%d %d %d\n", i, rows[j],
ex( X, i, j ) );

    }
    ret=0;
  }
  else
  { fprintf( OutputFile, "# dust\n" ); ret=2;}

  free(C); free(X); free(rows); free(cols);

  fclose( OutputFile );

  return( ret );

} /* Toudic */

#ifdef __MAIN__
int main( int argc, char * argv[] )
{
  if (argc < 3 )
  {
    printf( "Routine for linear homogenious Diophantine system solution by
Toudic method\n");
    printf( "Usage: toy <inputfile> <outputfile>\n" );
    printf( "<inputfile>, <outputfile> : SPM format: i, j, e(i,j)\n" );
    printf( "(c) 2005 by Dmitry Zaitsev\n" );
    return 3;
  }
```

```
  Toudic( argv[1], argv[2] );
}
#endif

/* Adriana project: uti.h                                    */
/* Utility functions                                         */

#define min(a,b) ((a<b)?a:b)
#define max(a,b) ((a>b)?a:b)

int isempty( char * s );
/* checks is the string s empty */

void SwallowSpace( char * str, int *i );
/* swallow blanks */

int IsSpace( char * str, int i );
/* check blank */

int CopyFile( char * fnameFROM, char * fnameTO, char * attr );
/* copy (append) file */

int TransposeMatrix( char * InpFileName, char * OutFileName );
/* transposes matrix in SPM format */

/* end uti.h */

/* Adriana utility */

#include <stdio.h>
#include <string.h>

#define MAXSTRLEN 1024

int isempty( char * s )
{
  int i=0;
  int empty=0;

  while( ( s[i]==' ' || s[i]==0xa || s[i]==0xd || s[i]==0x9 ) && s[i]!='\0'
) i++;
  if( s[i]=='\0' ) empty=1;

  return(empty);

} /* isempty */

void SwallowSpace( char * str, int *i )
{

  while( ( str[(*i)]==' ' || str[(*i)]==0xa || str[(*i)]==0xd ||
str[(*i)]==0x9 ) && str[(*i)]!='\0'  ) (*i)++;

} /* SwallowSpace */

int IsSpace( char * str, int i )
{

  if( str[i]==' ' || str[i]==0xa || str[i]==0xd || str[i]==0x9 ||
str[i]=='\0' )
    return( 1 );
  else
    return( 0 );
```

```
} /* IsSpace */

int CopyFile( char * fnameFROM, char * fnameTO, char * attr )
{
  FILE * fileFROM, * fileTO;
  char str[ MAXSTRLEN+1 ];

  fileFROM = fopen( fnameFROM, "r" );
  if( fileFROM == NULL ) {printf( "*** error open file %s\n", fnameFROM
);exit(2);}
  fileTO = fopen( fnameTO, attr );
  if( fileTO == NULL ) {printf( "*** error open file %s\n", fnameTO
);exit(2);}

 /* read solutions */
 while( ! feof( fileFROM ) )
 {
   fgets( str, MAXSTRLEN, fileFROM );

   if( feof( fileFROM ) ) break;

   fputs( str, fileTO );

 }

 fclose( fileFROM );
 fclose( fileTO );

 return(0);

} /* CopyFile */

int TransposeMatrix( char * InpFileName, char * OutFileName )
{
 char str[ MAXSTRLEN+1 ];
 FILE * iFile, * oFile;
 int i, j, v;

 iFile = fopen( InpFileName, "r" );
 if( iFile == NULL ) {printf( "*** error open file %s\n", InpFileName
);exit(2);}
 oFile = fopen( OutFileName, "w" );
 if( oFile == NULL ) {printf( "*** error open file %s\n", OutFileName
);exit(2);}

 while( ! feof( iFile ) )
 {
   fgets( str, MAXSTRLEN, iFile);

   if( feof( iFile ) ) break;

   if( str[0]=='#' || isempty(str) ){ fputs( str, oFile ); continue; }

   sscanf( str, "%d %d %d", &i, &j, &v );
   fprintf( oFile, "%d %d %d\n", j, i, v );
 }

 fclose( iFile );
 fclose( oFile );

