ПОДДЕРЖКА СУПЕРВЫЧИСЛЕНИЙ И ИНТЕРНЕТ-ОРИЕНТИРОВАННЫЕ ТЕХНОЛОГИИ

Серия

“КОНСТРУИРОВАНИЕ И ОПТИМИЗАЦИЯ ПРОГРАММ”

Под редакцией
доктора физ.-мат. наук, профессора, чл.-корр. РАЕН
В. Н. Касьянова

Выпуски серии:

1. Смешанные вычисления и преобразование программ (1991)
2. Конструирование и оптимизация программ (1993)
3. Интеллектуализация и качество программного обеспечения (1994)
4. Проблемы конструирования эффективных и надежных программ (1995)
5. Оптимизирующая трансляция и конструирование программ (1997)
6. Проблемы систем информатики и программирования (1999)
7. ***Поддержка супервычислений и Интернет-ориентированные технологии***

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

ПОДДЕРЖКА СУПЕРВЫЧИСЛЕНИЙ И ИНТЕРНЕТ-ОРИЕНТИРОВАННЫЕ ТЕХНОЛОГИИ

Под редакцией
проф. Виктора Николаевича Касьянова

Новосибирск 2001

УДК 519.68; 681.3.06
ББК З 22.183.49+ З 22.174.2

Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск: Ин-т систем информатики им. А. П. Ершова СО РАН, 2001. — 264 с.

Является седьмым в серии сборников, издаваемых Институтом систем информатики СО РАН по проблемам конструирования и оптимизации программ. Посвящен решению актуальных задач разработки методов и средств, повышающих эффективность использования супервычислителей и телекоммуникационных сетей.

Сборник представляет интерес для системных программистов, а также студентов и аспирантов, специализирующихся в области системного и теоретического программирования.

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

SUPERCOMPUTING SUPPORT AND INTERNET-ORIENTED TECHNOLOGIES

Edited by
prof. V. N. Kasyanov

Novosibirsk 2001

This volume is the seventh one among volumes published in A. P. Ershov Institute of Informatics Systems on problems of programs construction and optimization. The decision of actual problems of methods and tools elaboration increasing efficiency of supercomputers and telecommunication nets is described.

The volume is of interest for system programmers, students and postgratuate students working in the field of system and theoretical programming.

ПРЕДИСЛОВИЕ РЕДАКТОРА

Седьмой выпуск серии "Конструирование и оптимизация программ" посвящен решению актуальных задач разработки методов и средств, повышающих эффективность использования супервычислителей и телекоммуникационных сетей. Продолжая уже сложившиеся традиции, данный выпуск, как и предыдущие, базируется на результатах исследований, выполненных в лаборатории по конструированию и оптимизации программ ИСИ СО РАН совместно с НГУ при финансовой поддержке РФФИ и Минобразования.

Выпуск открывает статья В.Н. Касьянова, посвященная работе 16-го Всемирного компьютерного конгресса ИФИП (WCC-2000), который проходил в августе прошлого года в г. Пекине под лозунгом "Обработка информации. За рубежом 2000 года". Большое внимание в статье уделяется конференциям «Программное обеспечение: теория и практика» (ICS-2000) и «Использование в образовании информационных и коммуникационных технологий» (ICEUT-2000) — тем двум из восьми конференций конгресса, на которых автор статьи выступал с докладами.

Остальные статьи выпуска отражают результаты выполнения исследований, которые ведутся сотрудниками лаборатории "Конструирование и оптимизация программ" в форме трех плановых научно-исследовательских тем ИСИ СО РАН на 1999–2001 гг., и поэтому условно могут быть разбиты на три группы.

Первая группа состоит из 9 статей. Они посвящены проекту "Исследование методов распараллеливания алгоритмов и программ, разработка макета системы функционального программирования SFP", по которому получены следующие основные результаты.

Подготовлен аналитический обзор[1] задачи анализа зависимостей по данным, играющей важную роль в автоматическом распараллеливании последовательных программах. В общем случае задача является архитектурно независимой проблемой и предваряет фазы оптимизации и реструктуризации в системах распараллеливания. Основное внимание в обзоре уделялось основным понятиям (типы зависимостей, характеристики зависимостей, теоретико-графовое представление, конус зависимости и т.д.), существую-

[1] Обзор готовился В.А. Евстигнеевым по заказу редакции для данного сборника, но окончательный объем обзора вынудил передать его для опубликования в сборник «Системная информатика» (Новосибирск, Наука, 2000, Выпуск N 7). Тем самым, он повторил судьбу и других обзоров: В.К. Сабельфельд «Рекурсивные схемы: преобразования и отношения эквивалентности» (1995, Выпуск N 4), В.А. Евстигнеев «VLIW-машины: развитие архитектуры и принципов построения программного обеспечения» (1995, Выпуск N 4) и П.Г. Емельянов «Абстрактная интерпретация императивных программ» (1998, Выпуск N 6).

щих подходов, а также новым тестам с повышенной точностью, тестам для работы с символьными вычислениями, тестам с использованием теории многогранников ограничений и т.д.

Исследовались особенности компиляции и генерации кода для мультипроцессорных систем с неоднородным доступом к памяти (так называемые NUMA-машины), в том числе вопросы реструктуризации циклов, порогового планирования графа заданий, а также унифицирующих преобразований данных и управления.

Разработаны язык функционального программирования Sisal 3.0 и макет системы функционального программирования SFP. Система включает следующие компоненты: интерфейс, язык Sisal 3.0, отладчик, front-end транслятор, блоки промежуточных представлений IR1, IR2 и IR3, блоки анализа, преобразования и визуализации IR1-, IR2- и IR3-программ, конверторы промежуточных представлений, back-end трансляторы.

Исследован вопрос о распараллеливании трехмерного варианта PIC-метода применительно к задаче о взаимодействии потоков разреженной плазмы в самосогласованных электромагнитных полях. Изучались возможности эффективного распараллеливания метода "МЕДУЗА". Рассмотрен вопрос о распараллеливании вычислительной части алгоритма, получены новые оценки для коэффициента ускорения.

Разработаны язык и система GRAMAL, предоставляющие удобные средства для описания графовых моделей, конструирования их графических представлений, тестирования и отладки методов работы с графами и генерации программного кода для последующего применения.

В рамках информационной системы ТРАНСФОРМ по оптимизирующим и распараллеливающим преобразованиям программ создан программный комплекс, предоставляющий удаленному пользователю, связавшемуся с WWW-сервером ИС ТРАНСФОРМ, удобный интерфейс для регистрации и аутентификации пользователей, а также для поиска, ввода и модификации информации в базе данных.

Вторая группа состоит из 6 статей и связана с темой "Исследование теоретико-графовых моделей систем и процессов в задачах визуальной обработки и распределенных вычислений", исследования по которой характеризуют следующие основные результаты.

Выполнен цикл работ по анализу и систематизации теоретико-графовых алгоритмов и методов в информатике и программировании. Впервые подготовлена серия из 4 книг по алгоритмам на графах для программистов, охватывающая базовые алгоритмы теории графов и основные их приложения. В основу "энциклопедии" положено разбиение алгоритмов по классам графов (деревья, дэги, сводимые графы) с выделением в отдельный том толкового

словаря и использованием высокоуровневого представления алгоритмов. Начато издание "энциклопедии" на английском языке.

Проведено исследование теоретико-графовых моделей систем и процессов в задачах распределенных вычислений, показано совпадение классов полиномиально разрешимых задач для различных существующих моделей локальных вычислений: систем переписывания графов с запрещенными контекстами, локальных алгоритмов Журавлева и сетей конечных автоматов.

Получены верхние оценки для индекса асимметрии для 2-связных внешнепланарных графов и максимальных внешнепланарных графов, найдены новые оценки для хроматического числа (k,l)-раскрасок инциденторов ориентированных мультиграфов, и описаны свойства скрюченных 3-связных плоских графов с малым числом вершин и граней, в частности хроматических. Найдено значение реберного предписанного хроматического числа, и разработан метод вычисления кликоматического числа и плотности кубоподобного графа.

Проведено исследование теоретико-графовых моделей и методов в задачах визуальной обработки, подготовлен обзор систем рисования графов[2]. Разработаны методы визуальной обработки графов и подготовки графовых иллюстраций.

Создана рабочая версия системы HIGRES, ориентированная на поддержку конструирования, визуализации и изучения различных объектов и явлений в рамках их иерархических графовых моделей.

Разработана экспериментальная версия универсального и простого в использовании редактора атрибутированных графов VEGRAS, ориентированного на поддержку подготовки качественных штриховых иллюстраций в рамках среды Windows 95/98/NT.

Последняя группа — это 2 статьи по теме "Исследование и разработка программно-методического обеспечения систем обучения и принятия решений", в рамках которой получены следующие основные результаты.

Выполнен цикл работ по созданию программно-методическое обеспечения обучению доказательному и функциональному программированию.

В плане развития среды программирования на Лиспе выполнена русификация СП CLisp, создан конвертор исходных текстов в русифицированный вариант, разработаны библиотеки на Лиспе для поддержки CGI-программирования и SQL-запросов к СУБД Postgres.

[2] Обзор готовился для данного сборника, но из-за его большого объема он был опубликован в виде отдельного издания: Лисицын И.А.” Системы визуализации и редактирования графовых объектов”, Новосибирск, 2000, препринт ИСИ СО РАН N 76.

Исследовались вопросы обучения математиков информатике и программированию, подготовлен задачник по начальному обучению доказательному программированию на языке Паскаль[3].

Осуществлена реализация сопоставления с образцом в Лиспе с локальным связыванием переменных, входящих в образец, со значениями успешного сопоставления. Семантика сопоставления с образцом сконструирована и реализована на материале функциональных языков Refal и Haskell.

Созданы программы для синтаксического разбора HTML-текстов и конвертирования HTML в Latex, разработана технология перевода текстовых документов в гипертекст (HTML) со встроенными формулами, а также в форматы Latex, PostScript, PDF.

Продолжалась работа по системе "Кентавр" для надежного восстановления и переключения вариантов конфигурации операционных систем (ядро основано на специально скомплектованном варианте ОС Linux, оболочка — CLisp-программа). Создан минимальный комплект (5-15Мб) ОС Linux для работы на CLisp'е в среде Linux без специальной установки Linux.

На базе системы HIGRES выполнена экспериментальная реализация алгоритмов синтаксического анализа, ориентированная на их использование при обучении методам трансляции.

Проведено исследование систем принятия решений на основе технологии ТРИЗ, главным образом, в плане формализации понятий, используемых в методике ТРИЗ. За основу был взят подход Р. Акоффа, Ф. Эмери, расширенный посредством привлечения теоретико-множественных, логических и топологических понятий. Предложена структурная схема программной системы, которая будет работать на основе методики ТРИЗ. Она будет похожа на базу данных и ориентирована на использование в процессе ведения переговоров. Исследована методика ТРИЗ на примере процесса генерации и отгадывания загадок. Предложена формальная модель в терминах математической логики, и на ее основе начата программная реализация.

Выполнен цикл работ по развитию гипертекстовой информационной системы СИМИКС (SIMICS) в направлении ее использования для хранения материала, связанного со становлением информатики в Новосибирском научном центре.

Февраль 2001 г. Проф. В. Н. Касьянов

[3] Задачник объемом в 30 п.л. был подготовлен В.Н. Касьяновым и включен в план изданий СО РАН на 2001 год; его предварительное издание было осуществлено в НГУ в виде трех учебных пособий: «Курс программирования на Паскале в заданиях и упражнениях, Часть 1» (1999), «Курс программирования на Паскале в заданиях и упражнениях, Часть 2» (1999), «Практикум по программированию» (2001).

В. Н. Касьянов

О РАБОТЕ 16-ГО ВСЕМИРНОГО КОМПЬЮТЕРНОГО КОНГРЕССА ИФИП[1]

ВВЕДЕНИЕ

С 21 по 25 августа 2000 г. в Пекине прошел очередной, 16-й по счету, Всемирный компьютерный конгресс. Организатором конгрессов является IFIP — Международная федерация по обработке информации (ИФИП), которая в этом году отмечала свой 40-летний юбилей. Предыдущий 15-й конгресс ИФИП проходил в Вене (Австрия) и Будапеште (Венгрия), а следующий должен состояться в Монреале (Канада) в 2002 г. Компьютерные конгрессы, проводимые ИФИП, являются главным мировым научным форумом в области информационных технологий, на котором рассматриваются основные проблемы и наиболее важные новые результаты основных направлений современной информатики. Неслучайно их называют «олимпийскими играми» по информационным технологиям.

16-й конгресс WCC-2000 проходил под лозунгом "Обработка информации. За рубежом 2000 года", и в его работе приняло участие более 2 тысяч ученых и специалистов из 70 стран мира. Впервые конгресс проводился в развивающейся стране — Китае, что было не случайным, а подтвердило особый статус Китая в области развития информационных технологий. Главным организатором конгресса WCC-2000 с китайской стороны являлся Китайский институт электроники (CIE, см. http://www.cie-china.org) — неправительственная академическая и инженерная организация Китая с большим числом профессиональных и региональных отделений по всей стране. Основан CIE в 1956 г. и получил независимый академический статус в 1962 г. В 1979 г. он стал членом ИФИП, а в начале 80-х гг. установил отношения со всеми 15 основными международными организациями в области информатики. Китайский институт электроники издает 12 журналов и периодических изданий, в том числе журнал "Acta Electronic Sinica" и популярный еженедельник "PC Week".

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Форма проведения компьютерных конгрессов ИФИП со временем подверглась существенным изменениям. Нынешний конгресс состоял из восьми независимых конференций, каждая из которых была посвящена одному из наиболее важных направлений в области информационных технологий. Конференции, составляющие конгресс, работали параллельно и на одной территории — в так называемой «деревне азиатских игр», расположенной в северной части Пекина. Это позволило организаторам конгресса провести общие мероприятия конгресса, а также дать возможность каждому приехавшему на конгресс участвовать в любом интересном ему заседании любой конференции и встретиться с любым его участником.

Данная статья содержит краткий обзор работы 16-го Всемирного компьютерного конгресса WCC-2000 и дает некоторые сведения об его организаторах. В разд. 1 приводится краткая информация о ИФИП, ее структуре, целях и задачах. Разд. 2 содержит общее описание работы конгресса и его первого дня заседаний. Разд. 3 и 4 посвящены конференциям «Программное обеспечение: теория и практика» (ICS-2000) и «Использование в образовании информационных и коммуникационных технологий» (ICEUT-2000) — двум из восьми конференций конгресса, на которых автор статьи выступал с докладами.

1. МЕЖДУНАРОДНАЯ ФЕДЕРАЦИЯ ПО ОБРАБОТКЕ ИНФОРМАЦИИ (IFIP)

Международная федерация по обработке информации (ИФИП, см. http://www.ifip.or.at) — неправительственная некоммерческая рамочная организация национальных обществ, работающих в области информационной обработки, создана в 1960 г. под эгидой ЮНЕСКО как результат первого всемирного компьютерного конгресса, который состоялся в Париже в 1959 г.

Создание ИФИП отвечало насущным проблемам времени. В 60-е гг. в мире начался существенный рост компьютерной индустрии и стала быстро расширяться сфера применения ее продуктов. Таким образом, с началом работы ИФИП информационные технологии все в большей степени становятся эффективным инструментом, влияющим на жизнь людей, причем в разных направлениях: в науке и инженерии, в коммерции и индустрии, в образовании и управлении, а также в сферах досуга.

Основными целями ИФИП являются:

- способствование международной кооперации,

- стимулирование исследований и разработок,
- поддержка образования,
- распространение информации.

Своей миссией ИФИП считает право быть лидирующей истинно международной неполитической организацией, которая поощряет и поддерживает разработку, распространение и применение информационных технологий на пользу всему человечеству.

Членами ИФИП являются более 60 общественных организаций и академий наук, представляющих страны различных регионов мира, в том числе Россию, из которых 45 являются полными членами, 4 — членами-корреспондентами, 1 — ассоциативным членом и 11 — объединенными членами.

Среди индивидуальных членов ИФИП 18.1% специалистов из индустрии, 75% — из университетов, 3.8% занимаются управлением. Женщины в ИФИП составляют 12.4%, а молодежь (до 40 лет) — 19.4%.

ИФИП поддерживает дружественные связи со многими неправительственными организациями, первой из которых является ЮНЕСКО, и тесно взаимодействует с такими международными федерациями, как IFAC, IMACS, IFORS и IMEKO.

Главным событием ИФИП является Всемирный компьютерный конгресс, который в настоящее время проводится раз в два года. Помимо конгресса ИФИП поддерживает главные международные конференции по информационным технологиям, которые организуются техническими комитетами ИФИП и их рабочими группами. В настоящее время ИФИП включает 85 рабочих групп и состоит из 13 технических комитетов:

- основания информатики;
- программное обеспечение: теория и практика;
- образование;
- применения компьютеров в технологии;
- коммуникационные системы;
- системы моделирования и оптимизации;
- информационные системы;
- отношение между компьютерами и обществом;
- технология компьютерных систем;
- секретность и защита систем информационной обработки;
- искусственный интеллект;
- человеко-машинное взаимодействие.

2. ВСЕМИРНЫЙ КОМПЬЮТЕРНЫЙ КОНГРЕСС 2000 (WCC-2000)

16-й Всемирный компьютерный конгресс ИФИП проходил в виде следующих 8 отдельных конференций:

1. Международная конференция по коммуникационным технологиям (ICCT-2000).
2. Международная конференция по автоматизации проектирования чипов (ICDA-2000).
3. Международная конференция по использованию информационных и коммуникационных технологий в образовании (ICEUT-2000).
4. Международная конференция по программному обеспечению: теории и практике (ICS-2000).
5. Международная конференция по обработке сигналов (ICSP-2000).
6. Международная конференция по интеллектуальной обработке информации (IIP-2000).
7. Международная конференция по информационной технологии управления бизнесом (ITBM-2000).
8. Международная конференция по защите информации (SET-2000).

Программа каждой из конференций была сформирована своим международным программным комитетом, составленным из ведущих мировых специалистов в соответствующих областях. Каждый доклад, включенный в программу любой из конференций, прошел тщательный отбор и рассматривался не менее чем тремя рецензентами.

Конференциям предшествовал общий день заседаний, включающий церемонию открытия, на которой присутствовал и выступил с приветствием Председатель Госсовета КНР Цзян Цзэмин, а также ряд пленарных докладов.

Помимо приветствий руководителей ИФИП и организаторов конгресса на церемонии открытия состоялось вручение премии имени Исаака Ауербаха, основателя ИФИП. Премия была вручена проф. Асберну Ролстадесу (Норвегия), он стал четвертым обладателем этой престижной премии ИФИП.

В своем выступлении на церемонии открытия конгресса Цзян Цзэмин отметил, что мир охвачен технологической революцией, для которой информационные и генные технологии будут флагманскими кораблями. Он также отметил, что физические ресурсы в мире ограничены, а потому информационные ресурсы будут играть все более важную роль. Цзян Цзэмин считает, что сектор промышленности должен интегрироваться с сектором

программного обеспечения и необходимо соединение традиционной индустрии и информационных сетей. Он напомнил о все расширяющемся технологическом разрыве между развитыми и развивающимися странами и призвал развитые страны помочь развивающимся нациям преодолеть этот разрыв. Цзян Цзэмин предупредил, что бесконтрольные действия в Интернете хакеров приводят к нарушению неприкосновенности частной жизни, потерям информации и секретности, и призвал международную общественность к принятию международного соглашения по Интернету, которое позволило бы повысить уровень обслуживания пользователей Интернета и обеспечить гарантии их интересов.

После церемонии открытия состоялись пленарные доклады. *Lu Xinkui* (Министерство информационной индустрии, Китай) в своем докладе призвал объединить усилия, чтобы создать великолепное будущее для мировой информационной индустрии. *Mr. Christopher, B. Galvin* (Моторола) выступили с докладом «Радио, интернет, широкополосная связь и волнующее будущее информационных технологий». *Tony Hoar* (Майкрософт) посвятил свой доклад проблемам наследия в программном обеспечении. *Richard Tzar Kai Li* (Пасифик Сенчуре Групп) в своем докладе говорил о широкополосной связи в следующем тысячелетии. *Yang Yuanqing* (Легенд Холдингс Лимитед) рассказал о перспективах развития информационных технологий в Китае. *Henry W. K. Chow* (ИВМ) очертил будущее электронной коммерции. *Makoto Nagao* (Университет Киото, Япония) описал информационное будущее следующего десятилетия. *Daniel A. Reed* (Национальный Союз по Вычислительной Науке, США) выступил с докладом «Научные вычисления 21-го века: распределенные, совместные и междисциплинарные». *Yau Tsang, Ka Lai, Thomas Tang, H. M. Hui, Mak Shiv Tong, Sing Wang* (Гонконг) выступили с 5 содокладами о развитии и перспективе информационных технологий в Гонконге. *Sun Pishu* (Ланчао Электроник, Китай) представил доклад «Высопроизводительный сервер в интернетовские времена». *Wang Juntao* (Пекинская служба по электронной коммерции и сетям, Китай) говорил о возможностях и проблемах своей организации. *Zhu Jianqiu* (Группа по базисным наукам и технологии, Китай) в своем докладе рассказал о возможностях китайского предпринимательства в области информационных технологий в новые экономические времена.

Помимо указанного общего дня заседаний, параллельных заседаний 8 конференций, составляющих Конгресс, и общей церемонии закрытия программа Конгресса предусматривала целый ряд мероприятий, среди них:

- Генеральная ассамблея ИФИП, на которой, в частности, прошло избрание нового президента ИФИП, и проф. *Robert Aiken* стал новым президентом (ранее этот пост занимал проф. *Peter Bollerslev)*;
- лектории по основанным на агентах интеллектуальным информационным системам (*San Murugesan*, Австрия) и по состоянию и проблемам мультиагентной координационной технологии (*Mark Klein*, США);
- Третье рабочее совещание по информатике Ассоциации восточно-азиатских исследовательских университетов (см. http://www.adm.u-tokio.ac.jp/aearu);
- так называемый «День пионеров», на котором говорилось об истории и основателях информационных технологий, главным образом в Китае;
- Конгресс молодежи (см. http://www.futureforum.org);
- выставки и представления различных издательств, организаций и фирм;
- множество мероприятий (совещаний, рабочих групп, круглых столов и т.п.), организованных каждой из 8 конференций.

3. МЕЖДУНАРОДНАЯ КОНФЕРЕНЦИЯ ПО ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ: ТЕОРИИ И ПРАКТИКЕ (ICS-2000)

Программа конференции включала два приглашенных доклада: «Архитектура программного обеспечение и технология программирования» (*Dewayne Perry*, США) и «Интегрированный подход к совместному проектированию оборудования и программ» (*He Jifend*, Макао), 116 секционных докладов, которые были отобраны Международным программным комитетом (50 членов из 24 стран) из 340 заявок, поступивших из 34 стран, а также 46 стендовых докладов.

Секционные доклады были прочитаны на следующих 14 секционных заседаниях:

- технология требований к программному обеспечению;
- языки программирования;
- методология разработки программного обеспечения;
- архитектура и переиспользование программного обеспечения;
- процесс программного обеспечения;

- оценка программного обеспечения;
- формальные методы и спецификации;
- валидация и верификация программного обеспечения;
- распределенные системы и системы реального времени;
- сетевые вычисления и среды;
- человеко-машинное взаимодействие;
- параллельные вычисления;
- инженерия данных;
- массивные информационные системы в Интернете.

Помимо докладов программа конференции включала две панельные дискуссии "Совместное развитие теории и практики: настоящее и будущее" и "Электронная коммерция: проблемы и возможности для технологии программирования", а также два рабочих совещания — "Международный симпозиум по индустрии программирования в развивающихся странах (ISSIDeC)" и "Международное совещание по технологии программирования на пост-PC-стадии (IWSEPPA)".

На секции "Инженерия требований к программному обеспечению" были представлены 5 докладов. *Didar Zowghi, Ray Offen, Nurmuliani* (Австралия) доложили о предварительных результатах эмпирического исследования непостоянства требований и его влияний на различные аспекты жизненного цикла программного обеспечения. *Annette L. Steenkamp* (США) рассказал об использовании схемы ссылок для исследования программных систем. *Luisa Mich , Roberto Garigliano* (Англия) предложили схему, в которой определены меры неоднозначности — показатели структурной и семантической неоднозначности — в инженерии требований. *Mao Xin-Jun, Wang Huai-Ming, Chen Yue-Xin, Liu Feng-Qi* (Китай) предложили формальную схему для исследования поведения агентов в динамических и недетерминированных мультиагентных системах. *Mou Hu* (Канада) предложил новый метод построения тестов для тестирования программного обеспечения на основе требований.

На секции "Языки программирования" состоялось 7 докладов. *Takao Shimomura, Xinjun Zhang* (Япония) предложили эффективный метод слайсинга для библиотечных функций и описали его применение для программ, вызывающих библиотечные функции в виде объектного кода. *Wang Mingwen, Sun Yongqiang* (Китай) изложили эффективный подход к выполнению аккуратного анализа периода связывания для императивных языков. *Litong Song, Yoshihiko Futamura, Robert Glück, Zhenjiang Hu* (Япония) предложили методику оптимизации циклов на основе квазиинвариантов. *Harri*

Hakonen, Ville Leppanen, Tapio Salakoski (Финляндия) описали подход к интеграции объектов при разрешении синонимии (совмещения имен) для доступа к ним. *Lukman Efendy, Hashimoto Masaaki, Hirota Toyohiko* (Япония) привели формализацию метода обнаружения структурных расхождений при переходе от программных спецификаций к процедурной программе. *Yoshihiro Adachi, Yuichi Nakajima, Suguru Kobayashi* (Япония) описали контекстно-зависимую графовую грамматику с управляемым встраиванием соседей и ее применение для визуальных языков. *Qi Xuan, Wang Ting, Chen Huowang* (Китай) описали алгоритм для автоматического тегирования семантических классов, необходимого при реализации машинного перевода с китайского языка на английский.

На секции "Методология разработки программного обеспечения" были представлены 11 докладов. *Gianpaolo Cugola, Carlo Ghezzi, Mattia Monga, Gian Pietro Picco* (Италия) предложили многоаспектный язык Malaj для языка Джава, ориентированный на элиминацию расхождений между аспектно-ориентированным и объектно-ориентированным программированием. *Magda Huisman, Juhani Iivari* (Финляндия) рассказали о взаимосвязи между осознанной зрелостью отделений по информационным технологиям и развертыванием в них методологий разработки систем. *Tomokazu Arita, Kiyonobu Tomiyama, Takeo Yaku* (Япония) описали синтаксическую обработку диаграмм с помощью графовых грамматик. *George Spanoudakis, Kuriakos Kasis* (Англия) предложили естественную схему для диагностики явных несогласованностей в моделях, представленных на унифицированном языке моделирования UML. *John Trimble* (США) описал исследования, нацеленные на предоставление подходов к накоплению знаний, которые бы снижали риск, возникающий при разработке софтвера в ориентированных на исследования средах. *Pierre Bourque, Robert Dupuis, Alain Abran,James W. Moore, Leonard Tripp* (Канада) описали совместный проект IEEE и ACM по разработке путеводителя по знаниям в области программной инженерии и просуммировали текущие результаты проекта. *Serge Oligny, Pierre Bourque, Alain Abran, Bertrand Fournier* (Канада) представили подтверждающий анализ эмпирических моделей для предсказания длительности проектов по программной инженерии на основе проектных усилий, связанных с более современным и более большим образцом. *Victor N. Kasyanov, Ivan A. Lisitsyn* (Россия) рассказали об иерархических графовых моделях и визуальной обработке. *Gao Xiaolei, Miao Huaikou, Chen Yihai* (Китай) провели сравнительный анализ структурной методологии, объектно-ориентированной методологии и формальных методов, на основе которого

предложили язык и среду SOZL для поддержки разработки программного обеспечения. *J. L. Sierra, B. Fernandez-Manjon, A. Fernandez-Valmayor, A. Navarro* (Испания) предложили так называемый DTC-подход, интегрирующий языки разметки, преобразования документов и компоненты программного обеспечения при разработке приложений. *Dong Yunmei, Li Kaide, Chen Haiming, Hu Yongqian, Zhang Ruiling* (Китай) рассказали о разработке и реализации системы SAQ для построения и верификации формальных спецификаций.

На секции "Архитектура и переиспользование программного обеспечения" состоялось 9 докладов. *Robert M. Balzer, Neil M. Goldman* (США) предложили архитектуру для генерации анализаторов, редакторов и их графических интерфейсов в генераторе окружений проектирования. *Hong Mei, Jichuan Chang, Fuqing Yang* (Китай) рассказали о подходе к композиции компонент программного обеспечения на архитектурном уровне. *Maria Smolarova, Pavol Navrat* (Словакия) рассмотрели два возможных подхода к переиспользованию на основе образцов проектирования. *Chun Yuan, Yiyun Chen* (Китай) предложили формализм для оценки архитектур программного обеспечения на основе преобразования мультимножеств. *Yoonsoo Lee, Kyungseob Yoon, Heechang Koh, Changjong Wang* (Корея) предложили подход к сохранению компонент на основе сохранения архитектуры для переиспользования. *Li Jie, Hao AiLi, Mai ZhongFan* (Китай) описали характеристики архитектур программного обеспечения, причем не с точки зрения их статического описания, а по отношению к процессу их разработки. *Chidon Ahn, Sangkil Kim, Seunggeun Lee, Changjong Wang* (Корея) рассказали о методе репозитария для спецификации компонент на основе языка XML. *Zhang Weishi, Li Guanyu* (Китай) представили результаты исследований по верификации запрограммированных и переиспользуемых компонент программного обеспечения. *Fu Shao-yong, Liu Ji-ren, Tian Wen-hu* (Китай) представили результаты исследования и применения ключевых технологий сборочного программирования.

На секции "Процесс программного обеспечения" было заслушано 5 докладов. *Gerhard Chroust* (Австрия) посвятил свой доклад моделям процесса программного обеспечения: структуре и требованиям. *Nenad Cus Babic, Jozsef Gyorkos* (Словения) говорили о вопросах улучшения процессов программного обеспечения на основе предсказания возможных рисков. *Hong Guo, Graham King, Margaret Ross, Geoff Staples* (Англия) рассказали о последних достижениях по созданию европейской методологии оценки процесса программного обеспечения BOOTSTRAP, основанной на методах

CMM и SPICE. *Hyungwon Lee, Leon J. Osterweil* (США) представили результаты сравнительного исследования двух языков процессов HI-PLAN и Little-JIL, а также описали направление по созданию языков процессов следующего поколения. *Yanbo Han,Herbert Weber* (Германия) представили язык потока работ HOOM и основанный на этом языке подход к управлению адаптивными процессами.

На секции "Оценка программного обеспечения" были представлены 6 докладов. *Jukka Paakki, Anssi Karhinen, Juha Gustafsson, Lilli Nenonen, A. Inkeri Verkamo* (Финляндия) предложили метод автоматического анализа качества архитектуры путем поиска в ней архитектурных и проектных образцов. *Honghua Dai* (Австралия) описал ряд важных критериев для спецификации оценки производительности в разработке данных. *Emilia Mendes* (Новая Зеландия) представила опыт использовании метода по аналогии для оценки трудоемкости создания сетевых приложений. *Vergados D., Zevgolis D., Protonotarios E., Petrini G., Lami G.* (Греция) рассказали о разработке и учебном плане магистерского курса по оценке качества программного обеспечения. *Ernst Kesseler* (США) рассказал о применении теории к практике на примере анализа и оценки программного обеспечения летательных устройств. *Boyka Gradinarova, Dimitar Damianov* (Болгария) рассмотрели два метода, используемых при решении задачи управления процессом с одним входов и одним выходом, возникающей в среде обучения.

На секции "Формальные методы и спецификации" состоялось 9 докладов. *Tommi Mikkonen* (Финляндия) рассказал об управлении концептуальными абстракциями при спецификации систем, существенно использующих программное обеспечение. *Xudong Guan, Yiling Yang, Jinyuan You* (Китай) предложили подход к усилению исчисления мобильных амбиентов. *Yunming Wang, Jean-Pierre Talpin, Albert Benveniste, Paul Le Guernic* (Франция) рассмотрели подход к компиляции и распределению машин состояний на основе системы SPOTS . *Li Guangyuan, Tang Zhisong* (Китай) предложили линейную временную логику с непрерывной семантикой для спецификации гибридных систем. *Wang Yunfeng, Pang Jun, Zha Ming, Yang Zhaohui, Zheng Guoliang* (Китай) разработали исчисление уточнений для перехода от спецификаций к выполняемым программам с использованием программного оконного вывода. *Francois Siewe, Dang Van Hung* (Китай) описали подход к проектированию распределенных систем реального времени, который позволяет в одной унифицированной схеме обрабатывать модели с непрерывным и дискретным временем. *Guoliang Zhoujun Li, Huowang Chen* (Китай) исследовали отношения открытой бисимуляционной эквивалентно-

сти для исчисления передачи значений. *Zhu Huibiao, He Jifeng* (Макао) представили денотационную семантику языка описания оборудования Verilog с помощью исчисления вывода. *Qian Jun, Huang Tao and Feng Yulin* (Китай) предложили модель сборочного конструирования и семантику интерактивного вычисления для спецификации объектных систем.

На секции "Валидация и верификация программного обеспечения" были представлены 11 докладов. *Hong Zhu, Xudong He* (США) предложили теорию тестирования сетей Петри высокого уровня. *P.N.M. Sampaio, C.A.S. Santos, J.P. Courtiat* (Франция) рассказали об использовании формального метода для верификации временной семантики документов на языке интеграции синхронизованного мультимедиа CMIL. *Shaoying Liu, Jim Woodcock* (Англия) посвятили доклад методу тщательных просмотров для верификации формальных спецификаций, основанных на анализе дерева ошибок. *Antti Puhakka, Antti Valmari* (Финляндия) представили способ использования основанных на теории CFFD инструментов для верификации коммуникационных протоколов при предположении живости. *Pertti Kellomaki* (Финляндия) предложил метод для спецификации распределенных систем, позволяющий осуществлять более гибкую верификацию на основе модульных спецификаций. *Xudong He* (США) рассказал о формализованном использовании вариантных диаграмм в иерархических сетях с предикатами и переходами. *Chris George* (Макао) предложил простейший подход, основанный на множествах сообщений, для доказательства надежности протоколов идентификации. *Mourad OULD* (Франция) предложил формальную спецификацию и автоматическую валидацию интерактивного программного обеспечения с использованием языка потока данных Lustre. *W. T. Tsai, Baisu Huang, Raymond Paul, Mustafa Poonawala* (США) представили некоторую объектно-ориентированную схему для тестирования программного обеспечения, а также описали ее применение для тестирования систем реального времени. *Timo Aaltonen, Mika Katara, Risto Pitkanen* (Финляндия) предложили подход к верификации спецификаций реактивных систем реального времени с использованием временных автоматов. *Vicente Pelechano, Oscar Pastor, Juan Sánchez* (Италия) предложили процесс генерации кода по концептуальным моделям.

На секции "Распределенные системы и системы реального времени" состоялось 11 докладов. *Xinfeng Ye* (Новая Зеландия) предложил схему контрольных точек для систем вычисления, работающих в сети Интернет. *Qiang Li, May Allam* (США) предложили эффективный механизм для контроля доступа для дисковых драйверов, присоединенных к сети. *Liu Fuyan,*

You Jinyuan (Китай) обсуждали переход при проектировании операционных систем от многоадресного пространства к одноадресному и безадресному. *Wensong Zhang, Shiyao Jin, Quanyuan Wu* (Китай) представили виртуальный Linux-сервер, построенный на кластерах серверов для масштабируемых сетевых служб. *Wang Zhiping, Xiong Guangze, Zhou Wanlei* (Китай) предложили так называемый RTCC-протокол, гарантирующий производительность для распределенных приложений реального времени без требований каких-либо изменений оборудования. *Maryline Silly* (Франция) представил интерактивный алгоритм гибкого планирования для задач реального времени при ресурсных ограничениях. *Tang Kai, Xu Xin, Shao Junli* (Китай) рассказали о новой программной архитектуре при проектировании мультипротокольного маршрутизатора, при которой он разбивается на ряд функциональных единиц. *Thierry Villemur, Khalil Drira* (Франция) предложили формальную схему, основанную на структурных и поведенческих моделях, которая нашла свое применение в проектировании совместных обслуживаний с использованием языка Джава. *Cheng Qingzhang, Lin Jianming, Zhao Xinjian, Hu Tongsen* (Китай) представили новую концепцию механизма параллельного управления для случая, когда кооперативная работа основывается на Интернете. *Li Zhongwen, Xiong Guangze, Liu Jinde, Guo Bing* (Китай) предложили механизм качества обслуживания для использования в гетерогенных системах. *Fumio Aoki, Hiroki Nogawa* (Китай) описали подход к распределенным вычислениям для создания медицинских изображений с высоким разрешением.

На секции "Сетевые вычисления и окружения" заслушаны 8 докладов. *Saneyasu Yamaguchi, Hitoshi Aida, Tadao Saito* (Япония) описали подход к решению проблемы разделения данных в разъединенной среде посредством системы модифицируемых объектов и исследовали скорость распространения информации в ней при выполнении модификации. *Huan Zhou, Jing Li, Yulin Feng* (Китай) рассмотрели вопросы повышения степени согласованности при кешировании для сетевого просматривания в мобильной среде. *Luciano Baresi, Alberto Coen Porisini* (Италия) представили подход к проектированию и созданию распределенных моделирующих окружений на основе языка описания архитектуры моделирования SADL. *Dang Gang, Jin Shiyao* (Китай) представили реактивную мультиагентную среду DSEW-R для сетевого моделирования. *Konstantinos Raptis, Diomidis Spinellis, Sokratis Katsikas* (Греция) рассмотрели язык Джава как средство склеивания распределенных объектов, реализованных с помощью разных технологий. *Wu Gang, Wang Huiming, Wu Quanyuan* (Китай) представили архитектуру

управления распределенной сетью на основе технологий CORBA и мобильных агентов. *Wang Yu, Xue Wenge, Li Zengzhi, Li Gang* (Китай) предложили подход к управлению сетями на основе комбинации технологии CORBA и языка Джава. *Zhigang Luo, Jinde Liu* (Китай) описали общую модель динамической балансировки загрузки на основе торговой службы.

На секции "Человеко-машинное взаимодействие" состоялось 7 докладов. *Lorna Uden, Alan Dix* (Англия) представили визуальный иконный интерфейс, ориентированный на его использование в Интернете детьми в возрасте 5-6 лет. *Chuan-Chieh Jung, Hong-Nian Chen, Jaspher Wang, Y. S Kuo* (Китай) описали программируемые графические пользовательские интерфейсы, позволяющие работать без проблем на китайском языке. *Zhiwei Guan, Yang Li, Hongan Wang, Guozhong Dai* (Китай) представили результаты эмпирической оценки пяти интерактивных моделей для систем размещения объектов. *Jaakko Hakulinen, Jorma Sajaniemi* (Финляндия) провели эмпирический и теоретический анализ некоторых элементов пользовательских интерфейсов с точки зрения их влияния на скорость взаимодействия и появление в них ошибок. *Masatake Yamato, Akito Monden, Ken-ichi Matsumoto, Katsuro Inoue, Koji Torii* (Япония) описали подход к быстрому зрительному (с помощью взгляда) выбору кнопок в общих средах, основанных на графических интерфейсах, таких как, например, Microsoft Windows. *Chaomei Chen, Chiladda Chennawasin, Yue Yu* (Англия) описали разработку трехмерного пространства знаний на основе моделирования и анализа предметной области. *Kazutomo Uezono, Tomoko Kataoka, Katsuhiko Kakehi* (Япония) предложили многоязыковую систему для универсальной коммуникации и сделали веб-браузерную реализацию системы обработки документов со смешанными языками.

На секции "Параллельное вычисление" были представлены 11 докладов. *Lishan Kang, Yan Li, Zhuo Kang, Pu Liu, Yuping Chen* (Китай) предложили три асинхронных параллельно выполняемых алгоритма для решения задачи функциональной оптимизации. *Gu Xiao-Dong, Xu Yin-Long, Chen Guo-Liang* (Китай) рассмотрели подходы к решению на многопроцессорной системе задачи упаковки ящика с начальными ограничениями. *Hiromi Kobayashi* (Япония) предложил меру сложности программирования для параллельной обработки, основанную на мере сложности преобразования параллельного алгоритма в программу для мультипроцессорного параллельного компьютера. *Xie Xing, Zhou Zhi, Chen Guo-Liang, Gu Jun* (Китай) описали эффективную стратегию рестарта локальной оптимизации при решении задачи коммивояжера. *E. L. G. Saukas, S. W. Song* (Бразилия) рассмотрели методы

параллельного программирования для компьютеров с распределенной памятью с точки зрения размера обмениваемых данных и количества обменов. *Kai Hu, Jihong Wang, Jianping Hu* (Китай) описали алгоритм планирования, отображающий граф синхронизации заданий на кластер, который образуется в локальной сети из незанятых сетевых рабочих станций для выполнения ими параллельного вычисления. *Zhang Youhui, Pei Dan, Wang Dongsheng, Zheng Weiming* (Китай) представили систему для кластера компьютеров, обеспечивающую хорошую доступность во время выполнения. *Rong Hongbo, Tang Zhizhong* (Китай) предложили подход к группировке путей и преодоления зависимостей по данным при решении задач программной конвейерной обработки. *Xinda Lu , Jingbo Ye* (Китай) описали Джава-параллельный интерфейс JPI для реализации планирования задач, коммуникаций и функций редукции, подобных тем, которые предоставляются параллельной виртуальной машиной PVM и интерфейсом пропуска сообщений MPI. *Xiong Yuqing, Bai Shuo, Liang Liping, Zhou Xingming* (Китай) описали алгоритм передач вида один всем, основанный на общей логической топологии. *Jingui Huang, Jianer Chen, Songqiao Chen* (Китай) описали простой и хорошо аппроксимирующий алгоритм для планирования мультипроцессорных работ на трехпроцессорных системах.

На секции "Инженерия данных" были заслушаны 7 докладов. *Thanh N. Huynh, A Min Tjoa* (Австрия) описали архитектуру складов данных, основанных на системе управления реляционными базами данных. *Ning Chen, An Chen, Longxiang Zhou* (Китай) показали эффективный способ извлечения нечетких правил из больших реляционных баз данных. *Kewen Wang, Lizhu Zhou* (Китай) предложили подход к встраиванию логических программ одного вида в программы, основанные на логике другого вида, и доказали корректность указанной трансляции. *Hoang Kiem, Do Phuc* (Вьетнам) описали расширение зависимости атрибутов в теории нечетких множеств для задачи классификации в извлечении данных. *Ge Yu, Guoren Wang, Yubin Bao, Akifumi Makinouchi, Kunihiko Kaneko, Taiyong Jin* (Япония) предложили серверную систему управления распределенными и параллельными базами данных для Windows NT. *Mario Piattini, Antonnio Martínez* (Испания) описали три простые меры для оценки сопровождения SQL кода. *Yong Zhang, Lizhu Zhou, Jun Chen* (Китай) предложили пространственную базу данных трехмерной сети, основанную на связях и близости.

На секции "Массивные информационные системы в Интернете" были представлены 11 докладов. *Kurki-Suonio Reino, Tommi Mikkonen* (Финляндия) предложили подход, связанный с переходом от конструирования про-

грамм к инженерии абстракций. *Li Wei* (Китай) рассмотрел возникающие при проектировании, реализации и сопровождении главные проблемы массивных информационных систем. *Yoshikiyo Kato, William G. Griswold, Jimmy Yuan* (США) описали разработанные инструменты и проведенные эксперименты с программой большого масштаба, а также провели анализ результатов и их применений для проектирования инструментов. *Jin Shi, Lizhu Zhou* (Китай) описали организацию и управление устойчивыми объектами в архитектуре с многоуровневой памятью. *Qin Ding, William Perrizo, Kaushik Das, Qinghua Zou* (США) обсуждали приложения извлечения данных для больших данных у плохо распознанных образов. *Yunhai Tong, Shiwei Tang, Wei Xu, Kunqing Xie, Dongqing Yang* (Китай) описали двухуровневую семантически интегрированную схему географических данных для поддержки семантической интеграции пространственных информационных ресурсов. *Paula Leinonen, Eila Kuikka, Martti Penttonen* (Финляндия) охарактеризовали класс структурных трансформаций документов, распадающийся на так называемые плотные, иерархические и локальные трансформации, которые можно эффективно реализовать двухфазовой полуавтоматической процедурой. *Wei Yuan, Long Xiang, Li Wei* (Китай) предложили метод для оценки производительности параллельных приложений, основанных на распределенной системе с разделяемой памятью. *Lin Chuang, Shan Zhiguang, Yang Yang* (Китай) описали интегрированные схемы планирования и отбора запросов для веб-серверных кластеров и предложили для этих схем модели стохастических сетей Петри высокого уровня. *Zhang Dell, Dong Yisheng* (Китай) предложили эволюционный подход к совместной фильтрации. *Rakesh Mohan Bhatt* (Индия) обсудил вопросы расширяющихся обходов при использовании технологии асинхронного способа передачи (ATM).

4. МЕЖДУНАРОДНАЯ КОНФЕРЕНЦИЯ ПО ИСПОЛЬЗОВАНИЮ В ОБРАЗОВАНИИ ИНФОРМАЦИОННЫХ И КОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ (ICEUT-2000)

Работа конференции ICEUT-2000 велась по трем основным направлениям: пожизненное обучение (38 докладов), подготовка преподавателей (55 докладов), обучение информатике (39 докладов), каждое из которых заняло 4 секционных заседания. Всего состоялось 12 секционных заседаний. На этих заседаниях особое внимание уделялось новым формам обучения информатике, средам обучения, обучению в информационном обществе, тех-

ническим новациям и педагогике, методологии виртуальных классных кабинетов, вопросам использования информационных и коммуникационных технологий, общим вопросам обучения, инструментам обучения, проблемам дидактики и учебных планов, технологическим и педагогическим вопросам взаимодействия преподавателя и обучаемого, обучению с моделированием, инструментам поддержки обучения, дистанционному образованию для учителей.

На секции 1 "Пожизненное обучение и аспекты работы и расширенного гражданства" рассматривались доклады, посвященные углубленному пониманию взаимосвязей между непрерывным обучением на протяжении всей жизни (с детства до старости), дистанционным образованием и расширенным гражданством («расширенное гражданство» — это осознание человека членом виртуального информационного общества). *Mike Kendall* (Англия) рассказал об осуществляемых в Англии на политическом уровне действиях по углублению связей между расширенным гражданством и пожизненным обучением. *Yingbin Wei, Yaohong Kang, Zhaoqin Wang & Jianqin Huang* (Китай) рассмотрели китайские разработки по дистанционному обучению для поддержки пожизненного обучения. *Iris Braun, Katrin Borcea & Alexander Schill* (Германия) познакомили с текущими исследованиями по созданию интерфейсов для поддержки выполнения работ и проведения обучения телеработников на дому. *Zhang Chun-Feng & Guo Dong-Sheng* (Китай) изучали подтексты домашних использований и доступов при развитии обучения на дому. *Hadwig Kraeutler & Helmut Stemmer* (Австрия) рассказали о связях между музеями и школами, установленных в Австрии. *Gao Yuan, Song Jinyao, Pang Haizhen & Zhao You* (Китай) предложили проявлять определенную осторожность в культурных и образовательных факторах, например, ограничивать разработку в Китае дистанционного образования.

На секции 2 "Пожизненное обучение и аспекты педагогики и познания" были представлены доклады, посвященные большому разнообразию вопросов обучения и изучения, связанных с разработкой технологий пожизненного обучения. *Michel Arnaud* (Франция) рассмотрел факторы и черты познавательных взаимодействий, которые влияют на различные подходы к обучению. *Bernadette Charlier & Amaury Daele* (Бельгия) представили информацию о том, как студенты проходит совместное обучение, участвуя в проекте виртуального студенческого городка. *Carolyn Dowlin* (Австрия) рассмотрел социальные, эмоциональные и познавательные аспекты современных интеллектуальных педагогических агентов, используемых в программном обеспечении дистанционного обучения. *Hajime Saitoh, Noriko*

Tanaka, Takashi Ohno, Takashi Maeda, & Azuma Ohuchi (Япония) описали, как были разработаны и использованы так называемые концептуальные схемы для представления знаний. *Ru De Liu, Qi Chen & David Reid* (Англия) рассказали о влиянии компьютерных иллюстрационных инструментов на результаты обучения. *Antonio Simao Neto* (Бразилия) описал предварительные результаты исследований процесса познания у детей при использовании видеоигр. *Vittorio Midoro, Stefania Bocconi & Luigi Sarti* (Италия) предложили подходы к разработке инструментов оценки качества онлайновых курсов. *Ni Huilian* (Китай) описал текущее состояние исследований по разработке дистанционного обучения в Китае.

На секции 3 "Пожизненное обучение и связанные с ним изменения институтских и предметных учебных планов" были рассмотрены доклады о различных аспектах современных разработок по изменению учебных планов в связи с непрерывным пожизненным обучением. *Kjell Atle Halvorsen* (Норвегия) описал институтские изменения и влияния, связанные с созданием в Норвегии сетевого университета. *Lena Andersson-Skog, Ulf Hedestig & Ove Lundberg* (Швеция) обсуждали влияния технологий пожизненного обучения на обращение с научным знанием. *Lorna Uden & Alan Dix* (Англия) рассказали о введении обучения, основанного на решении проблем, и его влиянии на курсы для инженеров по программному обеспечению. *Erik De Corte, Lieven Verschaffel, Joost Lowyck, Stijn Dhert & Luc Vandeput* (Бельгия) обсуждали, как среды совместного обучения внедряются в среднюю школу для поддержки методов решения задач в математических курсах. *Robert Aiken, Munir Mandviwalla, Ned Kock & Cheryl Sandas* (США) рассказали, как информационные технологии интегрируются в американские курсы по трем разным предметным дисциплинам. *Teodora Bakardjieva & Boyka Gradinarova* (Болгария) доложили об использовании электронных журналов в поддержке реформы образования в небольших странах. *Rakesh Mohan Bhatt* (Индия) описал связи между потребностями учителей и технологическими инфраструктурами в национальных разработках в Индии. *Liu Huajun* (Китай) описал потребности учителей, использующих кабинеты с сетевыми компьютерами.

На секции 4 "Непрерывное обучение и технологические среды обучения" рассматривались доклады, охватывающие широкий спектр проектов, связанных с разработкой сред обучения, аккумулирующих потребности учителей. *Luis Anido, Martín Llamas, Manuel Fernández & M. Caeiro* (Испания) представили работы по созданию сетевой среды совместного обучения. *Chin-Hwa Kuo, Nai-Lung Tsao & David Wible* (Тайвань) рассмотрели

проектирование и использование сетевой среды для чтения для иностранцев, говорящих по-английски. *Harri Keiho, Jari Lahti & Jari Multisilta* (Финляндия) рассказали о том, как обучение WAP-технологии интегрируется в университетский мультимедийный курс. *Erik Dahl & Kurosh Bozorgebrahimi* (Норвегия) показали, как организуются сетевые семинары широкого профиля для поддержки профессионального развития преуспевающих служащих. *Tom Cunningham* (Шотландия) рассказал о разработке среды сетевого обучения, интегрирующей видеоконференции. *Yunsheng Liu, Zuoliang Cao, Changyun Yu & Zheming Wang* (Китай) описали факторы разработки, связанные с проектами организации дистанционного обучения. *Yinan Kang, Xiaojie Yuan & Xue-hu Feng* (Китай) представили, как можно интегрировать технологию интеллектуального гипермедиа в проекты дистанционного обучения. *J.C. Burguillo, P. Pavón, M.J. Fernández, M. Llamas & L. Anido* (Испания) рассказали о разработке гетеродинной технологической среды для поддержки дистанционного обучения.

На секции 5 "Информатика и развитие учебного плана по информатике" были представлены доклады, связанные с изменением аспектов учебных планов по информатике. *Fred Mulder & Tom van Weert* (Нидерланды) рассмотрели основу и детали Схемы Учебного Плана по Информатике 2000. *Monique Grandbastien & Jocelyne Rouyer* (Франция) рассказали о практике привлечения студентов в исследовательские лабораторные проекты. *Bo Han & Huazhu Song* (Китай) очертили основные составляющие курса по информатике. *Shiva Azadegan* (США) представил работы по конфигурируемому в сети учебному материалу, покрывающему вопросы этики и защиты. *Victor N. Kasyanov* (Россия) представил ориентированную на работу в сети Интернет информационную систему по истории информатики в Сибири. *David Carpenter, Dudley Dolan, Denise Leahy & Michael Sherwood-Smith* (Ирландия) рассказали о текущем состоянии работ по стандарту ECDL/ICDL, а также о планах по его применению и связанных с ними разработках. *Carl K. Chang, James Cross, Louis Hang, Hong Mei, Fred Mulder, Russ Shackelford & Pradip Srimani* (США) очертили основы и текущие работы по проекту "Учебный план 2001". *Yakov Fet* (Россия) говорил о моральных аспектах изучения истории информатики.

На секции 6 "Информатика и подходы к обучению и изучению" рассматривались доклады, посвященные проблемам, стоящим перед учителями, использующими или изучающими компьютер и связанными с необходимостью профессиональной реализации постоянно изменяющейся области. *Jill Slay* (Австрия) представил диапазон потребностей по адаптации и

дал введение в технологии решения проблем, связанных с внедрением сетевого обучения. *Johann S. Magenheim & Sigrid E. Schubert* (Германия) рассказали о разработке и опыте применения методов видеоанализа, отражающего практику эффективного обучения и изучения. *Harry Zhou & Doris Lidtke* (США) представили работы и результаты по интегрированному подходу к обучению и изучению в курсах по компьютерному обучению. *Guillermo León de la Barra, Ana María Urbina & Mario León de la Barra* (Чили) рассказали о разработке модели гибридного обучения и опыте ее применения при обучении математике с использованием компьютеров. *Frank Piessens & Bart De Decker* (Бельгия) представили эксперименты по сравнению кабинетного обучения и обучения с использованием школьных сред. *Chao Zeng, Li Xie, Guihai Chen, Setsuo Arikawa & Yoshihiro Ishihara* (Япония) рассказали об использовании он-лайновой системы для поддержки мониторинга студенческой производительности.

На секции 7 "Информатика и среды обучения" были заслушаны доклады, в которых изучались различные методологии обучения и изучения, а также рассматривались примеры технологической их поддержки. *Theda Thomas & Carina de Villiers* (Южная Африка) описали результаты применения технологии и технологически поддержанного кооперативного обучения в ряде институтов Южной Африки. *Antonio Navarro, Baltasar Fernandez-Manjon, Alfredo Fernandez-Valmayor & Jose Luis Sierra* (Испания) рассказали о разработке гипермедийных приложений. *Carlos Tadeu Q. de Morais, Júlio P. Machado, Paulo B. Menezes & Ricardo Reis* (Бразилия) исследовали потенциал сетевых обучающих систем, основанных на формальных методах, в Бразилии. *Kimio Sugita, Youzou Miyadera, Kensei Tsuchida & Takeo Yaku* (Япония) исследовали использования интегрированных обучающих сред со средствами визуализации. *Margaret Nagy, Elemér Nagy, Gyrgy Hampel Gábor Papp & Csilla Heves* (Венгрия) очертили использование моделирующих программ в деловых курсах. *Anna Grabowska* (Польша) исследовала потенциал новой среды обучения в одном из университетов Польши. *Xiao Dan & Tan Eng Chong* (Сингапур) описали использование технологии электронной белой доски. *Veaceslav Sidorenco* (Молдавия) рассказал об использовании виртуальных классных комнат в дистанционном обучении. *Nhon Van Do* (Макао) представил программу поддержки обучения математике, а также анализа геометрических задач. *Vladislav Katkov* (Белоруссия) говорил о разработке системы рисования функциональных графов.

Секция 8 "Информатика и развитие аспектов программирования" объединила доклады, посвященные различным аспектам программирования,

начиная от сопровождения современных курсов и кончая разработкой специфических проблем, связанных с целью разработки. *Rebeca Cortazar, Asuncion Barredo & Jose Luis Del Val* (Испания) описали содержимое основных курсов по технологии программирования в одном из университетов Испании. *Raul Sidnei Wazlawick & Antonio Carlos Mariani* (Бразилия) исследовали использование актеров и сцен для вводного курса по объектно-ориентированному программированию. *Tiffany Ya Tang & Albert Wu* (Гонконг) рассказали о достоинствах использования мультиагентной интеллектуальной обучающей системы в курсах по языкам программирования. *Youzou Miyadera, Ning Huang & Setsuo Yokoyama* (Япония) рассказали о преимуществах включения программной анимации в курсы по языкам программирования. *Ivan Kopecek & Karel Pala* (Чехия) рассказали об использовании диалоговой системы для обучения и изучения языка программирования. *Vladan Pantovic, Nikola Lazovic, Dusan Starcevic & Igor Uros* (Югославия) доложили об использовании различных программных агентов для поддержки схемы виртуального университета.

На секции 9 "Обучение преподавателей и стимулирование профессиональных разработок по информационным технологиям" были представлены доклады, в которых рассматривались пути, используемые разными странами для стимуляции преподавательских профессиональных разработок, и текущее состояние решения этой проблемы. *Ruud Brünemann, Pieter Hogenbirk & Hans Puper* (Нидерланды) описали подходы, применяемые в Нидерландах для проведения преподавательских профессиональных разработок. *Catherine P. Fulford, Curtis P. Ho & Ariana Eichelberger* (США) описали структуру и цель курсов, разрабатываемых в Гавайских университетах. *Lisbeth Appelberg, Eric Bruillard, Toni Downes Yaacov Katz, Tomás ó'Briain Baruch Offir, Don Passey & David Passig* (Швеция) представили имеющийся базис по подготовке преподавателей для дистанционного обучения. *Hernes, Morten Hestmann & Erna Haaland* (Норвегия) описали текущую ситуацию в данном вопросе в школах Норвегии. *Benigno Enza, Giorgio Olimpo & Mauro Tavella* (Италия) рассказали об использовании в Италии прототипной системы для стимулирования преподавательской компетенции и доверия. *Andrea Bartlett* (США) представил основу для использования электронных портфелей для поддержки обучения преподавателей. *Anne McDougall, Paul Nicholson & Alan Marshall* (Австралия) описали первые шаги по разработанной в Виктории схеме, направленной на обеспечение всех преподавателей портативными компьютерами. *Rosa Tripa & Isabel Chagas* (Португалия) описали структуру и базис португальской программы

непрерывного обучения преподавателей. *Marco Arscone & Rosa Maria Bottino* (Италия) рассказали об основе и развитии итальянского курса по обучению преподавателей с использованием информационных технологий. *Wang Li, Lin Zi & Liu Yumei* (Китай) рассмотрели вопросы, которые необходимо исследовать при разработке будущих программ по обучению электронным технологиям.

На секции 10 "Обучение преподавателей и аспекты познания" были представлены доклады, относящиеся к технологии обучения преподавателей и аспектам познания с помощью учителя. *Baruch Offir, Yael Lev & Yosi Lev* (Израиль) рассмотрели основу разработки эволюционной матрицы для изучения качества и частоты вербальных и невербальных взаимодействий при различных ситуациях, возникающих в процессе дистанционного обучения. *Li Zhiping & Li Chongrong* (Китай) исследовали связь между формами сетевой технологии и видами обучения. *Eric Bruillard & Georges-Louis Baron* (Франция) описали достоинства применения концептуальных схем в компьютерном обучении. *Timo Honkela, Teemu Leinonen, Kirsti Lonka & Antti Raike* (Англия) привели результаты исследований технологий обучения научным открытиям. *Jianwei Zhang, Qi Chen & David J. Reid* (Финляндия) поделились опытом использования самоорганизующихся схем в ситуациях конструктивного обучения. *John Murnane* (Австралия) рассмотрел фундаментальные аспекты использований основанного на информационных технологиях моделирования. *Chaobin Li & Wenxin Li* (Китай) представили работы по курсу дистанционного обучения для преподавания физики.

На секции 11 "Обучение преподавателей и мультимедийные среды обучения" были заслушаны доклады, посвященные различным аспектам использования мультимедийных сред в учебных кабинетах. *Margarete Grimus* (Австрия) представила результаты исследования применения мультимедийных средств большим числом средних школ в Австрии. *Giovanna Gazzaniga, Gaetano Morrone, Emanuela Ovcin & AnnaRosa Scarafiotti* (Италия) рассказали о некоторых разработках студенческой компании в рамках мультимедийной среды. *L. Uden, V.E. McGuinness & A. Alderson* (Англия) описали основу для исследования и его результаты при сравнении управления учителем и группой студентов, использующих мультимедийную программу. *Ernst Age Johnsen* (Норвегия) рассказал о разработке и использовании в Норвегии системы мультимедийных сетевых курсов по изучению английского языка. *Enrica Lemut, Bettina Pedemonte & Elisabetta Robotti* (Италия) описали использования мультимедийной геометрической софтверной программы для получения геометрических знаний. *Evgenia*

Sendova & Ivailo Ivanov (Болгария) представили текущие работы по мультимедийному основанному на языке Лого обучению в Болгарии. *Márta Turcsányi-Szabó* (Венгрия) рассказала о работах по мультимедийному Лого-обучению в Венгрии. *Edgar Weippl & Hans Lohninger* (Австрия) описали опыт применения мультимедийных смысловых схем. *Wang Shaofeng & Wang Kehong* (Китай) говорили о необходимости стандартизации курсов и программного обеспечения систем управления для разработки программ мультимедийного дистанционного обучения.

На секции 12 "Обучение преподавателей и взгляд в будущее" были представлены доклады по широкому кругу вопросов, связанных с будущими применениями и разработками информационных технологий в их приложениях для обучения преподавателей. *David Passig* (Израиль) очертил будущие потребности преподавателей в процессе интернет-обучения. *Sindre Rvik* (Норвегия) рассказал об управленческих приложениях в школах в терминах будущих эффективных использований информационных технологий. *M. J. Verdu R. Mompo M. A. Pez & B. Rodruez* (Испания) посвятили свой доклад различиям в будущих применениях глобальных и локальных сетей. *Alex C.W. Fung & Jims C.F. Yeung* (Гонконг) описали работы по сетевой адаптивной образовательной системе. *Ivan Kalas & Andrej Blaho* (Словакия) представили будущую версию языка Лого. *K. Maly, C. M. Overstreet, J. Brunelle, Y. Park & M. Ireland* (США) описали опыт применения новой интерактивной удаленной обучаемой системы. *M.P. Thapliyal* (Индия) обсуждал будущие приложения сетевых технологий для обучения преподавателей. *Jiang Dongxing, Tao Chaoquan & Fang Guowei* (Китай) представили разработку новой обучающей сетевой системы студенческого городка. *Wen-Chai Song & Shih-Ching Ou* (Тайвань) говорили о будущих потенциальных использованиях виртуальной реальности и приложений для обучения преподавателей. *J. A. Elorriaga, A. Arruarte and I. Fernndez-Cast* (Испания) рассказали о текущих работах по созданию интеллектуальных систем обучения и описали разработанный ими инструмент, позволяющий преподавателю легко создавать планы лекций и управлять ими. *K.Subramanian* (Индия) рассуждал о будущем Индии и об особенностях всеобщего обучения в новом тысячелетии. *Jin S, Wang Qiong, Li Xiaoming, Jim Gilligan, Stephen Ryrie, Song Jin, Wang Shengqing* (Англия, Китай) описали международную программу сетевого обучения, получившую название Internetics Scheme, и представили некоторые эксперименты, связанные с ее выполнением.

С.А. Логачева

АНАЛИЗ ЗАВИСИМОСТЕЙ ПО ДАННЫМ НА БАЗЕ АЛГОРИТМА ШОСТАКА[1]

1. ВВЕДЕНИЕ

При распараллеливании программы одной из основных проблем является выявление зависимости по данным. Существует три типа зависимостей: потоковая, антизависимость и выходная. Потоковая является истинной, остальные — «ложными». Далее мы будем говорить только об истинных зависимостях. Для их выявления существует ряд приближенных тестов, основанных на различных математических фактах.

В настоящей работе строится точный тест для анализа зависимостей по данным, основанный на исследовании разрешимости системы линейных неравенств с двумя переменными (полученной в результате выявления зависимостей) на базе алгоритма Шостака [1]. Тест состоит из двух этапов. Начальный этап включает в себя подбор пар операторов, “подозрительных” на зависимость. Второй этап состоит в применении модифицированного алгоритма Шостака.

В п. 2 данной работы приводится модель входной программы; в п. 3 описывается алгоритм Шостака и его модификация; в п. 4 производится сравнение полученных результатов алгоритма Шостака с результатами наиболее известных тестов на зависимость; в п. 5 описывается программная реализация проекта.

Все понятия, не определяемые в этой работе, могут быть найдены в [2].

2. МОДЕЛЬ ПРОГРАММЫ

Рассматриваются зависимости, порождаемые ссылками на элементы массива, имеющиеся в теле DO-цикла, поскольку при анализе зависимостей по данным решение этой задачи представляет основную трудность.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Предполагается, что не существует смешивания имен, таким образом, ячейки памяти, к которым обращаются по разным именам, являются различными.

В качестве модели программы рассматривается совершенное гнездо циклов (тело циклов находится внутри самого внутреннего цикла) таких, что приращения индексных переменных равны 1 и по каждой размерности используется только одна, отличная от других индексная переменная.

Наконец, предполагается, что индексные выражения линейные с целочисленными коэффициентами, а границы DO-циклов известные константы. (Введенные ограничения обеспечивают точность теста.)

Границы DO-циклов также могут линейно зависеть от индексной переменной любого внешнего цикла. Но в этом случае не всегда можно гарантировать точность теста, поскольку этот вопрос изучен не до конца.

3. АЛГОРИТМ ШОСТАКА

3.1. Определения

Пусть S — множество линейных неравенств вида $ax + by \le c$, где x, y — действительные переменные и a, b, c — действительные коэффициенты. Введем специальную нулевую переменную v_0 и будем считать, что она появляется в S только с нулевым коэффициентом. Таким образом, если неравенство содержит только одну переменную, то второй переменной для него будет v_0.

Построим для S неориентированный мультиграф G следующим образом: каждой переменной, встречающейся в S, сопоставим вершину в G, каждому неравенству — ребро, помеченное этим неравенством, например: ребро, помеченное неравенством $ax + by \le c$, связывает вершины x и y. Назовем

$G = G(S)$ графом для S.

Пусть P — путь в G, заданный последовательностью вершин $v_1, v_2, \ldots, v_{n+1}$ и последовательностью ребер $e_1, e_2, \ldots, e_n$, где $n \ge 1$. Назовем *последовательностью триплетов* последовательность вида $(a_1, b_1, c_1), (a_2, b_2, c_2), (a_n, b_n, c_n)$, где для любого i, $1 \le i \le n$, ребро e_i, помечено неравенством $a_i v_i + b_i v_{i+1} \le c_i$. Путь P является *допустимым*, если b_i и a_{i+1} имеют противоположные знаки для $1 \le i \le n$-1.

Понятно, что допустимые пути соответствуют последовательностям неравенств, которые образованы транзитивными цепями. Например, последовательность

$$x \le y, \quad y \le z, \quad z \le 3$$

образует допустимый путь, состоящий из последовательности вершин x, y, z и последовательности ребер $x - y \le 0$, $y - z \le 0$, $z + 0v_0 \le 3$, а последовательность

$$x \le y, \quad y \le z, \quad -z \le r$$

не образует допустимый путь, так как коэффициенты при z имеют одинаковые знаки.

Путь является *циклом*, если его первая и последняя вершины совпадают. Цикл называется *простым*, если все его промежуточные вершины различны.

Циклические перестановки цикла P являются *допустимыми* тогда и только тогда, когда a_1 и b_n имеют противоположные знаки, где (a_1, b_1, c_1), …, (a_n, b_n, c_n) — последовательность триплетов для P. В этом случае говорим, что цикл P — *перестановочный.*

Заметим, что допустимый цикл с начальной вершиной v_0 не является перестановочным, так как v_0 содержится в S только с нулевым коэффициентом.

Определим *остаточное неравенство* для допустимого пути P как неравенство, полученное с помощью последовательного применения транзитивности к неравенствам, помечающим ребра P. Например, если P образован цепочкой неравенств

$$x \le 2y + 1, \; y \le 2 - 3z, \; -z \le w,$$

то получим

$$x \le 2y + 1 \le 2(2 - 3z) + 1 \le 2(2 + 3w) + 1 = 6w + 5.$$

Таким образом, остаточное неравенство для P имеет вид $x - 6w \le 5$.

Дадим точное определение. *Остатком* r_P для пути P называется триплет (a_P, b_P, c_P), полученный следующим образом:

$$(a_P, b_P, c_P) = (a_1, b_1, c_1) * (a_2, b_2, c_2) * \ldots * (a_n, b_n, c_n),$$

где (a_1, b_1, c_1), …, (a_n, b_n, c_n) — последовательность триплетов для пути P и $*$ — бинарная операция:

$$(a, b, c) * (a', b', c') = (kaa', -kbb', k(ca' - c'b)) \qquad \text{и} \qquad k = \operatorname{sign}(a').$$

Остаточное неравенство для пути P есть $a_P x + b_P y \le c_P$, где x и y соответственно первая и последняя вершины пути P.

Достаточно просто показать, что операция $*$ ассоциативна, поэтому остаточное неравенство определено однозначно.

Индукцией по длине P легко показать, что любая точка (действительное число), удовлетворяющая неравенствам, помечающим ребра пути P, также удовлетворяет остаточному неравенству для P.

3.2. Алгоритм проверки разрешимости множества линейных неравенств с двумя переменными

Пусть P — цикл с начальной вершиной x, тогда точка, удовлетворяющая неравенствам вдоль P, должна удовлетворять остаточному неравенству $a_P x + b_P x \le c_P$. Если $a_P + b_P = 0$ и $c_P < 0$, то остаточное неравенство для P неверно, и мы говорим, что P — *невыполнимый цикл.*

Это значит, что множество неравенств S неразрешимо, если граф G для S имеет невыполнимый цикл. Обратное утверждение верно не всегда. Например, на рис. 1 показан граф G для $S = \{x \le y,\ 2x + y \le 1,\ z \le x,\ w \le z,\ z \le 1 + w,\ z \ge ½\}$, где S является неразрешимым, но граф не имеет невыполнимых циклов.

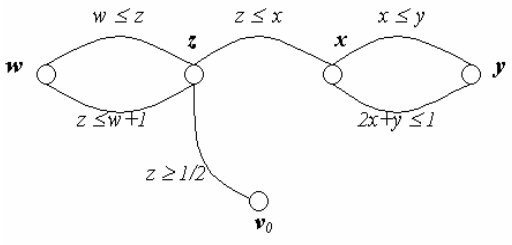


Рис. 1

Основная идея состоит в том, чтобы преобразовать граф G в граф G', который имел бы простой невыполнимый цикл тогда и только тогда, когда множество S было бы неразрешимо.

Пусть G — граф для S. Построим граф G' путем добавлением к графу G для каждого простого допустимого цикла P из G новой дуги, помеченной остаточным неравенством для P. Полученный граф назовем замыканием графа G.

Заметим, что замыкание не обязательно является единственным, так как начальная вершина перестановочного цикла может быть выбрана произвольным образом.

Теорема. *S является неразрешимым тогда и только тогда, когда G′ имеет простой невыполнимый цикл. [1]*

На рис. 2 показано единственное замыкание графа, изображенного на рис. 1. Заметим, что только *xyx*-цикл графа *G* добавляет ребро в *G′*. Цикл v_0xzv_0 графа *G′* является невыполнимым (имеет остаток (0, 0, –1/3)), значит, по теореме множество *S* неразрешимо.

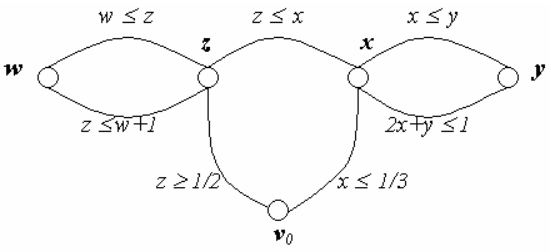


Рис. 2

Таким образом, имеем следующий **алгоритм проверки разрешимости множества неравенств *S***:

1) просматриваем все простые допустимые циклы и их циклические перестановки и вычисляем их остатки. Если находим невыполнимый цикл, тогда *S* неразрешимо;
2) в противном случае строим замыкание графа *G* добавлением новых ребер, помеченных остаточными неравенствами. Затем вычисляем остатки всех вновь сформированных простых допустимых циклов. Если находим невыполнимый цикл, тогда *S* неразрешимо. Иначе *S* имеет решение.

Если *S* является разрешимым, то построение решения приводится в [1]. Заметим, что новые допустимые циклы, образованные в п. 2), должны иметь начальную вершину v_0.

Алгоритм Шостака может быть обобщен для множества строгих неравенств и для множества неравенств с произвольным числом переменных [1].

3.3. Эффективность алгоритма

Метод Шостака требует нахождения простых циклов, для чего существуют несколько алгоритмов (Jonson, Szwarcfiter and Lauer [6], Read and Tarjan [7], [1]) со сложностью по времени $l(|V| + |E|)$ и сложностью по размерности $|V| + |E|$, где l — число порожденных циклов. Эти алгоритмы легко преобразуются для нахождения только простых допустимых циклов. Сложность полученных алгоритмов отличается от сложности исходных на константный множитель, так как каждый цикл имеет длину порядка $|V|$.

Заметим, что множество неравенств с целочисленными константами целочисленно разрешимо, если оно разрешимо в действительных переменных. Обратно не верно. Таким образом, алгоритм Шостака полезный, но не полный тест для проверки целочисленной разрешимости.

3.4. Модификация алгоритма

Для проверки разрешимости системы неравенств в целочисленной области все неравенства с одной переменной должны быть редуцированы путем деления свободного члена на коэффициент при переменной и взятия целой части, например, $10x \le 25$ преобразуется в $x \le 2$, т. е., просматривая данное множество линейных неравенств, редуцируем все неравенства с одной переменной и далее на каждом шаге алгоритма редуцируем полученные остаточные неравенства.

В данной работе из-за введенных ограничений на модель программы, таких, что по каждой размерности используется только одна индексная переменная, отличная от других, множества неравенств будут иметь три различных переменных, одна из которых будет v_0. Таким образом, построенный граф будет иметь три вершины и, соответственно, максимальный размер циклов будет равен трем.

Очевидно, что процесс редуцирования неравенств даст ответ на существование целочисленного решения. Пусть граф состоит из вершин x, y, v_0. Возможны три случая:

1. Если тестируется цикл (xyx), то создается остаточное неравенство, которое редуцируется, тем самым мы попадаем в целочисленную область.
2. Если тестируется цикл (v_0xv_0) или (v_0yv_0), то целочисленная разрешимость очевидна.

3. Если тестируется цикл (v_0xyv_0) (пусть он дан последовательностью неравенств $x \le c_1$, $ax+by \le c_2$, $y \le c_3$), то целочисленная разрешимость легко проверяется графически:

   а) пересечение соответствующих прямых образует треугольную область. Если существует вещественное решение, то существует и целочисленное, например точка $A=(c_1, c_3)$ на рис.3;

   б) в остальных случаях область решения содержит бесконечное число целочисленных точек (например, см. рис.4).

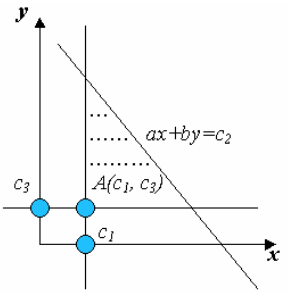


Рис. 3

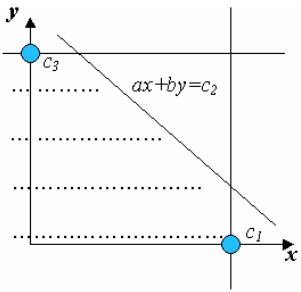


Рис. 4

Таким образом, модифицированный алгоритм Шостака при данных начальных ограничениях обеспечивает точную проверку целочисленной разрешимости для множества линейных неравенств с двумя переменными.

4. СРАВНЕНИЕ РЕЗУЛЬТАТОВ

Проведем сравнение результатов алгоритма Шостака с результатами наиболее известных алгоритмов, таких как НОД-тест, тест Банержи-Вольфа.

Рассмотрим пример:

```
for  (i=1; i<=25; i++)
{
     A[18i-26] = i+5;
     C[i]=A[-i+200];
}
```

Уравнение зависимости выглядит как

$$18x + y = 226$$

при следующих ограничениях:

$$\{x \geq 1,\ x \leq 25,\ y \geq 1,\ y \leq 25\}.$$

Представляя уравнение зависимости в виде двух неравенств, получаем систему линейных неравенств:

$$S = \{18x + y \leq 226,\ 18x + y \geq 226,\ x \geq 1,\ x \leq 25,\ y \geq 1,\ y \leq 25\}.$$

На рис.5 показан граф для S (только сплошные ребра). Тестирование графа указывает на наличие зависимости. Будем искать зависимость с использованием вектора направления (∗). Переход от (∗) к (<) добавляет неравенство $x < y$, которое преобразуется в $x \leq y - 1$ для целых x и y. Штрихованные ребра в графе представляют неравенства, которые выводятся из системы неравенств после добавления ограничения (<), например: неравенство x ≤ 11 получено из этого ограничения и уравнения зависимости.

Теперь рассмотрим ребра в следующем порядке: $-18x - y \leq -226,\ y \leq 25,\ x \leq 11$. Это простой допустимый цикл. Получаем остаточное неравенство $0 \leq -3$, которое является неверным, следовательно, цикл не выполним. Таким образом, зависимости в направлении (<) не существует. Однако, например, тест Банержи-Вольфа[3] не показал независимость в этом направлении, потому что его уравнение зависимости имеет рациональное решение внутри итерационного пространства, (рис.6).

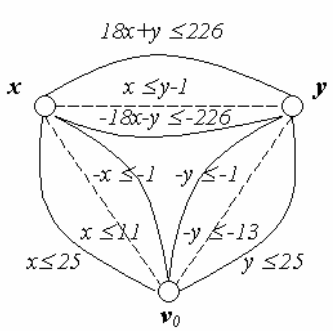


Рис. 5

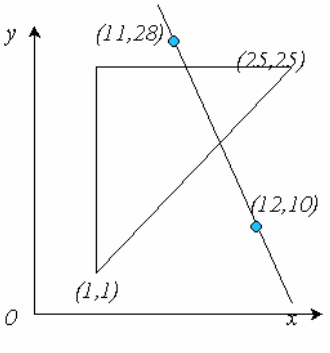


Рис. 6

Также не существует зависимости в направлении (=), значит, она должна быть в направлении (>). Если добавить соответствующие ограничение в граф (только сплошные ребра), то замыкание графа не будет содержать невыполнимые циклы.

Н.О.Д.-тест на этом примере показал зависимость, поскольку НОД(18, 1) = 1 и 226 делится на единицу. Н.О.Д.-тест, конечно, более быстрый тест, чем алгоритм Шостака, но на практике неэффективен, поскольку в большинстве случаев, как и в приведенном выше примере, НОД(a, b) = 1 и тест выдает зависимость там, где ее нет.

Аналогично тест Банержи показал зависимость по всем направлениям, (хотя она существует только в направлении (>)), потому что существуют вещественные решения диофантова уравнения в области итерационного пространства.

Рассмотрим еще один пример:

```
for(i=1; i≤10; i++)
  for(j=1; j≤i-1; j++)
  {
      A[i][j] = A[j][i];
  }
```

Заметим, что ни тест Банержи, ни Н.О.Д.-тест не позволяют вычислить зависимости с удовлетворительной точностью в треугольных циклах. Однако алгоритм Шостака позволяет находить зависимости в циклах такого вида, поскольку основан на проверке разрешимости системы линейных неравенств, например, в данном случае он показал, что зависимости нет.

Таким образом, алгоритм Шостака имеет следующие достоинства:

1) является точным тестом для выявления зависимости по данным при определенных начальных условиях, в отличие от других приближенных тестов, которые могут находить зависимость по данным там, где ее нет на самом деле;
2) позволяет тестировать циклы треугольного вида, что не удается делать многим другим тестам, хотя в данном случае не всегда можно гарантировать точность теста.

Стоит отметить, что алгоритм Шостака позволяет проконтролировать наличие промежуточных записей в ячейку памяти, относительно которой решается задача выявления зависимости (memory-based-dependence и value-based-dependence), что требует использования пресбургеровских формул [5] и представляет интерес для дальнейших разработок.

5. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Проект реализован на языке C++ с использованием библиотеки MFC и рассчитан на анализ С-программы с определенными ограничениями на синтаксис языка. Он состоит из двух этапов. Первый включает в себя подбор пар операторов, “подозрительных” на зависимость, второй состоит в применении модифицированного алгоритма Шостака. В проект также включены НОД-тест и тест Банержи для возможности сравнения результатов.

5.1. Синтаксис программы

Поскольку в качестве модели входной программы рассматривается совершенное гнездо циклов с определенными ограничениями, используется усеченный синтаксис.

Реализованы стандартные типы: int, float, char; стандартные арифметические операции; операторы: присваивания и for-цикла. Программа должна состоять из блока описаний и процедуры main. В блоке описаний должны быть описаны все переменные, а процедура main должна состоять из последовательности операторов for-цикла и операторов присваивания.

5.2. Подбор пар операторов, “подозрительных” на зависимость

Текст входной программы переводится в дерево внутреннего представления. Каждая вершина дерева имеет определенный тип (оператор, переменная, выражение), информационные поля (количество и значения которых зависят от типа), списки непосредственных потомков и предшественников.

Для каждой вершины оператора присваивания строится множество входных и выходных переменных. Далее происходит обход по дереву внутреннего представления, рассматривается вершина оператора присваивания и сравнивается множество входных и выходных переменных данной вершины с множествами входных и выходных переменных всех потомков данной вершины, которые являются операторами присваивания. В зависимости от пересечения этих множеств получаются пары операторов, “подозрительных” на зависимость, антизависимость или выходную зависимость.

Если встречается переприсваивание переменной, т. е. возникает выходная зависимость, то сравниваются индексные выражения массивов (по которым возникает выходная зависимость) в обоих операторах. Если индексные выражения совпадают, то происходит прерывание просмотра по этой ветке в данной вершине, что частично обеспечивает value-based dependence, например:

```
for (I=1; I<100; I++)
{
S1:   A[I]=…;
      …
S2:   A[I]=…;
S3:   …=A[I+1];
}
```

В данном случае зависимости между S1и S3 не существует, так как происходит переприсваивание переменной в операторе S2. И тест не выдаст пару S1 и S3 в качестве операторов, “подозрительных” на зависимость, потому что для оператора S1 произойдет прерывание просмотра в вершине оператора S2.

После того как произошел подбор пар операторов, “подозрительных” на зависимость, происходит разбор индексных выражений массивов, порождающих зависимость, и построение уравнения зависимости.

В случае гнезда циклов каждая размерность тестируется отдельно, что не нарушает точности теста, поскольку при данной модели программы граф неравенств в алгоритме Шостака распадается на отдельные графы, у которых общей является только одна вершина v_0. Таким образом, графы можно исследовать независимо друг от друга, а значит, можно применять тест к каждой размерности отдельно. Если хотя бы по одной размерности тест показал независимость, то и система уравнений зависимости не имеет решения, следовательно, зависимости в целом не существует.

5.3. Реализация алгоритма Шостака

Уравнение зависимости и соответствующие ограничения на индексные переменные преобразуются в систему линейных неравенств, поскольку алгоритм Шостака основан на построении неориентированного мультиграфа для системы линейных неравенств. Для нахождения всех простых циклов

графа был использован алгоритм Szwarcfiter and Lauer [6]. Граф задается массивом вершин, каждая из которых имеет список инцидентных ей ребер.

6. ЗАКЛЮЧЕНИЕ

Практическим результатом данной работы является построение точного теста для анализа зависимости по данным, написанного на языке C++ с использованием библиотеки MFC, на вход которого подается текст С-программы с усеченным синтаксисом: программа состоит из блока описаний и процедуры main. Процедура main содержит совершенное гнездо циклов, у которых границами являются известные константы; приращение индексной переменной равно единице; по каждой размерности используется только одна индексная переменная. (Эти ограничения обеспечивают точность теста.) Следует отметить, что тест работает и тогда, когда границы индексных переменных являются линейными выражениями от индексной переменной любого внешнего цикла (в этом случае нельзя гарантировать точность теста). На выходе получается список операторов, порождающих зависимость по данным.

В интерфейс системы встроен колоризированный текстовый редактор. Входная программа в результате лексического анализа разбивается на множество лексем, каждому типу которых соответствует свой цвет. Текст входной программы переводится в дерево внутреннего представления, в результате обхода которого происходит поиск операторов, “подозрительных” на зависимость. Далее происходит разбор индексных выражений и построение уравнения зависимости. Затем исполняются тест Шостака, Н.О.Д.-тест и тест Банержи и сравниваются результаты.

В дальнейшем данный тест может быть обобщен, что уменьшит начальные ограничения, и использован при написании компилятора в качестве блока анализа зависимости по данным.

СПИСОК ЛИТЕРАТУРЫ

1. **Shostak R.** Deciding linear inequalities by computing loop residues // J.ACM. — 1981. — Vol.28, №4. — P. 769–779.
2. **Евстигнеев В.А.** Основы параллельной обработки. Анализ программных зависимостей. — Новосибирск, 1996.

3. **Burke M., Cytron R.** Interprocedural dependence analysis and parallelization.: Tech. Rep. / IBM. — № 11794. — New-York, 1986.
4. **Banerjee U.** An introduction to formal theory of dependence analysis // J. Supercomputing. — 1988. — Vol. 2. — P. 133–149.
5. **Pugh W., Wonnacott D.** Static analysis of upper and lower bounds on dependences and parrallelism // J.ACM. — 1994. — Vol.16, № 4. — P. 248–278.
6. **Евстигнеев В.А.** Применение теории графов в программировании. — М.: Наука, 1985.
7. **Read R.C., Tarjan R.E.** Bounds on backtrack algorithms for listing cycles, paths, and spanning trees // ERL Memo M 433, Electronics Research Lab., Univ. Of California, Berkeley, Calif., 1973.
8. **Страуструп Б.** Язык программирования Си++. — М.: Радио и связь, 1991.
9. **Касьянов В.Н. Поттосин И.В.** Методы построения трансляторов. — Новосибирск: Наука, 1986.
10. **Girkar M., Polychronopoulos C.** Compiling issues for supercomputers: Rep. / CSRD, — №766. — Illinois, 1988.

В. А. Евстигнеев

NUMA-АРХИТЕКТУРА: НЕКОТОРЫЕ ОСОБЕННОСТИ КОМПИЛЯЦИИ И ГЕНЕРАЦИИ КОДА[1]

ВВЕДЕНИЕ

Известно, что мультипроцессорные системы характеризуются наличием множества идентичных процесоров, сообща решающих одну и ту же задачу и использующих общие ресурсы системы (например, память). Характерной особенностью мультипроцессоров является единое адресное пространство, поддерживаемое аппаратно. Конструктивно общая память может быть выполнена различно как в виде отдельной центральной памяти (как, например, в машинах Cray, NEC, Encore, Sequent и др.), так и в виде пространственно разделенной памяти (как в машинах Alliant, KSR, BBN, Cedar). В последнем случае машины приобретают свойства наращиваемой (scalable) вычислительной системы.

В машинах с общей памятью процессоры обмениваются информацией с помощью доступа к одной и той же ячейке памяти. В машинах с распределенной памятью процессоры с их локальной памятью обмениваются информацией путем явной посылки и получения сообщений. Накладные расходы поддержки обмена сообщениями — главная забота этих машин.