} /* TransposeMatrix */

/* Adriana project; writeinv.h                        */
```

```
/* write invariants (given in SPM format) into text file according to table
of names */

int WriteInv( char * TableFileName, char * InvFileName, char * OutFileName,
int verbose );

/* INPUT:  TableFileName - table of nodes names */
/*         InvFileName - basis invariants in SPM format */
/* OUTPUT: OutFileName - basis invariants in text form */
/* FUNCTION: transforms invariants from SPM format to text form according to
table of names */

/* end writeinv.h */

/* READNET - reads names' table and invariants in sparse matrix format writes
invariants */

#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>

#define MAXINPSTRLEN 1024
#define MAXFILENAME 256

#define mINIT 1024
#define namesINIT 16384

#define mDELTA 1024
#define namesDELTA 16384

static char str[ MAXINPSTRLEN + 1 ]; /* line buffer */

static int m, maxm; /* net size: trs, pls, arcs */
static int *pn; /* pls */

static char *names; /* all the names */
static int fnames, maxnames;

void GetName1( int *i, int *j )
{
 int state;

 if( str[(*i)]=='{' )
 {
   state=1;
   while( str[(*i)]!='\0' && state && str[(*i)]!=0xa && str[(*i)]!=0xd )
   {
    names[ (*j)++ ]=str[ (*i)++ ];
    if( str[(*i)-1]=='}' && state==1 ) state=0; else
      if( str[(*i)-1]=='\\' && state==2 ) state=1; else
        if( str[(*i)-1]=='\\' && state==1 ) state=2; else
        if( str[(*i)-1]=='}' && state==2 ) state=1; else
          if( state==2 ) state=1;
   }
   names[ (*j)++ ]='\0';
 }
 else
 {
   while( str[(*i)]!=' ' && str[(*i)]!='\0' && str[(*i)]!=0xa &&
str[(*i)]!=0xd && str[(*i)]!=0x9 && str[(*i)]!='*' )
    names[ (*j)++ ]=str[ (*i)++ ];
   names[ (*j)++ ]='\0';
 }
```

```
} /* GetName1 */

void ExpandNames1()
{
  char * newnames;

  if( fnames+MAXINPSTRLEN > maxnames )
  {
    maxnames+=namesDELTA;
    newnames = (char*) realloc( names, maxnames );
    if( newnames==NULL ) { printf( "*** not enough memory for names\n" );
exit(1); }
    else names=newnames;
  }

} /* ExpandNames1 */

void ExpandP1()
{
  int *newpn;

  if( m >= maxm - 2 )
  {
    maxm+=mDELTA;
    newpn = (int*) realloc( pn, maxm * sizeof(int) );
    if( newpn==NULL )
      { printf( "*** not enough memory for pn\n" ); exit(1); }
    else
      { pn=newpn; }
  }

} /* ExpandP1 */

void ReadTable( FILE * f )
{
  int i, len, p;

  while( ! feof( f ) )
  {
     fgets( str, MAXINPSTRLEN, f );
     if( feof( f) ) break;
     if( str[0]=='#' ) continue; /* comment line */

     len=strlen(str); i=0;
     SwallowSpace( str, &i );
     if( i==len ) continue; /*empty line */

     ExpandNames1();
     ExpandP1();

     p=atoi( str+i );
     while( ! IsSpace( str, i++ ));
     SwallowSpace( str, &i );
     pn[ p ]=fnames; m++;
     GetName1( &i, &fnames );

  }

}/* ReadTable */

int WriteInv( char * TableFileName, char * InvFileName, char * OutFileName,
int verbose )
{
```

```
 FILE * TableFile, * InvFile, * OutFile;
 int i, i0, j, v, len;

 /* open files */
 TableFile = fopen( TableFileName, "r" );
 if( TableFile == NULL ) {printf( "*** error open file %s\n", TableFileName
);exit(2);}
 InvFile = fopen( InvFileName, "r" );
 if( InvFile == NULL ) {printf( "*** error open file %s\n", InvFileName
);exit(2);}
 if( strcmp( OutFileName, "-" )==0 ) OutFile = stdout;
   else OutFile = fopen( OutFileName, "a" );
 if( OutFile == NULL ) {printf( "*** error open file %s\n", OutFileName
);exit(2);}