Можно утверждать, что различие между этими двумя семействами MIMD-машин исчезает. В машинах с общей памятью для сокрытия задержек при обращении к памяти вводятся такие механизмы, как хорошо организованная иерархическая система памяти. Для машин с распределенной памятью ожидается применение лучших схем выбора маршрутов для посылки сообщений для уменьшения накладных расходов, связанных с обменом сообщениями. Эта тенденция может привести к появлению машин с распределенной памятью, поддерживающих вычисления мелкозернистого уровня.

Единое глобальное пространство общей памяти на машинах с общей памятью обеспечивает более прозрачную модель программирования для MIMD-машин. Однако использование иерархической системы

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

памяти не позволит машинам с общей памятью быть наращиваемыми в достаточной степени. В противоположность этому машины с распределенной памятью могут легко наращиваться, но становятся более трудными для прикладного программирования. На конференции Supercomputing'91 фирма Alliant Computer Systems анонсировала систему Campus/800 с 800 процессорами. Она объявила, что Campus/800 есть первая система, поддерживающая и распределенную, и общую память. Ожидается, что система может облегчить программирование и повысить эффективность в реальных применениях.

Под наращиваемостью системы понимается возможность собирать ее из большого числа базисных компонент без принципиальной перестройки системы и ее программного обеспечения. Такими компонентами могут быть процессоры, блоки памяти, блоки сети связи и т.д. Ключ к наращиваемости системы — наличие пространственной и временной локальности вычислений. Естественно, что для каждой системы имеются пределы наращиваемости. Так, Cray Y-MP допускает конфигурации от одного до восьми процессоров, CM-2 — от 8K до 64K процессорных элементов, Cray C-90 имеет коэффициент наращиваемости 16, KSR-1 — 128 (от 8 до 1088 процессоров) и т.д.

1. ПОНЯТИЕ NUMA-АРХИТЕКТУРЫ

Масштабируемые (scalable) параллельные машины часто организуются как сети пар процессор-память; примерами таких машин являются машины фирмы BBN (Butterfly, TC 2000, GP 2000), KSR-1 и KSR 2 фирмы Kendall Square Research, NCUBE 2, а также мультикомпьютеры, подобные Intel iPSC/i860. Такие машины называются *машинами с неоднородным доступом к памяти*, или NUMA-машинами (*non-uniform memory access*), поскольку процессор получает доступ к своей локальной памяти в сотни и тысячи раз быстрее, нежели доступ к нелокальным данным. Например, в KSR-1 доступ к локальной памяти составляет 18 тактов, в то время как к нелокальной памяти — 175 тактов. Машины с распределенной памятью, подобные Intel iPSC/i860, имеют значительно большую неоднородность относительно времени доступа, так как доступ к нелокальным данным должен сопровождаться обменом сообщений. Если нелокальные доступы располагаются на критическом пути в программе, то превращение их в локальные путем управления данными будет ускорять исполнение программы.

Следующая особенность большинства NUMA-архитектур состоит в

том, что пересылка данных между процессорами блоками более эффективна, нежели посылка этих данных большим количеством маленьких сообщений. Пересылка данных между процессорами может рассматриваться как конвейер с большим временем разгона, сравнимым с временем работы одной ступени конвейера. Например, для Intel iPSC/i860 требуется 70 микросекунд для начала коммуникации, и лишь одна микросекунда для пересылки числа с плавающей точкой двойной точности между ближайшими соседями, если коммуникация между ними была уже налажена. Поэтому, когда некоторое число элементов данных должно быть отправлено между двумя процессорами, желательно использовать одно длинное сообщение для того, чтобы погасить время разгона.

1.1. Мультикомпьютеры

Итак, кроме машин, представляющих собой мультипроцессоры с общей разделенной памятью, к машинам с NUMA-архитектурой относятся практически все мультикомпьютеры. Различие между мультипроцессорными и мультикомпьютерными системами заключется в следующем:

- В мультикомпьютерах отсутствует аппаратная поддержка единого адресного пространства. Каждый компьютер в мультикомпьютере имеет свое адресное пространство. Общее глобальное адресное пространство формируется программными средствами как конкатенация номера компьютера представляющего собой узел мультикомпьютерной системы, и адреса внутри узлового адресного пространства для поддержки SPMD-модели вычислений.
- Мультикомпьютеры представляют собой семейство независимых, связанных между собой ЭВМ под управлением распределенной ОС сетевого типа. Каждый компьютер имеет копию ядра ОС.
- Распределение работ (программ и относящихся к ним данных) в мультикомпьютере происходит в основном статически, при изменении нагрузки компьютеров закрепленная задача может быть передвинута в другой узел.

  Размер задачи ограничен размером памяти узла. Мультикомпьютер не способен (или не очень эффективен) исполнять семейство больших скалярных программ, которые типичны для таких областей применения, как цифровое моделирование.
- Обмен данными в мультикомпьютерах осуществляется путем явной посылки сообщений, которые могут быть скрыты от пользо-

вателя аппаратно или программно. При этом возникает проблема такого распределения задач и данных по узлам, при котором расходы на организацию посылки сообщений были бы минимальны. Для минимизации расходов на обмен сообщениями данные могут перемещаться или переименовываться так же, как в системе виртуальной памяти.

- В отличие от мультипроцессоров, где данные хранятся в общей памяти в единственном экземпляре, в мультикомпьютерах из-за использования механизма посылки сообщений в одно и то же время в разных компьютерах могут храниться копии одних и тех же переменных.
- Каждое нелокальное (т.е. не к своей локальной памяти) обращение к данным в мультикомпьютере требует специального программного механизма для вычисления адреса, организации посылки сообщения и управления памятью, чтобы иметь возможность обрабатывать копии данных.
- В отличие от мультипроцессоров, где механизм посылки сообщений представляет собой работу с указателями, мультикомпьютеры требуют обязательного перемещения данных.
- Мультикомпьютеры хорошо работают на очень больших параллельных программах, исполняемых до полного завершения. Если любой из узлов испытывает недостаток некоторых средств, которые должны быть получены от других узлов, то возникают трудно устранимые узкие места.
- Мультикомпьютеры чаще всего не являются системами общего назначения в терминах области применения или размера задачи.

Примечание. Для уменьшения внутрисистемных задержек процессоры и устройства локальной памяти объединяются для создания эффективных вычислительных узлов мультикомпьютера. Используется также размещение одной и той же программы во всех узлах и рассылка данных по всем компьютерам для уменьшения расходов на перемещение данных. Последнее основано на принципе широковещания.

1.2. Прочие машины с NUMA-архитектурой

Среди новейших машин к классу NUMA-машин следут отнести суперкомпьютеры NCUBE 2, IBM SP-2, Convex SPP 1200/XA, Intel Paragon, Thinking Machine CM5, IBM RP3, проект DASH (Стенфорд), проект ALEWIFE (МТИ), Horizon/Tera.

2. ОСОБЕННОСТИ ПО NUMA-МАШИН

Для разработчика ПО следствием указанных выше особенностей NUMA-машин является то, что программы должны не только эксплуатировать параллелизм, но и управлять данными там, где это возможно, для исключения нелокальных ссылок; там, где нелокальные ссылки необходимы, они должны быть сгруппированы для блочной пересылки. Существующая компиляторная технология ориентирована большей частью на машины с *однородным* доступом к памяти, в которых одна забота — эксплуатация параллелизма. Параллельный код генерируется путем распределения среди процессоров итераций самого внешнего цикла в гнезде циклов вместе с вставкой команд синхронизации, чтобы позаботиться о зависимостях, порожденных этим циклом. Для уменьшения синхронизации применяются преобразования типа перестановки циклов для перемещения параллельных циклов в сторону самого внешнего, где это возможно. Этот подход не исполняет любое управление данными; он не пригоден для генерации хорошего кода для NUMA-архитектур.

Альтернативный подход, реализованный в языке Фортран-D, состоит в том, чтобы дать программисту управление тем, как структуры данных распределяются по процессорам. Компилятор использует информацию о *декомпозиции данных* для определения того, как распределять работу по процессорам. Один простой способ сделать это — использовать так называемое правило *собственности* — процессор исполняет оператор присваивания, если левосторонняя переменная оператора отображается в локальную память этого процессора. Процессор исполняет итерацию цикла, если он способен выполнить любую работу в теле цикла на этой итерации. Хотя эта стратегия принимает в расчет отображения данных, компилятор может генерировать неэффективный код, в котором все процессоры исполняют все итерации “разыскивая работу для выполнения”, если структура гнезда циклов не соответствует распределению данных. Во многих таких случаях реструктуризация цикла может улучшить качество кода, но никакой общий подход к преобразованиям цикла недопустим в этом контексте.

2.1. Реструктуризация циклов

В работе [1] представлен систематический подход к реструктуризации циклов для параллельных машин с иерархией памяти. Как и в под-

ходе, основанном на собственности, наша стартовая точка есть язык типа Фортран-D со специфицированной пользователем декомпозицией данных. Однако прежде чем использовать эту информацию непосредственно для генерации кода, мы используем информацию о распределении данных для управления *нормализацией доступа*, которая представляет собой метод реструктуризации цикла, включающий в себя перестановку циклов, скашивание цикла (loop skewing), обращение цикла (loop reversal) и масштабирование цикла (loop scaling). Цель рестуктуризации — преобразовать гнезда циклов так, чтобы код мог бы быть сгенерирован распределением итераций самого внешнего цикла по процессорам без компроментации локальности. Структура внутренного цикла выбирается так, что данные могут быть перенесены с использованием, где это возможно, блока передачи (block transfers).

Указанная статья содержит следующие результаты.

- Описан новый метод, называемый *нормализацией доступа*, обеспечивающий компиляцию программ для параллельных машин с неоднородным доступом к памяти.
- Наши преобразования циклов выражаются в рамках *обратимых* матриц и теории целочисленных решеток, что является важным обобщением шаблона Банержи для работы с унимодулярных матриц. Это обобщение интересно само по себе и имеет приложения в других областях.

2.2. Пороговое планирование графа заданий

Оптимальное планирование в его наиболее общей форме — NP-трудная проблема. Известно несколько схем планирования. В работе [2] рассматривается проблема статического планирования бесконтурного графа заданий с ненулевыми вершинной и реберной стоимостями для машин с распределенной памятью.

Исходя из теоретической сложности проблемы планирования, предлагаются различные эвристики, которые обеспечивают получение ответа за полиномиальное время, но не гарантируют оптимальность. Эвристики могут сравниваться по их близости к оптимальному решению и вычислительной сложности.

Известные подходы не касаются сущности отображения заданий на машины с распределенной памятью как компромисса между длиной расписания и числом требуемых процессоров. Эта сущность приобрела важность из-за доступности систем с распределенной памятью, по-

добных Intel Paragon, который имеет низкую коммуникационную стоимость и большое число процессоров. В работе [3] разработана схема планирования времени компиляции, которая порождала масштабируемый (scalable) код для различного числа процессоров, для параллельного функционального языка Sisal [4,5]. Планировщик времени компиляции стремится значительно уменьшить длину расписания для систем с большим числом доступных процессоров. Для систем с малым числом доступных процессоров схема пытается сократить, насколько это возможно, число требуемых процессоров. Для обеих целей вначале предполагается бесконечное число процессоров при реализации планирования во время компиляции. Однако если во время прогона число доступных процессоров меньше, чем число требуемых процессоров, то предпрогоночный планировщик перестраивает расписание, подгоняя его к числу доступных процессоров. Наша стратегия планирования применима в общем случае для отображения функционального параллелизма в любом языке для машин с распределенной памятью.

Другой важной стороной дела является стоимость разбиения программ и планирования для семейства систем с распределенной памятью, которые имеют одну и ту же стоимость вычислений, но различную стоимость коммуникаций. Разница в стоимости коммуникаций может быть атрибутирована как разница соответствующих архитектур, аппаратуры маршрутизации и алгоритмов. Примером таких популярных систем служит семейство систем Intel iPSC на базе микропроцессоров i860. Эти системы имеют одну из следующих трех конфигураций:

- Intel Gamma — гиперкуб;
- Intel Delta — сеть (mesh);
- Intel Paragon — сеть с микроядрами OSF/1.

Таким образом, можно увидеть, что стоимость вычислений на этих системах одна и та же, так как все они базируются на микропроцессорах i860, но коммуникационная стоимость убывает в указанном выше порядке. Коммуникационная стоимость на системе Delta меньше, чем на системе Gamma главным образом из-за лучшей архитектуры, тогда как лучшая аппаратура маршрутизации служит причиной того, что коммуникационная стоимость в Paragon меньше, чем в Delta.

В работе [2] показано, что для крупнозернистых заданий, каковыми являются, например, вершины графа IF-2 для языка SISAL, можно в определеной степени избежать разбиения программ и регенерации расписания при компиляции для семейства систем, имеющих равные

вычислительные, но различные коммуникационные стоимости.

Имеет смысл сделать некоторые предположения относительно исполнения графов заданий Sisal IF-2 на целевой машине Intel iPSC/860.

1. Задания строгие (или, другими словами, задание не может начать исполнение, пока все входы не станут доступными), невытесняемые (non-preemptive) и имеют небольшие накладные расходы. Эти ограничения определяются семантикой Sisal.
2. Значения обмениваются между двумя процессорами в виде сообщений, используя блокирующие вызовы send() и receive(). Это предположение сделано специально для iPSC/860.
3. При реализации планирования во время компиляции предполагается бесконечное число процессоров.

2.3. Унифицирующие преобразования данных и управления

Большинство машин с общей памятью, базирующейся как на аппаратуре, так и на ПО, полагаются на кэширование данных для эксплуатации их локальности и на уменьшение коммуникаций. Когерентность кэша должна быть поддержана для многих копий одних и тех же данных на многих процессорах. Сейчас существует большое число работ в области когерентных протоколов в крупномасштабных мультипроцессорах. Эти протоколы варьируются от чисто аппаратной до программной реализации эмуляции общей памяти для машин с посылкой собщений. Существуют гибридные методы, которые содержат в себе и миграцию, и удаленные ссылки на NUMA-машинах. Для уменьшения стоимости когерентности данных и коммуникаций должна использоваться локальность данных.

Когда локальность эксплуатируется только с помощью времени прогона, ядер и политики уровня аппаратуры, которые наблюдают поведение программы снизу, то ложное общее использование (false sharing) становится важной проблемой. Неформально, ложное общее использование может быть описано следующим образом: два или более процесса осуществляют доступ к неперекрывающимся областям одного и того же когерентного блока (по крайней мере один из них пишет), вызывая ненеобходимый когерентный трафик и перемещение данных. Ложное общепользование является серьёзной помехой для высокой производительности машин с распределенной общей памятью.

В работе [6] предлагается алгебраическое представление отображений данных, модели локальности данных, новый алгоритм преобразо-

вания данных для локальности, а также унифицированный подход к улучшению локальности с преобразованиями данных и управлению для машин с общей распределенной памятью. Экспериментальные результаты, использующие множество приложений на параллельных машинах, показывают, что новые оптимизации значительно улучшают производительность.

3. БЛИЗКИЕ РАБОТЫ

Из многочисленных работ, касающихся машин с распределенной общей памятью, выделим следующие, имеющие непосредственное отношение к реструктуризации циклов и преобразованиям данных и управления: [7–13].

СПИСОК ЛИТЕРАТУРЫ

1. **Li Wei, Pingali K.** Access normalization: loop restructuring for NUMA compilers: Techn. Rep. / Cornell Univ., Comp. Sci. — N TR 92-1278. — Ithaca, 1992.
2. **Pande S., Psarris K.** A compilation technique for varying communication cost NUMA architectures // Proc. 6th Intern. PARLE Conf. Athens, Greece, 1994. — Berlin a.o.: Springer-Verlag, 1994. — P. 49–60. — (Lect. Notes Comput. Sci.; Vol. 817).
3. **Pande S., Agrawal D.P., Mauney J.** A fully automatic compiler for distributed memory machines // Proc. of the 26th Hawaii Intern. Conf. on System Sciences, January 1993. — P. 536–545.
4. **Евстигнеев В.А., Городняя Л.В., Густокашина Ю.В.** Язык функционального программирования SISAL // Интеллектуализация и качество программного обеспечения. — Новосибирск: ИСИ СО РАН, 1994. — С. 21–42.
5. **Густокашина Ю.В., Евстигнеев В.А.** IF1 — промежуточное представление SISAL-программ // Проблемы конструирования эффективных и надежных программ. — Новосибирск: ИСИ СО РАН, 1995. — С. 70–78.
6. **Cierniak M., Li Wei.** Unifying data and control transformations for distributed shared-memory machines: Techn. Rep. / Univ. of Rochester, Comp. Sci. — N 542. — Rochester, 1994.
7. **Balasundaram V., Fox G., Kennedy K., Kremer U.** An interactive environment for data partitioning and distribution // Proc. 5th Distributed Memory Comput. Conf., April 1990.
8. **Hudak D., Abraham S.** Compiler techniques for data partitioning of sequentially iterated parallel loops // Proc. ACM Intern. Conf. Supercomputing, June 1990.
9. **Knobe K., Lucas J., Steele G.** Data optimization: allocation of arrays to reduce communication on SIMD machines // J. of Parallel and Distrib. Computing. — Vol. 8, Feb. 1990. — P. 102–118.
10. **Li J., Chen M.** Index domain alignment: minimizing cost of cross-referencing between distributed arrays: Techn. Rep. / Yale Univ., 1989.

11. **Ramanujam J., Sadayappan P.** Compile-time techniques for data distribution in distributed memory machines // IEEE Trans. on Parallel and Distrib. Systems. — 1991. — Vol. 2, Oct. — P. 472–482.
12. **Wolf M., Lam M.** A data locality optimizing algorithm // Proc. ACM SIGPLAN 91 Conf. on Progr. Lang. Design and Impl., June 1991. — P. 30–44.
13. **Gannon D., Jalby W., Gallivan K.** Strategies for cach and local memory management by global program transformations // J. of Parallel and Distrib. Comp. — 1988. — Vol. 5. — P. 587–616.

В. Н. Касьянов, Ю. В. Бирюкова, В. А. Евстигнеев

ФУНКЦИОНАЛЬНЫЙ ЯЗЫК SISAL 3.0[1]

ВВЕДЕНИЕ

Название языка Sisal является аббревиатурой английского выражения Streams and Iterations in a Single Assignment Language [1–11]. Создание языка — результат сотрудничества четырех организаций: Ливерморской национальной лаборатории имени Лоренца, Университета штата Колорадо, Манчестерского университета (Великобритания) и Digital Equipment Corporation (DEC). Язык ориентирован на поддержку научных вычислений и представляет собой дальнейшее развитие языка VAL.

Впервые о проекте по созданию языка Sisal упоминается в работе 1983 г. [1], а в 1985 г. появилась первая версия языка Sisal 1.2 [2], которая была реализована на ряде машин, включая Denelcor HEP, Vax 11-780, Cray-1, Cray-X/MP [3]. Для нее имеется также работающий оптимизирующий транслятор OSC [4, 5]. В 1991 г. Ливерморская лаборатория и университет Колорадо опубликовали новую версию языка Sisal 2.0 [6], которая содержала в себе такие идеи современных функциональных языков, как функции высшего порядка, конструкция type set и, наконец, модули. Краткое описание этой версии можно найти в работе [7]. В дальнейшем работа продолжалась, и описание результатов работ над новой (пока никем не реализованной) версией языка — Sisal 90 — увидело свет в 1995 г. [8, 9].

Язык Sisal достаточно широко распространился по США [10]. В целом, по данным Ливерморской лаборатории (на июль 1993 г.), язык использовался в 79 организациях, из которых 16 находились за пределами США.

При разработке языка Sisal авторами проекта преследовались следующие цели:

1. Создать универсальный функциональный язык, который мог бы эффективно исполняться на машинах как последовательной, так и параллельных архитектур, включая новейшие.

2. Создать для представления программ независимое от языка и целевой архитектуры промежуточное представление типа потоковых графов.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

3. Разработать технику оптимизации для высокопроизводительных параллельных прикладных программ.

4. Разработать микрозадачное окружение для поддержки потокового стиля параллельных вычислений на мультипроцессорах с общей памятью.

5. Добиться эффективности последовательного и параллельного исполнения, сравнимой с императивными языками.

6. Внедрить функциональный стиль программирования для сложных научных применений.

С точки зрения семантики язык Sisal обладает рядом важных свойств. Во-первых, он математически правильный, т. е. функции языка отображают входы в выходы без побочных эффектов. Во-вторых, имена прозрачны для ссылок, т. е. они олицетворяют значения, а не ячейки памяти. В-третьих, Sisal — это язык однократного присваивания.

Обладая всеми качествами, присущими функциональным языкам программирования, Sisal способствует разработке корректных детерминированных программ, которые свободны от совмещения имен, побочных эффектов и ошибок, зависящих от реального времени. Результаты детерминированы, невзирая на архитектуру, операционную систему или обстановку исполнения. В отличие от императивных языков Sisal уменьшает нагрузку на программирование. Пользователь должен определить, что может быть вычислено, и ему достаточно только представлять зависимости по данным между операциями. Компилятор отвечает за планирование операций, передачу значений данных, синхронизацию операций, управление памятью. Легко написать параллельную функциональную программу, так как она кодируется как последовательные императивные программы. Пользователь освобожден от большинства сложностей параллельного программирования и поэтому получает возможность больше сконцентрироваться на конструкции алгоритмов и разработке прикладных программ [12].

В данной работе представлен язык функционального программирования Sisal 3.0, выбранный в качестве начальной версии входного языка системы функционального программирования SFP (СФП), разрабатываемой в ИСИ СО РАН при финансовой поддержке РФФИ (грант N 01-01-00794) и Минобразования. Цель работ — создание системы параллельного программирования SFP на базе языка Sisal, позволяющей прикладному программисту разрабатывать и отлаживать параллельную программу на своем рабочем месте с последующим ее исполнением на супервычислителе, доступном прикладному программисту по сети.

В разд. 2 кратко описана система SFP. Раздел 3 посвящен основным синтаксическим и семантическим характеристикам языка Sisal 3.0. В разд. 4 дается обзор доступных материалов, связанных с проблематикой языка Sisal.

1. СИСТЕМА SFP

Система SFP разрабатывается в рамках проекта ПРОГРЕСС [13, 14] и должна предоставить прикладному программисту на его рабочем месте удобную среду для разработки функциональных программ, предназначенных для последующего исполнения на ЭВМ с параллельными архитектурами, доступными через телекоммуникационные сети. В рамках этой среды программист должен иметь возможность, с одной стороны, создавать и отлаживать программу без учета целевой параллельной архитектуры, а с другой — производить настройку отлаженной программы на ту или другую целевую параллельную архитектуру с целью достижения высокой эффективности исполнения разработанной программы на суперЭВМ.

Процедура настройки состоит в реализации оптимизирующей кросс-трансляции разработанной функциональной программы в программу на языке супервычислителя (например, языке Си), в процессе которой программа подвергается необходимым оптимизирующим и реструктурирующим преобразованиям под управлением пользователя. При этом степень участия программиста может быть различной — от предоставления дополнительной информации до прямого управления производимыми преобразованиями. В частности, программист должен иметь возможность визуальной обработки создаваемой Sisal-программы в рамках ее внутреннего представления.

Система SFP включает следующие компоненты: интерфейс, отладчик, front-end транслятор, блоки промежуточных представлений IR1, IR2 и IR3, блоки анализа, преобразования и визуализации IR1-, IR2- и IR3-программ, конверторы промежуточных представлений, back-end трансляторы. Представления IR1 и IR2 соответствуют по своим функциям и возможностям известным промежуточным представлениям IF1 и IF2 для функциональных программ, а IR3 — графовое представление императивных программ.

2. КРАТКОЕ ОПИСАНИЕ ЯЗЫКА SISAL 3.0

Язык Sisal 3.0 продолжает традицию предыдущих версий и остается языком, ориентированным на написание больших научных программ. В нем адаптированы операции по обработке массивов из Фортрана 90, поддерживается мультиязыковое программирование и включены черты, упрощающие работу пользователя, хотя некоторые из них не укладываются в строгую парадигму функционального программирования. Большинство научных приложений имеет отношение к проблемам спецификации, завершения работы программ, ввода/вывода и обработки ошибок. По своей специфике эти задачи не функциональны и не параллельны, поэтому предполагается кодировать их на языке C. Однако ядро языка Sisal 3.0 параллельно и функционально. В таком определении языка можно усмотреть слияние парадигм императивного и функционального программирования, когда конструкции языка либо принадлежат интегрированному ядру, либо являются языковыми расширениями [15].

Наиболее важными и полезными чертами языка являются:

1. Статический полиморфизм и конструкция **type set,** поддерживающая функциональный полиморфизм и повышающая возможности языка по переиспользованию кода. При отсутствии этой конструкции в блоке **program**, компилятор гарантированно разрешит все типы и создаст эффективный код.

2. Функции высших порядков. Это подразумевает, что функции могут быть параметрами и результатами других функций, а также выступать в качестве значений выражений. Преимущества использования таких функций при научном программировании хорошо известны.

3. Определяемые пользователем редукции. Редукции предназначены для работы с циклическими значениями. Теперь наряду со стандартными редукциями пользователь может создать свои, что облегчает его работу, с одной стороны, и может явиться причиной недетерминированности — с другой.

4. Модули. Они введены в язык для того, чтобы можно было поддерживать несколько стилей программирования, в частности функциональный и модульный.

5. Возможность встраивания в Sisal-программы кода, написанного на языке программирования C.

6. Возможность аннотирования программ, что позволяет описывать контекст ее исполнения. Указанное свойство языка важно для эффективного применения преобразований на этапе трансляции программ, а также для их конкретизации.

2.1. Краткий обзор синтаксиса языка Sisal 3.0

Исполняемая единица языка состоит из одной или нескольких единиц компиляции, каковыми являются программа, интерфейсы и модули. Все эти единицы могут компилироваться отдельно. Простейшей из них является программа. Она состоит из множества деклараций, которые определяют импортируемые из других модулей единицы, некоторые типы данных и функции. Любая функция в программе может быть точкой начала исполнения. На этом уровне (самом внешнем) параметры функций представляют собой значения, получаемые на уровне операционной системы; результаты исполнения функции также относятся к этому уровню.

Модуль — синтаксический аналог программы, который не может служить начальной точкой для исполнения. Он объединяется с интерфейсом для экспорта некоторых своих типов и имен функций. Интерфейс модуля и сам модуль имеют одинаковое имя.

Язык Sisal 3.0 поддерживает стандартные скалярные типы данных: булевские, целые, действительные одинарной и двойной точности, комплексные одинарной и двойной точности и null. Из структурных типов в языке имеются массивы, потоки, записи, союзы и type set. Для определения сложной структуры данных необходимо указать типы составляющих ее элементов, число элементов при этом не указывается и становится известным только при присвоении структуре значения. Массивы и потоки состоят из однотипных элементов и различаются прямым и последовательным доступом к элементам. Записи и союзы состоят из разнотипных элементов и различаются способом хранения.

Основной синтаксической единицей языка является выражение, оно представляет список значений некоторых типов. Арность выражения — это размер списка значений. Основной конструкцией языка является let-выражение, которое вводит имена значений и заголовки функций, производит над ними некоторые действия и выдает результаты.

В языке присутствуют достаточно большое разнообразие видов циклов: итерационные циклы с инициализацией значений, циклы с предусловием и постусловием, а также явные параллельные циклы, экземпляры тела которых могут параллельно выполняться на различных процессорах.

К выражениям выбора, присутствующим в языке, относятся конструкции if и case, причем в последней имеется возможность выбирать ветвь не только по значению, но и по типу имени.

Функция языка Sisal состоит из заголовка и тела. В заголовке объявляются имена и типы формальных параметров, а также типы возвращаемых результатов. Существует возможность предописания функций и создания рекурсивных функций, а также описания функций, закодированных на языке программирования С.

Достаточно подробно многие из указанных конструкций языка описаны в руководстве пользователя [9], поэтому ниже мы остановимся лишь на тех структурах, о которых не упоминается в этом руководстве.

2.2. Средства модульного программирования

Так как язык Sisal 3.0 поддерживает принципы модульного программирования и раздельной компиляции, введем понятие единицы компиляции.

Единица компиляции — это текст, который может компилироваться без связи с другими текстами. Простейшей формой такой единицы является программа (**program**), содержащая только функции и определения типов. Эти функции вызываются с уровня операционной системы. Входом в программу являются фактические параметры вызываемых функций, а результаты вызываемых функций становятся выходом программы.

Важно отметить, что типы параметров и результатов не могут быть членами конструкции **type set**, функциями и составными значениями, включающими **type set** и функции, т. е. в этом контексте не допустим любой “непрозрачный” тип данных.

Программа может состоять одной или более единиц компиляции, при этом среди них обязательно должна существовать одна, которая использует служебное слово **program** и содержит функции, запускающие вычисления. Некоторые функции и типы, определенные в одной единице, могут использоваться (импортироваться) другими. В общем случае существует пара единиц компиляции: одна — **module**, которая содержит множество функций и определений типов, и другая — **interface**, которая определяет подмножество функций и типов конкретного модуля, доступных другим единицам компиляции. Образование пары осуществляется использованием обеими единицами одного и того же имени, например:

```
interface X
        % Объявление функций и типов,
        % доступных для других единиц компиляции
end interface
module X
```

```
        % Здесь содержатся полные определения функций и типов
        % в соответствующем интерфейсе и, может быть, другие опре-
          деления
end module
```

Интерфейс определяет связи модуля с внешним миром. Объявления импорта позволяет использовать функции или типы, определенные в одном модуле, в других единицах компиляции с помощью служебного слова **from**. Если не указано, какие именно функции и значения импортируются, то предполагается, что используются все описанные в интерфейсе значения и функции, например:

```
module Y
    from X: a, b, c
    . . .
end module
```

При компиляции *Y* некоторое представление о декларации *a, b* и *c* должно быть доступным. Поэтому компиляция *Y* возможна, если интерфейс модуля *X* уже написан и заранее откомпилирован, хотя исходный текст модуля *X* может быть в данное время недоступен.

2.3. Средства препроцессирования

В интегрированную среду подготовки программ на языке Sisal 3.0 как обязательный компонент предполагается включить препроцессор, назначением которого является обработка исходного текста программы до ее компиляции. Он также расширяет возможности языка, поддерживая:

1) подстановку имен и макросы,
2) включение файлов,
3) условную компиляцию,

что позволяет сберечь время при разработке программ и создать гибкие, удобочитаемые и более пригодные для сопровождения и отладки программы.

Для управления препроцессором используются директивы. Директивой служит строка исходного файла, в первой позиции которой указан символ #. Директивы могут размещаться в любом месте исходного файла, их действие остается в силе вплоть до конца файла. В соответствии со сказанным ранее имеются три типа директив препроцессора:

1) #define, #undef — первая предусматривает определение макросов или препроцессорных идентификаторов, каждому их которых ставится в соответствие некоторая символьная последовательность. В последующем тексте программы эти идентификаторы заменяются на указанные последовательности символов. Вторая директива отменяет действие первой;

2) #include — позволяет включать в текст программы текст из выбранного файла;

3) #if, #ifdef, #ifndef, #else, #endif, #elif — позволяют организовать условную обработку текста программы. Условность состоит в том, что компилируется не весь текст, а только его части, которые так или иначе выделены с помощью перечисленных директив.

Предполагается, что препроцессор будет работать аналогично препроцессору языка С, поэтому более подробную информацию о его работе можно найти в [16].

2.4. Средства аннотирования программ

Для более гибкого применения оптимизирующих преобразований на этапе трансляции и эффективного использования программ предполагается поддерживать возможность создания на языке Sisal аннотированных программ [17], т.е. таких программных текстов, в которых содержатся как собственно программные части (базовые программы), так и аннотации — формализованные комментарии, предназначенные для спецификации контекстов применений базовых программ.

Синтаксически каждая аннотация — это фрагмент Sisal 3.0 программы, оформленный в виде комментария, в котором в каждой строке после символа % расположен символ $.

Различаются глобальные и локальные аннотации.

Глобальные аннотации формируют полные единицы компиляции (модули аннотаций), которые вполне аналогичны единицам компиляции базовой программы и в которых практически без каких-либо ограничений могут использоваться любые конструкции базового языка Sisal 3.0.

Локальные аннотации являются частью единиц компиляции базовой программы и могут иметь вид объявлений, утверждений и директив (следует отличать их от ранее описанных директив препроцессора).

Объявления позволяют определять дополнительные имена, а также связи модуля с внешним миром.

Каждое утверждение — это логическое выражение, которое всегда истинно при любом допустимом исполнении базовой программы. Утверждения могут использоваться для описания свойств не только объектов (например, диапазонов входных и выходных значений), но и конструкций аннотированной программы.

Синтаксически каждая директива — это либо оператор присваивания, либо вызов функции. С помощью директив можно управлять процессом обработки транслируемой программы. Например, при помощи директив можно указывать характеристики целевой архитектуры или включать применение такого оптимизирующего (или реструктуирующего) преобразования, для которого автоматически не удается проверить справедливость корректности или целесообразности его выполнения.

Каждая локальная аннотация ассоциируется компилятором с первым зарезервированным словом, предопределенной функцией, константой, именем или арифметическим символом, следующими за ней в исходном тексте. Семантика локальных аннотаций зависит от применения.

2.5. Проблемные области

С точки зрения реализации языка Sisal 3.0 можно выделить две проблемные области. Первая затрагивает вопросы ввода/вывода. Все предыдущие версии языка не содержат операторов ввода/вывода, программа (версия Sisal 2.0) или функция main (версии Sisal 1.2 и Sisal 90) получают входные параметры и выдают выходные значения на уровне операционной системы; вывести промежуточные данные практически невозможно. Этот факт затрудняет процесс отладки и сопровождения программ. Поэтому в версии Sisal 3.0 предлагается использовать стандартную библиотеку функций ввода/вывода на языке C. Однако это порождает вторую проблемную область — из-за введения императивных процедур невозможно гарантировать детерминированность программ, которая заложена в семантике языка Sisal.

3. БЛИЗКИЕ РАБОТЫ

Приведем краткую характеристику доступных работ, которые так или иначе связаны с языком Sisal. Чтобы как-то систематизировать этот материал, выделим организации, работающие над проблематикой языка, для каждой из которых дадим обзор имеющихся опубликованных материалов.

3.1. Ливерморская национальная лаборатория им. Лоренца

Организация является одним из основных разработчиков языка Sisal и методов его реализации, базирующихся на промежуточных представлениях транслируемых программ IF1 [18, 19] и IF2 [20].

Оба представления являются иерархической графовой формой и подобны потоковым графам. Представление IF2 отличается от IF1 тем, что поддерживает прямые операции с памятью. Над представлениями можно провести машинно-независимые оптимизирующие преобразования, например исключение общих подвыражений, удаление инвариантов цикла. К ним возможно применение и машинно-зависимого анализа: нахождение векторных операций или распараллеливание графа для мультипроцессорных систем.

Силами Ливерморской лаборатории реализована версия Sisal 1.2, для нее существуют оптимизирующий компилятор OSC [4, 5], инструментальная система TWINE [21], а также интерпретатор.

Аппликативные языки, каковым является Sisal, по своей семантике идеально подходят для разработки алгоритмов для параллельных архитектур. Однако, чтобы гарантировать отсутствие сторонних эффектов и наличие прозрачности для ссылок в аппликативной программе, операции, которые изменяют значения данных, вынуждены работать с копиями этих данных. С этой точки зрения использование массивов, что характерно для научных вычислений, очень дорого по расходам времени и памяти, причем эти расходы иногда перекрывают выгоды от параллельного вычисления. Поэтому необходимо решать задачу эффективного использования памяти вычислительной машины. Существует несколько подходов к разрешению данной проблемы.

Ренеллетти предложил алгоритм, который работает на внутреннем представлении IF1. Граф исходной программы анализируется в два прохода с целью получения для каждого значения, вычисляемого программой, выражения его размера. Эта информация о размере, вычисляемая во время исполнения программы, используется для выделения буферов памяти для агрегатных структур, которые вычисляются позже и записывают свои элементы в уже отведенную для них память. Такое раннее выделение областей памяти под элементы данных делает ненужным размещение в памяти промежуточных буферов, тем самым сокращая количество операций копиро-

вания. Алгоритм не работает только в тех случаях, когда невозможно вычислить размер данных до их непосредственного вычисления [22].

3.2. Университет штата Колорадо

Основные направления исследований ученых университета штата Колорадо связаны с эффективной реализацией языка Sisal 1.2.

С этой проблематикой тесно связана задача эффективного использования памяти вычислительной машины, которая обсуждалась в предыдущем пункте. Девид Канн предложил алгоритм, работающий на промежуточном представлении IF2 с теми агрегатными структурами, для которых не пригоден алгоритм Ренеллетти. Размер этих структур не известен во время компиляции, поэтому невозможно выделить буфер в памяти, в который во время вычисления записывается структура. Алгоритм делится на три фазы: первая фаза готовит каждый граф к анализу, вторая удаляет ненужные операции по подсчету ссылок на агрегаты, а третья удаляет ненужные операции копирования и идентифицирует агрегаты, на которые ссылается только одна операция. Практические результаты показали, что две последние фазы алгоритма удаляют около 98% операций по подсчету ссылок и около 82% операций копирования [5].

3.3. Манчестерский университет (Великобритания)

Исследования сотрудников Манчестерского университета связаны с разработкой формального алгоритма эффективного использования памяти под агрегатные структуры для языков однократного присваивания [23, 24]. Данный алгоритм рассматривает все операции над агрегатными структурами. Для них вычисляется несколько формальных характеристик, которые помогают выделить те операции, которые могут выполняться над самими данными, а не над их копиями. Алгоритм может работать с вложенными агрегатами, используя метод подсчета числа операций, работающих с этими данными. Этот же алгоритм предусматривает, где это возможно, применение стратегии предраспределения памяти под агрегатные структуры, сходной со стратегией Ренеллетти, описанной выше.

3.4. Университет Nice Sophia Antipolis (Франция) и университет Аделаиды (Австралия)

Внимание ученых этих университетов привлекла версия языка Sisal 2.0, реализацией которой они и занимаются [25, 26]. К 1996 г. разработано формальное определение динамической семантики значительной части языка с использованием правил вывода Typol системы Centaur, что послужило строгому пониманию дизайна языка, т.е. исключило двоякую трактовку конструкций, обеспечило ценной информацией как разработчиков, так и пользователей языка, помогло сравнить Sisal с другими функциональными языками. К тому же система спецификаций Centaur позволяет автоматически создать структурный редактор и интерпретатор, которые в дальнейшем могут развиться в интерактивную оболочку для программирования на языке Sisal. Система Centaur дает возможность также разработать инструменты анимации для показа процесса вычислений.

Конечной целью данных исследований является формальная характеристика методов распараллеливания и алгоритмов компиляции языка с целью создания гибкой оболочки с элементами визуализации для программирования на языке Sisal.

3.5. Кипрский университет

Сотрудники Кипрского университета работают над созданием системы Mustang для автоматического распараллеливания Fortran-программ путем отображения их в семантику языка однократного присваивания [26–31]. Предполагается, что в начале исходная Fortran-программа транслируется в промежуточное представление IF1. Во время трансляции используется система Parafrase 2 для грамматического разбора исходной программы, проведения анализа зависимостей по данным и определения возможностей для распараллеливания, которые потом явно представляются в IF1-программе. Затем IF1-программа пропускается через соответствующие блоки оптимизирующиго Sisal-компилятора (OSC). На данный момент создан и оттестирован действующий прототип системы Mustag.

СПИСОК ЛИТЕРАТУРЫ

1. **McGraw, J. R. et. al.** Sisal: Streams and iterations in a single assignment language, Language Reference Manual, Version 1.1 / Lawrence Livermore Nat. Lab. Manual M-146. — Livermore, CA 1983.
2. **McGraw, J. R. et. al.** Sisal: Streams and iterations in a single assignment language, Language Reference Manual, Version 1.2 / Lawrence Livermore Nat. Lab. Manual M-146 (Rev. 1). — Livermore, CA 1985.
3. **Skedzielewski S. K., Welcome M. L.** Data flow graph optimization in IF1 // Lect. Notes Comput. Sci. — 1985. — Vol. 201. — P. 17–34.
4. **Cann D.C.** The optimizing SISAL compiler: Version 12.0. — Livermore, Apr.2, 1992. — (Prepr. / Lawrence Livermore National Laboratory; UCRL-MA-110080).
5. **Cann D. C.** Compilation techniques for high performance high performance applicative compilation // Technical Rep. CS-89-108. — Colorado State University, 1989.
6. **Bohm A. P. W., Oldenhoeft R. R., Cann D. C., Feo J. T.** The SISAL 2.0 Reference Manual. — Livermore, CA, 1991. — (Prepr. / Lawrence Livermore Nat. Lab.; UCRL-MA-109098, LLNL).
7. **Евстигнеев В. А., Городняя Л. В., Густокашина Ю. В.** Язык функционального программирования SISAL // Интеллектуализация и качество программного обеспечения. — Новосибирск, 1994. — С. 21–42.
8. **Feo D. T., Miller P. J., Skedzielewski S. K., Denton S. M.** Sisal 90 User's Guide / Lawrence Livermore Nat. Lab. Draft 0.96. — Livermore, CA, 1995.
9. **Бирюкова Ю. В.** SISAL 90 руководство пользователя. — Новосибирск, 2000. — 84с. — (Препр./ РАН. Сиб. Отд-е. ИСИ; № 72).
10. **Feo J. T.** Sisal. — Livermore, CA, 1992. — (Prepr. / Lawrence Livermore Nat. Lab.; UCRL-JC-110915, LLNL).
11. **Feo J. T., Cann D.C, Oldehoeft R.R.** A report on the Sisal language project // J. on Parallel and Distributed Computing. — 1990. — Vol. 10. — P. 349–366.
12. **McGraw J. R.** Parallel functional programming in Sisal: fictions, facts, and future. — Livermore, CA, 1993.— (Prepr. / Lawrence Livermore Nat. Lab.; LLNL).
13. **Kasyanov V. N., Evstigneev V.A. et al.** The system PROGRESS as a tool for parallelizing compiler prototyping // Proc. of Eighth SIAM Conf. on Parallel Processing for Scientific Computing (PPSC-97). — Minneapolis, 1997. — P. 301–306.
14. **Kasyanov V.N., Evstigneev V.A. et al.** Support tools for supercomputing and networking // Lect. Notes Comput. Sci. — 1999. — Vol.1593. — P. 431–434.
15. **Feo D. T., Miller P. J., Skedzielewski S. K., Denton S. M., Solomon C. J.** Sisal 90 // Proc. High Performance Functional Computing — Livermore, 1995. — P. 35-47.
16. **Трой Д.** Программирование на языке Си для персонального компьютера IBM PC / Пер. Б.А. Кузьмина под ред. И. В. Емелина. — М.: Радио и связь, 1991.

17. **Касьянов В. Н.** Оптимизирующие преобразования программ. — М.: Наука, 1988.
18. **Skedzielewski S. K., Glauert J.** IF1 — An intermediate form for applicative languages. Manual M-170 / Lawrence Livermore National Laboratory — Livermore, CA, 1985.
19. **Густокашина Ю.В., Евстигнеев В.А.** IF1 — промежуточное представление Sisal-программ // Проблемы конструирования эффективных и надежных программ. — Новосибирск, 1995. — С. 70–78.
20. **Welcome M., Skedzielewski S., Yates R.K., Ranelletti J.** IF2 — An applicative language intermediate form with explicit memory management / Lawrence Livermore Nat. Lab. Manual M-195. — Livermore, CA, 1986.
21. **Miller P. J.** TWINE: a portable, extensible SISAL execution kernel // Proc. Second SISAL User's Conf. — Livermore, 1992. — P. 243–256.
22. **Ranelletti J. E.** Graph transformation algorithms for array memory optimization in applicative languages. — Livermore, CA, 1987. — (Prepr. / Lawrence Livermore National Laboratory; UCRL-53832).
23. **Li Z., Kirkham C.** Efficient implementation of aggregates in united functions and obiects. — University of Manchester, 1995.
24. **Li Z., Kirkham C.** Efficient storage reuse of aggregates in single assignment languages. — University of Manchester, 1996.
25. **Attali I., Caromel D., Guider R., Wendelborn A. L.** Optimizing Sisal programs: a formal approach // Proc. Europar'96. –1996. — P. 1123–1124.
26. **Attali I., Caromel D., Wendelborn A. L.** A formal semantics and an interactive environment for Sisal // Tools and Environments for Parallel and Distributive Systems. — Kluver Academic Publishers, 1996.– P. 231–258.
27. **Cann D. C., Evripidou P.** Advanced Array Optimizations for high performance functional languages // IEEE transactions on parallel and distributed systems. — 1995. — Vol. 6, No. 3.
28. **Evripidou P., Gaudiot J.** Incorporating input/output operations into dynamic dataflow graphs // Parallel computing. — 1995. — Vol. 21. — P. 1285–1311.
29. **Evripidou P., Barry R.** Mapping Fortran programs to single assignment semantics for efficient parallelization // Parallel Processing Letters. — 1998. — Vol.8, N 3. — P. 407–418.
30. **Lachanas A., Evripidou P.** Exploiting course grain parallelism from Fortran by mapping it to IF1 // Lect. Notes Comput. Sci. — 1998. — Vol. 1470.
31. **Evripidou P., Lachanas A.** Venus compiler: mapping Fortran to single assignment semantics for efficient parallelization / Tech. Rep. TR-99-3. — University of Cyprus, 1999.

В. А. Вшивков, И. В. Лобив, Ф. А. Мурзин

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ О ВЗАИМОДЕЙСТВИИ ПОТОКОВ РАЗРЕЖЕННОЙ ПЛАЗМЫ[1]

ВВЕДЕНИЕ

Известные под общим названием методы “частиц в ячейках”, или PIC-методы (Particles In Cells) [1], широко применяются в вычислительной математике при моделировании различных процессов.

Большой интерес представляет вопрос об эффективном распараллеливании данного алгоритма.

В работе [2] содержится краткий обзор реализаций PIC-метода на параллельных ЭВМ. Из него следует, что возникают большие проблемы, связанные с балансировкой загрузки процессоров и сокращением коммуникационных издержек. Эти трудности обусловлены тем, что в процессе работы программы частицы могут переходить из одной пространственной подобласти в другую, поэтому через некоторое число шагов обычно производят перераспределение частиц по процессорам.

Процесс вычислений также значительно зависит от топологии вычислительной системы. Например, в работе [3] сделан вывод, что SOS — коммерческая программа моделирования трехмерной электромагнитной плазмы — на MIMD-мультипроцессорах с распределенной памятью и массовыми параллельными вычислениями, в том числе на гиперкубе с большим количеством вершин, работает неэффективно.

В данной работе рассматривается простейшая параллельная архитектура, содержащая коммутатор определенного вида. Подробно описан процесс отображения алгоритма на вычислительную систему данной архитектуры. В рамках некоторых естественных предположений оценены время выполнения алгоритма в параллельном и последовательном случаях, а также коэффициент ускорения.

Все это позволило более четко понять ядро проблемы, основные трудности и наметить направления дальнейших исследований. Выводы приведены в конце статьи.

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

1. ПОСТАНОВКА ЗАДАЧИ И ДИСКРЕТИЗАЦИЯ

Рассматривается задача о взаимодействии потоков разреженной плазмы в самосогласованных электромагнитных полях. Такие задачи возникают при изучении динамики солнечных вспышек, обтекании солнечным ветром магнитосферы Земли, взаимодействии летательных аппаратов с космической плазмой, активных экспериментах с облаками плазмы в космосе.

Полная система уравнений, используемая нами для решения поставленных задач, состоит из кинетического уравнения Власова для ионов, уравнения движения для электронной компоненты, а также уравнений Максвелла для элекромагнитного поля:

$$
\begin{array}{l}
\frac{\partial f_i}{\partial t} + \overline{v_i}\frac{\partial f_i}{\partial \overline{r_i}} + \frac{\overline{F}}{m_i}\frac{\partial f_i}{\partial \overline{v_i}} = 0, \\
\overline{F} = e(\overline{E} + \frac{1}{c}[\overline{v_i}\overline{B}]), \\
m_e(\frac{\partial \overline{u_e}}{\partial t} + (\overline{u_e}\nabla)\overline{u_e}) = -e\overline{E} - \frac{e}{c}[\overline{u_e}\overline{B}], \\
rot\overline{B} = \frac{4\pi n}{c}(\overline{u_e} - < \overline{v_i} >), \\
rot\overline{E} = -\frac{1}{c}\frac{\partial \overline{B}}{\partial t}, \\
div\overline{B} = 0.
\end{array}
$$

Область решения имеет форму параллелепипеда. В ней введена декартова система координат (x, y, z). Вся область разбита на одинаковые ячейки такой же формы с размерами h_x, h_y, h_z вдоль соответствующих направлений. Для повышения точности аппроксимации электромагнитных полей внутри ячейки используется их представление в сдвинутых сетках, показанное на рис. 1. Частицы, моделирующие плазму, располагаются внутри ячеек в соответствии с необходимой плотностью. Каждая частица имеет свою скорость, которая изменяется под действием электрического и магнитного полей. Друг с другом частицы не взаимодействуют.

Этап 0. На этом этапе производится рассылка начальных данных.

Этап 1. Находятся предварительные значения компонент магнитного поля BX, BY, BZ из уравнений

$$
BX^m_{i-\frac{1}{2},l,k} = BX^{m-\frac{1}{2}}_{i-\frac{1}{2},l,k} - \frac{c\tau}{2}\left(\frac{EZ^m_{i-\frac{1}{2},l+\frac{1}{2},k} - EZ^m_{i-\frac{1}{2},l-\frac{1}{2},k}}{h_y} - \right.
$$
$$
\left. - \frac{EY^m_{i-\frac{1}{2},l,k+\frac{1}{2}} - EY^m_{i-\frac{1}{2},l,k-\frac{1}{2}}}{h_z}\right),
$$

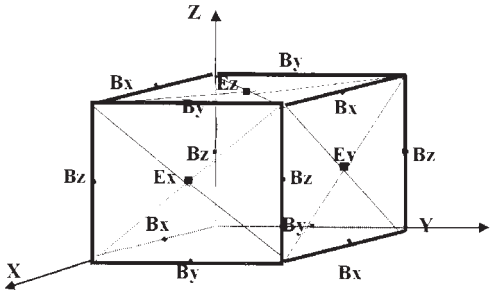


Рис. 1. Задание величин на сетке

$$BY^m_{i,l-\frac{1}{2},k} = BY^{m-\frac{1}{2}}_{i,l-\frac{1}{2},k} - \frac{c\tau}{2}\Big(\frac{EX^m_{i,l-\frac{1}{2},k+\frac{1}{2}} - EX^m_{i,l-\frac{1}{2},k-\frac{1}{2}}}{h_z} - \frac{EZ^m_{i+\frac{1}{2},l-\frac{1}{2},k} - EZ^m_{i-\frac{1}{2},l-\frac{1}{2},k}}{h_x}\Big),$$

$$BZ^m_{i,l,k-\frac{1}{2}} = BZ^{m-\frac{1}{2}}_{i,l,k-\frac{1}{2}} - \frac{c\tau}{2}\Big(\frac{EY^m_{i+\frac{1}{2},l,k-\frac{1}{2}} - EY^m_{i-\frac{1}{2},l,k-\frac{1}{2}}}{h_x} - \frac{EX^m_{i,l+\frac{1}{2},k-\frac{1}{2}} - EX^m_{i,l-\frac{1}{2},k-\frac{1}{2}}}{h_y}\Big).$$

Здесь нижние индексы i, j, k показывают положение соответствующей величины на пространственной сетке в направлениях (x, y, z) соответственно. Верхний индекс m указывает шаг по времени, h_x, h_y, h_z — пространственные шаги по осям x, y, z.

Этап 2. По найденным значениям магнитных полей и значениям электрического поля на предыдущем слое вычисляются новые скорости частиц:

$$VX_j^{m+\frac{1}{2}} = VX_j^{m-\frac{1}{2}} + \frac{\tau e}{m_i}\Big[EX_j^m + \frac{1}{c}\Big(\frac{VY_j^{m-\frac{1}{2}} + VY_j^{m+\frac{1}{2}}}{2} \times BZ_j^m - \frac{VZ_j^{m-\frac{1}{2}} + VZ_j^{m+\frac{1}{2}}}{2} \times BY_j^m\Big)\Big],$$

$$VY_j^{m+\frac{1}{2}} = VY_j^{m-\frac{1}{2}} + \frac{\tau e}{m_i}\Big[EY_j^m + \frac{1}{c}\Big(\frac{VZ_j^{m-\frac{1}{2}} + VZ_j^{m+\frac{1}{2}}}{2} \times BX_j^m - \frac{VX_j^{m-\frac{1}{2}} + VX_j^{m+\frac{1}{2}}}{2} \times BZ_j^m\Big)\Big],$$

$$VZ_j^{m+\frac{1}{2}} = VZ_j^{m-\frac{1}{2}} + \frac{\tau e}{m_i}\Big[EZ_j^m + \frac{1}{c}\Big(\frac{VX_j^{m-\frac{1}{2}} + VX_j^{m+\frac{1}{2}}}{2} \times BY_j^m - \frac{VY_j^{m-\frac{1}{2}} + VY_j^{m+\frac{1}{2}}}{2} \times BX_j^m\Big)\Big].$$

Нижний индекс j указывает номер частицы. Значения напряжен-

ностей электрического и магнитного полей определяются для каждой частицы отдельно путем интерполяции по восьми значениям в ближайших к частице узлах пространственной сетки. Поскольку все сеточные величины заданы на сдвинутых относительно друг друга сетках, то интерполирующие множители для каждой величины различны. Формулы для определения скоростей записаны неявно, в виде системы линейных алгебраических уравнений, которая решается аналитически, что дает явные выражения для компонент скорости на новом временном слое.

Этап 3. Данный этап содержит несколько подэтапов.

Подэтап 3.1. Определяются координаты частиц на слое $(m+\frac{1}{2})$:

$$\begin{aligned} X_j^{m+\frac{1}{2}} &= X_j^m + \frac{\tau}{2} V X_j^{m+\frac{1}{2}}, \\ Y_j^{m+\frac{1}{2}} &= Y_j^m + \frac{\tau}{2} V Y_j^{m+\frac{1}{2}}, \\ Z_j^{m+\frac{1}{2}} &= Z_j^m + \frac{\tau}{2} V Z_j^{m+\frac{1}{2}}. \end{aligned}$$

Подэтап 3.2. Плотности заряда вычисляются по формуле

$$N^{m+\frac{1}{2}}(x,y,z) = \sum_j M_j R(x - X_j^{m+\frac{1}{2}}) R(y - Y_j^{m+\frac{1}{2}}) R(z - Z_j^{m+\frac{1}{2}}),$$

где M_j — заряды отдельных частиц, R — ядро PIC-метода. Для равномерной сетки с шагом h ядро R имеет вид:

$$R(x) = \begin{cases} 1 - |x|/h, & |x| < h, \\ 0, & |x| > h. \end{cases}$$

Из этой формулы видно, что функция $R(x)$ отлична от нуля только на интервале длиной $2h$. Поэтому суммирование ведется только по частицам, расстояние которых до узла сетки не превышает шага h.

Подэтап 3.3. Находим средние скорости частиц на сетке. Вычисление компонент средней скорости ионов (UX, UY, UZ) можно записать с помощью следующих формул:

$$\begin{aligned} UX^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}} &= (N^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}})^{-1} \textstyle\sum_j M_j V X_j^{m+\frac{1}{2}} R(x_i - X_j^{m+\frac{1}{2}}) \times \\ &\qquad \times R(y_{l-\frac{1}{2}} - Y_j^{m+\frac{1}{2}}) R(z_{k-\frac{1}{2}} - Z_j^{m+\frac{1}{2}}), \\ UY^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}} &= (N^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}})^{-1} \textstyle\sum_j M_j V Y_j^{m+\frac{1}{2}} R(x_{i-\frac{1}{2}} - X_j^{m+\frac{1}{2}}) \times \\ &\qquad \times R(y_l - Y_j^{m+\frac{1}{2}}) R(z_{k-\frac{1}{2}} - Z_j^{m+\frac{1}{2}}), \\ UZ^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k} &= (N^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k})^{-1} \textstyle\sum_j M_j V Z_j^{m+\frac{1}{2}} R(x_{i-\frac{1}{2}} - X_j^{m+\frac{1}{2}}) \times \\ &\qquad \times R(y_{l-\frac{1}{2}} - Y_j^{m+\frac{1}{2}}) R(z_k - Z_j^{m+\frac{1}{2}}). \end{aligned}$$

Этап 4. Координаты частиц на слое $(m+1)$ определяются из следующих формул:

$$X_j^{m+1} = X_j^m + \frac{\tau}{2} V X_j^{m+\frac{1}{2}},$$
$$Y_j^{m+1} = Y_j^m + \frac{\tau}{2} V Y_j^{m+\frac{1}{2}},$$
$$Z_j^{m+1} = Z_j^m + \frac{\tau}{2} V Z_j^{m+\frac{1}{2}}.$$

Этап 5. Вычисляются окончательные значения компонент напряженности:

$$BX^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k} = BX^{m-\frac{1}{2}}_{i-\frac{1}{2},l,k} - \frac{c\tau}{2}\left(\frac{EZ^m_{i-\frac{1}{2},l+\frac{1}{2},k} - EZ^m_{i-\frac{1}{2},l-\frac{1}{2},k}}{h_y} - \frac{EY^m_{i-\frac{1}{2},l,k+\frac{1}{2}} - EY^m_{i-\frac{1}{2},l,k-\frac{1}{2}}}{h_z}\right),$$
$$BY^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k} = BY^{m-\frac{1}{2}}_{i,l-\frac{1}{2},k} - \frac{c\tau}{2}\left(\frac{EX^m_{i,l-\frac{1}{2},k+\frac{1}{2}} - EX^m_{i,l-\frac{1}{2},k-\frac{1}{2}}}{h_z} - \frac{EZ^m_{i+\frac{1}{2},l-\frac{1}{2},k} - EZ^m_{i-\frac{1}{2},l-\frac{1}{2},k}}{h_x}\right),$$
$$BZ^{m+\frac{1}{2}}_{i,l,k-\frac{1}{2}} = BZ^{m-\frac{1}{2}}_{i,l,k-\frac{1}{2}} - \frac{c\tau}{2}\left(\frac{EY^m_{i+\frac{1}{2},l,k-\frac{1}{2}} - EY^m_{i-\frac{1}{2},l,k-\frac{1}{2}}}{h_x} - \frac{EX^m_{i,l+\frac{1}{2},k-\frac{1}{2}} - EX^m_{i,l-\frac{1}{2},k-\frac{1}{2}}}{h_y}\right).$$

Этап 6. Вычисляются скорости электронов (VX, VY, VZ) :

$$VX^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}} = UX^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}} - (c/4\pi e N^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}}) \times \left(\frac{BZ^{m+\frac{1}{2}}_{i,l,k-\frac{1}{2}} - BZ^{m+\frac{1}{2}}_{i,l-1,k-\frac{1}{2}}}{h_y} - \frac{BY^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k} - BY^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-1}}{h_z}\right),$$
$$VY^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}} = UX^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}} - (c/4\pi e N^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}}) \times \left(\frac{BX^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k} - BX^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-1}}{h_z} - \frac{BZ^{m+\frac{1}{2}}_{i,l,k-\frac{1}{2}} - BZ^{m+\frac{1}{2}}_{i-1,l,k-\frac{1}{2}}}{h_x}\right),$$
$$VZ^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k} = UX^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k} - (c/4\pi e N^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k}) \times \left(\frac{BY^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k} - BY^{m+\frac{1}{2}}_{i-1,l-\frac{1}{2},k}}{h_x} - \frac{BX^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k} - BX^{m+\frac{1}{2}}_{i-\frac{1}{2},l-1,k}}{h_y}\right).$$

Этап 7. Находится электрическое поле:

$$
\begin{aligned}
EX^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}} &= \tfrac{1}{c}(BY^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}} VZ^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}} - BZ^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}} VY^{m+\frac{1}{2}}_{i,l-\frac{1}{2},k-\frac{1}{2}}),\\
EY^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}} &= \tfrac{1}{c}(BZ^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}} VX^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}} - BX^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}} VZ^{m+\frac{1}{2}}_{i-\frac{1}{2},l,k-\frac{1}{2}}),\\
EZ^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k} &= \tfrac{1}{c}(BX^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k} VY^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k} - BY^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k} VX^{m+\frac{1}{2}}_{i-\frac{1}{2},l-\frac{1}{2},k}).
\end{aligned}
$$

В этих формулах компоненты электрического и магнитного поля и скорости электронов в точках, где определяются электричские поля, находятся как среднеарифметическое значений в соседних двух (для поля) или четырех узлах (для скорости), в которых они определены, например:

$$
\begin{aligned}
BY_{i,l-\frac{1}{2},k-\frac{1}{2}} &= \tfrac{1}{2}(BY_{i,l-\frac{1}{2},k} + BY_{i,l-\frac{1}{2},k-1}),\\
VZ_{i,l-\frac{1}{2},k-\frac{1}{2}} &= \tfrac{1}{4}(VZ_{i-\frac{1}{2},l-\frac{1}{2},k} + VZ_{i-\frac{1}{2},l-\frac{1}{2},k-1} + VZ_{i+\frac{1}{2},l-\frac{1}{2},k} +\\
&\qquad + VZ_{i+\frac{1}{2},l-\frac{1}{2},k-1}).
\end{aligned}
$$

На этом цикл вычислений заканчивается.

Итак, исходная постановка задачи в семимерном пространстве путем использования метода частиц свелась к более сложной постановке, но в трехмерном пространстве. Решение уравнения Власова заменилось решением уравнений движения для большого количества частиц.

Частицы и поля, описывающие физическую часть задачи, в алгоритме представляются двумя вычисляемыми обьектами — множеством частиц и равномерной эйлеровой сеткой (узлами сетки являются вершины ячеек, на которые делится пространство моделирования). Каждая частица находится внутри определенной ячейки сетки.

2. ОТОБРАЖЕНИЕ АЛГОРИТМА НА ПАРАЛЛЕЛЬНУЮ ВЫЧИСЛИТЕЛЬНУЮ СИСТЕМУ

Предполагаем, что размер сетки равен $N = (n_1+1)\times(n_2+1)\times(n_3+1)$.

Считаем, что узлы сетки занумерованы от нуля до n_1, n_2, n_3 соответственно. Число частиц $M >> N$. Сформулируем требования на архитектуру ЭВМ, которая могла бы быть пригодна для реализации PIC-метода.

2.1. Считаем, что каждой ячейке сетки соответствует процессор, который вычисляет напряженность электричского поля и другие параметры, связанные с данной ячейкой. Предполагаем, что число процессоров

равно K и $n_1 \cdot n_2 \cdot n_3$ делится на K, т.е. $n_1 \cdot n_2 \cdot n_3 = l \cdot K = N^*$. Обозначим Π_i процессор с номером i, $0 \le i \le K-1$.

2.2. Ячейка характеризуется тройкой чисел $< i, j, k >$, соответствующих левому нижнему ее углу. Предполагаем, что зафиксирована нумерация ячеек, т.е. взаимно-однозначная функция

$$\nu : \Pi_{i=1}^{3}\{0, \ldots, n_i - 1\} \to \{0, \ldots, K-1\},$$

которая тройке $< i, j, k >$ сопоставляет $\nu_{ijk} = \nu(i, j, k)$ — номер процессора, о котором идет речь в п. 2.1, и для любого k верно $|(\nu)^{-1}(k)| = l$.

Процессор с номером ν_{ijk} рассчитывает величины

$$\begin{array}{ccc} BX_{i-\frac{1}{2},l,k}, & BY_{i,l-\frac{1}{2},k}\ , & BZ_{i,l,k-\frac{1}{2}}, \\ UX_{i,l-\frac{1}{2},k-\frac{1}{2}}, & UY_{i-\frac{1}{2},l,k-\frac{1}{2}}, & UZ_{i-\frac{1}{2},l-\frac{1}{2},k}, \\ VX_{i,l-\frac{1}{2},k-\frac{1}{2}}, & VY_{i-\frac{1}{2},l,k-\frac{1}{2}}, & VZ_{i-\frac{1}{2},l-\frac{1}{2},k}, \\ EX_{i,l-\frac{1}{2},k-\frac{1}{2}}, & EY_{i-\frac{1}{2},l,k-\frac{1}{2}}, & EZ_{i-\frac{1}{2},l-\frac{1}{2},k}, \\ NX_{i,l-\frac{1}{2},k-\frac{1}{2}}, & NY_{i-\frac{1}{2},l,k-\frac{1}{2}}, & NZ_{i-\frac{1}{2},l-\frac{1}{2},k}. \end{array}$$

При вычислении некоторых величин требуются данные из соседних ячеек.

2.3. Для простоты считаем, что M делится на K и $R = M/K$. Каждый процессор будет рассчитывать данные в точности для R частиц. Распределение частиц по процессорам считаем фиксированным, например: процессор с номером ноль обрабатывает частицы с номерами $1, \ldots, R$, второй — с номерами $R+1, \ldots, 2R$ и т.д. Обозначим $\pi(i)$ — множество номеров частиц, которые рассчитывает i-й процессор.

2.4. Обмен данными между процессорами производится через коммутатор, структуру которого не уточняем, но считаем, что если задана функция $\sigma : \{0, \ldots, K-1\} \to \{0, \ldots, K-1\}$, то коммутатор способен обеспечить связь всех процессоров Π_k с процессорами $\Pi_{\sigma(k)}, (0 \le k < K)$ одновременно в параллельном режиме, после чего произойдет передача данных. Структура алгоритма такова, что конфликты при записи исключены. Несущественно, происходят ли обмены между процессорами непосредственно или через общую память.

Архитектура ЭВМ, удовлетворяющая перечисленным выше условиям, показана на рис. 2.

При вычислении некоторых величин требуются данные из соседних ячеек, т.е. фактически из других процессоров.

Используя коммутатор, их можно получать в параллельном режиме.

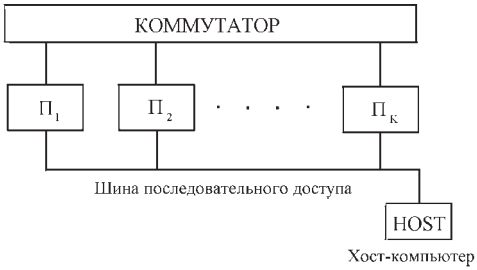


Рис. 2. Структура вычислительной системы

Поясним, как это делается. Допустим, что процессор $\Pi_{\nu(i,l,k)}$ вычисляет величину $BX_{i-\frac{1}{2},l,k}$. Можно считать, что он "знает" (т.е. в нем хранятся) значения $\nu(i+1,l,k), \nu(i,l+1,k), \nu(i,l,k+1)$.

Возникают отображения

$$\sigma_{100} : \nu(i,l,k) \to \nu(i+1,l,k),$$
$$\sigma_{010} : \nu(i,l,k) \to \nu(i,l+1,k),$$
$$\sigma_{001} : \nu(i,l,k) \to \nu(i,l,k+1).$$

Все процессоры Π_k, за исключением тех, которые ответственны за примыкающие к границе ячейки, через коммутатор могут одновременно подключиться к процессорам $\Pi_{\sigma_{i_1 i_2 i_3}(k)}$ и считать необходимые величины.

Например, подключившись к $\Pi_{\sigma_{010}(k)}$, можно считать величину $EZ_{i-\frac{1}{2},l+\frac{1}{2},k}$, необходимую для вычисления упомянутого выше $BX_{i-\frac{1}{2},l,k}$.

Трижды повторив подобные действия, все процессоры получат все необходимые для вычисления величины.

Опишем процесс обработки данных, касающихся частиц, в параллелльном режиме. Все процессоры одновременно производят сортировку частиц и предварительные вычисления.

Каждый процессор Π_k разбивает множество $\pi(k)$ на дизъюнктивные компоненты $\pi(k) = \bigcup_{\nu=1}^{N^*} \pi_\nu(k)$, где $\pi_\nu(k)$ — множество номеров частиц $\mu \in \pi(k)$, таких, что частица с номером μ находится в ячейке, имеющей номер ν.

Далее каждым процессором Π_k вычисляются величины

$$N_{\nu k}^{m+\frac{1}{2}}(x,y,z) = \sum_{j \in \pi_\nu(k)} M_j R(x - X_j^{m+\frac{1}{2}}) R(y - Y_j^{m+\frac{1}{2}}) R(z - Z_j^{m+\frac{1}{2}}).$$

При некоторых i, l, s имеем $\nu = \nu_{ils} = \nu(i,l,s)$. Величина $N_{\nu k}^{m+\frac{1}{2}}$ с точностью до коэффициента пропорциональности представляет собой часть суммы, которую необходимо вычислить, чтобы получить $N_{ils}^{m+\frac{1}{2}}$.

Легко видеть, что $N_{ils}^{m+\frac{1}{2}} = \sum_{k=0}^{N-1} N_{\nu k}^{m+\frac{1}{2}}$, т.е. другие части требуемой суммы находятся в других процессорах.

Далее требуется осуществить доступ к необходимым частичным суммам, что возможно сделать за $K-1$ подэтап.

Каждый процессор Π_k считывает у процессора $\Pi_{k'}$, $k' = k \oplus 1 (mod K)$, величину $N_{\nu k'}^{m+\frac{1}{2}}$.

Аналогично, Π_k считывает у процессора $\Pi_{k''}$, $k'' = k \oplus 2 (mod K)$, величину $N_{\nu k''}^{m+\frac{1}{2}}$.

Всего необходимо $K-1$ подэтап.

По завершению всех подэтапов обмена каждый из процессоров будет иметь все необходимые данные для вычисления величин $N_{ils}^{m+\frac{1}{2}}$.

На рис. 3 приведена схема, показывающая зависимость по данным в описаном выше алгоритме. На схеме видно, какие данные необходимы, для вычисления той или иной величины.

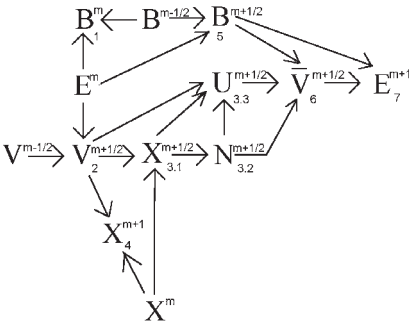


Рис. 3. Схема зависимостей по данным

Входными данными являются E^m, X^m и $B^{m-\frac{1}{2}}, V^{m-\frac{1}{2}}$. По ним вычисляются E^{m+1}, X^{m+1} и $B^{m+\frac{1}{2}}, V^{m+\frac{1}{2}}$ соответственно. Все остальные величины промежуточные. Вблизи каждой величины стоит номер этапа, на котором она вычисляется.

В приведенную ниже таблицу включены все используемые величины, а также показано, на каком временном шаге они используются.

$m-\frac{1}{2}$	m	$m+\frac{1}{2}$	$m+1$
B	B	B	
V		V	
	E		E
	X	X	X
		N	
		U	
		$\overline{V}$	

3. ОЦЕНКИ ВРЕМЕНИ ВЫПОЛНЕНИЯ АЛГОРИТМА

Оценим время выполнения алгоритма в параллельном и последовательном случаях и коэффициент ускорения, допустив ряд естественных предположений. Обозначим t_i — время вычислений на i-м этапе в параллельном режиме, T_i — время вычислений на i-м этапе в последовательном режиме, $0 \le i \le 7$, $\kappa_i = T_i/t_i$ — коэффициент ускорения.

Заметим, что при вычислении скорости частицы, компонент поля и т.д. употребляется всякий раз конечное число операций, определяемое конкретной формулой, что позволяет учесть вклад этих вычислений соответствующими константами.

Можно также пренебречь временами, необходимыми для обменов, ввиду того, что время срабатывания коммутатора в параллельной системе часто бывает меньше времени доступа к памяти последовательной машины.

Для соответствующих этапов имеем:

0.	$t_0 = T_0$.		
1.	$t_1 = c_1 l$,	$T_1 = c_1 N^*$,	$\kappa_1 = K$.
2.	$t_2 = c_2 R$,	$T_2 = c_2 M$,	$\kappa_2 = K$.
3.1	$t_{3.1} = c_{3.1} R$,	$T_{3.1} = c_{3.1} M$,	$\kappa_{3.1} = K$.
3.2	$t_{3.2.1} = c_{3.2.1} R$,	$T_{3.2.1} = c_{3.2.1} M$,	$\kappa_{3.2.1} = K$,
	$t_{3.2.2} = c_{3.2.2} N^*$,	$T_{3.2.2} = 0$.	
3.3	$t_{3.3.1} = c_{3.3.1} R$,	$T_{3.3.1} = c_{3.3.1} M$,	$\kappa_{3.3.1} = K$,
	$t_{3.3.2} = c_{3.3.2} N^*$,	$T_{3.3.2} = 0$.	
4.	$t_4 = c_4 R$,	$T_4 = c_4 M$,	$\kappa_4 = K$.
5.	$t_5 = c_5 l$,	$T_5 = c_5 N^*$,	$\kappa_5 = K$.
6.	$t_6 = c_6 l$,	$T_6 = c_6 N^*$,	$\kappa_6 = K$.
7.	$t_7 = c_7 l$,	$T_7 = c_7 N^*$,	$\kappa_7 = K$.

Заметим, что этапы 3.2.2 и 3.3.2 в последовательном случае отсутствуют. С другой стороны, они осуществимы на последовательной машине, только если все частицы помещаются в оперативной памяти.

Суммарное время равно соответственно:

$$\begin{aligned} t &= \sum_{i=1}^{7} t_i = \alpha l + \beta R + \gamma N^*, \\ \alpha &= c_1 + c_5 + c_6 + c_7, \\ \beta &= c_2 + c_{3.1} + c_{3.2.1} + c_{3.3.1} + c_4, \\ \gamma &= c_{3.2.2} + c_{3.3.2}, \\ T &= \alpha N^* + \beta M = (\alpha l + \beta R) K. \end{aligned}$$

Коэффициент ускорения:

$$\begin{aligned} \kappa &= \frac{\alpha N^* + \beta M}{\alpha l + \beta R + \gamma N^*} = \\ &= \frac{(\alpha l + \beta R) K}{\alpha l + \beta R + \gamma N^*} = \\ &= K(1 - \frac{\gamma l}{\alpha l + \beta R + \gamma N^*}) = \\ &= K(1 - \varepsilon(l, R, K)). \end{aligned}$$

Ниже на графиках показана зависимость коэффициента ускорения для конкретных случаев.

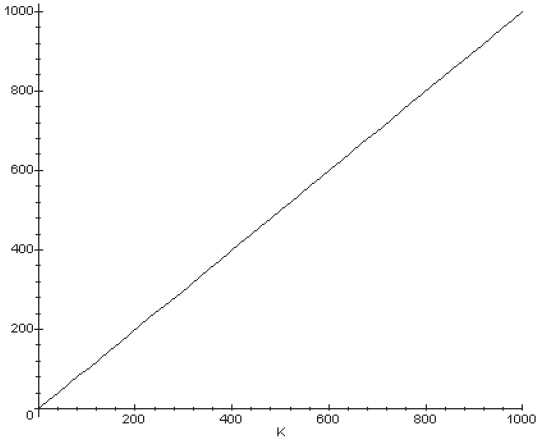


Рис. 4. График зависимости коэффициента ускорения κ от числа используемых процессоров K

Пример 1.
$\alpha = 20,\ \beta = 20,\ \gamma = 200,$
$N = K = 1, \ldots, 1000,$
$R = M/N = 100.$

В данном примере количество ячеек равно количеству процессоров и меняется от 1 до 1000. График зависимости коэффициента ускорения κ от числа используемых процессоров K показан на рис. 4.

Пример 2.
График на рис. 5 показывает зависимость коэффициента ускорения от количества ячеек N и количества частиц M, при фиксированном числе процессоров $K = 1000$. Остальные параметры остались прежними.

4. ЗАКЛЮЧЕНИЕ

Из предыдущего пункта видно, что имеет место практически линейная зависимость коэффициента ускорения от входного параметра

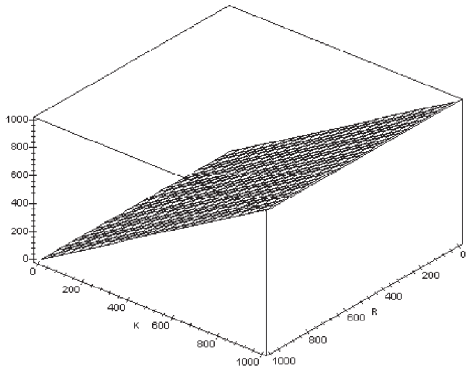


Рис. 5. График зависимости коэффициента ускорения κ от $R = M/N$ и числа процессоров K

задачи K. Это обусловлено тем, что в алгоритме отсутствуют этапы, на которых используются малоэффективные итерационные процессы, как, например, в работе [4].

Кроме того, коэффициент ускорения зависит также от констант α, β, γ, которые определяются сложностью вычислений, возникающих при расчете одной ячейки или одной частицы. В конечном итоге, эта сложность определяется сложностью соответствующих формул.

Отметим, что при увеличении α, β, γ эффективность вычислений, как правило, возрастает. С другой стороны, для большинства задач важных на практике, значения α, β, γ небольшие.

Можно выделить несколько возможных направлений дальнейших исследований.

Интересно было бы получить аналогичные оценки для других архитектур ЭВМ, например матричной и гиперкуба.

Большое практическое значение имеет исследование ситуации, когда данные, соответствующие частицам, не входят в суммарную локальную память процессоров. В этом случае приходится производить непрерывные обмены данными с внешними носителями.

Целесообразно развить исследования, учитывающие статистику распределения частиц. В этом случае следует говорить о математическом

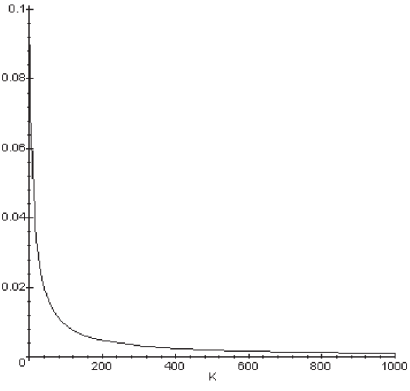


Рис. 6. График зависимости ε от числа процессоров K

ожидании времени исполнения алгоритма. При этом могут быть использованы различные методы теории информации, теории массового обслуживании, теории случайных блужданий и т.д.

Статистическая информация задается априорно или накапливается в процессе расчета. Такой подход может оказаться плодотворным для следующих случаев:

1) частицы сгруппированы в некоторой подобласти расчетной области;

2) частицы распределены более или менее равномерно, но потоки между ячейками небольшие;

3) скорости частиц небольшие.

СПИСОК ЛИТЕРАТУРЫ

1. **Хокни Р., Иствуд Дж.** Численное моделирование методом частиц. — М.: Мир, 1987. — 638 с.
2. **Малышкин В.Э., Вшивков В.А., Краева М.А.** О реализации метода частиц на мультипроцессорах. — Новосибирск, 1995. — 37 с. — (Препр. / СО РАН. ВЦ; N 1052).
3. **Christiansen J.P.** Numerical simulation of hydrodynamics by the method of point vortices // J. Comp. Phys. — 1973. — Vol. 13, N 3. — P.363–379.
4. **Поттер Д.** Вычислительные методы в физике. — М.: Мир, 1975. — 392 с.

И. В. Бурдонов, Ф. А. Мурзин

О РАСПАРАЛЛЕЛИВАНИИ МЕТОДА МЕДУЗА[1]

1. ВВЕДЕНИЕ

Для решения ряда задач математической физики используется методика МЕДУЗА [1]. Ее сущность состоит в том, что область исследования представляется в виде точек и окружающих их областей. На первом этапе область заполняется точками, а потом строится сетка таким образом, чтобы в каждой ее ячейке находилась ровно одна точка. Для вычисления величин в узле (центре ячейки) используются данные из вершин (многоугольника, образующего данную ячейку), вычисленные на предыдущем шаге, и наоборот.

В работе рассматривались различные вопросы, связанные с распараллеливанием методики МЕДУЗА на многопроцессорной вычислительной системе, использующей коммутатор. Были найдены способы распределения данных по процессорам и способы связи между ними.

Во второй части работы приводится краткое описание методики МЕДУЗА в применении к двумерным газодинамическим задачам.

Далее исследуется отображение метода на параллельную вычислительную машину. В конце работы приведены оценки времени выполнения алгоритма в параллельном и последовательном случаях и коэффициента ускорения.

2. МЕТОДИКА МЕДУЗА РАСЧЕТА ДВУМЕРНЫХ ГАЗОДИНАМИЧЕСКИХ ЗАДАЧ

2.1. Постановка дифференциальной задачи

Основой постановок задач, решаемых предложенной методикой в данной работе, является лагранжева система:

$$\begin{array}{lll} \frac{du}{dt} = -\sigma\frac{\partial p}{\partial x}, & \frac{dv}{dt} = -\sigma\frac{\partial p}{\partial y}, & \frac{dx}{dt} = u, \\ \frac{dy}{dt} = v, & \frac{dE}{dt} = -p\frac{d\sigma}{dt}, & \frac{d\sigma}{dt} = \sigma div(u,v). \end{array} \qquad (1.1)$$

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Здесь $\sigma = \frac{1}{\rho}$ — удельный объем. Система (1.1) замыкается уравнением состояния в форме

$$\mathrm{E} = \mathrm{E}(\mathrm{p}, \rho). \tag{1.2}$$

В начальный момент времени $t = t_0$ состояние среды в некоторой области G переменных (x, y) предполагается известным:

$$u = u_0(\mathrm{x}, \mathrm{y}), v = v_0(\mathrm{x}, \mathrm{y}), \rho = \rho_0(\mathrm{x}, \mathrm{y}), \mathrm{p} = \mathrm{p}_0(\mathrm{x}, \mathrm{y}). \tag{1.3}$$

На границе Γ области G задано какое-нибудь правильное граничное условие, корректно замыкающее задачу, например:

$$\mathrm{p}|_\Gamma = p(t, M), M \in \Gamma. \tag{1.4}$$

Требуется рассчитать решение эволюционной системы (1.1), (1.2) с условиями (1.3), (1.4).

В действительности рассчитываемая система от указанной отличается вязкой добавкой к давлению

$$q = -c\frac{d\sigma}{dt}|\frac{d\sigma}{dt}|$$

всюду, кроме (1.2). В дальнейшем об этом упоминаться не будет, и все выкладки будут проводиться с точностью до этого слагаемого.

2.2. Сетка

Конструирование разностной схемы связано с идеями Паста и Улама представления среды в виде глобул. Точнее, среда имеет представление в виде точек и окружающих их областей. Каждая точка несет следующую информацию:

X, Y — эйлеровы координаты;
u, v — компоненты скорости;
p — давление;
σ — удельный объем;
m — масса точки; (2.1)
q — вязкость;
N — номер уравнения состояния;
A_i — адреса “соседей” данной точки.

Указание “соседств” A_i обозначает, что в расчете используется сетка, представляющая собой граф с вершинами в лагранжевых точках и

ребрами, обозначающими “соседство”. В процессе расчета архитектура графа меняется в соответствии с метрическими отображениями о близости точек. В этом смысле методика может быть названа методикой на нерегулярной сетке.

Для указания необходимых свойств сетки введем отношение “соседства” $A \to$ B (A является “соседом” B). Элементарные свойства этого отношения состоят в следующем:

а) $A \to$ B $\Rightarrow B \to A$, т. е. всегда имеет смысл $A \Leftrightarrow$ B, что означает неориентированность графа, выражаемую предложением: если A является “соседом” B, то и B является “соседом” A.

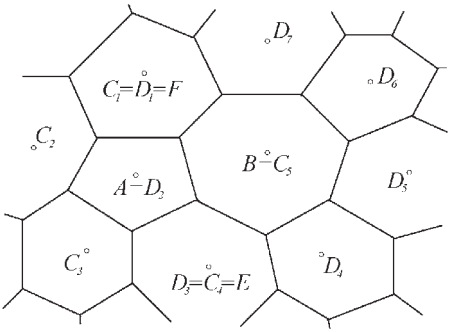


Рис. 1.

б) Если $C_j \leftarrow$ A, $D_k \leftarrow B, AB, j, k = 1..l > 3$, то найдутся две и только две точки E и F такие, что E $\in C_j, E \in D_k, F \in C_j, F \in D_k$, и F не является “соседом” E. Иными словами: если A и B — “соседи”, то в пересечении множества “соседей” A с множеством “соседей” B найдутся две точки E и F, не являющиеся “соседями” между собой (см. рис. 1).

Это свойство важно в дальнейшем для перестройки “соседств”, происходящей по стандартному алгоритму при числе “соседей” большем трех. Перестройка графа при числе “соседств” равном трем возможна, но при этом следует использовать другой алгоритм. Чтобы без необходимости не усложнять программу, лучше сконструировать сетку с числом “соседей” не меньшим четырех.

Каждой точке сеточного графа ставится в соответствие некоторый многоугольник, именуемый далее областью соответствия. Сама точка будет называться узлом, или центром, своей области соответствия, а вершины многоугольника — вершинами области соответствия.

При этом требуется выполнение следующих условий:

1) точки сеточного графа должны принадлежать к внутренним точкам своей области соответствия;

2) вся исследуемая область движения должна быть покрыта множеством областей соответствия;

3) вершины областей соответствия должны быть точками лагранжевой сетки, т. е. должны перемещаться вместе со средой;

4) области соответствия должны быть выпуклыми.

Очевидно, требованию 3) можно удовлетворить, полагая координаты вершин областей соответствия линейными комбинациями координат исходных точек, не зависящими от времени. В реализованном алгоритме эти веса выбраны по 1/3, но, вообще говоря, ими можно распорядиться по-иному для достижения определенных целей, например улучшения аппроксимации границ.

Резюме высказанных положений о сетке может быть сформулировано следующим образом: дискретизация движущегося континуума осуществляется точками, являющимися вершинами плоского неориентированного графа.

2.3. Дискретизация задачи

Расчет по методике МЕДУЗА начинается с расстановки точек A_i в области G. Хорошая расстановка точек помогает получать более аккуратную информацию о движении. Этот момент не алгоритмизирован и диктуется опытом и вкусами вычислителя. Из общих соображений можно высказать лишь следующее: точки, сближающиеся в расчете сильнее других, разумно расставлять реже, а расширяющиеся — гуще. В случае, когда информативность различных участков G должна быть различной, следует прибегнуть к соответствующей расстановке.

После расстановки точек производится установление “соседств”. В работе [1] отмечено, что начальные “соседства” могут быть получены, например, на основе разбиения G на области Дирихле, каждая из которых является континуальным односвязным подмножеством G, соотнесенным к A_i по принципу метрической близости. Иными словами, каждая точка $Q \in G$ и $Q \notin A_i$ относится к той точке A_i, к которой она расположена ближе, чем к остальным. Такое разбиение G позволяет однозначно установить “соседей” для каждой точки A_i, поскольку границами областей Дирихле являются линии, равноудаленные от соседних точек. Одним из наиболее сложных вопросов, связанных с ме-

тодом, является геометрическая задача построения данного разбиения (в геометрии — построение диаграммы Вороного множества точек).

В различных исследованиях по вычислительной геометрии приводятся некоторые способы решения этой задачи. В работе [2] приводится способ построения диаграммы Вороного методом "разделяй и властвуй". Графическая интерпретация данного метода состоит из нескольких шагов.

Шаг 1. Множество точек $\{A_i\}$ делится на два приблизительно равных подмножества $\{A_i^{'}\}$ и $\{A_i^{''}\}$. Для этого используется медиана по x-координате.

Шаг 2. Рекурсивно строятся диаграммы Вороного для $\{A_i^{'}\}$ и $\{A_i^{''}\}$.

Шаг 3.1. Строится разделяющая цепь — некоторая ломаная, разделяющая $\{A_i^{'}\}$ и $\{A_i^{''}\}$. Ее звенья — ребра областей соответствия, пересеченные медианой.

Шаг 3.2. Ребра $\{A_i^{'}\}$, которые оказались за границей, в области $\{A_i^{''}\}$ удаляются. Аналогично с $\{A_i^{''}\}$.