 /* init net size  */
 maxm=mINIT;
 maxnames=namesINIT;

 /* allocate arrays */
 pn = (int*) calloc( maxm, sizeof(int) ); m=1;

 names = (char*) calloc( maxnames, sizeof(char) ); fnames=0;

 if(
     pn==NULL ||
     names==NULL )
   { printf( "*** not enough memory for pn,names\n" ); return(1); }

 ReadTable( TableFile );
 fclose( TableFile );

 i0=0;
 while( ! feof( InvFile ) )
 {
   fgets( str, MAXINPSTRLEN, InvFile );
   if( feof( InvFile ) ) break;
   if( str[0]=='#' ) continue; /* comment line */

   len=strlen(str); i=0;
   SwallowSpace( str, &i );
   if( i==len ) continue; /*empty line */

   sscanf( str, "%d %d %d", &i, &j, &v );

   if( i != i0 ) { i0=i; if(verbose) fprintf( OutFile, "\n" ); }

   if(verbose) fprintf( OutFile, "%s", names+pn[j] );
   if( v > 1 ) {if(verbose) fprintf( OutFile, "*%d", v );}
   if(verbose) fprintf( OutFile, " " );

 }/* while */
 if(!verbose) fprintf( OutFile, "\n%d semiflow(s)", i0 );
 fprintf( OutFile, "\n\n" );

 fclose( InvFile );

 if( OutFile != stdout ) fclose( OutFile );

 free( pn );

 free( names );

}/* WriteInv */
```

```
#ifdef __MAIN__
int main( int argc, char *argv[] )
{
  if( argc>1 ) { TableFileName=argv[1]; } else exit( 2 );
  if( argc>2 ) { InvFileName=argv[2]; }
  if( argc>3 ) { strcpy(OutFileName,argv[3]); } else { sprintf( OutFileName,
"%s.txt", InvFileName ); }
  WriteInv( TableFileName, InvFileName, OutFileName );
}
#endif

# Adriana makefile

PROJ=Adriana
CC=gcc
CFLAGS=
LDFLAGS=

SRC = deborah.c toy2.c cor.c coy.c fui.c mui.c smmul2.c smb2.c fic.c uti.c
ana.c Adriana.c readnet.c writeinv.c chi.c

.c.o:
      @rm -f $@
      $(CC) $(CFLAGS) -c $*.c

OBJ = $(SRC:.c=.o)

CLEANFILES = $(PROG) $(OBJ)

all: $(PROJ)

$(PROJ): $(OBJ)
      @rm -f $@
      $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $(OBJ)

clean:
      rm -f $(CLEANFILES)

# Dependencies (.c and .h files)

deborah.o: deborah.c
toy2.o: toy2.c
cor.o: cor.c
coy.o: coy.c
fui.o: fui.c
mui.o: mui.c
smmul2.o: smmul2.c
smb2.o: smb2.c
uti.o: uti.c
ana.o: ana.c
readnet.o: readnet.c
writeinv.o: writeinv.c
chi.o: chi.c