По указанному алгоритму строится диаграмма Вороного. В работе [2] приведены все формулы и подробное описание построения разделяющей цепи. В нашей программе не важна полученная картина, необходимо лишь определить "соседей" для каждого узла. Так как ребро является показателем "соседства", то из указанного выше алгоритма путем введения (обоюдного) в отношение "соседства" узлов при определении ребра, которое их разделяет, получаем алгоритм определения "соседей". Если к нему сделать небольшую добавку в виде сортировки, то в конце получим упорядоченный список "соседей". Аналогично с точками пересечения ребер. На многопроцессорной вычислительной системе данный алгоритм в параллельном режиме может реализоваться лишь частично (процессы разделения множеств). Построение разделяющих цепей возможно только при последовательном продвижении по координатам, поэтому приведенные здесь выдержки из алгоритма носят характер примера.

В настоящее время известны алгоритмы построения диаграммы Вороного на основе сортировки точек через введение лексикографического порядка. Алгоритмы сортировки на данный момент изучены довольно хорошо, многие из них можно распараллелить (в данной работе это не рассматривается).

Будем говорить, что три соседних между собой узла "образуют" вершину, если она принадлежит границам областей этих узлов. Иными

словами, вершина находится в треугольнике (а в нашей реализации — в точке пересечения серединных перпендикуляров к его сторонам), образованном этими узлами при триангуляции. Каждую из вершин необходимо в процессе работы алгоритма занумеровать и составить для каждой список номеров трех "образующих" узлов. При этом номеру вершины однозначно соответствует тройка номеров узлов. Будем считать, что задана функция, которая по трем номерам узлов выдает номер вершины, которую они образуют: $m(n_1, n_2, n_3)$. Эта функция понадобится в дальнейшем для связи.

Таким образом, строится начальный граф, удовлетворяющий требованиям пп. 1), 2), 4) для областей соответствия, поскольку области Дирихле обладают свойствами разбиения G на выпуклые полигоны. Использованию областей Дирихле в качестве областей соответствия в задачах гидродинамики препятствует п. 3) о лагранжевом описании вершин областей соответствия. Поэтому после установления "соседств" производится разбиение на области соответствия с последующим вычислением масс.

Если среди областей соответствия встречаются треугольники, то точки либо слегка смещаются для получения четырех "соседей", либо им приписывается четвертый "сосед" из числа "соседей соседей". Дискретизация гидродинамических величин по начальным данным производится далее очевидным образом. Таким образом, начальная информация (2.1) установлена для каждой внутренней точки. Специфику информации о граничных точках мы рассмотрим позже.

2.4. Расчет внутренних точек

Расчет заключается в последовательном переходе с одного временного слоя на последующий в соответствии с разностной аппроксимацией системы (1.1—1.2).

Одна из разновидностей аппроксимации заключается в разностном представлении объемных законов сохранения. Термин "объемные" подчеркивает то, что они относятся к телам вращения вокруг оси x, полученным из многоугольников соответствия. Приведем расчетные формулы для точки с номером n. Определим s_n как число "соседей" для узла с номером n.

Пусть X_n, Y_n обозначают координаты рассчитываемой точки, а X_i, Y_i, $1 \le i \le s_n$ — координаты ее "соседей". Обозначим координаты вершин многоугольников соответствия через x_j, y_j, $1 \le j \le s_n$.

Необходимо установить соответствие между номерами вершин для n-го узла и номерами соседей (список которых уже упорядочен). Предположим, что такое соответствие установлено и, например, узлы $i, i+1$, $1 \le i \le s_n - 1$ "образуют" i-ю вершину, а узлы s_n и 1 — вершину с номером s_n (см. рис. 2).

Тогда координаты вершин рассчитываются по формуле

$$x_j = \frac{X_n + X_j + X_{j+1}}{3}, y_j = \frac{Y_n + Y_j + Y_{j+1}}{3}. \tag{4.1}$$

Далее рассчитываются площади S_j треугольников, являющихся частью области соответствия (см. рис. 2):

$$S_j = \frac{1}{2}[(x_j - X_n)(\mathrm{y}_{j+1} - Y_n) - (x_{j+1} - X_n)(\mathrm{y}_j - Y_n)]. \tag{4.2}$$

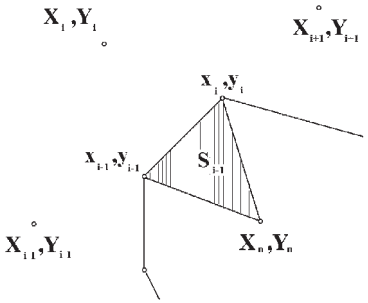


Рис. 2.

Затем рассчитываются объем тора многоугольного сечения, нормированный на единичный азимутальный угол и удельный объем:

$$V^n = \sum_{i=1}^{s_n} S_i \frac{Y_n + y_i + y_{i+1}}{3}; \tag{4.3}$$

$$\sigma^n = \frac{V_n}{m}. \tag{4.4}$$

Здесь и в дальнейших формулах нижний индекс означает соотнесение к нижнему временному слою, верхний — к верхнему.

Давление P^n вычисляется из следующей системы уравнений:

$$\begin{array}{l} E^n - E_n + \frac{P^n+P_n}{2}(\sigma^n - \sigma_n) = 0, \\ E^n = E(P^n, \sigma^n), \\ E_n = E(P_n, \sigma_n). \end{array} \tag{4.5}$$

Давление в вершинах многоугольника соответствия

$$p^i = \frac{P^n + P^i + P^{i+1}}{3}. \tag{4.6}$$

Линейно интерполируя давление с вершин на грани тора, получаем силу, действующую на него. Используя разностную форму уравнений Ньютона, получаем аппроксимацию уравнений Эйлера:

$$\frac{u^n - u_n}{\Delta t} = \frac{1}{6}\sum_{i=1}^{s_n}(y_i - y_{i+1})[(2y_i + y_{i+1})p^i + (y_i + 2y_{i+1})p^{i+1}]; \tag{4.7}$$

$$\frac{v^n - v_n}{\Delta t} = \frac{1}{6}\sum_{i=1}^{s_n}(x_{i+1} - x_{i-1})[(2y_{i-1} + y_{i+1})p^i + (y_i + 2y_{i+1})p^{i+1}] + \\ + \sum_{i=1}^{s_n} S_i \frac{P^n + p^i + p^{i+1}}{3}. \tag{4.8}$$

Отсюда, вычислив скорости u^n, v^n, получаем координаты сдвинутых точек:

$$\begin{array}{rcl} X^n & = & X_n + u^n \Delta t, \\ Y^n & = & Y_n + v^n \Delta t. \end{array} \tag{4.9}$$

Этим расчет временного шага внутренней точки заканчивается.

2.5. Особенности расчета граничных точек

Достаточно качественные способы расчета граничных точек для рассматриваемой методики не разработаны до сих пор. Проще рассчитываются задачи с заданной нормальной скоростью. Задание граничного давления может приводить к сильным немонотонностям в геометрии границ, поэтому в рассчитываемых ранее задачах приходилось использовать регуляризационные механизмы. В некоторых задачах этих явлений можно избежать, вставив дополнительные ряды фиктивных точек, что не всегда возможно.

Далее приводится одна из реализаций условий с давлением.

При начальном разбиения области G на области Дирихле может оказаться, что некоторые из них будут рассечены границей Г. Точки, которым соответствуют такие ячейки Дирихле, назовем граничными. Сам алгоритм определения граничных точек вписывается в алгоритм определения "соседей", поэтому к началу расчета полагаем, что разбиение на внутренние и граничные точки осуществлено.

Со стороны границы, противоположной рассматриваемой области G, расставляются дополнительные фиктивные точки Φ_n так, чтобы обычный алгоритм построения областей соответствия для всех точек, включая фиктивные, давал бы удовлетворительную аппроксимацию границы. В действительности фиктивные точки расставляются априорно и снабжаются признаком особого счета. Сами граничные точки при этом считаются в обычном режиме внутренних точек.

Каждая из фиктивных точек Φ_n несет следующую информацию: X_n^{Φ}, Y_n^{Φ} — координаты, Σq_i — вес, $\Sigma q_i u_i$, $\Sigma q_i v_i$ — взвешенные скорости, P_n — давление и номера двух фиктивных "соседей".

Если фиктивным точкам приписать скорости, близкие к скоростям граничных точек, то сжатие ячейки будет определяться, грубо говоря, ее половиной. В двумерном случае это приведет к чрезмерному заполнению границы внутренними точками. Только за счет этого давление в них может подняться до заданного в фиктивных точках. Избежать этих явлений можно, увеличив подвижность фиктивных точек. Реализация этого соображения осуществляется следующим образом.

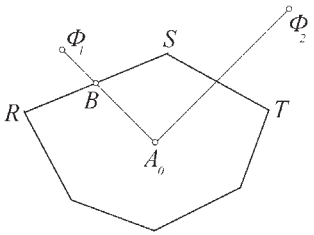


Рис. 3.

Масса граничной точки A_0 распределяется неким способом по границе ее области соответствия, но так, что $m_{RS} < m_0, m_{ST} < m_0$ (рис. 3).

Масса стенки RS приписывается точке B. Вычисляется сила, действующая на B по направлению $\Phi_1 A_0$, как разница давлений в точках

Φ_1 и A_0, помноженная на произведение площади конуса от вращения отрезка RS на косинус угла между этим отрезком и нормалью к отрезку $\Phi_1 A_0$ (проекцию площади RS на нормаль к $\Phi_1 A_0$). Полученные сила и масса позволяют вычислить ускорение точки B и приписать его точке Φ_1. Отсюда вычисляется составляющая скорости точки Φ_1 по направлению $A_0\Phi_1$. Касательная составляющая скорости берется та же, что и в точке A_0.

Если фиктивная точка обслуживает несколько граничных, то ее различные скорости взвешиваются с коэффициентами q_i типа угла зрения линии SR из Φ_1.

Чтобы предотвратить выскакивание точки A_0 из ее области соответствия, формула распределения масс сконструирована так, что $lim_{B\to A_0} m_{RS} = m_0$.

Осевые точки в отличие от граничных считаются неплохо. Основное отличие счета таких точек состоит в наличии естественной симметрии (рис. 4).

Координата x точки C полагается равной полусумме координат осевых соседей A и B. Так же вычисляется давление в данной точке. Это изменение вызвано потерей условия стыковки трех линий в вершине области соответствия из-за симметрии вращения. Второе отличие заключается в том, что возникающие в процессе счета отрицательные координаты y точки A заменяются положительными с одновременным изменением знака скорости v, т.е. происходит как бы обмен местами симметричных точек A и A'. В остальном расчет осевых точек идентичен расчету внутренних.

Таким образом, все граничные точки считаются в режиме внутренних точек, для этого им приписываются фиктивные соседи, которые расставляются априорно.

2.6. Об аппроксимации и устойчивости метода

Полностью доказательство аппроксимации системы (1.1) формулами (4.1)—(4.8) приведено в [1]. Приведем результат. Пусть L обозначает оператор левой части первого уравнения системы (1.1), а L^* — оператор левой части схемы (4.7), деленной на m. Тогда для счетного соотношения шагов порядка τ справедлива формула $(L - L^*)\{u, p, \sigma\} = O(\tau)$, т.е. в общем случае имеет место аппроксимация первого порядка. Аналогичный результат имеет место и для (4.8).

Основным вопросом, поставленным при изучении устойчивости дан-

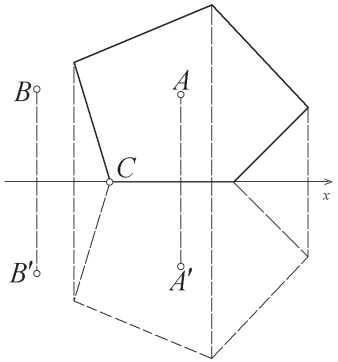


Рис. 4.

ной методики, был вопрос о том, как влияет на устойчивость приближние точки к границе области соответствия. Если бы это состояние привело к падению временного шага до 0, то расчеты по данной методике были бы значительно затруднены. В результате проведенных расчетов установлено, что близкое расположение счетной точки к границе области соответствия и выскакивание наружу слабо сказывается на устойчивости и формально не влияет на порядок аппроксимации уравнений. Таким образом, методика МЕДУЗА не обесценивается в описанных ситуациях, тем не менее их следует избегать из-за неконтролируемых потерь точности. Поэтому приближение точки к границе области соответствия приводит либо к изменению “соседства”, либо к пересадке точек вглубь области соответствия с последующими пересчетами некоторых величин. Иными словами, в конце временного шага сетка нуждается в коррекции. Соответствующие выкладки будут приведены в п. 3.3.

3. ОТОБРАЖЕНИЕ АЛГОРИТМА НА ПАРАЛЛЕЛЬНУЮ ВЫЧИСЛИТЕЛЬНУЮ СИСТЕМУ

3.1. Описание алгоритма параллельного вычисления и требования к вычислительной системе

Сформулируем требования к архитектуре ЭВМ, которая может быть пригодна для реализации метода МЕДУЗА.

Введем ряд обозначений.

Пусть $\{A_i\}_{1\le i\le I}$ — множество расчетных узлов (центров ячеек) и $\{B_j\}_{1\le j\le J}$ — множество вершин областей соответствия. Размерность этих множеств (величины I и J) связаны следующим образом. В работе [2] доказана теорема о соответствии диаграммы Вороного и триангуляции множества узлов и по ее непосредственному следствию. Диаграмма Вороного множества I точек имеет не более $2I-5$ вершин. Таким образом, имеем выражение для числа вершин J : $J=\xi I$, где $\xi=2-\frac{5}{I}\to 2$ с ростом I. В дальнейшем по умолчанию считаем, что $\xi=2$.

Предположим, что в распоряжении имеются K процессоров. Считаем, что процессоры занумерованы и обозначим их $\{\mathcal{P}_k\}_{1\le k\le K}$.

Для простоты считаем, что $\frac{I}{K}$ — целое число, и в последующих вычислениях будем считать распределение узлов по процессорам равномерным. Такое распределение узлов задается пользователем в явном виде. Полагаем, например, что $\mathcal{P}_k$ отвечает за узлы, имеющие номера $nK+k, 0\le n\le \frac{I}{K}-1$. В этом случае процессор $\mathcal{P}_1$ отвечает за узлы с номерами $1, K+1, ..., (\frac{I}{K}-1)K+1$, процессор $\mathcal{P}_2$ — за узлы с номерами $2, K+2, ..., (\frac{I}{K}-1)K+2$ и т.д. Когда мы говорим, что процессор отвечает за какой-то узел или какой-то узел содержится в процессоре, то подразумеваем, что в процессоре хранятся все характеристики данного узла. Таким образом имеем представление узла в виде записи, т.е. если A_i хранится в $\mathcal{P}_k$, то в $\mathcal{P}_k$ хранится запись $A_i.X, A_i.Y, A_i.P$ и т.д. В реальной ситуации эти величины могут иметь другой тип данных, но в данном алгоритме удобно представить их в виде записи.

Определим функцию $\nu : (i,K)\to(1,..,K)$ для нахождения номера процессора, хранящего узел A_i следующим образом: $\nu(i,K)=i-[\frac{i-1}{K}]\,K$, где квадратными скобками обозначена целая часть числа.

В результате имеем соответствие между начальным номером узла и его адресом (номером процессора). По одной нумерации однозначно находится другая.

Далее рассмотрим начало алгоритма вплоть до нумерации вершин.

Сначала происходит расстановка узлов (способом, указанным пользователем). Далее выполняется алгоритм построения диаграммы Вороного, “соседств” и вершин, в конце выполнения которого мы имеем следующую информацию: процессор $\mathcal{P}_k$ содержит записи $A_{nK+k}.X$, $A_{nK+k}.Y$, $A_{nK+k}.m_i$, $0\le n\le \frac{I}{K}-1$. Здесь $m_i, 1\le i\le s_{nK+k}$ — упорядоченный список номеров “соседей” для узла A_{nK+k}. А также имеем занумерованные вершины и списки “образующих” их узлов.

Далее рассмотрим распределение вершин по процессорам. Возникает вопрос о целесообразности данного распределения: не меньше ли будет затрачено всеми процессорами времени на простой пересчет координат для всех своих узлов, ведь при таком способе счета будет намного меньше обменов между процессорами? В принципе понятно, что при таком подходе число вычислений возрастает в три раза (что очень невыгодно с точки зрения экономии времени), так как для каждой вершины считать будут три ее ближайших смежных между собой узла. Детально данный вопрос рассмотрен в п. 3.4.

Заметим, что число вершин в общем случае не делится на K нацело, поэтому равномерного распределения не получится, но его характер будет таким же, как и при распределении узлов. Максимальное количество вершин в каждом процессоре равно $2\frac{I}{K}$. Таким образом, в процессоре $\mathcal{P}_k$ находятся вершины $mK + k, 0 \le m \le 2\frac{I}{K} - 1$. т.е. записи $B_{mK+k}.n_i, 0 \le m \le 2\frac{I}{K} - 1$. Здесь $n_i, 1 \le i \le 3$ — список номеров "образующих" узлов для вершины B_{mK+k}. Для нахождения номера процессора, хранящего вершину B_j, используется определенная функция ν.

Таким образом, распределение всех компонентов сетки по процессорам завершено.

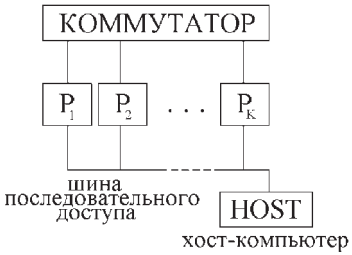


Рис. 5.

Обмен данными между процессорами производится через коммутатор, структуру которого не уточняем, но считаем, что если задана функция $\phi : \{1, .., K\} \to \{1, .., K\}$, то коммутатор способен связать процессоры $\mathcal{P}_k$ с процессорами $\mathcal{P}_{\phi(k)}$ одновременно в параллельном режиме. После коммутации происходит передача данных. Структура алгоритма должна быть такой, чтобы не возникали конфликты при записи. Несущественно, происходят ли обмены между процессорами непосредственно или через общую память.

Такая архитектура ЭВМ (см. рис. 5) ранее описывалась и использовалась в некоторых работах, например в [3].

Построим алгоритм расчета на данной вычислительной системе, реализуем его в несколько этапов.

На нулевом этапе рассматриваются распределение начальных данных и другие необходимые вопросы.

На этапах 1—4 производятся вычисления соответствующих формул из предыдущей части (4.1—4.4). Данные этапы, вместе с нулевым, исполняются только в начале программы. Таким образом, рассылка начальных данных, построение "соседств", вычисление координат вершин, площадей областей соответствия, объемов и удельных объемов производятся только один раз. Далее некоторые из заданных величин будут исправляться при коррекции сетки, но в целом их расчет закончен.

Далее считаются $iter$ итераций, по ходу которых последовательно исполняются этапы 5—9 по вычислению формул 4.5—4.9 и этапы 10—11, на которых производится коррекция сетки способами, рассмотренными в п. 3.3.

3.2. Этапы алгоритма параллельного вычисления

На **нулевом** этапе происходит задание начальных данных (1.3), (1.4).

1. На данной стадии расчета, кроме распределения узлов по процессорам, уже имеются координаты всех узлов и списки "соседей" для каждого узла.

На этом этапе предстоит вычислить координаты вершин областей соответствия $A_{nK+k}.x_j, A_{nK+k}.y_j$ для всех узлов по формуле (4.1).

Для этого необходимо всеми процессорами последовательно обработать списки своих вершин и вычислить $B_{mK+k}.x, B_{mK+k}.y$, затем переслать координаты в соответствующие процессоры.

Рассмотрим пошаговое вычисление координат $A_{nK+k}.x_j, A_{nK+k}.y_j$.

Шаг 1. Перед тем как посчитать координаты вершины $B_{mK+k}.x$, $B_{mK+k}.y$, необходимо за три этапа считать соответствующие координаты узлов $A_{m_1}.X, Y$ из $\mathcal{P}_{\nu(m_1)}$, $A_{m_2}.X, Y$ из $\mathcal{P}_{\nu(m_2)}$ и $A_{m_3}.X, Y$ из $\mathcal{P}_{\nu(m_3)}$. Процессор $\mathcal{P}_{\nu(mK+k)}$ может через коммутатор подключиться к процессору $\mathcal{P}_{\nu(m_1)}$ и считать $A_{m_1}.X, Y$ и т.д. Все процессоры делают это одновременно и синхронно.

Шаг 2. Происходит собственно вычисление координат вершины

B_{mK+k}. Все процессоры проводят вычисления синхронно, так как вычисляют одну и ту же формулу.

Шаги 1 и 2 выполняются, пока все процессоры не обработают все свои врешины. При равномерном распределении вершин все процессоры закончат работу одновременно. Таким образом, для подсчета координат всех вершин необходимо J/k последовательных повторений шагов 1 и 2.

Шаг 3. Процессор $\mathcal{P}_{\nu(nK+k)}$ начинает последовательное считывание величин $A_{nK+k}.x_j$, $A_{nK+k}.y_j$ посредством подключения через коммутатор к $\mathcal{P}_{\nu(m(n_j,n_{j+1},nK+k))}$, где n_j, n_{j+1} — номера соответствующих "соседей", $m(n_1, n_2, n_3)$ — функция связи.

Таким образом, после того как посчитаны координаты всех вершин, следует считывание данных. Все процессоры последовательно обрабатывают списки своих узлов. Для каждого считываются соответствующие координаты. Все процессоры делают описанные действия параллельно, на считывание координат одной вершины тратится один этап. Так как в этом случае имеем зависимость от числа всех вершин для всех узлов в процессоре, то общее время исполнения 1-го этапа будет зависеть от их наибольшего числа.

2. Вычисление площадей областей соответствия. На самом деле, как видно из формул, вычисляются только все составляющие площади ячейки (площади треугольников). При вычислении каждый процессор обрабатывает все свои узлы, используя при этом уже имеющиеся координаты вершин, по формулам (4.2). Таким образом считаются $A_{nK+k}.S_j$, где $1 \le j \le s_{nK+k}$. На данном этапе все процессоры работают параллельно. Время работы будет зависеть от времени вычисления процессором с максимальным числом всех вершин.

3. Нахождение объемов $A_{nK+k}.V^*$ производится в параллельном режиме, при этом используются уже посчитанные координаты вершин и площади частей ячеек. Каждый процессор обрабатывает все свои узлы, считая при этом одну формулу (4.3) для каждого из них. И хотя мы имеем равномерное распределение узлов, время исполнения данного этапа также будет зависеть от процессора с наибольшим числом вершин всех узлов, так как в формуле (4.3) фигурирует суммирование по всем вершинам.

4. Удельные объемы $A_{nK+k}.\sigma^*$ считаются параллельно и за сравнительно небольшое, одинаковое для всех узлов время. Формула (4.4).

На последующих этапах производятся итерационные вычисления. Сначала находятся значения гидродинамических величин, потом производится проверка новых координат на близость к границе и коррекция сетки в необходимых местах.

5. Вычисляем давление $A_{nK+k}.P^*$ в центре ячейки из неявной схемы. При этом заметим, что процесс нахождения давления не зависит от числа вершин в областях соответствия. Система (4.5). Сделаем допущение, что время вычисления независимо от его вида (будь то простая формула или некоторый прцесс, включающий в себя итерации и подстановки) и в последовательном, и в параллельном режиме для всех узлов одно и то же и является постоянным. Все процессоры на данном этапе работают параллельно.

6. На данном этапе необходимо посчитать давление $A_{nK+k}.p_j$ в каждой вершине области соответствия. Сами вычисления (расчетная формула (4.6)) и их порядок идентичны вычислениям на 1-м этапе (координаты). Сначала так же за три шага (для каждой вершины) считываются значения давлений в "образующих" узлах. Потом вычисляется давление в вершине и осуществляется переход к следующей вершине. После того как обработаны все вершины, начинается считывание данных (обрабатываются все узлы). Время исполнения на данном этапе будет аналогично времени исполнения 1-го этапа.

7. Находим скорость $A_{nK+k}.u^*$ по формуле (4.7). При этом все процессоры вычисления производят параллельно, в формуле имеется зависимость от числа вершин области соответствия для каждого узла. Время работы на данном этапе также будет зависеть от этого числа.

8. По формуле (4.8) находим скорость $A_{nK+k}.v^*$. Эта формула похожа на предыдущую, в ней также имеется подсчет суммы по всем вершинам.

9. Подсчет координат $A_{nK+k}.X^*, Y^*$ состоит из вычислений по одинаковым формулам (4.9) и производится на одном этапе. Все процессоры в параллельном режиме обрабатывают все свои узлы.

На этом расчет временного шага заканчивается. После вычисления новых координат необходимо произвести проверку каждого узла на близость к границе и откорректировать сетку.

3.3. Коррекция сетки

Далее необходимо разобрать ситуацию с коррекцией сетки. С определением узлов, центры масс в которых сместились близко к границе (проверкой на близость), и их пометкой все понятно — необходимо просто найти расстояния до каждой стороны области соответствия (для каждого узла) и сравнить их с некоторым параметром ϵ, который зависит от длины временного шага, скоростей точек и размеров сетки. Случаи, когда ϵ будет больше, чем найденное расстояние, необходимо пометить.

10. Процессор $\mathcal{P}_k$ параллельно обрабатывет все свои узлы, для каждого вычисляет расстояния до всех сторон границы области соответствия $A_{nK+k}.\rho_j$ и сравнивает их с параметром ϵ. При этом используются следующие (либо подобные) формулы (индексы опущены):

$\rho_j = |A_jX + B_jY + C_j|$, где X, Y — координаты центра ячейки;

$A_j = \frac{1}{x_{j+1}-x_j}$, $B_j = -\frac{1}{y_{j+1}-y_j}$;

$C_j = -\frac{x_j}{x_{j+1}-x_j} + \frac{y_j}{y_{j+1}-y_j}$.

После сравнения необходимо пометить узел. Для этого введем некоторый целочисленный признак $A_{nK+k}.\rho$: либо $\rho = 0$, тогда узел находится в обычном состоянии, либо $\rho = j$, где j — номер стороны границы. После переопределения ρ на очередном шаге будем иметь информацию о том, к какой именно стороне приблизилась точка (в некоторых способах корректировки эта информация необходима). Время исполнения на данных этапах будет зависеть от наибольшего количества вершин всех узлов в процессоре.

Самый простой с точки зрения формул для координат способ — пересадка точки вглубь области соответствия с последующими интерполяциями. Методы и формулы интерполяций могут быть самыми различными, и в данной работе их анализ не приводится. Время, которое будет затрачено на интерполяции, для всех узлов считаем одинаковым. Процессоры исполняют данный этап в параллельном режиме, последовательно обрабатывая списки узлов, помеченных ранее. Общее время работы зависит от количества помеченных узлов $I_{\text{п}} = bpI$, где величина bp определяет долю помеченных узлов. Чтобы не усложнять расчеты, будем считать, что шаг по времени выбран так, что bp сравнительно невелико по сравнению с 1. При таком подходе зависимость времени исполнения от распределения помеченных узлов по процессорам не сильная, приближенно будем считать, что они распределены равномерно.

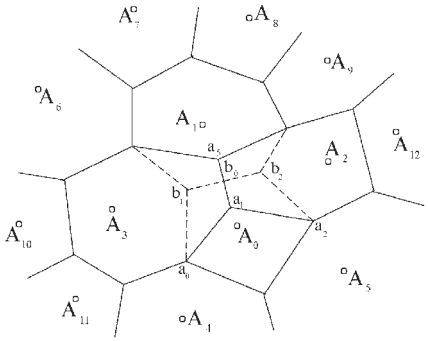


Рис. 6.

Далее приведем выдержки из способа корректировки "изменение соседства", предложенного авторами применения методики МЕДУЗА для решения газодинамических задач [1].

Изменение "соседства" предусмотрено структурой графа, которая может быть переменной. Техника реализации изменения структуры не очень сложна, поскольку новые "соседи" могут появиться лишь из числа "соседей соседей". Можно пользоваться лишь этим приемом, и тогда пересадка не потребуется. При перестройке "соседств" скачкообразно меняются площади областей соответствия, в связи с чем резко меняется плотность и другие гидродинамические параметры, что приводит к усилению пиков немонотонности. Для их уменьшения используются интерполяционные приемы, укладывающиеся в рамки законов сохранения. В целом в этом случае оценка влияния интерполяций на конечный результат расчета труднодоступна, но известно, что чем меньше произведено интерполяций на одну точку за расчет, тем надежнее результаты.

Техника изменения "соседств" заключается в следующем (см. рис. 6).

Пусть для точек A_0, A_1, A_2, A_3 указаны следующие соседства:

для $A_0 : A_2, A_3, A_4, A_5,$

для $A_1 : A_9, A_8, A_7, A_6, A_3, A_2,$

для $A_2 : A_0, A_5, A_{12}, A_9, A_1, A_3,$

для $A_3 : A_{11}, A_4, A_0, A_2, A_1, A_6, A_{10}.$

Пусть теперь точка A_0 находится близко к границе a_0a_1 и в то же время близко к вершине a_1.

По вершине a_1 определяются те "соседи" точки A_0, которые "образуют" данную вершину. В примере это точки A_2 и A_3. По свойству областей соответствия для точек A_2 и A_3 найдется "сосед" A_1 такой, что он не будет "соседом" A_0. Эта точка и вводится в "соседство" к A_0, а точки A_2 и A_3 из взаимного "соседства" выводятся. Исправленная таблица "соседств" теперь будет выглядеть так:

для $A_0 : A_2, A_1, A_3, A_4, A_5$,
для $A_1 : A_9, A_8, A_7, A_6, A_3, A_0, A_2$,
для $A_2 : A_0, A_5, A_{12}, A_9, A_1$,
для $A_3 : A_{11}, A_4, A_0, A_1, A_6, A_{10}$.

Перестройка свелась к тому, что в "соседство" A_0 введена A_1, в "соседство" A_1 введена A_0, из "соседства" A_2 выведена A_3, из "соседства" A_3 выведена A_2. При этом все свойства областей соответствия сохранились. Этот пример показывает, что граф "соседств" в процессе перестройки числа ребер не меняет.

Интерполяции, сопутствующие перестройке, состоят в следующем. Масса $a_0a_1b_0b_1$, рассчитанная по точке A_3, из массы этой точки удаляется, а к массе точки A_0 добавляется. То же самое относится и к количествам движения и энергиям. Аналогичные приемы применяются ко всем четырем перестроенным точкам.

Данную процедуру разумно проводить, когда исходная точка приближается к границе, но не к вершине, что означает попарное сближение точек. В таких случаях разумнее проводить раздвижку сблизившейся пары без изменения "соседств". Интерполяция остается вследствие изменения областей соответствия.

Указанные процедуры при расчете газов с различными свойствами приводят к необходимости рассчитывать ячейки смешанных составов и, в отличие от (2.1), хранить указания о разных составах смешанных ячеек.

Одним из слабых мест данной методики, как отмечено в [1], является недоказанность корректности механизма перестроек "соседств". Действительно, в указанном выше примере точка A_3 может приблизиться к границе a_1a_5 (и к вершине a_5) одновременно с A_0. И тогда после проведения указанной выше перестройки A_3 окажется за пределами своей области соответствия. Потребуется пересаживать точку. Из подобных затруднений авторы вышли путем снабжения узлов особым признаком, который принимал несколько целочисленных значений (характеризующих состояние узла). Изменение его значения для каждого узла зависе-

ло от значений этого признака у всех его "соседей" (а в ряде случаев и от состояния всех "соседей соседей"). Указанный способ не эффективен при вычислении на параллельной вычислительной системе, так как все узлы в этом случае должны обрабатываться последовательно несколько раз.

Напомним, что основа механизма перестройки "соседств" — свойство б) из п. 2.2. В работе [1] утверждается, что это свойство верно, если число "соседей" каждой точки не меньше четырех, доказательство данного предложения не приводится. Тем не менее, если использовать метод построения "соседств" на основе метрической близости, нужно добавить еще одно условие: никакие четыре точки, являющиеся "соседями", не лежат на одной окружности. Проверка данного положения трудна в алгоритмическом смысле и требует затрат большого количества ресурсов. В целом такое расположение точек не влияет на точность счета, если оно не является общим для всей области, например в случае задания целочисленных координат точек и т.д.

Таким образом, получаем, что при распараллеливании метода МЕДУЗА при коррекции сетки метод перестройки "соседств" (использующийся с самых ранних реализаций) неприемлем. Гораздо выгоднее использовать способ пересадки.

3.4. Оценка времени выполнения

Отметим, что в расчетах принимается оптимальное время для последовательного случая и приведенное (с округлением вверх) для параллельного. Так, например, в случае, когда требуется выполнить какие-либо (одинаковые) действия со всеми "соседями" (вершинами) для каждого узла, необходимо $\sum_{i=1}^{I} s_i$ таких действий. При этом затраченное время в последовательном режиме будет равно произведению этого числа на время исполнения данного действия. В параллельном случае в различных процессорах находятся узлы с различными числами s_i, поэтому некоторые (имеющие узлы с меньшим количеством вершин) процессоры завершат работу раньше и общее время исполнения будет зависеть от времени, затраченного процессорами с наибольшим количеством вершин всех узлов. Такая ситуация возникает при обычном распределении узлов по процессорам или, например, в ситуации, когда $I = K$. Таким образом, общее время равно произвдению времени исполнения одного действия на $max_k(\sum_{n=0}^{\frac{I}{K}-1} s_{nK+k})$, где s_j — количество вершин

j-го узла. Это число оценивается сверху, оно не больше чем $\frac{I}{K}s_{max}$, где $s_{max} = max_{i=1}^{I} s_i$ — максимальное число вершин для всех узлов.

Формулы и обозначения:

$iter$ — число итераций;

t_j — время исполнения на j-м этапе в параллельном режиме;

T_j — время исполнения на j-м этапе в последовательном режиме;

$t = \sum_{j=1}^{9} t_j, T = \sum_{j=1}^{9} T_j$ — общее время исполнения (соответственно);

$k_j = \frac{T_j}{t_j}$ — ускорение на j-м этапе;

$k = \frac{T}{t}$ — общее ускорение;

τ_k — время обмена (считывание) данными, при котором каждый процессор считывает k величин за одно соединение (в данной реализации $k = 1, 2$);

Σ — краткая запись $\sum_{i=1}^{I}$;

c_k — некоторые константы времени выполнения процессором соответствующих формул, зависящие только от времени выполнения процессором простейших математических действий, т.е. от свойств конкретного процессора.

На **нулевом** этапе производится рассылка начальных данных, построение "соседств", t_0 и T_0 — время, которое необходимо затратить на данный этап в параллельном и последовательном режимах соответственно.

1. Координаты вершин:
$t_1 = (2c_1 + 3\tau_2)J\frac{1}{K} + I\tau_2 s_{max}\frac{1}{K}$;
$T_1 = 2c_1 J$;
$k_1 = \frac{2c_1 JK}{(2c_1+3\tau_2)J+I\tau_2 s_{max}} =$
$= \frac{2c_1\xi}{(2c_1+3\tau_2)\xi+\tau_2 s_{max}}K.$

2. Площади частей области соответствия:
$t_2 = c_2 I s_{max}\frac{1}{K}$;
$T_2 = c_2\Sigma s_i$;
$k_2 = \frac{s_{mid}}{s_{max}}K.$

3. Объем:
$t_3 = c_{31} I s_{max}\frac{1}{K} + c_{32} I\frac{1}{K}$;
$T_3 = c_{31}\Sigma s_i + c_{32} I$;
$k_3 = \frac{c_{31}s_{mid}+c_{32}}{c_{31}s_{max}+c_{32}}K.$

4. Удельный объем:
$t_4 = c_4 I\frac{1}{K}$;

5. Давление в узлах неявной схемы:
$t_5 = c_5 I\frac{1}{K}$;

$T_4 = c_4 I;$
$k_4 = K.$

$T_5 = c_5 I;$
$k_5 = K.$

6. Давление в вершинах:
$t_6 = (c_6 + 3\tau_1) J \frac{1}{K} + I \tau_1 s_{max} \frac{1}{K};$
$T_6 = c_6 J;$
$k_6 = \frac{c_6 JK}{(c_6+3\tau_1)J + I\tau_1 s_{max}} =$
$= \frac{c_6 \xi}{(c_6+3\tau_1)\xi + \tau_1 s_{max}} K.$

7. Скорость u:
$t_7 = c_{71} I s_{max} \frac{1}{K} + c_{72} I \frac{1}{K};$
$T_7 = c_{71} \Sigma s_i + c_{72} I;$
$k_7 = \frac{c_{71} s_{mid} + c_{72}}{c_{71} s_{max} + c_{72}} K.$

8. Скорость v:
$t_8 = c_{81} I s_{max} \frac{1}{K} + c_{82} I \frac{1}{K};$
$T_8 = c_{81} \Sigma s_i + c_{82} I;$
$k_8 = \frac{c_{81} s_{mid} + c_{82}}{c_{81} s_{max} + c_{82}} K.$

9. Координаты:
$t_9 = 2 c_9 I \frac{1}{K};$
$T_9 = 2 c_9 I;$
$k_9 = K.$

10. Проверка на близость:
$t_{10} = c_{10} I s_{max} \frac{1}{K};$
$T_{10} = c_{10} \Sigma s_i;$
$k_{10} = \frac{s_{mid}}{s_{max}} K.$

11. Коррекция сетки (пересадка):
$t_{11} = 2 c_{11} bp\ I \frac{1}{K};$
$T_{11} = 2 c_{11} bp\ I;$
$k_{11} = K.$

Теперь рассмотрим вопрос о целесообразности распределения вершин по процессорам. Для его решения необходимо просуммировать время исполнения на этапах, где оно будет различаться для случев с распределением и без него. Далее нужно сравнить результаты (искать минимум).

Учитывая ранее указанные замечания, пренебрегаем временем, затраченным на начальное распределение вершин по узлам.

Время исполнения будет различным только на этапах, где требуется произвести вычисления в вершинах областей соответствия, — это 1-й и 6-й этапы алгоритма. Заметим, что на этих этапах расчетные формулы одни и те же с точностью до идентификаторов, поэтому константы времени вычисления c_i на этих этапах равны, таким образом, $c_1 = c_6 = c$.

Введем обозначения: время исполнения на соответствующих этапах в случае с распределением обозначим со штрихом, а без распределения — с двумя. Сравним величины $time_1 = t_1^{'} + t_6^{'}$ и $time_2 = t_1^{''} + t_6^{''}$:

$$\begin{aligned} time1 &= (2c_1 + 3\tau_2) J \tfrac{1}{K} + I\tau_2 s_{max} \tfrac{1}{K} + ((c_6 + 3\tau_1) J \tfrac{1}{K} + I\tau_1 s_{max} \tfrac{1}{K})\ iter = \\ &= (2c + 3\tau_2)\xi \tfrac{I}{K} + \tau_2 s_{max} \tfrac{I}{K} + (c + 3\tau_1)\xi\ iter \tfrac{I}{K} + \tau_1 s_{max}\ iter \tfrac{I}{K} = \\ &= [(\tau_2 + \tau_1 iter)(3\xi + s_{max}) + (2 + iter)\xi c] \tfrac{I}{K}; \end{aligned}$$

$$\begin{aligned} time2 &= (2c_1 + 2\tau_2) I s_{max} \tfrac{1}{K} + (c_6 + 2\tau_1) I s_{max} \tfrac{1}{K}\ iter = \\ &= (2c + 2\tau_2) s_{max} \tfrac{I}{K} + (c + 2\tau_1) s_{max}\ iter \tfrac{I}{K} = \\ &= [2(\tau_2 + \tau_1 iter) s_{max} + (2 + iter) s_{max} c] \tfrac{I}{K}. \end{aligned}$$

Рассмотрим знак разности $time_1 - time_2$:

$$\begin{aligned} sgn(time1 - time2) &= \\ &= sgn((\tau_2 + \tau_1 iter)(3\xi + s_{max}) + (2 + iter)\xi c - \\ &\quad -2(\tau_2 + \tau_1 iter)s_{max} + (2 + iter)s_{max}c) = \\ &= sgn((\tau_2 + \tau_1 iter)(3\xi - s_{max}) - (2 + iter)(s_{max} - \xi)c). \end{aligned}$$

Так как время τ_2 не может быть больше $2\tau_1$, то знакоопределяемое выражение не превосходит $(2 + iter)(3\xi - s_{max})\tau_1 - (2 + iter)(s_{max} - \xi)c$.

Тогда $sgn(time1 - time2) \le sgn((3\xi - s_{max})\tau_1 - (s_{max} - \xi)c)$.

При рассмотрении этого выражения видно: когда $3\xi \le s_{max}$, что наиболее вероятно, получаем отрицательное выражение (при этом $time_1 < time_2$), поэтому рассмотрим случай, когда $s_{max} < 3\xi$, т. е., принимая во внимание предыдущие оценки для ξ, допускаем, что $s_{max} < 6$ или $s_{max} \in \{4, 5\}$. В этом случае при делении на положительную величину — множитель при τ_1 получаем:

$$sgn(time1 - time2) = sgn(\tau_1 - \Omega\ c), \quad \Omega = (s_{max} - \xi)/(3\xi - s_{max}).$$

Покажем, что $\tau_1 < \Omega c$, т.е. докажем, что время исполнения меньше в варианте с распределением вершин. Необходимо оценить коэффициент Ω, который точно посчитать невозможно, так как он зависит от числа соседей для каждого узла. Заметим, что $\Omega(\xi)$ убывает с ростом ξ и достигает минимума при максимальном значении $\xi = 2$: $\Omega(2) = (s_{max} - 2)/(6 - s_{max})$. При подстановке различных значений s_{max} получаем, что $\Omega > (4 - 2)/(6 - 4) = 1$ в случае с одними четырехугольниками.

В реальных многопроцессорных системах время взаимодействия между процессорами соотносится со временем выполнения элементарных операций следующим образом [4,5]. Обозначим через τ_+ и τ_* время выполнения процессором сложения и умножения двух чисел соответственно. Тогда для известных систем имеют место соотношения: $2\tau_+ \le \tau_* \le 4\tau_+$ и $0,1\tau_+ \le \tau_1 \le \tau_+$. В нашем примере τ_1 сравнивается с константой c, которая не меньше чем $2\tau_+ + \tau_*$. Таким образом, с точки зрения затраченного времени выгоднее распределять вершины по процессорам.

Теперь рассмотрим коэффициент ускорения.

Сначала найдем t и T. Известно, что $c = c_1 = c_6$, введем обозначения:

$a = c_{71} + c_{81} + c_{10}$,

$b = c_5 + c_{72} + c_{82} + 2c_9 + 2c_{11}\ bp$,

$d = c_2 + c_{31}$,

$e = c_{32} + c_4$.

Тогда

$$\begin{aligned} t &= t_0 + (2c_1 + 3\tau_2)\xi\tfrac{I}{K} + \tau_2 s_{max}\tfrac{I}{K} + c_2 s_{max}\tfrac{I}{K} + c_{31}s_{max}\tfrac{I}{K} + c_{32}\tfrac{I}{K} + \\ &\quad + c_4\tfrac{I}{K} + [c_5\tfrac{I}{K} + (c_6 + 3\tau_1)\xi\tfrac{I}{K} + \tau_1 s_{max}\tfrac{I}{K} + c_{71}s_{max}\tfrac{I}{K} + c_{72}\tfrac{I}{K} + \\ &\quad + c_{81}s_{max}\tfrac{I}{K} + c_{82}\tfrac{I}{K} + 2c_9\tfrac{I}{K} + c_{10}s_{max}\tfrac{I}{K} + 2c_{11}\ bp\tfrac{I}{K}]\ iter = \\ &= t_0 + \tfrac{I}{K}[(\tau_2 + c_2 + c_{31})s_{max} + (\tau_1 + c_{71} + c_{81} + c_{10})s_{max}\ iter(c_5 + (c_6 + \\ &\quad + 3\tau_1)\xi + c_{72} + c_{82} + 2c_9 + 2c_{11}\ bp)\ iter + (2c_1 + 3\tau_2)\xi + c_{32} + c_4] = \\ &= t_0 + \tfrac{I}{K}[(\tau_2 + d)s_{max} + (\tau_1 + a)s_{max}\ iter + ((c + 3\tau_1)\xi + b)\ iter + \\ &\quad + (2c + 3\tau_2)\xi + e], \end{aligned}$$

$$\begin{aligned} T &= T_0 + 2c_1\xi\ I + c_2 s_{mid}\ I + c_{31}s_{mid}\ I + c_{32}\ I + c_4\ I + [c_5\ I + c_6\xi\ I + \\ &\quad + c_{71}s_{mid}\ I + c_{72}\ I + c_{81}s_{mid}\ I + c_{82}\ I + 2c_9\ I + c_{10}s_{mid}\ I + \\ &\quad + 2c_{11}\ bp\ I]\ iter = \\ &= T_0 + I[ds_{mid} + as_{mid}\ iter + (c\xi + b)\ iter + 2c\xi + e]. \end{aligned}$$

Далее введем ряд предложений. На нулевом этапе, при рассылке начальных данных, затраченное время одинаково и в параллельном, и в последовательном случае и сравнительно мало по сравнению со временем исполнения алгоритма на последующих этапах. Поэтому при вычислении ускорений пренебрегаем длительностью нулевого этапа, т. е. величинами t_0 и T_0.

Найдем выражения для коэффициента ускорения, для чего введем дополнительные обозначения :

$\alpha(iter) = d + iter\ a$,

$\beta(iter, \tau_1, \tau_2) = \tau_1 iter + \tau_2$,

$\gamma(iter) = e + iter\ b + iter\ c\xi + 2c\xi$.

Тогда при подстановке получим

$$\begin{aligned} k &= \frac{T}{t} = \\ &= K\ \frac{ds_{mid} + as_{mid}\ iter + (c\xi + b)\ iter + 2c\xi + e}{(\tau_2 + d)s_{max} + (\tau_1 + a)s_{max}\ iter + ((c + 3\tau_1)\xi + b)\ iter + (2c + 3\tau_2)\xi + e} = \\ &= K\ [\frac{\alpha s_{mid} + \gamma}{\alpha s_{max} + \beta s_{max} + 3\beta\xi + \gamma}] \end{aligned} \qquad (1)$$

Таким образом, получено выражение для коэффициента ускорения при распараллеливании методики МЕДУЗА на многопроцессорной машине указанной архитектуры. Интересен тот факт, что число точек I в него не входит.

Далее приведем приблизительную оценку для коэффициента ускорения.

Сразу заметим, что функция β относительно мала по сравнению с α и γ, так как в ней в качестве временных коэффициентов при $iter$ присутствует только τ_1, который мал по сравнению с a, b и c. Свободные члены в α и γ также превосходят τ_2 в несколько раз.

Величину s_{mid} можно оценить в среднем как $(s_{max} + s_{min})/2$. Такая оценка будет неплохой для большинства случаев при достаточном количестве точек. С другой стороны, $s_{min} = 4$ практически в любой расстановке точек. Подставим $s_{mid} = (s_{max} + 4)/2$ в (1):

$$k = K\ [(\frac{1}{2}\alpha(s_{max} + 4) + \gamma)/(\alpha s_{max} + \beta s_{max} + 3\beta\xi + \gamma)] =$$
$$= K\ [1 - \frac{1}{2}(\alpha s_{max} - 4\alpha + (2\beta s_{max} + 6\beta\xi))/(\alpha s_{max} + \gamma + (\beta s_{max} + 3\beta\xi))].$$

В этом выражении члены с относительно малой β сгруппированы отдельно. Для достижения более четкой оценки ими можно пренебречь. В таком случае

$$k \approx K\ [1 - \frac{1}{2}\frac{s_{max} - 4}{s_{max} + \frac{\gamma}{\alpha}}] = K\ [1 - \varepsilon(s_{max}, \frac{\gamma}{\alpha})].$$

4. ЗАКЛЮЧЕНИЕ

Несмотря на многочисленные трудности, которые возникают при создании программ расчета по методике МЕДУЗА, к настоящему времени уже созданы функционирующие программы. В частности в [1] проанализированы результаты работы и приведены сравнения с расчетами по другим методикам. Успешным является применение методики МЕДУЗА к решению различных задач ядерной физики, примером служат расчеты уравнений теплопроводности И.Д. Софронова [6].

Отметим, что предварительные расчеты времени работы параллельного алгоритма дали вполне удовлетворительные результаты. Полученные выражения для времени и коэффициента ускорения характеризуют сложность данной методики. Вместе с тем очевидны многие ее недостатки, такие, например, как нечеткая доказанность в отдельных случаях и отсутствие строгого и эффективного алгоритма. Все это влияет на создание полноценной функционирующей программы. В рамках работы в случае получения приемлемых результатов планируется создание программы, реализующей алгоритм. В настоящее время создана некоторая

ее часть, описаны структуры (на языке C++), которые должны использоваться в программе, рассмотрена реализация построения диаграммы Вороного, основанного на сортировке. Для получения каких-либо значимых результатов требуются большие усилия не только в программировании алгоритма, но и в доработке различных его частей.

В дальнейших теоретических исследованиях можно выделить два направления. Первое связано с разработкой эффективных алгоритмов для расчета вообще, включая поиск более четких путей (с доказательствами), второе — с распараллеливанием этих алгоритмов на различных архитектурах ЭВМ, например матричной и гиперкубе.

СПИСОК ЛИТЕРАТУРЫ

1. **Бабенко К.И.** Теоретические основы и конструирование численных алгоритмов задач математической физики. — М.: Наука, 1979. — 295 с.
2. **Препарата Ф., Шеймос М.** Вычислительная геометрия. — М.: Мир, 1989. — 478 с.
3. **Лобив И.В., Мурзин Ф.А.** О распараллеливании PIC-метода. //Проблемы систем информатики и программирования. — Новосибирск: ИСИ СО РАН, 1998. — С. 146—155.
4. **Системы** параллельной обработки: Сб. статей/ Под ред. Д. Ивенса. — М.: Мир, 1985. — 415 с.
5. **Высокоскоростные** вычисления: Сб. статей/ Под ред. Я. Ковалика. — М.: Радио и связь, 1988. — 385 с.
6. **Софронов И.Д.** О численном решении уравнений теплопроводности на неортогональных сетках// Тр. Моск. Междунар. математического конгресса, 1966 г., Секция 14. — М.: Изд. МГУ, 1966.

Ф.А. Мурзин, Д.Ф. Семич

ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ ТЕСТИРОВАНИЯ АЛГОРИТМОВ ПО ОБРАБОТКЕ ИЗОБРАЖЕНИЙ[1]

1. ВВЕДЕНИЕ

При создании прикладного программного обеспечения, ориентированного на обработку изображений, возникает необходимость в специальных программах, представляющих собой в некотором смысле “испытательные стенды”.

Такие программы должны иметь модульную структуру и развитый интерфейс, позволяющие легко подключать к ним новые программные блоки и отключать старые. Блоки, как правило, имеют конкретный содержательный смысл: фильтры, спектральные преобразования и т.д.

Реализованный в данном блоке алгоритм после испытаний на стенде и при установлении его полезности может быть включен в основные разрабатываемые программы в сильно измененном виде. Последнее обстоятельство связано с глубокой оптимизацией конечного программного продукта.

В данной работе кратко сообщается об алгоритмах, положенных в основу подобного “испытательного стенда”.

2. ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ ЭТАПЫ

2.1. Преобразование заданного в цветовом пространстве RGB изображения в систему координат YUV производится по формулам:

RGB −> YUV

$$Y = 0.114*B + 0.587*G + 0.299*R$$
$$U = 0.5000*B + 0.3313*G + 0.1687*R$$
$$V = 0.0813*B + 0.4187*G + 0.5000*R$$

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

YUV –> RGB

G = Y - 0.3441∗(U-128) - 0.7141∗(V-128)
B = Y + 1.772∗(U-128)
R = Y + 1.402∗(V-128)

Используются следующие приближения:

0.1145 = 3735/32768
0.2989 = 9798/32768
0.5866 = 19235/32768

0.5 = 16384/32768
0.3313 = 10855/32768
0.1687 = 5529/32768

0.0813 = 2264/32768
0.4187 = 13720/32768
0.5 = 16384/32768

1.772 = 14516/8192
1.402 = 11485/8192
0.3441 = 2819/8192
0.7141 = 5850/8192

2.2. Далее могут применяться различные дискретные интегральные преобразования: косинусное, Адамара, Хаара, Габора, Добеши, различные специфические вейвлет–преобразования типа 9/7 и другие. Например, дискретное косинусное преобразование (DCT) задается с помощью некоторой матрицы B и ее транспонированной матрицы B^T. Для блока размером 8×8 матрица B имеет вид:

$$B(i, j) = 0.5\,C \cos\{i\pi(2j+1)/16\},$$

где строки и столбцы имеют номера i и j соответственно, меняющиеся от 0 до 7, и

$$C = \begin{cases} 1/\sqrt{2} & \textit{if } i = 0, \\ 1 & \textit{otherwise}. \end{cases}$$

Заметим, что DCT является ортогональным преобразованием, и поэтому имеет место равенство $B^{-1} = B^T$.

Пусть f обозначает 8×8 блок изображения, и F — результат его преобразования. Блоки f и F могут быть получены один из другого посредством применения прямого (forward) (FDCT) и обратного (inverse) (IDCT) дискретных косинусных преобразований соответственно, которые задаются следующим образом

- $FDCT:\ F = BfB^T$,
- $IDCT:\ f = B^T FB$.

2.3. Квантование. На этом этапе вычисляется матрица квантования, в соответствии со следующим псевдокодом:

```
for(i=0;i<8;i++)
{
        for(j=0;j<8;j++)
        Q[i][j] = 1+((1+i+j)*q);
}
```

где q — коэффициент качества, от которого зависит степень потери качества сжатого изображения.
Для $q = 2$ имеем матрицу квантования:

$$Q = \begin{bmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{bmatrix}.$$

Квантование, описанное выше, называется грубым (coarse). В программе предусмотрено также тонкое (fine) квантование.

2.4. Один из интересных возникающих вопросов состоит в определении того, сколько коэффициентов, полученных в результате тех или иных преобразований, достаточно оставлять, чтобы не потерять качества изображения. Например, косинусное преобразование лучше применять к изображениям, заданным в координатах YUV. При этом для Y желательно сохранять все коэффициенты, лежащие в левом верхнем углу выше диагонали, и эле-

менты, лежащие на самой диагонали. Таким образом, получаем 36 коэффициентов. Остальные можно вычеркнуть, даже если они остаются после квантования, описанного выше. Для компонент U, V достаточно оставить по одному угловому коэффициенту. Всего получим 38 коэффициентов.

Преобразование Адамара лучше применять к изображениям, заданным в координатах RGB. При этом для каждой координаты достаточно оставлять 22 коэффициента, которые будут располагаться в левом верхнем углу в виде сектора круга. В итоге получим, что необходимо хранить $22 \times 3 = 66$ коэффициентов.

Квантование в этом случае применяется более простое, а именно: все числа делятся на одно и то же число, обычно на 4. Можно делить на 8 или 16, но при этом теряется качество.

С другой стороны, преобразование Адамара довольно хорошо себя ведет относительно разреживания исходного изображения и последующей интерполяции восстановленного изображения. Разреживание можно произвести по обеим пространственным координатам, т.е. в четыре раза. Учитывая тот факт, что коэффициентов необходимо хранить примерно в два раза больше, чем для косинусного преобразования, получаем, что разреживание дает выигрыш только в два раза, при этом происходит падение качества.

Таким образом, целесообразно не применять разреживание, а работать с косинусным преобразованием в координатах YUV.

2.5. Далее исследовался следующий метод компрессии компонент U, V. Рассмотрим дискретную функцию, характеризующую изменение спектрального коэффициента при движении вдоль блоков в горизонтальном направлении. Напомним, что для U, V оставляем только один коэффициент.

Данная функция определена в точках $0, \cdots, K = 720/8 - 1 = 89$ ввиду того, что исходное изображение имеет размер 720×480.

Аппроксимируем функцию ломаной линией по алгоритму, напоминающему алгоритм Брезенхема, известный в компьютерной графике. В отличие от классических интерполяционных задач, значения в конкретных точках аппроксимирующей функции могут отличаться от значений исходной дискретной функции, но на небольшую величину, которая задается и называется в программе Accuracy. С другой стороны, количество изломов должно быть как можно меньше (именно последнее обстоятельство важно для компрессии). Для уменьшения величин чисел будем запоминать расстояния между соседними точками, в которых наблюдаются изломы, и разности между соответствующими значениями аппроксимирующей ломаной.

2.6. На следующем этапе применяется LZW–метод кодирования. Такой алгоритм основан на динамически формируемом словаре, который состоит из различных строк. Строки заменяются кодами переменной длины.

В программе применяется один из алгоритмов, разработанных А.В. Кадачем*. Скорость компрессии этого метода — 20–25 Мбайт/с на Pentium–3 с тактовой частотой 500 Мгц, скорость декомпрессии — 80–120 Мбайт/с, степень компрессии — примерно такая же, как у PkZIP.

Описанная методика позволяет для U, V легко достигать степень компрессии в 350 раз.

2.7. В программе аналогично интерполируются старшие коэффициенты косинусного преобразования для компоненты Y. Предусмотрены три варианта: интерполируется один старший коэффициент, а также три или шесть коэффициентов.

2.8. Для высокочастотных коэффициентов компоненты Y такая методика не работает, так как данным коэффициентам соответствуют низкоамплитудные, но высокочастотные функции. Методика применима только к старшим коэффициентам.

2.9. По причине, указанной выше, в программе пока стоит “затычка”. В матрице коэффициентов размером 720×480 в каждом блоке размером 8×8 старшие коэффициенты зануляются: один, три или шесть соответственно (ранее они были уже проинтерполированы и закомпрессованы).

Полученная матрица младших коэффициентов компрессируется методом RLE. Были испробованы также другие методы: LZW, Huffman. Результаты были близкими и значительно зависящими от конкретного файла. Размер полученных файлов колебался от 1,6 до 30 Кбайт.

2.10. Более подробно опишем метод интерполяции (см. рисунок).

Для простоты будем считать, что задана непрерывная функция $f(x)$.

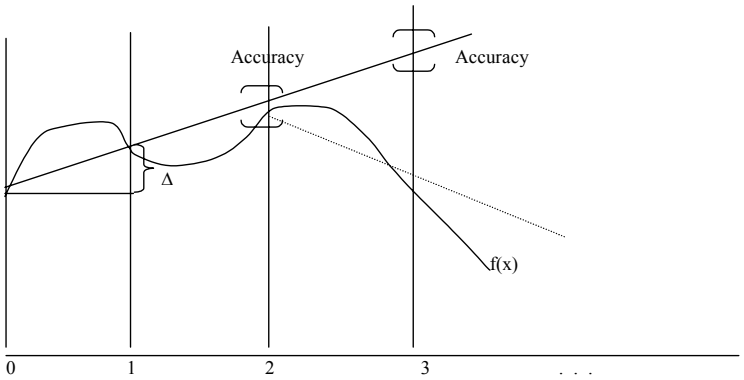
Вычисляем величину приращения $\Delta = f(1) - f(0)$. Соединим точки $f(0)$ и $f(1)$ отрезком прямой и продолжим данную прямую направо. Обозначим через $L(x)$ ее уравнение. Тогда легко видеть, что

$$L(2) = f(0) + 2\Delta = f(1) + \Delta,$$
$$L(3) = f(0) + 3\Delta = f(2) + \Delta.$$

Одновременно контролируем неравенство

* Кадач А. В. Эффективные алгоритмы неискажающего сжатия текстовой информации: Дис. канд. физ.-мат. наук: 05.13.11. — Новосибирск, 1997. — 186 с.

$$|f(k) - L(k)| \le \text{Accuracy}.$$



Величина Accuracy задает точность аппроксимации. Для компонент UV она может равняться 10 и более. Для Y ее желательно брать поменьше.

Таким образом будем продвигаться направо, пока не нарушится данное неравенство. В примере, показанном на рисунке, оно нарушается в точке $k = 3$. В этом случае отступим на шаг назад, запомним пару $(2, f(2))$, и алогичный процесс начнем сначала, т.е. будем искать новую Δ, и т.д. Продолжение процесса на рисунке изображено пунктирной линией.

Для хранения пар используются два различных массива: массив индексов и массив значений, которые дифференцируются, чтобы достичь дополнительной компрессии. Здесь, конечно, речь идет о разностной производной.

3. ЗАКЛЮЧЕНИЕ

Реализованная программа тестирования алгоритмов по обработке изображений показала свою полезность при разработке прикладного программного обеспечения. С помощью нее можно исследовать различные фильтры, алгоритмы скалирования, спектральные преобразования, варианты межкадрового сжатия для видео, методики вычисления векторов смещений и другие вопросы.

Программа реализована в системе Microsoft Visual C++ 6.0 с применением технологии MFC. Объем кода — 2.5 Мбайт.

В.А. Маркин

ЯЗЫК ОПИСАНИЯ ГРАФОВЫХ МОДЕЛЕЙ И АЛГОРИТМОВ GRAMAL[1]

ВВЕДЕНИЕ

При построении программных систем различных уровней сложности часто и широко используются графовые модели и различные методы их обработки. Идея использования графовых схем для языков спецификации или в языках высокого уровня обсуждается более 30 лет. Но только недавно начали бурно развиваться различные средства разработки, анализа и тестирования на базе систем переписывания графов. Многие считали, что моделирование графов и использование систем переписывания графов ведет в тупик из-за NP-сложности многих графовых алгоритмов. Ситуация кардинально изменилась с появлением систем переписывания графов или систем на базе графовых грамматик, таких как PROGRES [1], PAGG [2] ,GraphEd [3], или подобных рассмотренным в [4, 5].

Графы, являясь очень удобным инструментом описания структур данных, различных видов связей, информационных потоков, широко используются в теории компиляции, в различных математических задачах. Часто при описании алгоритма на графах требуется наглядно представить, увидеть пошаговые фазы алгоритма и т.д.

Именно для этих целей создана система GRAMAL, преследующая четыре цели: предоставить инструмент для описания графовых моделей; графически представить данную модель, предоставить средства тестирования и отладки методов работы с графами, а также возможность генерации программного кода для последующего применения.

В основе системы GRAMAL лежит разработанный язык, поддерживающий средства описания соответствующих графовых схем и средства описания преобразований на графах. Помимо этого, интегрированная среда разработки, написанная для использования одноименного языка GRAMAL, содержит в себе текстово-графический редактор языка, средства просмотра структур языка, а также средства интерпретации и генерации конечного

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

кода на языке С. Используя систему GRAMAL, можно получить прототип независимого приложения на языке С, который в дальнейшем может быть откомпилирован внешним компилятором с языка С. Тем самым спецификация конструируемой системы может быть исполнена или протестирована конечным пользователем без специальных знаний по теории графовых грамматик.

Далее будут даны описания языковых конструкций языка GRAMAL, определены формат описания предопределенных типов языка (вершин, дуг и графа), способы инициализации и методы манипулирования данными структурами. Будет также приведено краткое описание инструментальной системы GRAMAL с кратким же изложением режимов системы, ее целей и решаемых ею задач.

1. ОПИСАНИЕ ЯЗЫКОВЫХ КОНСТРУКЦИЙ ЯЗЫКА GRAMAL

Система GRAMAL предоставляет средства для представления различных атрибутированных графовых моделей (состоящих из атрибутированных вершин и дуг, как ориентированных, так и неориентированных). Атрибуты прописываются как для вершин, так и для дуг, что отличает GRAMAL от многих систем. В атрибутах хранится дополнительная информация, которая напрямую не относится к структуре графа, такая как, например, в таблице идентификаторов. Другими словами, атрибуты содержат информацию, локальную для объекта. Дополнительные связи между вершинами, помимо задания типов инцидентных ребер, можно задавать атрибутами соответствующих типов ребер. Как и во всех различных системах [1–5], связанных с системами описания и манипулирования графовыми моделями bkb системами переписывания графов, выделяются три основных объекта — вершина, ребро (либо дуга) и сам граф. Тем самым в языке GRAMAL определяются три основных класса: node_class, edge_class и graph_class. По умолчанию в системе определены три базовых типа: Node, Edge и Graph, от которых создаются все соответствующие производные типы. Эти базовые типы отвечают за инициализацию и базовые настройки (какие именно, укажем позднее) объектов соответствующих типов. В отношении всех основных классов язык GRAMAL поддерживает принцип наследования. Так, описав один из типов вершин, мы можем создать производный класс от данного типа, наследующий все атрибуты и методы базового типа.

Любой производный тип (тип вершины, ребра или графа) содержит описания атрибутов и методов работы с данным типом. Дадим, например краткое описание графовой модели Бинарного дерева с соответствующими методами работы с ними.

```
// Схема Бинарного дерева

Node_class Node{  // Вершина дерева
        Attrib:
                Key int num;
        Condition:
                Numeration(int n){this.num==n};
                Leave{(not with –R_Rebro-) and (not with -L_Rebro-)};
};

Node_class Root:Node{ //Отдельно выделен тип корень дерева
        Condition:
                Root{(not with -L_Rebro->) and (not with –R_Rebro->) };
};

Edge_class Rebro(node_class Node => node_class Node){ //Соответствующий
//тип ребра дерева
        Attrib:
                String Label;
};

Edge_class L_Rebro:Rebro{ //Левое ребро
        Condition:
                L_Rebro{(<-).num <(->).num};
};

Edge_class R_Rebro:Rebro{ //Правое ребро
        Condition:
                R_Rebro{(<-).num >(->).num};
};
```

```
graph_class BiTree{

        attrib:
        Node Nodes;          //Описание множества вершин
        Root Roots;//Описание корней
        L_Rebro L_Edges;   //Множество левосторонних дуг
        R_Rebro R_Edges;   //Множество правосторонних дуг

        methods:

                path Node Find(int n):Node{
                        ( L_Rebro(n) | R_Rebro(n) )
                        };
                rule Root Create (void)
                        { //Пустая часть инициализации
                        }=>{ //Вторая часть правила
                        Root uzel = new Root; //Создаем вершину типа Root
                        Return uzel; //Возвращаем значение новой вершины
                        };
                rule void Insert ( Node Uzel, int N)
                        {
                                Node tmp;
                                Set Uzel;
                                Tmp=Find(N);
                        }=>{
                                Node tmp2=new Node;
                                Tmp2.num=N;
                                If (tmp.num>N)
                                        New  L_Rebro(tmp,tmp2);
                                Else
                                        New  R_Rebro(tmp,tmp2)
                        }

};
```

Описанная выше схема бинарного дерева практически в полной мере описывает многие аспекты языка GRAMAL.

Как видно из схемы, GRAMAL содержит синтаксические конструкции для определения типизированной графовой схемы. В такой схеме описаны соответствующие типы вершин и дуг, возможности и условия возникающих связей между ними. Раздел TYPES_NODES содержит объявления типов вершин (метки и атрибуты вершин, состоящие из имен и областей видимости). Раздел TYPES_EDGES определяет метки ребер и, что важно, типы вершин, которые данный тип ребер соединяет.

Ниже приведем пример схемы, описывающей систему таблиц при компиляции программ:

```
Section  CLASS_NODES
{
    node_class GRAPH_NODE;
    node_class MODULE: GRAPH_NODE;
    node_class LIST_HEAD: GRAPH_NODE;
}
Section TYPES_NODES
{
    typedef node_type Ident GRAPH_NODE;
    typedef node_type Function_Module MODULE;
    typedef node_type Data_Type_Module MODULE;
    typedef node_type Data_Object_Module MODULE;
    typedef node_type Ident_List LIST_HEAD;
    typedef node_type Proc_List LIST_HEAD;
    typedef node_type Implementation GRPH_NODE;
}

Section TYPES_EDGES
{
    typedef edge_type ToIdent: node_class MODULE => node_class Ident;
    typedef edge_type ToBaseOn: node_class MODULE => node_class
Ident_List;
    typedef edge_type  ToExport: node_class MODULE => node_class Proc_List;
    typedef edge_type  ToContains: node_class MODULE => node_class
Ident_List;
    typedef edge_type ToImport: node_class MODULE => node_class Ident_List;
    typedef edge_type  ToImplementation: node_class MODULE => node_class
Implemetation;
}
```

Разделы TYPES_NODES и TYPES_EDGES необходимы для более упрощенного задания формализма при описании графовой схемы. Очень часто встречается, что при определении графовой схемы в общем случае определяется большое число типов вершин и ребер, отличающихся друг от друга частичными параметрами. Поэтому систему GRAMAL отличает от других возможность задания базовых и порожденных типов объектов, что делает описываемую схему более компактной и легко определяемой. Так, например, во время трансляции можно получить несколько типов таблиц (Function_Module, Data_Type_Module, Data_Object_Module). В нашем примере все они наследуются от типа MODULE, чем мы уменьшили описание допустимых типов ребер в разделе TYPES_EDGES, потому что такие типы ребер, как ToIdent, ToBasedOn, ToExport и т.д., для всех описанных типов модулей определили один раз вместо трех.

Как и в объектно-ориентированных языках GRAMAL, предоставляет возможность строить иерархию типов вершин и ребер.

Наследование позволяет уменьшить размер описания графовой схемы, по сравнению с тем, как это происходило бы при использовании только базовых типов вершин и ребер.

Помимо этого наследование позволяет выделить общие атрибуты объектов. Так, например, если мы определим две таблицы — Типов и Идентификаторов — как производные от типа LIST_ELEM, то общим у них будет атрибут — имя объекта NAME, который мы и можем приписать к множеству атрибутов типа LIST_ELEM.

И наконец, в GRAMALe можно определить область определения атрибута. Допускаются два возможности: 1) user тип — атрибут определяется пользователем, его можно инициализировать и задавать значение; 2) derived тип — атрибут задается или вычисляется системой по определенным правилам, заданным при описании типа объекта. Примером первого типа атрибутов может служить Наименование Идентификатора NAME_Ident в заданной таблице идентификаторов, а второй — заданная на графе разметка.

```
Section CLASS_NODES
{
        node_class IDENT:GRAPH_NODE
        {
                user Name: string="";
        }
…….
```

```
        node_class LIST_ELEM:GRAPH_NODE
        {
                derived Num: int = [<=ToNext=.Num +1 | 1];
        }
}
```

Из приведенной в начале главы схемы бинарного дерева видно, что описание любого из видов предопределенных классов состоит из трех частей: раздела атрибутов, раздела условий (правил специального вида) и раздела методов (процедур, связанных с данным классом).

В разделе атрибутов описываются обычные информационные поля, имеющие предопределенные типы данных с дополнительным ключом Key. Данный ключ просто указывает системе, что при создании определенного объекта надо явно указать значение данного поля. При создании, скажем, вершины во множестве вершин данного класса система должна проверить уникальность данного идентификатора (используя терминологию баз данных). Данное поле может быть только одно, а при его отсутствии считается, что существует целочисленный нумератор (еще раз напомним, что производные классы наследуют атрибуты базового класса).

В разделе условий описываются специальные правила, которые, в зависимости от значения условного выражения, меняют состояние (контекст) класса.

Под состоянием класса мы подразумеваем:

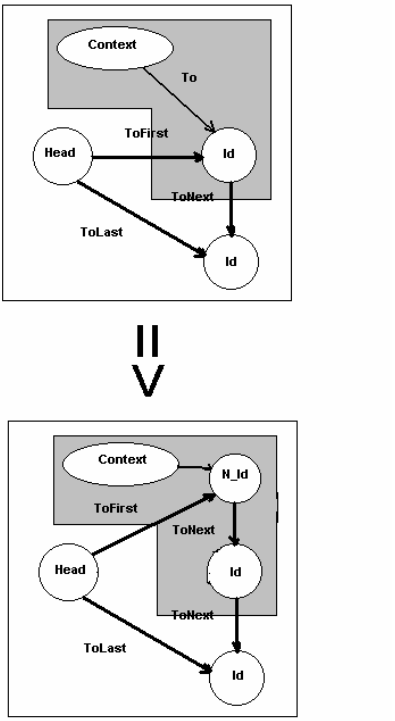
- для типов вершин — указатель на данную вершину;
- для типов дуг — указатели на смежные вершины и указатель на текущую вершину;
- для графов — совокупность состояний всех типов вершин и дуг.

В описанном выше примере два условия L_Rebro и R_Rebro на входе получают целочисленное значение, и если условие верно, то меняют контекст и возвращают указатель на смежную вершину.

В разделе методов, существующих только для графовых классов, описываются правила преобразования графа, где под преобразованием будем понимать изменение контекстов и множеств вершин либо дуг. Все методы работы с графом разделим на три вида: path (изменение контекста), rule (правило преобразования графа), и transform (более сложное преобразование, состоящее из совокупности правил (rules)).

Для методов типа *path* необходимо указать два контекста: входной контекст входной вершины и контекст выходной вершины. Если при вызове данного метода на входе указать явно объект данного типа вершины, то контекст, связанный с данным типом вершин, меняется на входной объект. В приведенном примере при описании метода Find встречаем альтернативный оператор, где перебираются все выполнимые альтернативы.

Основными методами являются правила, методы типа *rule*, каждое из которых состоит из двух частей: выделение подграфа для преобразования и само преобразование. Если первая часть — выделение подграфа для преобразования — не выполняется, то правило отклоняется. В одном из методов используется оператор Set, который устанавливает соответствующий контекст в системе. Графически правило преобразования rule на примере вставки элемента в список приведено на рисунке.



Последний вид методов — *transform* (преобразования) — является совокупностью правил либо преобразований.

Любая программа на языке GRAMAL состоит из последовательности операторов описания типов, объектов и процедур. В языке, помимо описанных выше специализированных типов, переменные могут быть стандартными типами: int, char, float, string и т.д.

Входом в точку программы является процедура *main*.

2. ФУНКЦИОНАЛЬНЫЕ ОСОБЕННОСТИ СИСТЕМЫ GRAMAL

Помимо задачи описания грамматических особенностей специализированного языка вторым этапом работы над данным проектом являлось создание инструментальной системы GRAMAL.

Основными целями данной системы являются средства отладки и визуализации входных задач, написанных на языке GRAMAL, а также генерация программного кода (например, на языке C) для дальнейшего использования наработанной библиотеки в других программных проектах.

Система, таким образом, находится в одном из трех состояний: отладки, трассировки (либо визуализации) и генератора. Соответственно среда GRAMAL состоит из трех основных частей: редактора, интерпретатора и back-end на языке C.

В режиме отладки пользователь находится в окне редактирования, редактируя программный код. По возможности пользователю предоставляется графический контекстный интерфейс с выделением специализированных конструкций языка. Используя средства графического представления графовых объектов, пользователь может быстро создать прототип графовой модели своей задачи либо описать входные данные для нее. Спецификация в системе GRAMAL создается в две стадии. Первая стадия описывает саму графовую схему, а во второй создаются все операции над этой схемой. Виды данных операций можно найти в описании грамматики языка. Типы узлов в описании графовой схемы могут рассматриваться как абстрактные классы в объектно-ориентированных языках, которые включают в себя необходимые атрибуты данного типа узла. Все типы узлов обозначаются прямоугольниками. Если между типами вершин существуют связи наследования, то они указываются жирными стрелками. Для обозначения связей между вершинами или ребрами используются одинарные стрелки, двойной щелчок на которых вызывает дополнительное окно, содержащее атрибуты данного ребра. Построив схему, пользователь переходит к построению пра-

вил преобразований над ней. Каждое такое правило состоит из двух частей: левой и правой (смотри описание грамматики). Для данной задачи пользователь вызывает дополнительное окно, в котором можно описать данное правило как графически, так и через текстовое представление. Нужно отметить, что в любой момент, пользователь для любого объекта схемы может получить back-end на языке C.

По окончании редактирования пользователь должен оттранслировать программу, получив необходимую информацию об ошибках, для дальнейшей отладки.

Как только программа будет отлажена, пользователь может перейти в режим интерпретации программы для анализа запрограммированной графовой задачи (описанных графовых моделей и методов). Данный режим называется еще режимом визуализации, в котором пользователю должен быть предоставлен графический вид графовой задачи с соответствующими методами печати, просмотра и сохранения в графических форматах. При интерпретации программы пользователь с помощью расставленных брейкпойнтов либо пошаговой отладки вызывает дополнительные окна, в которых показано текущее состояние схемы, а цветом указаны текущие контексты вершин и ребер.

И последний режим генератора переводит программу на языке GRAMAL в конечный язык программирования C, где описаны соответствующие структуры и методы, а точкой входа является процедура GRAMAL. В дальнейшим пользователь может использовать сгенерированную процедуру в иных программных проектах.

ЗАКЛЮЧЕНИЕ

Целью системы GRAMAL является достижение простоты в реализации и анализе различных алгоритмов на графовых моделях. С помощью данной системы пользователь может быстро и без каких-либо дополнительных знаний в программировании создать прототип своей задачи, если она описывается графовыми схемами. При необходимости пользователь сможет получить прототип своей задачи, написанный на языке C++, для дальнейшего использования в каких-либо других внешних программных системах.

Развитие системы рассматривается в двух направлениях. Первое включает введение в систему дополнительных функциональных инструментов, таких как анализатор корректности правил трансформации схемы, создание библиотек графовых схем, экспорт в язык Java. Помимо этого рассматрива-

ется исследование по явному описанию и внедрению в систему средств объектно-ориентированного программирования. Второе направление включает реализацию многоплатформенности инструментальной системы, т.е. написание графических компонент системы под Windows и Linux.

СПИСОК ЛИТЕРАТУРЫ

1. **Engels G. et al.** Building integrated software development environments. Part 1: Tool specification // ACM Trans. on Software Engineering and Methodology. — 1992. — Vol. 1, N 2. — P. 135–167.
2. **Gottler H., Gunther J., Nieskens. G.** Use graph grammars to design CAD-systems // Lect. Notes Comput. Sci. — 1991. — Vol. 532. — P. 396–410.
3. **Himsolt M.** GraphEd: An interactive graph editor // Lect. Notes Comput. Sci. — 1989. — Vol. 349. — P. 532–533.
4. **Nagl M., Schurr A., Munch M.** Applications of graph transformations with industrial relevance (International Workshop, AGTIVE 99) // Lect. Notes Comput. Sci. — Vol. 1779.
5. **Theory** and application of graph transformations / Ed. By Ehrig H. et al. // Proc. 6th Internat. Workshop TAGT 98. — Berlin a.o.: Springer-Verlag, 1998. — (Lect. Notes Comput. Sci.; Vol. 1764).

Т.А. Волянская, Ю.В. Малинина

ТРАНСФОРМ: ИНТЕРФЕЙС ДЛЯ ВВОДА ИНФОРМАЦИИ[1]

ВВЕДЕНИЕ

Информационная система (ИС) ТРАНСФОРМ, создаваемая в ИСИ СО РАН при финансовой поддержке РФФИ и Минобразования, предназначена для накопления и систематизации информации по преобразованиям программ и ориентирована на работу в среде Интернет. Основой ИС является база данных, содержащая описания публикаций по преобразованиям программ и сами описания преобразований программ.

Информационная система ТРАНСФОРМ представляет собой информационно-поисковую систему; источник данных в информационной системе — публикации по преобразованиям программ (статьи, монографии, отчеты, диссертации); для реализации используется СУБД POSTGRES95; режим поиска организован на основе стандарта языка запросов POSTQUEL путем заполнения соответствующих форм запроса.

Цель работы — создание программного комплекса, предоставляющего удаленному пользователю, связавшемуся с WWW-сервером ИС Трансформ, удобный интерфейс для регистрации в системе, ввода информации в базу данных, формирования запросов поиска, осуществления этого поиска и выдачи результатов в удобном для пользователя виде. Программа должна перефразировать запрос пользователя в SQL-запрос к базе данных и получить результаты его выполнения, а также диагностировать возникающие ошибки.

НАЗНАЧЕНИЕ РАЗРАБОТКИ

Функционально, разрабатываемый «Web-интерфейс к ИС Трансформ» будет служить для связи сервера базы данных СУБД Postgres95 с Internet-протоколами и предоставления удобного пользовательского интерфейса, позволяющего осуществлять ввод новой информации в БД для ведения базы данных, модификацию уже существующей в БД информации, поиск и генерацию отчетов по запросам пользователей.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

После окончания разработки Web-интерфейс будет применяться для ведения базы данных по преобразованиям программ. Информацию планируется сделать доступной пользователям, универсальность обеспечивается за счет разграничения полномочий пользователей.

Эксплуатационное назначение Web-интерфейса к ИС Трансформ — сбор и хранение информации по преобразованиям программ и предоставление этой информации для просмотра внешними клиентами

МЕТОДЫ И СРЕДСТВА РАЗРАБОТКИ

WWW-технология предоставляет средства для разработки простого удобного интерфейса пользователя для доступа к базе данных. Для общения клиента с сервером используется протокол HTTP, для указания ресурсов — технология URL, для взаимодействия WWW-сервера с внешними прикладными программами — CGI, для создания и использования гипертекстовых документов определен язык HTML. С помощью данной технологии реализуется взаимодействие WWW- сервера с базами данных Postgres95.

Инструментальными средствами решения задачи построения Web-интерфейса являются:

- язык сценариев PHP — используемый для написания CGI-программ;
- язык запросов POSTQUEL — для реализации доступа к серверу БД Postgres95;
- язык гипертекстовой разметки документов HTML — для разработки интерфейса пользователя с использованием HTML-форм.

WWW-доступ к базе данных осуществляется по второму из трех рассмотренных выше сценариев: путем динамического создания гипертекстовых документов на основе содержимого БД. Для реализации этой технологии используется взаимодействие WWW-сервера с запускаемыми программами CGI. Доступ к БД осуществляется на стороне WWW-сервера специальной CGI-программой (написанной на PHP с использованием существующих функций для работы с Postgres95), запускаемой WWW-сервером в ответ на запрос WWW-клиента. Эта программа, обрабатывая запрос, просматривает содержимое БД и создает выходной HTML-документ, возвращаемый пользователю.

Требования к функциональным характеристикам

- отображение интерфейса пользователя в виде HTML- документа;
- аутентификация пользователей с целью разграничения полномочий;
- обработка введенных пользователем данных с проверкой заполнения обязательных полей в формах, исключением неправильно заданных форматов, неверно заполненных полей запроса и т. д.;
- в соответствии с запросом поиск в БД, ввод или модификация информации в БД;
- отображение результатов работы.

Все выходные данные представляются в формате HTML. Входные данные — ключевые слова, список целевых полей, дальнейшие действия получаются после заполнения пользователем формы.

Требования к программному обеспечению

- WWW-сервер должен работать под управлением ОС UNIX;
- на стороне сервера должна быть установлена СУБД Postgres95;
- на стороне сервера должен быть установлен PHP в виде модуля Apache;
- на стороне клиента требуется подключение к сети и наличие браузера, поддерживающего HTML не ниже версии 2.0.

Требования к информационному обеспечению

- Web-интерфейс к ИС должен отображать всю информацию в браузере пользователя в формате HTML;
- проводить аутентификацию пользователя с целью выяснения его привилегий;
- для пользователей с низким уровнем привилегий давать возможность поиска и просмотра информации, для пользователей с высоким уровнем — возможность ввода и модификации информации.

Требования к информационной и программной совместимости

Программный модуль совместим с интерпретатором PHP версии не ниже 3.0. Для работы необходимо выполнение PHP в качестве модуля Apache и нужно, чтобы на сервере был запущен монитор БД (фоновый процесс, реагирующий на обращения к БД и обрабатывающий их).

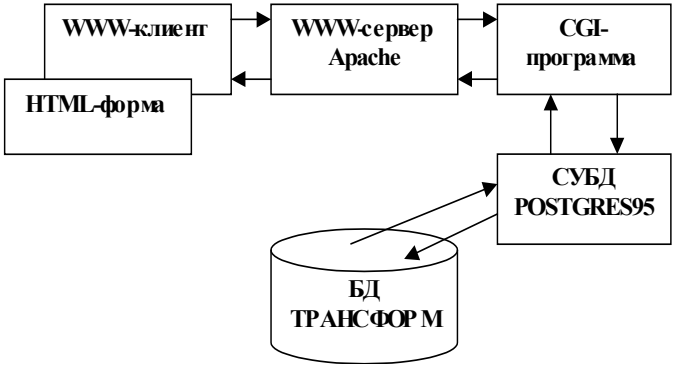


Рис. 1

СТРУКТУРА ВЗАИМОДЕЙСТВИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Структура взаимодействия программного обеспечения информационной системы показана на рис. 1. Согласно технологии WWW сервер протокола HTTP Apache, работающий, как правило, по 80-му порту стека протоколов TCP-IP, принимает запросы от пользователя с помощью клиентских программ просмотра гипертекстовых документов. Формализованный доступ к данным в рамках информационной системы осуществляется на основе HTML-форм. С их помощью введенные в поля формы данные передаются на сервер Apache, который вызывает указанный в форме CGI-скрипт (реализованный на PHP) для обработки этих параметров и передает ей управление. CGI-скрипт с помощью функций прикладного интерфейса СУБД POSTGRES95 преобразует данные в SQL-запрос, устанавливает соединение с сервером СУБД и передает ему запрос на выполнение. Сервер СУБД выполняет запрос, обращаясь к БД «Трансформ», и возвращает результат CGI-скрипту, который, в свою очередь, формирует на лету HTML-документ и через сервер Apache передает его клиенту.

Описание взаимодействия ТО системы

Приведенная на рис. 2 схема показывает, как работает система в целом. Пользователь на клиентском компьютере в программе просмотра заполняет предложенную форму или выбирает дальнейшее действие. Браузер по нажатию одной из кнопок в форме пересылает данные из заполненной формы или отображает вновь полученные в результате какой-либо операции. Программа принимает данные, проверяет их и формирует запрос к монитору БД или получает от него результат. Получив запрос, монитор обрабатывает его; если не произошло ошибок, ждет запроса от программы на отправку ей результата. На диске сервера хранится БД, модифицируемая по запросу клиента.

ОПИСАНИЕ ФУНКЦИОНИРОВАНИЯ WEB-ИНТЕРФЕЙСА ИС ТРАНСФОРМ

Разработанный интерфейс ИС Трансформ выполняет следующие функции:

- регистрацию пользователей;
- аутентификацию пользователей;
- протоколирование сеанса работы пользователей;
- работу с публикациями: поиск, добавление, модификация, генерирование кода, генерирование ключевых слов, ведение словаря стоп-слов;
- работу с преобразованиями: поиск, добавление и модификация.

РЕГИСТРАЦИЯ ПОЛЬЗОВАТЕЛЕЙ

В информационной системе предусмотрена регистрация пользователей. Зарегистрированный пользователь имеет возможность не только поиска и просмотра информации, но также возможность ввода информации в базу данных и модификации хранящейся в базе данных информации. Незарегистрированный пользователь имеет возможность только просматривать информацию.

Процедура регистрации нового пользователя состоит в заполнении соответствующей формы для регистрации, ссылка на которую находится на

главной странице. Также пользователь попадает на регистрационную форму при отказе от аутентификации. (при нажатии в окне для ввода логина и пароля кнопки “Cancel”).

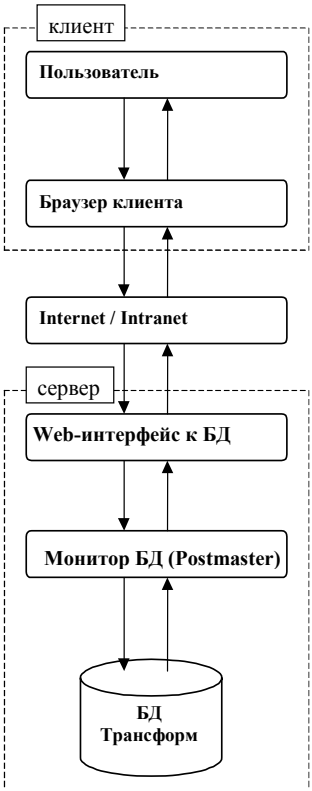


Рис. 2

Регистрационная форма содержит обязательные для заполнения поля: реальное имя пользователя (Ф.И.О), имя пользователя в системе — логин и e-mail-адрес.