# end Adriana Makefile
```

Приложение Е. Документы о внедрении

УТВЕРЖДАЮ
Председатель Правления
ОАО Укртелеком
_____Дзекон Г.Б.
«21» 03 2006г.

АКТ
о внедрении результатов научных работ Д.А.Зайцева в ОАО Укртелеком

Диссертационная работа Зайцева Д.А. посвящена разработке модели сетей с коммутацией меток MPLS.

Библиотека стандартных компонентов модели, включающих модели IP-маршрутизатора, LSR-маршрутизатора, LSR/LER-маршрутизатора, терминальной сети позволяют выполнять сборку модели по структурной схеме сети с минимальными трудозатратами.

Параметры реального оборудования и трафика вводятся как атрибуты элементов модели и вычисляются с помощью представленной соискателем методики масштабирования времен.

Измерение функциональных характеристик модели с помощью специальных измерительных фрагментов позволяют представить результаты моделирования в удобной форме, не требующей дополнительной обработки.

Отличие функциональных характеристик, таких как время отклика и трафик, полученных с помощью модели, от результатов измерения реальных сетей составляет не более пяти процентов.

Указанная модель была использована при проектировании магистральных транспортных сетей Укртелеком.

Применение предложенных Д.А.Зайцевым методов и моделей позволило сократить сроки проектирования магистральных сетей и повысить достоверность предварительных оценок пропускной способности и качества обслуживания.

Экономический эффект от использования работ соискателя составил около 50 тыс. грн.

Директор департамента информационных
и перспективных технологий , к.т.н., с.н.с. О.В.Копейка

«Утверждаю»
Генеральный директор
ООО «НПО Сигма-Т»

_____ В.Я. Смола

«25» _____ 2006 г.

АКТ
о внедрении результатов научных работ Д.А.Зайцева
в ООО «НПО Сигма-Т»

Научно-технический совет предприятия в составе: председатель - технический директор Тимченко С.И., членов совета – заместитель технического директора по развитию Хребтов Ю.В, главный инженер проектов Орлова И.А., конструктор Гусев В.Я., составили настоящий акт в том, что:

1. Разработанное Д.А.Зайцевым программное обеспечение Deborah и Adriana используются в процессе проектирования нового поколения цифровых АТС (ЦАТС) для анализа свойств моделей Петри протоколов работы ЦАТС.

2. Модели Петри и указанное программное обеспечение применены также для выявления возможных негативных последствий Наложения Возможностей (Feature Interaction) при добавлении новых видов сервисов в процессе модернизации ЦАТС.

Применение предложенных Д.А.Зайцевым композиционных методов анализа сетей Петри позволило существенно ускорить процессы анализа крупномасштабных моделей телефонных протоколов и сократить сроки разработки и модернизации ЦАТС СТМ-256х.

Экономический эффект от использования работ соискателя составил около 4 000 грн. на единицу изделия.

Председатель **С.И. Тимченко**

Члены НТС

Ю.В. Хребтов

И.А. Орлова

В.Я. Гусев

УТВЕРЖДАЮ
Проректор ОНАС
_____ Н.В.Захарченко
«__» _____ 2006 г.

АКТ
о внедрении результатов научных работ Д.А.Зайцева
в учебный процесс ОНАС им. А.С. Попова

Настоящий акт составлен комиссией в составе: председатель – нач. учебного отдела В.Д.Кузнецов, членов комиссии – декан фак. ТКС А.В.Онацкий, зав. каф. Сетей связи Л.А.Никитюк, доц. каф. Сетей связи Л.В.Бубенцова.

Комиссия рассмотрела материалы по внедрению результатов научных работ Д.А.Зайцева в учебный процесс. Основные выводы комиссии следующие:

1. Основы теории функциональных сетей Петри и модели телекоммуникационных систем в форме раскрашенных сетей Петри изучаются в курсах «Математическое моделирование информационных/телекоммуникационных систем» подготовки магистров.

2. Подготовлено и издано Учебное пособие «Математические модели дискретных систем», подготовлены Методические указания к лабораторным работам и практическим занятиям «Сети Петри и моделирование систем», в которых представлены упрощённые модели Петри коммутируемых сетей, телекоммуникационных протоколов, систем управления.

3. Изучение моделей Петри телекоммуникационных систем позволило повысить качество подготовки магистров, так как управляемое моделью проектирование телекоммуникационных и информационных систем является перспективным современным направлением, практически применяемым в десятках проектах ведущими производителями телекоммуникационного оборудования, например, таких как компания Nokia.

4. Использование в учебном процессе специфических методов, моделей и программного обеспечения Deborah и Adriana, предложенных Д.А.Зайцевым, позволило приблизить подготовку магистров к переднему краю научных исследований.

5. Совместно с магистрами и специалистами 2005 года выпуска по материалам дипломных работ подготовлены публикации в изданиях перечня ВАК Украины.

Председатель комиссии:

В.Д.Кузнецов

Члены комиссии:

А.В.Онацкий

Л.А.Никитюк

Л.В.Бубенцова