Что касается пароля, то он генерируется автоматически по следующему алгоритму: сначала берется хэш-функция MD5 от имени пользователя (логина), затем от полученной строки берутся первые 8 символов. Созданный таким образом набор символов назначается в качестве исходного пароля и

отправляется по указанному пользователем e-mail-адресу. В дальнейшем пользователь может изменить данный ему пароль, заполнив соответствующую форму.

Также предлагается заполнить (не обязательно) поля, такие как: страна проживания, город, организация (место работы) и контактный телефон.

В форме регистрации предусмотрена также возможность сохранения имени и пароля по желанию пользователя, что позволяет при последующих входах в систему не проходить процесс аутентификации, т.е. не вводить каждый раз свое имя и пароль.

Это осуществляется с помощью Cookies-механизма для сохранения переданных пользователем данных в удаленном браузере (или на локальном диске пользователя). Cookies можно устанавливать на период одной сессии — тогда они удаляются после закрытия броузера. При установлении Cookies на некоторый определенный период времени переданные данные записываются в файл на локальном диске удаленного пользователя.

Таким образом, при выборе опции сохранения имени и пароля последние сохраняются на локальном диске пользователя в файле cookies и при каждом запросе передаются браузером серверу как часть HTTP-заголовка.

После заполнения формы введенные регистрационные данные проверяются и выдается сообщение об ошибке в случае, когда не заполнены обязательные поля, так же осуществляется запрос к базе данных для проверки того, что логин не используется уже другим пользователем. При отсутствии ошибок регистрационные данные вносятся в БД, пользователю выдается сообщение об успешной регистрации и уведомление о том, что пароль отправлен по указанному адресу.

АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЕЙ

Теперь рассмотрим, как происходит аутентификация пользователей в системе. Как уже было сказано выше, ИС Трансформ предоставляет возможность как просмотра информации, хранящейся в БД, так и возможность ее ввода и модификации. Зарегистрированные пользователи входят в систему с помощью своего логина и пароля, только после этого им предоставляется возможность добавления или модификации информации. Пользователям, не зарегистрированным в системе или не прошедшим аутентификацию, доступен только просмотр и поиск информации.

При входе в систему пользователь должен пройти процесс HTTP-аутентификации (проверку правильности имени и пароля пользователя).

HTTP аутентификация в PHP доступна только в случае, если пакет выполняется как модуль Apache. Эта процедура осуществляется путем вызова соответствующего PHP-скрипта для аутентификации, который предоставляет пользователю окно ввода с запросом Пользователь/Пароль (Username/Password). После введения пользователем своего имени и пароля и нажатия на кнопку “Ок” содержащий PHP-скрипт URL будет вызван снова с переменными $PHP_AUTH_USER, $PHP_AUTH_PW и $PHP_AUTH_TYPE, установленными соответственно имени пользователя, его паролю и типу аутентификации. Далее осуществляется запрос к базе данных и проверка правильности имени и пароля пользователя.

В случае успешной аутентификации внутренним PHP-переменным $PHP_AUTH_USER, $PHP_AUTH_PW присваиваются соответственно значения имени и пароля пользователя. Они сохраняются в течение всего сеанса работы и передаются каждому скрипту как часть HTTP-заголовка, в результате осуществляется автоматическая аутентификация пользователя при обращении к защищенным паролем формам (insert, update).

В случае несоответствия имени или пароля аутентификации не происходит, пользователю снова предоставляется окно запроса Username/Password. При нажатии кнопки “Cancel” пользователю выдается страничка с приглашением зарегистрироваться в системе с соответствующей ссылкой на регистрационную форму.

Доступ к формам для ввода и изменения информации возможен только после прохождения пользователем процесса HTTP аутентификации. Если пользователь не прошел процесс аутентификации при входе в систему, то при обращении к формам для ввода и модификации будет вызван PHP-скрипт, который запросит аутентификацию пользователя. Аналогично аутентификации при входе появится окно ввода с запросом Пользователь/Пароль. Как только пользователь ввел свое имя и пароль, URL, содержащий PHP-скрипт, будет вызван снова с переменными $PHP_AUTH_USER, $PHP_AUTH_PW, установленными соответственно имени пользователя и его паролю. После получения введенных пользователем данных $PHP_AUTH_USER и $PHP_AUTH_PW осуществляется запрос к базе данных и проверка правильности имени пользователя и пароля. При успешной проверке пользователю будет предоставлен доступ к соответствующим формам.

В том случае, если при регистрации пользователь выбрал опцию сохранения имени\пароля, при каждом входе в систему процесс аутентификации будет происходить автоматически: по запросу сервера удаленный браузер

передаст вызванному PHP-скрипту значения переменных $PHP_AUTH_USER и $PHP_AUTH_PW (логин и пароль), которые хранятся в файле cookies на машине пользователя. Таким образом, окно ввода запроса не появится, пользователю не потребуется вводить свое имя/пароль.

ПРОТОКОЛИРОВАНИЕ СЕАНСА РАБОТЫ ПОЛЬЗОВАТЕЛЯ

В системе реализован механизм протоколирования сеанса работы пользователя. Протоколируются следующие действия пользователя: вход в систему, поиск, добавление и модификация информации. При каждом входе пользователя в систему генерируется уникальный номер Web-сессии (ID-номер), который сохраняется в удаленном браузере в течение всего сеанса работы пользователя с системой с помощью Cookies-механизма, и удаляется после закрытия им браузера. ID-номер генерируется из IP-адреса удаленного пользователя с помощью функции PHP (uniquid), которая возвращает префиксед-уникальный идентификатор, основанный на текущем времени в микросекундах. В соответствующую таблицу БД вносится следующая информация о текущем сеансе: логин пользователя, если он прошел процесс аутентификации, или anonymous, в противном случае — имя хоста или IP-адрес, текущая дата, произведенное пользователем действие — вход, поиск, добавление или модификация, а также таблица, с которой связано это действие — публикации или преобразования, и oid-номер в случае добавления или модификации.

Также реализован механизм update (обновления) уже имеющейся запротоколированной информации о текущем сеансе. Точнее говоря, если сначала пользователь вошел в систему как anonymous, т. е. не прошел процесса аутентификации (в этом случае в качестве логина ему присваивается значение anonymous), и соответственно все его действия протоколировались под именем anonymous, а затем прошел аутентификацию, например при доступе к формам insert/update, то во всех осуществленных им действиях, запротоколированных в системе, значение anonymous будет заменено на логин.

ИНТЕРФЕЙС ДЛЯ РАБОТЫ С ОПИСАНИЯМИ ПУБЛИКАЦИЙ

Основными функциями интерфейса для работы с описаниями публикаций являются:

- поиск публикаций;
- добавление публикаций;
- модификация публикаций;

Две последние функции доступны только зарегистрированным в ИС пользователям.

Поиск публикаций. Основными функциями поискового интерфейса являются построение пользователем запроса на поиск требуемой информации в БД и преобразование результатов поиска к удобному для восприятия пользователем представлению. В интерфейсе для поиска публикаций реализована возможность осуществления двух типов поиска: простого и с предварительными запросами.

Рассмотрим вначале, как осуществляется простой поиск публикаций. Пользователю предлагается ввести в поле ввода запроса ключевые слова. Среди слов для поиска могут быть слова, входящие в названия публикаций, имена авторов. Есть возможность фиксирования количества выводимых на страницу результатов поиска (от 5 до 50). Пользователь также может выбрать вид представления найденных результатов в виде таблицы, записей и страницы.

При выборе формата представления результатов поиска в виде таблицы на страницу выводится таблица, состоящая из двух столбцов: название публикации и имена авторов.

При выборе формата представления результатов поиска в виде записей информация о каждой найденной публикации отображается на отдельной странице. Каждая запись представляет собой таблицу из трех строк: код публикации, название и имена авторов. Навигация по записям осуществляется с помощью кнопок перехода (предыдущая, следующая запись).

При выборе формата представления результатов поиска в виде страницы на экран выводятся все удовлетворяющие запросу публикации в виде записей, точнее, максимальное количество выводимых публикаций, заранее определенное пользователем.

Поиск с предварительными запросами. В отличие от рассмотренного выше простого поиска, поиск с предварительными запросами позволяет формировать запрос по двум критериям с использованием логических операций. Форма для поиска содержит два поля для ввода поисковых слов. В первом поле среди слов для поиска могут быть слова, входящие в названия публикаций, а во втором — входящие в имена авторов. Как уже было сказано, поиск допускает использование логических операций “и” и ”или”. В

первом случае будут найдены все публикации, удовлетворяющие обоим критериям одновременно, а во втором - хотя бы одному критерию.

Возможно проведение предварительного запроса, в результате которого отображается количество удовлетворяющих запросу публикаций.

Добавление описания публикации. При входе на страницу для добавления публикации пользователю предоставляется список полей для выбора. Список разделен на поля по умолчанию (тема, год, авторы, название), которые обязательно должны быть заполнены для каждой публикации, и поля, выбираемые по желанию пользователя (WWW-адрес, аннотация на английском и русском языках). После выбора пользователем нужных ему полей генерируется форма для ввода значений соответствующих полей в БД.

В качестве поля для ввода темы публикаций используется список с соответствующими значениями для выбора. После заполнения пользователем формы проверяется, все ли обязательные поля заполнены; если хоть одно поле не заполнено, выдается сообщение об этом.

Помимо вышеперечисленных обязательных полей (тема, год, авторы, название) в БД вносится так называемый код публикации, который автоматически генерируется по значениям, введенным в поля «авторы» и «год издания публикации».

Каждый код состоит примерно из 5-6 символов и генерируется по следующему алгоритму. Первые символы берутся из первых символов фамилий авторов; если у публикации только один автор, берутся первые три символа его фамилии, два автора - по два первых символа, три и более авторов - по первому символу первых трех авторов. К этим символам добавляется две последние цифры года издания публикации.

Также автоматически генерируются и вносятся в таблицу с описанием публикаций так называемые ключевые слова, которые в дальнейшем будут использоваться для поиска этой публикации. Ключевые слова извлекаются из названия публикации следующим образом: берутся все слова, за исключением так называемых стоп-слов (например, союзов, предлогов и т.д.).

Для хранения и накопления стоп-слов ведется специальный словарь, который может редактировать сам пользователь. Интерфейс для работы со словарем предусматривает добавление новых слов в словарь и просмотр его содержания. Добавление осуществляется путем введения слов через запятую в соответствующее поле ввода. Реализован механизм просмотра всех слов в словаре, начинающихся с данного символа. Просмотр осуществляется с помощью выбора нужного символа из списка алфавитных символов. Таким образом, прежде чем добавить новое слово в словарь, пользо-

ватель может проверить его наличие в словаре.

Модификация описания публикаций. Теперь рассмотрим, как организована работа пользователя по модификации описаний публикаций. Сначала пользователь выбирает ту публикацию, информацию о которой он хочет изменить, что осуществляется выбором из соответствующего списка кода нужной публикации. После выбора кода публикации генерируется форма, содержащая как текущие значения описания публикации, так и пустые поля для ввода новых значений. При оставлении поля новых значений пустым сохраняется прежнее значение. После введения новых данных по нажатию кнопки форма отправляется на сервер БД.

ОРГАНИЗАЦИЯ РАБОТЫ С ПРЕОБРАЗОВАНИЯМИ

Основными функциями интерфейса для работы с преобразованиями являются:

- поиск преобразований;
- добавление преобразований;
- модификация преобразований;

Поиск преобразований. Для преобразований реализован простой поиск, который осуществляется по введенным в поле ввода запроса ключевым словам. Среди слов для поиска могут быть слова, входящие в названия, цель, способ преобразований, участок экономии, способ реализации, языковую и архитектурную ориентацию. Аналогично поиску публикаций реализованы три формата представления найденных результатов (в виде таблицы, записей и страницы), есть возможность ограничения максимального количества отображаемых на странице результатов запроса.

Добавление преобразований. Аналогично интерфейсу для добавления описаний публикаций пользователь сначала должен выбрать список полей для добавления. Список разделен на поля по умолчанию (название преобразования, цель преобразования, способ преобразования, способ реализации, участок экономии, содержание преобразования, формализованное описание) и поля, выбираемые по желанию пользователя (варианты преобразования, известные синонимы, языковая ориентация, архитектурная ориентация, используемое промежуточное представление, контекстные условия, иллюстративный материал, программная реализация преобразования — название реализации, алгоритм, комментарии к алгоритму, оценка сложно-

сти алгоритма, название компилятора содержащего реализацию, название компьютера, название файла с кодом реализации; взаимосвязь с другими преобразованиями). После выбора пользователем нужных ему полей генерируется форма для ввода значений соответствующих полей в БД.

В качестве полей для ввода цели преобразования, способа преобразования, способа реализации участка экономии используются списки с соответствующими значениями. После заполнения пользователем формы проверяется, все ли обязательные поля заполнены; если хоть одно поле не заполнено, выдается сообщение об этом.

Модификация преобразований. Теперь рассмотрим, как организована работа пользователя по модификации преобразований. Сначала пользователь выбирает то преобразование, информацию о котором он хочет изменить, что осуществляется выбором из соответствующего списка названия нужного преобразования. После выбора редактируемого преобразования генерируется форма, содержащая как текущие значения о преобразовании, так и пустые поля для ввода новых значений. Если поле новых значений остается пустым, сохраняется прежнее значение. После введения новых данных по нажатию кнопки форма отправляется на сервер БД.

ЗАКЛЮЧЕНИЕ

В работе были исследованы современные методы и средства построения интерфейсов информационных систем, основанные на использовании WWW-технологии.

В процессе работы разработан и реализован Web-интерфейс пользователей ИС Трансформ, выполняющий следующие функции:

- регистрацию пользователей;
- аутентификацию пользователей;
- протоколирование сеанса работы пользователей;
- работу с публикациями: поиск публикаций, добавление публикаций, модификация публикаций, генерирование кода публикации, генерирование ключевых слов для публикации, ведение словаря стоп-слов;
- работу с преобразованиями: поиск преобразований, добавление преобразований, модификация преобразований.

В качестве средств разработки использовались HTML, CGI, PHP и Postquel.

Данная разработка будет использоваться в качестве Web-интерфейса ИС Трансформ, создаваемой в ИСИ СО РАН и предназначенной для накопления и систематизации информации по преобразованиям программ.

СПИСОК ЛИТЕРАТУРЫ

1. **Малинина Ю.В.** Информационная система ТРАНСФОРМ по преобразованиям программ // Проблемы конструирования эффективных и надежных программ. — Новосибирск, 1995. — С. 128—136
2. **Малинина Ю.В.** ИС ТРАНСФОРМ: описание инфологической схемы базы данных // Оптимизирующая трансляция и конструирование программ. — Новосибирск, 1997. — С. 60—79.
3. **Малинина Ю.В.** ИС ТРАНСФОРМ: прототип интерфейса для визуального исследования БД // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 78—106.

В. Н.Касьянов

ПРИМЕНЕНИЕ ГРАФОВ В ПРОГРАММИРОВАНИИ[1]

1. ВВЕДЕНИЕ

Теория графов стала активно применяться в программировании одновременно с использованием ЭВМ в силу удобного выражения задач обработки информации на теоретико-графовом языке. Среди первых работ, существенно использующих теоретико-графовые методы в решении задач программирования, можно отметить широко известные работы А.П. Ершова по организации вычисления арифметических выражений (1958 г.) и оптимальному использованию оперативной памяти (1962 г.), а также заметку Р. Карпа (1960 г., на русском языке — 1962 г.), в которой была введена теоретико-графовая модель программы в виде управляющего графа. Эта модель стала к настоящему времени классической для решения задач трансляции и конструирования программ [1, 2].

Модель программы в виде управляющего графа, модель арифметического выражения в виде ориентированного дерева, синтаксические деревья, деревья сортировки, сети Петри и другие теоретико-графовые конструкции внесли свой существенный вклад в развитие программирования и его автоматизации. Появление суперкомпьютеров и сетей и возникшая при этом проблема эффективной организации параллельных и распределенных вычислений над информационными массивами большого объема подтвердили тенденцию использования графов как наиболее эффективного средства автоматизации программирования.

Расширение круга задач, решаемых на ЭВМ, потребовало выхода на модели дискретной математики, что привело к подлинному расцвету теории графов и комбинаторики, которые за сорок лет трансформировались из разделов “досуговой” (по выражению Андрея Петровича Ершова) математики в основной инструмент решения огромного числа задач.

Современное состояние информатики и программирования нельзя представить себе без теоретико-графовых методов и алгоритмов. Широкая применимость графов связана с тем, что они являются очень

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 00-07-90296) и Министерства образования РФ.

естественным средством объяснения сложных ситуаций на интуитивном уровне. Эти преимущества представления сложных структур и процессов графами становятся еще более ощутимыми при наличии хороших средств их визуализации. Поэтому неслучайно в настоящее время в мире растет интерес к методам и системам визуальной обработки графов и графовых моделей.

Многие программные системы, особенно те, которые используют информационные модели, включают элементы визуальной обработки графовых объектов. Среди них — системы и окружения программирования, инструменты CASE-технологии, системы автоматизации проектирования и многие другие. По направлению, связанному с исследованием алгоритмов и методов визуализации графов, за которым в мировой практике закрепилось название “Graph Drawing”, ежегодно проводятся международные конференции, первая из которых состоялась в 1994 г. Материалы этих конференций регулярно публикуются в серии книг Lecture Notes in Computer Science, а обзор алгоритмов рисования графов [3] содержит более 300 работ, написанных до первой конференции по данной тематике.

Поскольку информация, которую желательно визуализировать, постоянно увеличивается и усложняется, возникает все больше ситуаций, в которых классические графовые модели перестают быть адекватными.

Требуются и возникают более мощные графовые формализмы для представления информационных моделей, обладающих иерархической структурой, такие как например хиграфы [25] и составные графы [26]. Хиграфы являются обобщением понятия гиперграфа и представляют сложные отношения, используя многоуровневые “блобы”, которые могут содержать друг друга и пересекаться между собой. Составные графы являются расширением класса ориентированных графов за счет введения между вершинами дополнительного отношения включения и представляют собой менее общий формализм, чем хиграфы. Одним из современных неклассических формализмов являются кластерные графы [24]. Кластерный граф состоит из неориентированного графа и рекурсивного разбиения его вершин на подмножества, называемые кластерами. Это относительно общий графовый формализм, который может использоваться во многих приложениях и хорошо приспособлен к задачам рисования графов.

Помимо иерархичности, которая является основой многочисленных методов анализа и синтеза сложных информационных моделей в различных областях применения ЭВМ, новые формализмы должны в большей степени поддерживать визуализацию семантических аспектов информационных моделей. Понятно, что визуальная обработка информационных моделей не сводится только к проблемам визуализации их структурных свойств, а включает также широкий круг вопросов, связанных с поддержкой различных процессов анализа и синтеза графовых моделей с использованием человеко-машинного интерфейса, основанного на их наглядных представлениях.

В статье описываются текущие работы по созданию средств поддержки применения графов в программировании, которые ведутся сотрудниками лаборатории конструирования и оптимизации программ Института систем информатики им. А. П. Ершова СО РАН при финансовой поддержке РФФИ и Минобразования.

Статья состоит из шести разделов. Разд. 2 посвящен работам по созданию "энциклопедии" алгоритмов на графах для программистов. В разд. 3 обсуждаются понятия иерархических графов и графовых моделей, включая вопросы их представления и визуальной обработки. В разд. 4 рассматриваются инструментальные системы HIGRES и VEGRAS для поддержки визуальной обработки графов и графовых моделей. Разд. 5 посвящен работам по созданию толкового словаря по теории графов для программистов и его электронной версии. Разд. 6 — заключение.

2. НА ПУТИ К "ЭНЦИКЛОПЕДИИ" АЛГОРИТМОВ НА ГРАФАХ ДЛЯ ПРОГРАММИСТОВ

2.1. Вопросы применения графов в программировании

Современное состояние программирования нельзя представить себе без теоретико-графовых алгоритмов. Хорошо известно, что многие задачи повышения качества трансляции, как в смысле улучшения рабочих характеристик транслятора, так и в смысле повышения качества получаемых машинных программ формулируются и решаются как задачи на графах. Сюда относятся в первую очередь задачи, связанные с представлением программ в виде теоретико-графовых моделей, важнейшей из которых является управляющий граф. Кроме того, необходимо указать на такие области применения граф-моделей, как эффективное ис-

пользование ресурсов вычислительной системы (оптимизация использования памяти, регистров, уменьшение обменов между оперативной и внешней памятью и т.д.), организация больших массивов информации (деревья и, вообще, графы данных для повышения эффективности информационного поиска), увеличение степени параллелизма программы, повышение эффективности работы многопроцессорных и многомашинных систем (распределение загрузки процессоров, обмен сообщениями между процессами, синхронизация, конфигурация сетей связи между процессорами и т.д.). Решение этих и подобных задач привело к появлению множества граф-моделей, связанных как с программами и структурами данных, так и с вычислительными системами, в том числе параллельными.

Как уже отмечалось выше, теория графов из академической дисциплины все более превращается в средство, владение которым становится решающим для успешного применения ЭВМ во многих прикладных областях. Причем совершенно ясно, что, несмотря на наличие обширной специальной литературы по решению задач на графах, широкое применение в практике программирования полученных математических результатов затруднено в силу отсутствия их систематического описания, ориентированного на программистов. Поэтому значительный класс практических задач, по существу сводящихся к простому выбору подходящего способа решения и к построению конкретных формулировок абстрактных алгоритмов, для многих программистов все еще остается полем для интеллектуальной деятельности по переоткрытию методов. Известный фундаментальный труд Д. Кнута “Искусство программирования для ЭВМ” [4], если бы даже он и продолжился, как планировалось, не сможет решить эту проблему в силу ориентации Д. Кнута на низкоуровневое (в терминах машины MIX) описание алгоритмов. В изданных же трех томах серии излагается достаточно узкий класс используемых в программировании граф-моделей (фактически Д. Кнут ограничился рассмотрением деревьев и не планирует продолжать работу над серией).

В отличие от Д. Кнута, в своих книгах мы ориентируемся на высокоуровневое описание алгоритмов в терминах специального псевдоязыка (лексикона) программирования с использованием традиционных конструкций математики и языков программирования высокого уровня. Такой подход позволяет формулировать алгоритмы в естественной форме, допускающей прямой анализ их корректности и сложности, а

также простой перенос алгоритмов на традиционные языки программирования и ЭВМ с сохранением полученных оценок сложности. Кроме того, подобный стиль описания алгоритмов является базой для доказательного стиля программирования: он позволяет понять алгоритм на содержательном уровне, оценить пригодность его для решения конкретной задачи и осуществить модификацию алгоритма, не снижая степень математической достоверности окончательного варианта программы.

Среди теоретико-графовых алгоритмов и методов, применяемых в программировании, естественно выделяются классы алгоритмов, общим для которых является тот или иной тип графов, используемых в качестве модели. Здесь в первую очередь следует назвать класс алгоритмов обработки деревьев, класс алгоритмов обработки бесконтурных графов (ациклические графы, DAG), моделирующих частично упорядоченные множества, решетки и полурешетки, а также класс алгоритмов обработки регуляризуемых графов (сводимые графы), моделирующих программу при потоковом анализе, оптимизирующей трансляции и распараллеливании и являющихся основой современных технологий программирования, таких как структурное программирование, трансформационное программирование и др. Поэтому нам показалось естественным при изложении теоретико-графовых методов и алгоритмов для программистов использовать указанное их разделение на классы. Текущим результатом данной работы явились три книги [5-7].

2.2. Класс алгоритмов на деревьях

Книга [5] представляет собой систематическое изложение алгоритмов на деревьях, образующих один из наиболее важных и широко используемых в программировании классов алгоритмов теории графов. Эти алгоритмы по своей фундаментальности для задач обработки информации можно сравнить только с алгоритмами вычисления функций анализа или алгоритмами линейной алгебры в вычислительной математике.

В книге даются основные математические понятия и модели, методы и алгоритмы, связанные с различными приложениями теории графов. В ней рассматриваются задачи обходов и генерации деревьев, отыскания каркасов, построения структурных деревьев, изоморфизма, унификации и преобразования деревьев, организации и представления информации, а также синтаксического анализа. Наряду с описанием алгоритмов книга содержит большое количество сведений о свойствах деревьев

и областях их применения, а также обширную библиографию с подробными комментариями.

Книга состоит из 8 глав, которые объединены в три части.

В части I, содержащей три главы, излагаются основные сведения о деревьях и приводятся базисные алгоритмы их обработки. Открывает эту часть глава 1, в которой приведены основные определения, рассмотрены проблемы представления деревьев, перечисления и подсчета. Глава 2 посвящена обходам и генерации деревьев, а также алгоритмическим вопросам обработки деревьев. Глава 3 рассматривает каркасы графов и содержит описание алгоритмов их отыскания, перечисления и поиска каркасов с заданными свойствами.

В части II, также объединяющей три главы, рассматриваются вопросы применения деревьев для решения задач, связанных со структуризацией программ, унификацией, системами переписывания термов, синтаксическим анализом и пр. Глава 4 посвящена структурным деревьям, отражающим внутреннее строение управляющих графов, в том числе деревьям обязательного предшествования (доминаторным деревьям) и деревьям обязательного преемственности (постдоминаторным деревьям). В главе 5 изучаются задачи изоморфизма и унификации, а также системы переписывания термов. В главе 6 рассматриваются синтаксические деревья, играющие большую роль в синтаксическом анализе программ.

Часть III посвящена вопросам хранения и поиска информации; она состоит из двух глав. Глава 7 рассматривает информационные деревья для одноуровневой памяти, которые включают в себя балансированные относительно различных критериев и многомерные деревья. Глава 8 посвящена деревьям для многоуровневой памяти, включая различные варианты B-деревьев, в том числе и многомерные.

2.3. Класс алгоритмов на бесконтурных графах

Книга [6] является второй в серии книг, посвященных алгоритмам на графах, решающих задачи из различных областей информатики и программирования. Данная книга, как и предыдущая, не является только книгой по теории графов, она насыщена результатами из теоретического и системного программирования, полученными либо на основе теории графов, либо существенно использующими теоретико-графовый язык. Эти результаты разнесены по главам, что позволит использовать книгу многими специалистам. В ней изложены необходимые определения, основополагающие факты и свойства, относящиеся к бесконтур-

ным графам и частично упорядоченным множествам и их приложениям в задачах синтаксического анализа и генерации кода.

В отличие от деревьев, бесконтурные графы (DAG — в англоязычной литературе) представляют более сложный и поэтому менее широко используемый класс графов, количество общедоступных публикаций по алгоритмам для этого класса заметно меньше. Помимо основных определений и таких универсальных задач, как например топологическая сортировка или нахождение кратчайшего пути в главу по базисным алгоритмам обработки бесконтурных графов, мы включили в книгу алгоритмы построения транзитивного замыкания, выделения бикомпонент, нахождения конгруентного замыкания, выявления ближайших общих предков и изображения бесконтурных графов на плоскости. Отдельные главы книги мы посвятили подробному рассмотрению алгоритмов и методов таких основных этапов трансляции, как семантический анализ и кодогенерация, которые, в отличие от более изученного этапа синтаксического анализа, рассмотренного в первой книге, опираются на класс бесконтурных графов. Не менее просто оказалось выбрать материал, относящийся к частично упорядоченным множествам, в частности к решеткам и полурешеткам. Этим вопросам также посвящена отдельная глава книги.

Следует признать, что для некоторых задач, сводящихся к задачам на бесконтурных графах, требовался большой вступительный материал, в силу чего рассмотрение этих задач не было осуществлено в данном издании. Эта касается, например, использования частично упорядоченных множеств в теории параллельных процессов и искусственном интеллекте.

2.4. Класс алгоритмов для сводимых и регуляризуемых графов

Книга [7] посвящена сводимым графам и граф-моделям в программировании. В ней изложены необходимые определения, основополагающие факты и свойства, относящиеся к базисным алгоритмам обработки сводимых и регуляризуемых графов, приведены описания важных апробированных, а также новейших алгоритмов, привлекших внимание авторов. Рассмотрен ряд широко используемых в программировании граф-моделей, связанных с оптимизацией и автоматическим распараллеливанием последовательных программ, а также моделированием программ и систем при параллельной и распределенной обработке.

Книга состоит из одиннадцати глав, образующих две части: сводимые графы и граф-модели в программировании.

Первая часть посвящена изучению свойств класса сводимых и регуляризуемых графов. В ней рассмотрены определения сводимых и регуляризуемых графов, их свойства и ряд алгоритмов решения возникающих при этом теоретико-графовых задач. Регуляризуемые графы представляют собой наиболее общий тип граф-моделей структурированных программ. Они поддерживают эффективное проведение оптимизирующих и распараллеливающих преобразований программ и являются основой трансформационного подхода к конструированию надежного и эффективного программного обеспечения. Класс сводимых и регуляризуемых графов играет чрезвычайно важную роль в программировании в силу того, что программа, управляющий граф которой принадлежит этому классу, допускает применение более эффективных алгоритмов анализа и оптимизации. Так, задача нахождения минимального множества дуг, удаление которых разрывает все контуры в орграфе, является NP-полной для графа общего вида, но имеет полиномиальную сложность для сводимых графов.

Эта часть состоит из четырех глав. В первой главе изучаются интервалы как частный случай “хорошо устроенных” графов, исследуются их свойства, описывается алгоритм выделения максимальных интервалов в управляющем графе. Вводится понятие интервального представления управляющего графа, на основе которого определяется класс сводимых графов. Здесь же изучаются обобщенная сводимость (регуляризуемость) графов, разборность графов, а также другие свойства, эквивалентные понятию сводимости. Здесь же описывается алгоритм проверки графов на сводимость, а также излагается связанный с ним способ определения порядка втягиваемых вершин. Решается задача регуляризации несводимых графов. В главе 2 рассматривается важная проблема разрезания контуров в сводимых графах. Она состоит в определении минимального множества вершин или дуг, удаление которых разрывает все контуры в управляющем графе. В главу включен играющий важную роль в данном вопросе алгоритм Бергера-Шора, а также алгоритм Шамира. Глава 3 посвящена анализу циклической структуры уграфов и циклически сводимым уграфам. В ней рассматриваются решения двух основных задач, связанных с выявлением участков повторяемости в программе, моделируемой уграфом: оценки частоты выполнения операторов и переходов в уграфе и выделения иерархии циклов

в нем. Здесь же рассматриваются циклически сводимые уграфы. К ним относятся уграфы, для которых задача разрушения контуров решается эффективно; приводится соответствующий алгоритм. Завершается первая часть главой 4, в которой изучается проблема перечисления путей, важная для эффективной организации решения задач потокового анализа программ. Вначале рассматривается более простая проблема построения сильных и слабых укладок — последовательностей вершин специального вида, позволяющих успешно решать проблему перечисления путей.

2.5. Граф-модели в программировании

Во второй части книги [7] представлен ряд широко используемых в программировании граф-моделей, связанных с оптимизацией и автоматическим распараллеливанием последовательных программ, а также моделированием программ и систем при параллельной и распределенной обработке. Основное внимание в ней уделяется тематике, мало исследованной в отечественных монографиях. Сюда в первую очередь относятся такие понятия, как структурная сложность программ, зависимость по управлению и теоретико-графовые формы представления программ в распараллеливающих компиляторах и пр.

Данная часть состоит из семи глав. Глава 5, начинающая данную часть, посвящена управляющему графу — основной модели последовательных программ. На его основе формулируется понятие структурной сложности программы, вводится цикломатическая мера сложности, анализируются ее слабые и сильные стороны, описывается ряд других мер, в том числе мер, несвязанных прямо с управляющим графом. В главе 6 рассматриваются графовые модели для программ с процедурами: граф процедур, граф вызовов процедур и граф зацепленности. Изучаются свойства графа процедур, и описываются методы и алгоритмы его построения. Рассматриваются схемы с косвенной адресацией, и описаны методы нахождения информационных связей и множеств аргументов и результатов в программе, содержащей указатели. В главе 7 описываются различные графовые промежуточные представления программ, используемые в основном при автоматическом распараллеливании программ. Подробно рассмотрены граф зависимостей по данным, граф программных зависимостей, базирующийся на теоретико-графовом определении зависимости по управлению, иерархический граф заданий. Глава 8 посвящена операторным моделям программ как теоретической осно-

ве оптимизирующих и распараллеливающих преобразований итеративных программ. Рассматривается два направления в теории операторных схем: семантические и формальные модели. Описывается класс крупноблочных схем и его основные подклассы. Изучаются вопросы представления памяти в операторных схемах. Глава 9 посвящена сетям Петри, относящимся к числу наиболее важных и распространенных моделей в области параллельной и распределенной обработки информации. Они обеспечивают формальное описание как алгоритмов и программ, так и собственно вычислительных систем и устройств. Здесь изучаются основные свойства сетей Петри и методы их анализа, дается характеризация классов языков и рассматриваются регулярные и иерархические сети Петри. Глава 10 посвящена графовым формализмам, ориентированным на поддержку визуальной обработки информационных моделей, обладающих иерархической структурой. В ней рассматриваются иерархические графы и графовые модели, исследуются вопросы их использования для визуальной обработики сложных структур и процессов. Завершается вторая часть книги рассмотрением графов адресуемых данных (глава 11) как моделей структур данных, позволяющих изучать свойства структур данных безотносительно содержания самих данных.

3. ИЕРАРХИЧЕСКИЕ ГРАФОВЫЕ МОДЕЛИ И ВИЗУАЛЬНАЯ ОБРАБОТКА

3.1. Иерархические графы

Иерархические графы и граф-модели рассматривались в работах [8, 9].

Пусть G обозначает граф произвольного вида, элементы (вершины и ребра) которого отличаются один от другого какими-либо пометками, называемыми их *именами*, например: G может быть неориентированным графом, орграфом (ориентированным графом), мультиграфом (с кратными ребрами), псевдографом (с петлями) или гиперграфом.

Граф C называется *фрагментом* графа G, обозначаем $C \subseteq G$, если C — часть графа G, т. е. C образован подмножеством элементов графа G. F — *иерархия фрагментов* графа G, если F — такое множество фрагментов графа C, что $G \in F$ и для любых двух фрагментов C_1 и C_2 из F либо фрагменты C_1 и C_2 не пересекаются, либо один из них является частью (*подфрагментом*) другого. Фрагмент G — *основной* (*главный*)

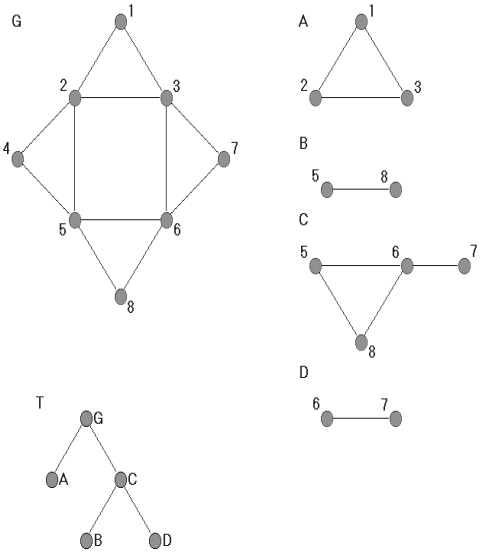


Рис. 1. Пример связного иерархического графа

фрагмент иерархии F. Фрагмент $C \in F$ — *элементарный*, если в F нет фрагментов G, являющихся подфрагментами фрагмента C. Для любых $C_1, C_2 \in F$ фрагмент C_1 — *прямой* подфрагмент C_2 (или, что то же самое, фрагмент, *непосредственно* вложенный в C_2), если C_1 — подфрагмент C_2 и не существует такого $C_3 \in F$, отличного от C_1 и C_2, что $C_1 \subseteq C_3 \subseteq C_2$.

Иерархический граф $H = (G, T)$ состоит из графа G и корневого дерева T, вершины которого соответствуют элементам некоторой иерархии в G, а дуги отражают отношение их непосредственной вложенности. T называется *деревом вложенности*, а G — *основным графом* иерархического графа H.

Пример иерархического графа $H = (G, T)$ приведен на рис. 1. Здесь и в дальнейшем рядом с вершинами дерева T указаны имена фрагментов, которым они соответствуют.

Граф H называется иерархическим графом с тривиальной иерархией, если дерево T — тривиальный граф, т. е. состоит из единствен-

ной вершины G. Указанный класс иерархических графов представляет обычное понятие графа.

Граф $H = (G, T)$ называется *связным*, если для любой вершины p дерева T фрагмент $G(p)$ основного графа G, соответствующий p, является связным. Нетрудно представить себе пример несвязного иерархического графа, основной граф которого является связным. Важный частный случай иерархических графов образуют такие $H = (G, T)$, что каждая вершина дерева вложенности T соответствует некоторому подграфу основного графа G. Такие графы будем называть простыми иерархическими графами.

В классе простых иерархических графов естественно выделяется подкласс *иерархических деревьев* $H = (G, T)$, в которых основной граф G не содержит ребер. Указанный класс иерархических графов представляет обычное понятие корневого дерева. Другой важный подкласс простых иерархических графов — так называемые иерархические упорядоченные деревья $H = (G, T)$, в которых G является линейным орграфом. Линейное упорядочение вершин основного орграфа G, отражающее достижимость одной вершины орграфа G из другой, индуцирует естественное линейное упорядочение на множестве сыновей любой вершины дерева T. Таким образом, класс иерархических упорядоченных деревьев представляет обычное понятие упорядоченного корневого дерева. Класс простых иерархических графов $H = (G, T)$, в которых G — неориентированный граф, а все листья T соответствуют тривиальным подграфам G, представляет понятие кластерного графа [24].

3.2. Изображения иерархических графов

Пусть задан некоторый иерархический граф $H = (G, T)$, у которого G не является гиперграфом (вопросы изображения гиперграфа рассматриваются в [9]). Изображением иерархического графа H называется представление его элементов (вершин, фрагментов и ребер) на плоскости в соответствии со следующими тремя правилами.

1. Каждая вершина графа G представляется точкой или некоторой простой замкнутой областью R, определяемой простой (не имеющей самопересечений) замкнутой кривой — ее *границей*. Изображения разных вершин не пересекаются.

2. Каждый фрагмент C, являющийся вершиной дерева T, представляется простой замкнутой областью таким образом, что представления всех вершин и фрагментов графа H, содержащихся в C, целиком рас-

положены строго внутри области, представляющей C (т. е. они принадлежат этой области и не содержат точек ее границы), а представления всех вершин и границ фрагментов графа H, не содержащихся в C, целиком расположены вне области, представляющей C (т. е. они не имеют с ней общих точек).

3. Каждое ребро графа G представляется простой кривой, соединяющей по одной точке из представлений соответствующих вершин графа G, которым она инцидентна, и пересекающейся с этими представлениями в точности по этим двум точкам (эти точки являются концами кривой, представляющей ребро). Дуга графа G представляется аналогично, за исключением указания направления на кривой. Кривые, представляющие разные ребра и дуги, пересекаются между собой и с границами фрагментов лишь в конечном числе точек и не имеют общих точек с изображениями вершин, которым они не инцидентны.

Изображение D графа H называется *структурным*, если представление любого ребра (и дуги), соединяющего некоторые вершины p и q в H, пересекает границу некоторого фрагмента C только тогда, когда ребро не принадлежит фрагменту C, причем само пересечение не содержит больше чем $\mid \{p, q\} \cap C \mid$ точек, и *неструктурным* — в противном случае. Изображение D графа H называется *плоским*, если в нем нет пересекающихся ребер или дуг.

Граф H называется *планарным*, если он имеет плоское структурное изображение. Понятно, что каждое иерархическое дерево аналогично обычному "дереву" является планарным. Вместе с тем понятие планарности иерархического графа не совпадает с понятием планарности его основного графа. Например, существуют непланарные иерархические графы, основные графы которых являются планарными, а также непланарные иерархические графы, основные графы которых имеют плоские структурные изображения. Справедлива теорема о том (см. [9]), что простой связный иерархический граф $H = (G, T)$ является планарным тогда и только тогда, когда существует такое плоское изображение графа G, что для любой вершины $p \in T$ все элементы графа $G \setminus G(p)$ находятся на внешней грани изображения графа $G(p)$.

Наряду с полным изображением иерархического графа, определенным выше, можно рассматривать и частичные его изображения, в которых некоторые элементы графа не имеют изображений. Например, во многих приложениях нет необходимости изображать главный фрагмент. Другой пример — частичные изображения иерархического гра-

фа, определяемые разными сечениями его дерева вложенности T. В каждом таком изображении все представления его фрагментов, соответствующих элементам сечения, являются как бы "непрозрачными", скрывающими представления всех элементов графа, содержащихся в них. Понятно, что при переходе от полного изображения основного графа к некоторому его частичному изображению могут возникать кратные ребра, даже если основной граф не является мультиграфом. Их можно изображать независимо друг от друга либо одним ребром. Выбор в пользу того или иного представления указанных ребер зависит от класса решаемых задач.

Вопросы построения изображений иерархических графов на плоскости весьма важны и трудны как в смысле сложности формализации "хорошего" (наглядного) изображения того или иного класса графов, так и в смысле сложности алгоритмов построения указанных "хороших" изображений. Нужно сказать, что различные приложения накладывают дополнительные ограничения на изображения планарных графов и даже деревьев. При этом большинство графов, возникающих на практике, не планарны, т.е. не могут быть изображены на плоскости без пересечений ребер. Поэтому возникает задача нахождения такого изображения непланарного графа, ребра которого имеют пересечения, но изображение обладает как можно большей наглядностью, например имеет наименьшее число пересечений ребер. Такие обычно используемые при оценке качества изображения характеристики графа, как род, число скрещиваний, толщина или искаженность, также естественным образом обобщаются на класс иерархических графов в соответствии с введенным выше понятием укладки иерархического графа на плоскости.

3.3. Иерархические графовые модели

Под графовой моделью в общем случае мы понимаем класс графовых объектов, имеющих вид помеченных графов, с заданным на нем отношением эквивалентности [9]. При этом при задании графовой модели мы различаем статическую (или синтаксическую) часть описания, определяющую класс помеченных графов, образующих указанную модель, и динамическую (или семантическую) часть, задающую разбиение данного класса графов на подклассы попарно эквивалентных. Заметим, что термин "графовая модель" используется ниже в двух смыслах: для обозначения как всего класса объектов, составляющих ее, так и отдельных

элементов этого класса. Пусть имеется множество объектов V, называемых *метками*, распадающееся на попарно непересекающиеся подмножества *классов* меток. В качестве классов меток могут использоваться определенные множества чисел, символов, строк (цепочек символов), формул, графов и объектов других видов. Пусть также задано множество объектов W, называемых *типами*, и пусть каждому элементу $w \in W$ поставлено в соответствие множество *пометок* $V(w)$, имеющее вид декартового произведения $V_{i_1} \times V_{i_2} \times \ldots \times V_{i_s}$, где $V_{i_j} \subseteq V$ — некоторый класс меток для любого j.

Со статической точки зрения *иерархическая графовая модель* — это тройка (H, M, L), где H — иерархический граф, M — *функция типа*, приписывающая каждому элементу (вершине, ребру и фрагменту) h иерархического графа H его тип $M(h) \in W$, а L — *функция меток*, приписывающая каждому элементу h графа H его пометку — некоторый элемент $L(h) \in V(M(h))$. Что касается динамической части иерархической графовой модели, то она может быть задана разными способами и привносит в визуализацию графовых моделей различные анимационные аспекты. Можно выделить два разных подхода к заданию семантической части графовой модели: путем явного задания набора инвариантов (свойств, присущих всем эквивалентным между собой моделям), который различает классы эквивалентности графовых моделей, либо через так называемые эквивалентные преобразования графовых моделей, которые сохраняют указанный набор инвариантов.

Оба подхода к заданию семантической части графовой модели опираются на преобразования графов и активно развиваются в рамках теории схем программ. Здесь первый подход приводит к семантическим моделям программ, используемым главным образом для исследования проблем разрешимости и обоснования корректности преобразований, а второй — к формальным моделям программ, ориентированным на исследование различных формализаций применяемых на практике способов улучшения качества транслируемых программ — так называемых оптимизирующих преобразований.

Первый подход к заданию семантической части графовой модели характеризуется рассмотрением некоторых порождающих процессов, тем или иным способом описывающих функционирование модели и позволяющих (не всегда конструктивно) связать с моделью набор инвариантов (инвариантных свойств), различающих классы эквивалентности. Например, при представлении в виде графовой модели некоторого

класса программ, реализующих некоторые функции (а это, как правило, так), имеет место естественное и наиболее общее (слабое) определение эквивалентности, требующей лишь равенства функций, вычисляемых программами, — так называемая *функциональная эквивалентность*. Однако в силу известных результатов об алгоритмической неразрешимости распознавания любого внутреннего (инвариантного относительно функциональной эквивалентности) свойства программ в любом достаточно содержательном (например вычисляющем все рекурсивные функции) классе программ рассматриваются более сильные отношения эквивалентности. Основным методом сужения понятия эквивалентности является сравнение не функций, реализуемых программами, а некоторого вида историй их вычисления в процессе выполнения программ. Вид истории может быть произвольной степени детальности, лишь бы по ней однозначно восстанавливался результат выполнения программы. Тем самым программы с совпадающими историями автоматически имеют совпадающие результаты, т.е. являются функционально эквивалентными. Поэтому в графовой модели, представляющей класс программ, обычно используется некоторый вид таких историй.

Второй подход к заданию семантической части графовой модели — это задание системы преобразований графовых моделей, сохраняющих эквивалентность (так называемых *эквивалентных* преобразований). При этом желательно иметь систему преобразований, полную в том смысле, что любая пара эквивалентных моделей может быть сведена одна к другой с помощью этой системы. Однако далеко не всегда конечные полные системы преобразований существуют, поэтому рассматриваются и такие системы эквивалентных преобразований, которые не обладают свойством полноты. Как правило, используемые на практике системы эквивалентных преобразований не обладают также свойством Черча-Россера, позволяющим не следить за порядком применения преобразований.

3.4. Вопросы визуализации и визуальной обработки

Существующие системы визуализации графов можно условно разделить на два класса.

К первому (более широкому) классу относятся узкоспециальные системы, которые ориентированы на графовые модели с определенной семантикой и топологией. Каждая такая система является, как правило, частью некоторого большого проекта, включающего ее в качестве ви-

зуализатора каких-либо специфических для данного проекта данных. Более широкое использование таких систем либо очень затруднено, либо просто невозможно.

Второй класс — это универсальные системы визуальной обработки графовых моделей, такие как, например, системы daVinci [16], GraphEd [21], Graphlet [13], GraVis [17], VCG [14], а также библиотеки LEDA [15], AGD [20], ffGraph [19], Graph Layout Toolkit, Graph Editor Toolkit [18] и др. Несмотря на значительный прогресс в создании универсальных систем, следует отметить ряд недостатков подавляющего большинства из них. Прежде всего для российского пользователя общим недостатком этих систем является их ориентация на работу в ОС UNIX на больших рабочих станциях, которыми в достаточной степени оснащены иностранные университеты, их разрабатывающие. Как правило, универсальность существующих систем весьма ограничена, в частности, ни одна из них не позволяет графовым моделям быть иерархическими и не поддерживает визуализацию алгоритмов их обработки.

Требования к универсальным системам визуализации и визуальной обработки информационных моделей сложны для формализации и нуждаются в отдельном исследовании. Отметим лишь, что такие системы должны обладать базовыми возможностями поддержки визуальной обработки, связанной с решением широкого круга задач анализа и синтеза иерархических графовых моделей в их конструкторском, исследовательском и учебном применении.

Система должна поддерживать визуализацию и редактирование помеченных иерархических графов. В частности, при изображении графовой модели тип ее элементов может быть связан с определенной геометрической формой соответствующих представлений и/или их цветовой гаммой, а также местом и способом представления пометок, относящихся к элементам соответствующего типа. Система должна обладать современным графическим интерфейсом, ориентированным на иерархичность графовых моделей и поддерживающим их многооконное редактирование и визуализацию с возможностями регулирования масштаба, использования разных геометрических и цветовых образов для формирования изображения графа, работы с прямоугольной сеткой для выравнивания объектов и т.п.

Система должна обладать рядом возможностей, облегчающих и автоматизирующих отдельные операции конструирования, визуализации и изучения различных объектов и явлений в рамках их иерархических

графовых моделей, включая возможности отката (функция Undo), генерации случайных графов, автоматического расположения графов на плоскости, анимации процессов обработки графовых моделей и специализации системы.

При использовании правил преобразований для задания семантической части графовой модели возникают вопросы поддержки визуальной обработки системы преобразований, связанные с их визуальным конструированием, применением и анализом. В частности, весьма важной является возможность поддержки управляемого визуального применения системы преобразований, позволяющего в каждый момент задавать, какое преобразование и каким образом нужно применять к текущему виду модели, а также возвращаться к виду модели, имеющемуся у нее до преобразования. Нужно подчеркнуть важную роль вопросов визуальной обработки систем преобразований и для случая явного описания семантической части графовой модели. Многие из порождающих процессов удобно не рассматривать как неделимое глобальное преобразование, а задавать с помощью системы локальных преобразований. Именно такая ситуация, например, имеет место для сетей Петри, функционирование которых описывается в терминах локальных преобразований, представляющих срабатывания переходов.

4. СИСТЕМЫ ВИЗУАЛИЗАЦИИ И ВИЗУАЛЬНОЙ ОБРАБОТКИ HIGRES И VEGRAS

4.1. Система HIGRES

Система HIGRES — универсальный визуализатор и редактор иерархических графовых моделей [10, 22, 23]. Основным отличием данной системы от ее аналогов является возможность сохранять во внутреннем представлении и визуализировать не только сам граф, но и его семантику, представленную в виде системы атрибутов вершин, фрагментов и дуг графа и библиотекой алгоритмов обработки — так называемых внешних модулей. При этом пользователь может корректировать и доопределять семантику графа с помощью введения новых атрибутов и новых внешних модулей. Такой подход обеспечивает, с одной стороны, универсальность системы, с другой — возможность ее специализации.

Система работает под Microsoft Windows 95/98/NT, что выгодно отличает ее от западных аналогов, которые в подавляющем большинстве ориентированы на использование графической оболочки XWindow опе-

рационной системы UNIX. Microsoft Windows, имея предпочтительные графические возможности, гораздо шире распространена на территории России и других стран бывшего СССР. Достоинствами системы HIGRES являются качественный интуитивный интерфейс, стандартный для Windows-приложений такого типа, широкий набор графических средств визуализации, а также наличие дополнительных средств, облегчающих и автоматизирующих процесс редактирования графа.

В настоящий момент не существует общепринятого формата файлов для хранения графов, тем более иерархических, поэтому в системе HIGRES используется свой внутренний формат. Проблема конвертации решается с помощью специальной библиотеки HGL, написанной на языке C++. С ее помощью внутри любой программы можно создать иерархический граф и записать его в файл того формата, который воспринимается системой. Таким образом, HGL служит интерфейсным звеном между программами, генерирующими иерархические графы, и системой HIGRES, выступающей в роли их визуализатора.

При визуализации автоматически сгенерированных графов всегда встает проблема их автоматического расположения на плоскости. Это расположение в общем случае должно быть наиболее удобным и наглядным для пользователя, т. е. пользователь должен получать из него налучшее представление о структуре графа. В зависимости от семантики графа к этому требованию могут добавляться и более специфические, например можно потребовать расположить рядом какую-нибудь пару семантически связанных вершин. Проблема автоматического расположения графа на плоскости решается с помощью так называемых методов рисования графов [27]. В систему HIGRES включено несколько таких методов.

Система позволяет создавать качественные изображения графов и записывать их в файлы форматов Windows BMP и PCX. Последний формат может служить для создания иллюстративного материала для книг и статей, подготовленных в системе TeX. Важной особенностью системы HIGRES является поддержка ею визуальной обработки иерархических графовых моделей. Для этого в системе имеется возможность визуализации процесса исполнения любого ее внешнего модуля и предусмотрены средства, позволяющие легко расширять систему библиотеками модулей, создаваемыми пользователями в соответствии со своими индивидуальными потребностями.

4.2. Визуализация графовой модели и интерфейс в системе HIGRES

Визуализация графа в системе HIGRES организована таким образом, чтобы максимально наглядно изобразить как иерархическую структуру графа, так и его семантику.

Каждому фрагменту графа соответствует некоторый прямоугольник плоскости, внутри которого располагаются все его вершины (см. рис. 2). Кроме того, для каждого фрагмента можно открыть отдельное окно, в котором видны только вершины данного фрагмента и его подфрагменты. При этом каждый подфрагмент можно объявить закрытым — тогда изображаются только его контуры, либо открытым — тогда изображаются все его вершины и инцидентные им дуги. Для изображения контуров фрагментов в системе используется прием создания эффекта тени. Закрытые фрагменты выглядят слегка выступающими вверх, как будто они закрыты крышками, открытые же слегка утоплены вниз.

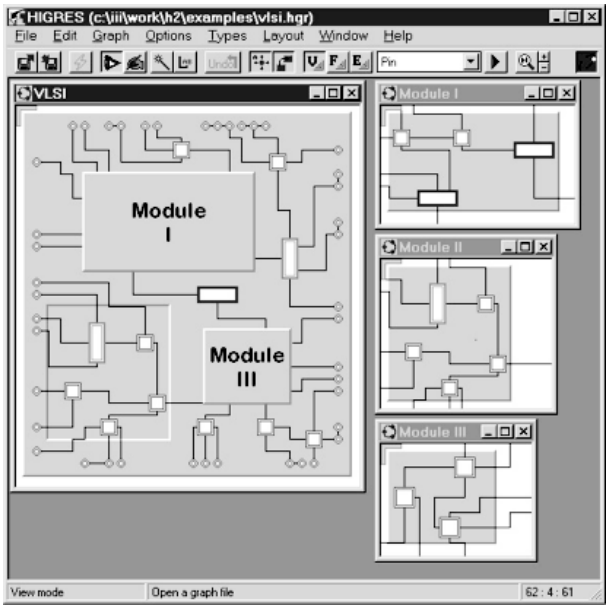


Рис. 2. Модель СБИС, представленная иерархическим графом

Для изображения вершин используется несколько вариантов геометрических фигур. Кроме того, можно выбирать стиль и цвет контура и заливки вершины. Эти параметры задаются отдельно для каждого типа вершин. Таким образом, вершины, принадлежащие к одному типу, визуально похожи друг на друга, хотя могут отличаться текстом меток. Ребра (дуги) графа изображаются либо ломаными линиями, задаваемыми точками сгиба, либо гладкими кривыми, для задания которых также используется некоторый набор точек на плоскости, вдоль которых проходит дуга (эти точки по аналогии также называются точками сгиба). Вариант изображения дуги, стиль ее линии и стиль стрелки на конце (если граф ориентированный) задается отдельно для каждого типа дуг. Для визуализации атрибутов объектов (вершин, ребер, фрагментов) используется гибкая система, позволяющая изображать их не просто как список значений меток, а более наглядным и естественным образом.

Для каждого типа объектов задается "строка визуализации", представляющая собой шаблон текста, изображающего атрибуты каждого объекта данного типа. В этой строке определяются места, в которые при визуализации подставляются значения меток конкретного объекта. Кроме того, строка может содержать специальные символы, отмечающие места начала новой строки. В конечном итоге для каждого объекта определяется некоторый текст, который называется текстом меток. Более точно, для типов вершин задаются две строки визуализации и соответственно два текста меток: внутренний и внешний. При визуализации вершины внутренний текст меток располагается внутри нее, а внешний — рядом с ней. Точно так же для типов фрагментов задаются две строки визуализации, порождающих "закрытый" и "открытый" текст меток. Первый изображается внутри фрагмента, когда он закрыт, а второй — когда открыт. Единственная строка визуализации, задаваемая для каждого типа дуг, порождает текст меток для каждой дуги данного типа, который располагается рядом с одной из точек сгиба дуги либо рядом с одной из конечных точек данной дуги.

Интерфейс системы придерживается основных общепринятых стандартов для приложений Windows 95. Система имеет главное окно, внутри которого расположены окна фрагментов, которые пользователь может открывать и закрывать по мере надобности, переходя вверх и вниз по иерархии. Главное окно системы содержит меню и toolbar, обеспечивающий быстрый доступ к часто используемым операциям. Пользователь может переключаться с режима просмотра на режим редактирования графа.

В режиме просмотра можно только открывать и закрывать фрагменты графа и их окна, просматривая содержимое, скроллировать изображение в окнах и изменять его масштаб.

В режиме редактирования левая кнопка мыши служит для выделения объектов, а правая — для высвечивания всплывающего меню, с помощью которого можно выбрать операцию, производимую с выделенным объектом. Кроме того, выбрав соответствующий пункт в этом меню, можно добавить в граф новую вершину, фрагмент или дугу. С помощью левой кнопки мыши пользователь может перемещать вершины, фрагменты и сгибы дуг, а также изменять размеры вершин и границы фрагментов. При этом никакие две вершины не должны накладываться друг на друга и каждый фрагмент должен целиком включать в себя все свои вершины и подфрагменты. По выбору пользователя система может поддерживать выполнение данного условия одним из двух способов. Она может либо автоматически слегка корректировать расположение графа после каждого его изменения пользователем, нарущающего данное условие, либо просто запретить любые такие изменения.

Существуют также два дополнительных режима редактирования, использование которых может ускорить выполнение некоторых часто производимых операций, — режим создания объектов и режим редактирования меток.

4.3. Визуальная обработка графов в системе HIGRES

С интерфейсной точки зрения процесс визуальной обработки иерархического графа в системе HIGRES выглядит следующим образом. Пользователь с помощью системы выбирает нужный ему внешний модуль и запускает его. Система передает текущий граф обрабатывающему модулю и открывает специальное окно, предоставляющее пользователю интерфейс для управления работой модуля. Пользователь может регулировать параметры обработки, прерывать ее на любом шаге, просматривать промежуточные результаты в любую сторону в форме анимации либо в покадровом режиме. Для каждого модуля определяется набор числовых и текстовых параметров, которые пользователь может изменять в процессе работы. Кроме того, модуль может сам инициировать запрос необходимых ему данных, а также генерировать сообщения, которые вместе с другой рабочей информацией записываются в протокол, выдаваемый на экран.

От программиста, создающего свой собственный внешний модуль, не требуются знания деталей внутреннего представления иерархических графов в системе, поскольку все взаимодействие внешнего модуля с системой обеспечивается функциями библиотеки HGL. Создавая модуль, программист должен позаботиться, кроме программирования непосредственно самого процесса обработки, лишь о разбиении этого процесса на шаги, число которых не фиксируется. Внешний модуль представляет собой отдельное Windows-приложение и может быть получен с использованием любого компилятора C++ для Windows, понимающего шаблоны классов. В настоящий момент для иллюстрации возможностей визуальной обработки создано несколько примеров графовых моделей и внешних модулей к ним. К этим моделям относятся конечные автоматы (см. рис. 3), сети Петри и простейшие схемы программ.

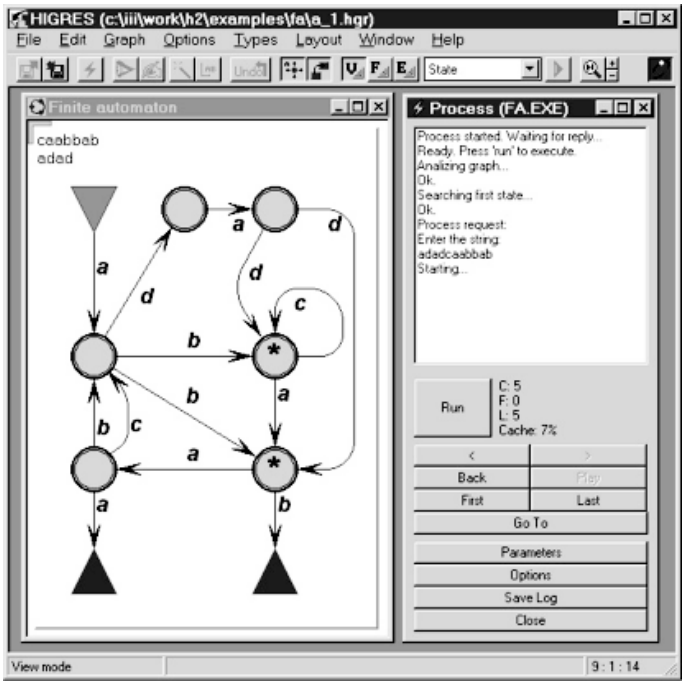


Рис. 3. Моделирование работы конечного автомата

4.4. Система VEGRAS

Система VEGRAS — это универсальный и простой в использовании редактор атрибутированных графов, в том числе иерархических, ориентированный на поддержку подготовки качественных штриховых иллюстраций в рамках среды Windows 95/98/NT. Система написана на языке C++ с использованием компилятора Microsoft Visual C++ Version 4.2 и библиотеки MFC.

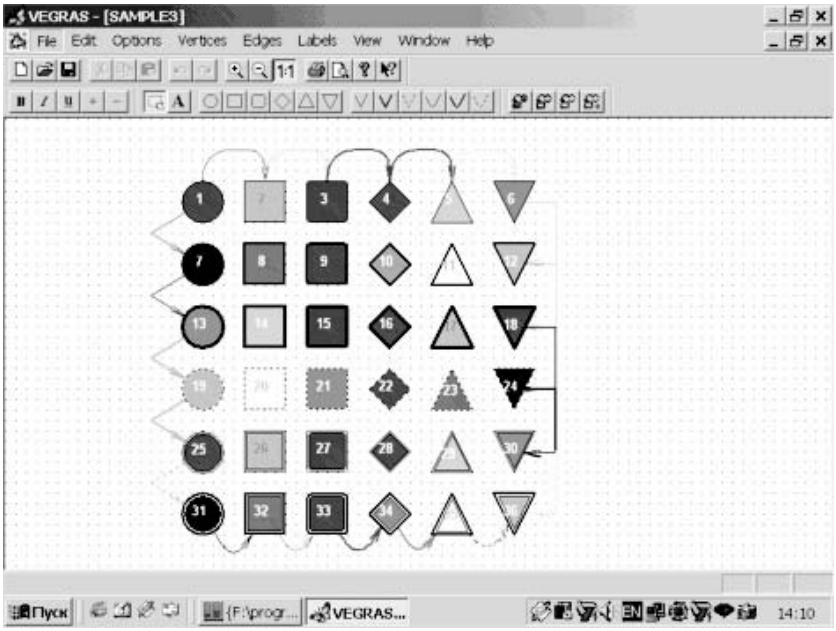


Рис. 4. Система VEGRAS

Вершины графа (см. рис. 4) изображаются в виде различных геометрических фигур, таких как эллипс, прямоугольник, скругленный прямоугольник, ромб, треугольник или перевернутый треугольник. Кроме этого, каждая вершина имеет каемку (простую, жирную, очень жирную, пунктирную или двойную). Цвет каемки выбирается отдельно. Ребра и дуги графа изображаются либо ломаными линиями, задаваемыми точками сгиба, либо гладкими кривыми, для задания которых используется некоторый набор точек на плоскости, которые также на-

зываются точками сгиба. Линии, изображающие ребра и дуги графа, могут различаться по толщине, цвету виду (сплошные, пунктирные) и могут снабжаться стрелками. Число точек сгиба практически не ограничено.

Фрагменты в системе представлены своими границами, имеющими вид "замкнутых" ребер. Все вершины и ребра графа, а также границы фрагментов могут иметь произвольное количество меток, место расположения которых пользователь может изменять. Цвет и шрифт можно задавать отдельно для каждой метки. Метки могут иметь верхние и нижние многоэтажные индексы, что весьма полезно при создании иллюстраций к научным публикациям.

В процессе редактирования графа можно изменять размеры, форму, цвет вершин и дуг, перемещать и удалять вершины, дуги, метки или точки сгиба ребер и дуг, копировать выделенные объекты в буфер (операции Cut/Copy/Paste), добавлять и удалять точки сгиба дуги. Можно выделить группу объектов графа и производить какие-либо действия сразу над всей группой (изменять цвета, форму, добавлять метки). При этом те операции, которые не добавляют и не удаляют вершины и дуги графа, меняют лишь изображение графа, а не сам граф. Причем, само изображение графа при таких операциях сохраняет определенные свойства, связанные с наглядностью. В частности, при перемещении вершин, дуг или точек сгиба дуги, соответствующие им метки тоже передвигаются так, чтобы их относительное месторасположение не изменялось. Полезным свойством является наличие операций Undo/Redo, позволяющих отменить или повторить до 32 последних действий (причем эти действия могут быть сложными, как, например, удаление группы объектов или изменение каких-либо параметров сразу для всей группы).

В системе VEGRAS реализованы операции ZoomIn/ZoomOut, позволяющие менять масштаб изображения от 1 до 1000%, и имеется возможность использовать прямоугольную сетку, внешний вид и параметры которой можно изменять. VEGRAS позволяет записывать изображения графов в файлы форматов Windows BMP и PCX, а также обеспечивает возможность вставки построенных в системе изображений графов в документы других приложений, поддерживающих механизм OLE, таких как Microsoft Word и Excel. Кроме этого, система VEGRAS воспринимает формат файлов системы HIGRES, а также поддерживает конвертацию построенного изображения атрибутированного иерархического

графа в формат системы HIGRES.

5. ТОЛКОВЫЙ СЛОВАРЬ ПО ТЕОРИИ ГРАФОВ ДЛЯ ПРОГРАММИСТОВ

Проблема терминологии является одной из основных проблем в применении теоретико-графовых методов в программировании и информатике. Терминология в теории графов пока еще не устоялась, при написании статей требуется терминологическая привязка к одной из существующих на русском языке монографий, что является довольно трудным делом из-за сокращения числа издающихся книг, в том числе переводных, и резкого сокращения их тиража.

Недавно опубликованный словарь [11], предварительная версия которого была издана в 1995–96 гг. тремя выпусками в Новосибирском государственном университете, призван если не решить, то хотя бы значительно облегчить эту проблему. В словаре собраны термины, использованные в таких известных монографиях по теории графов, как книги Ф. Харари, К. Бержа, О. Оре, А.А. Зыкова и др., а также в доступных для отечественного читателя публикациях по информатике и программированию с указанием источника и вариантов. Статьи словаря снабжены иллюстрациями, перекрестными ссылками и ссылками на доступную литературу. Русские термины сопровождаются их английскими эквивалентами, что позволяет использовать книгу как русско-английский словарь, а прилагаемый англо-русский словарь может помочь при чтении англоязычной литературы. Последнее, на наш взгляд, позволит сократить размножение вариантов используемых в литературе терминов. Кроме собственно теоретико-графовых терминов в книгу включены необходимые для понимания термины из программирования, комбинаторного анализа, прикладной алгебры и исследования операций, что расширяет круг пользователей словаря.

При отборе терминов авторы словаря исходили из следующих соображений. В качестве основного было выбрано множество понятий, представленных в известной монографии “Лекции по теории графов” [12], как наиболее полного и доступного для отечественного читателя издания по теории графов. Затем оно пополнялось терминами из переводных и других отечественных книг по теории графов, а также монографий по информатике и программированию, существенно использующих методы теории графов. Чтобы как-то уменьшить разрыв между терминологией монографий и терминологией, используемой в статьях и еще

не успевшей попасть в монографии, словарь был расширен за счет тех терминов, которые встречаются в докладах на ежегодной конференции “Graph Theory Concepts in Computer Science” и в статьях, опубликованных в ведущих по данной тематике журналах “Discrete Mathematics”, “J. Graph Theory” и др. В последнем случае при включении того или иного термина в словарь делалась лишь общая ссылка на название журнала или конференции в соответствующей статье словаря, но не добавлялась ссылка в список литературы в конце словаря на конкретный номер журнала или труды конкретной конференции.

Создана начальная Web-версия словаря (см. рис. 5), поддерживающая взаимодействие с пользователем с помощью HTML-форм со встроенными сценариями на языках Java и C++. Указанная система получила название GRAPP. В настоящее время электронный словарь GRAPP соответствует первому изданию словаря и находится на сервере лаборатории по адресу http://pco.iis.nsk.su/grapp.

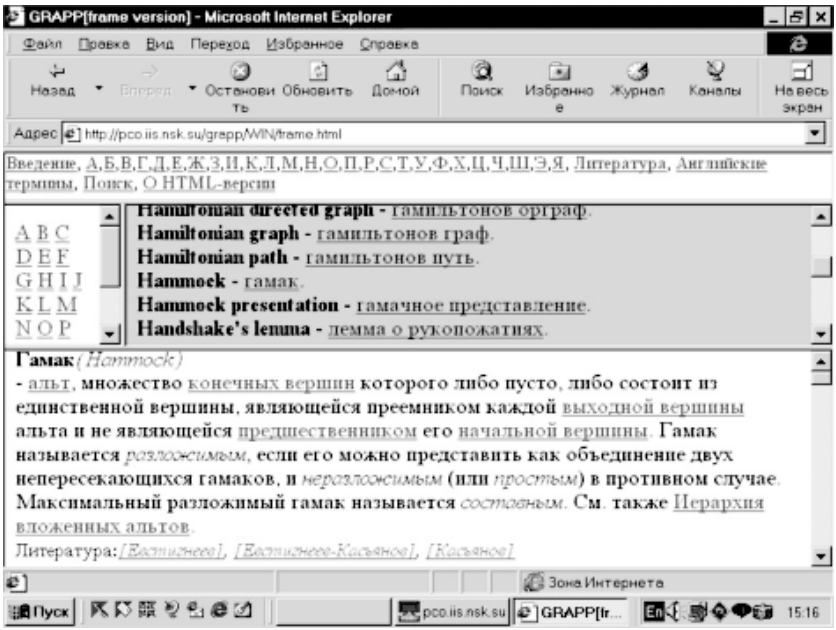


Рис. 5. Система GRAPP

6. ЗАКЛЮЧЕНИЕ

Автор благодарен всем сотрудникам лаборатории конструирования и оптимизации программ ИСИ СО РАН, принимавшим участие в выполнении проектов, рассмотренных в данной статье, в первую очередь проф. В.А. Евстигнееву, а также И.А. Лисицыну , Е.С. Мердишевой, Т.С. Мердишевой и В.Е. Казанцеву.

СПИСОК ЛИТЕРАТУРЫ

1. **Касьянов В.Н.** Оптимизирующие преобразования программ. — М.: Наука, 1988.
2. **Касьянов В.Н., Поттосин И.В.** Методы построения трансляторов. — Новосибирск: Наука, 1986.
3. **Di Battista G., Eades P., Tamassia R., Tollis I.G.** Algorithms for drawing graphs: an annotated bibliography // Comput. Geom. Theory Appl. — 1994. — Vol. 4. — P. 235—282.
4. **Кнут Д.** Искусство программирования для ЭВМ. — М.: Мир; Том 1: Основные алгоритмы, 1976; Том 2: Получисленные алгоритмы, 1977; Том 3: Сортировка и поиск, 1978.
5. **Евстигнеев В.А., Касьянов В.Н.** Теория графов: алгоритмы обработки деревьев. — Новосибирск: Наука, 1994.
6. **Евстигнеев В.А., Касьянов В.Н.** Теория графов: алгоритмы обработки бесконтурных графов. — Новосибирск: Наука, 1998.
7. **Евстигнеев В.А., Касьянов В.Н.** Сводимые графы и граф-модели в программировании. — Новосибирск: Изд-во ИДМИ, 1999.
8. **Kasyanov V.N.** Hierarchical graphs and visual processing // International Congress of Mathematicians (ICM98): Abstracts of Short Communications and Poster Sessions. — Berlin, 1998. — P. 292.
9. **Касьянов В.Н.** Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. — Новосибирск: ИСИ СО РАН, 1999. — С. 7–32.
10. **Лисицын И.А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей // Там же. — С. 64–77.
11. **Евстигнеев В.А., Касьянов В.Н.** Толковый словарь по теории графов в информатике и программированиию. — Новосибирск: Наука, 1999.
12. **Лекции** по теории графов / В.А. Емеличев, О.И. Мельников, В.И. Сарванов, Р.И. Тышкова. — М.: Наука, 1990.
13. **Himsolt M.** The Graphlet system (system demonstration) // Lect. Notes Comput. Sci. — 1996. — Vol. 1190. — P. 233–240.
14. **Sander G.** Graph layout through the VCG tool // Lect. Notes Comput. Sci. — 1994. — Vol. 894.— P. 194—205.
15. **Mehlhorn K., Näher S.** LEDA: a platform for combinatorial and geometric computing // Communs. ACM. — 1995. — Vol. 38. — P. 96–102.
16. **Fröhlich M., Werner M.** Demonstration of the interactive graph visualization system daVinci // Lect. Notes Comput. Sci. — 1994. — Vol. 894. — P. 266–269.

17. **Lauer H., Ettrich M., Soukup K.** GraVis — System Demonstration // Lect. Notes Comput. Sci. — 1997. — Vol. 1353. — P. 344–349.
18. **Madden B., Madden P., Powers S., Himsolt M.** Portable Graph Layout and Editing // Lect. Notes Comput. Sci. — 1995. — Vol. 1027.— P. 385–395.
19. **Информация** о библиотеке ffGraph доступна по адресу: http://www.fmi.uni-passau.de/~friedric/ffgraph/main.shtml
20. **Информация** о библиотеке AGD доступна по адресу: http://www.mpi-sb.mpg.de/~mutzel/dfgdraw/agdlib.html
21. **Himsolt M.** GraphEd: A graphical platform for the implementation of graph algorithms (extended abstract and demo) // Lect. Notes Comput. Sci. — 1994. — Vol. 894.— P. 182–193.
22. **Kasyanov V.N., Lisitsyn I.A.** On support tools for visual processing of hierarchical graph models // Joint Bull. Nov. Comput. Center and A.P. Ershov Inst. Informatics Sys. , Ser.: Comp. Science. — 1999.— Vol. 12. — P. 19–23.
23. **Kasyanov V.N., Lisitsyn I.A.** Hierarchical graph models and visual processing // Proc. 16th IFIP Cong. — Beijing, 2000.— P. 179–182.
24. **Feng Q.W., Cohen R.F., Eades P.** Planarity for clustered graphs // Lect. Notes Comput. Sci. — 1995. — Vol. 979.—P. 213–226.
25. **Harel D.** On visual formalism // Commun. ACM. — 1988.— Vol. 31, N 5.—P. 514–530.
26. **Sugiyama K., Misue K.** Visualization of structured digraphs // IEEE Trans. on Systems, Man and Cybernatics. — 1991. — Vol. 21, N 4. — P. 876–892.
27. **Di Battista G., Eades P., Tamassia R., Tollis I.G.** Graph drawing: Algorithms for vizualization of graphs. — Prentice Hall, 1999.

В. А. Бояршинов

ЭКВИВАЛЕНТНОСТЬ МОДЕЛЕЙ ЛОКАЛЬНЫХ ВЫЧИСЛЕНИЙ[1]

В настоящее время существует несколько моделей локальных вычислений на графах: системы переписывания графов с приоритетом [1], системы переписывания графов с запрещенными контекстами [2], локальные алгоритмы Журавлева [3, 4], (см. также [5]), сети конечных автоматов [7]. Представляет интерес вопрос сравнения классов задач, разрешимых за полиномиальное время в различных моделях локальных вычислений.

Известно [2], что модели переписывания графов с запрещенными контекстами и переписывания графов с приоритетом эквивалентны между собой. В настоящей работе доказывается, что класс задач, разрешимых за полиномиальное время с помощью систем переписывания графов с запрещеными контекстами, совпадает с классом задач, разрешимых за полиномиальное время с помощью локальных алгоритмов Журавлева и сетей конечных автоматов.

Статья является продолжением работ автора [8,9].

Предложение 1. Пусть проблема K разрешима над семейством графов $I(P)$ с помощью системы переписывания графов с запрещенными контекстами S_0 за время T_0. Тогда существует конечный автомат ω_0 такой, что проблема K разрешима над семейством графов $I(P)$ с помощью конечного автомата ω_0 за время $O(V(G) \cdot T_0)$.

Доказательство. В сети конечных автоматов моделируется повторяющийся обход графа фишкой. Каждый раз, как фишка меняет свою позицию, инициируется процесс проверки: содержит ли окрестность первого порядка фишки вхождение левой части некоторого правила переписывания из S. Если такое вхождение есть, то производится изменение состояний автоматов окрестности согласно правилу переписывания. Затем обход графа сети продолжается. Алгоритм завершает свою работу и сеть переходит в стационарное состояние, когда в течение некоторого обхода графа сети фишка не обнаружит ни одного вхождения левой части правила из S.

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

В левой части каждого правила из S выделим вершину, которую в дальнейшем будем называть **центральной**. Центральная вершина правила выделяется согласно следующим правилам:

• Если левая часть правила является звездой с двумя и более лучами, то в качестве центральной вершины выбирается всесмежная вершина (центр) звезды.

• Если левая часть правила представляет собой ребро, то в качестве центральной вершины правила выбирается произвольный конец этого ребра.

• Если левая часть правила представляет собой одновершинный граф, то в качестве центральной вершины выбирается вершина правила.

Пусть L_V — алфавит вершинных меток, L_E — алфавит реберных меток системы переписывания с запрещенными контекстами S; $N(S)$ — число правил переписывания в S; $q(G)$ — число вершин графа G; $Q(S)$ — максимальное число вершин в левой части правила переписывания из S.

Работу системы переписывания с запрещенными контекстами S моделирует конечный автомат $R_2 = (E_2, L_2, \psi_2)$, где

$$E_2 = L_2 = \{(s,t,a,b,c,d,e) \mid \begin{array}{l} s \in \{0,1,\Delta\}, t \in \{A,B,CWait,\Delta\}, a \in L_V, \\ b \in L_E, c \in \{\Delta, cktar_1 \ldots, clear_N, c_1, \ldots, c_N, \\ \qquad draw_1, \ldots, draw_N, ver_1, \ldots, ver_N\}, \\ d \in \{\Delta, ok, no, 1, \ldots, Q\}, e \in \{yes, no\}\}. \end{array}$$

Первые две позиции вектора состояния ветви автомата предназначаются для моделирования обхода в глубину графа сети; третья и четвертая позиции предназначаются для хранения меток системы переписывания S; пятая и шестая предназначены для проверки наличия вхождения левой части допустимого правила переписывания и организации непосредственно процесса переписывания допустимого вхождения; седьмая позиция предназначена для запоминания факта переписывания вхождения некоторого правила во время последнего обхода графа сети.

Функция изменения состояния автомата ψ_2 определяется множеством правил изменения состояний, которое состоит из двух групп.

Первая группа правил — правила, моделирующие обход графа сети с учетом двух замечаний: всякий раз, изменяя позицию активной вершины, передаем управление второй группе правил:

$$e^2_{i,t+1} := Wait, \forall i; e^5_{j,t+1} := c1, \forall j;$$

кроме того, производится еще один обход графа сети в том и только том случае, если в седьмой позиции состояния фишки, вернувшейся в начальную вершину, стоит отметка о моделировании применения некоторого правила переписывания в течение последнего обхода.

Вторая группа правил — правила, отвечающие за поиск и переписывание допустимого вхождения, — строится следующим образом.

Занумеруем вершины левых частей всех правил номерами из $\{1, \ldots, q(\lambda D_i)\}$. При этом центральной вершине правила приписывается номер 1; все остальные вершины нумеруются произвольным образом.

Пусть λD — помеченная звезда на n вершинах $\{v_1, \ldots, v_n\}$, где вершина v_1 смежна со всеми остальными вершинами. Сопоставим графу λD пропозициональную формулу

$$\begin{aligned} P(\lambda D) = {} & (\forall_j (e^3_{j,t} = \lambda(v_1))) \wedge (\exists i_1, \ldots, i_{n-1}((l^3_{i_1,t} = \lambda(v_2)) \wedge \\ & (l^4_{i_1,t} = \lambda((v_1, v_2))) \wedge \ldots \wedge (l^3_{i_{n-1},t} = \lambda(v_n)) \wedge \\ & (l^4_{i_{n-1},t} = \lambda((v_1, v_n))))), \end{aligned}$$

которую будем называть **характеристической формулой** графа λD.

Для каждого правила $r_i = (\lambda_i D_i, \lambda'_i D_i, F_{i,l}, \ldots, F_{i,k_i}), i \neq N$, добавим в ψ_2 правила распознавания и переписывания вхождения λD_i, а именно:

• Если левая часть правила r_i содержит вершины $\{v_1, \ldots, v_{q(\lambda D_i)}\}$, $q(\lambda D_i) \geq 3$, то добавляются правила

◇ $(e^5_{j,t} = ci, \forall j) \wedge (\neg P(\lambda D_i) \vee P(F_{i,l} \vee \ldots \vee P(F_{i,k_i})) \quad \Longrightarrow$
$e^5_{j,t+1} := c(i+1), \forall j$
(если нет вхождения левой части i-го правила или оно включено в запрещенный контекст, то переходим к поиску вхождения следующего правила);

◇ $(e^5_{j,t} = ci, \forall j) \wedge \neg P(F_{i,l} \wedge \ldots \wedge \neg P(F_{i,k_i}) \wedge$
$(\exists j_1, \ldots j_{q(D_i)-1}((l^3_{j_1 t} = \lambda_i(v_2)) \wedge$
$(l^4_{j_1,t} = \lambda_i((v_1, v_2))) \wedge \ldots \wedge (l^3_{j_{q(D_i)-1} t} = \lambda_i(v_{q(D_i)})) \wedge$
$(l^4_{jq(D_i)-1,t} = \lambda_i((v_1, v_{q(D-i)})))) \quad \Longrightarrow$
$e^5_{k,t+1} := drawi, \forall k; e^6_{j1,t+1} := 2, e^6_{j_2 t+1} := 3, \ldots, e^6_{j_{q(D_i-1},t+1} := q(D_i)$
(если существует вхождение левой части i-го правила и оно не включено ни в один из запрещенных контекстов, то даем соответствующим вершинам команду изменить свое состояние согласно i-му правилу переписывания);

◇ $\exists j (l^5_{j,t} = drawi) \wedge (l^6_{j,t} = k) \quad \Longrightarrow$
$e^3_{m,t+1} := \lambda'_i(v_k), \forall m; e^4_{j,t+1} := \lambda'_i((v_1, v_k))$

(если существует соседний автомат, который дает команду переписать вхождение правила r_i, при этом сообщает, что нужно сопоставить себя вершине с номером k, то изменяем свое состояние так же, как правило переписывания r_i изменяет метку k-й вершины и ребра (v_1, v_k));

$\diamond$ $\forall k(e^5_{k,t+1} = drawi \wedge \exists j_1, \ldots, j_{q(D_i)-1}((e^6_{j_1,t} = 2) \wedge, \ldots, \wedge(e^6_{j_{q(D_i)-1},t} = q(D_i))) \quad \Longrightarrow$

$e^3_{k,t+1} := \lambda'_i(v_1), \forall k;$ $\quad e^6_{k,t+1} := \Delta, \forall k;$

$e^4_{j_1,t+1} := \lambda'_i((v_1, v_2)),$ $\quad \ldots,$

$e^4_{j_{q(D_i)-1},t+1} := \lambda'_i((v_1, v_{q(D_i)}));$ $\quad e^5_{k,t+1} := \Delta, \forall k;$

$e^7_{k,t+1} := yes, \forall k;$ $\quad e^2_{k,t+1} := A, \forall k$

(отдав команду начать переписывание, центральная вершина правила изменяет свое состояние согласно правилу r_i; отмечает, что в течение последнего обхода графа сети было переписано по крайней мере одно вхождение и разрешает продолжить обход графа).

• Если левая часть правила r_i содержит $q(\lambda D_i) = 1$ вершину, то производятся следующие преобразования. Пусть $F_{i,l}, \ldots, F_{i,m}$ — запрещенные контексты для r_i, в которых центральная вершина правила сопоставлена вершине, отличной от центра звезды (в частности, $q(F_{i,j} \geq 3, \forall i \in \{1, \ldots, m\})$; $F_{i,m+1}, \ldots, F_{i,k_i}$ — все остальные запрещенные контексты. Занумеруем вершины графов $F_{i,l}, \ldots, F_{i,m}$ таким образом, чтобы центр звезды в каждом контексте получил номер 1, а вершина, которой сопоставлена центральная вершина правила, получила номер 2; вершины графов $F_{i,m+1}, \ldots, F_{i,k_i}$ нумеруются таким образом, чтобы номер 1 получила вершина, которой сопоставлена центральная вершина правила. В определение ψ_2 добавляются следующие правила:

$\diamond$ $(e^5_{j,t} = ci, \forall j) \wedge (\neg P(\lambda_i(D_i)) \vee P(F_{i,m+1}) \vee \ldots \vee P(F_{i,k_i}) \quad \Longrightarrow$

$e^5_{j,t+1} := c(i+1), \forall j$

(если нет вхождения левой части i-го правила или оно включено в запрещенный контекст, то переходим к поиску вхождения следующего правила);

$\diamond$ $(e^5_{j,t} = ci, \forall j) \wedge P(\lambda(D_i)) \wedge \neg P(F_{i,m+1}) \wedge \ldots \wedge \neg P(F_{i,k_i}) \quad \Longrightarrow$

$e^6_{j,t+1} := 1, \forall j; e^5_{j,t+1} := ver_i, \forall j$

(если существует вхождение левой части правила r_i и оно не включено ни в один из запрещенных контекстов среди $F_{i,m+1}, \ldots, F_{i,k_i}$, то переходим к проверке, не включено ли это вхождение в некоторый запрещенный контекст среди $F_{i,l}, \ldots, F_{i,m}$);

- $\exists j \quad ((l^6_{j,t} = 1) \wedge (l^5_{j,t} = ver_i) \wedge ((P(F_{i,l}) \wedge (l^3_{j,t} = \lambda_{F_{i,1}}(v_2))$
  $\wedge (l^4_{j,t} = \lambda_{F_{i,l}}((v_1, v_2)))) \vee \ldots \vee (P(F_{i,m} \wedge (l^3_{j,t} = \lambda_{F_{i,m}}(v_2))$
  $\wedge (l^4_{j,t} = \lambda_{F_{i,m}}((v_1, v_2))))) \quad \Longrightarrow$
  $e^6_{j,t+1} := yes$
  (если сосед-автомат запросил проверить, не включен ли он в некоторый запрещенный контекст среди $F_{i,l}, \ldots, F_{i,m}$, и такое включение удалось обнаружить, то сообщить ему об этом);
- $\exists j \quad ((l^6_{j,t} = 1) \wedge (l^5_{j,t} = ver_i) \wedge ((\neg P(F_{i,1}) \vee \neg (l^3_{j,t} = \lambda_{F_{i,1}}(v_2))$
  $\vee \neg (l^4_{j,t} = \lambda_{F_{i,l}}((v_1, v_2)))) \wedge \ldots \wedge (\neg P(F_{i,m} \vee \neg (l^3_{j,t} = \lambda_{F_{i,m}}(v_2))$
  $\vee \neg (l^4_{j,t} = \lambda_{F_{i,m}}((v_1, v_2))))) \quad \Longrightarrow$
  $e^6_{j,t+1} := no$
  (если сосед-автомат запросил проверить, не включен ли он в некоторый запрещенный контекст среди $F_{i,1}, \ldots, F_{i,m}$, и такого включения не удалось обнаружить, то сообщить ему об этом);
- $\forall j (e^5_{j,t} = ver_i) \wedge \forall j (l^6_{j,t} = no) \quad \Longrightarrow$
  $e^5_{j,t+1} := draw_i, \forall j; e^6_{j,t+1} := \Delta, \forall j$
  (если вхождение правила r не включено ни в один запрещенный контекст среди $F_{i,l}, \ldots, F_{i,k_i}$, то перейти к переписыванию вхождения);
- $\exists k (l^5_{k,t} = draw_i) \quad \Longrightarrow \quad e^6_{k,t+1} := \Delta$
  (если соседний автомат перешел к переписыванию вхождения, занулить вспомогательную информацию);
- $\forall j (e^5_{j,t} = draw_i) \quad \Longrightarrow$
  $e^3_{j,t+1} := \lambda'_i(v_1), \forall j; e^5_{j,t+1} := \Delta, \forall j; e^7_{k,t+1} := yes, \forall k; e^2_{k,t+1} := A, \forall k$
  (переписав вхождение правила, разрешить продолжить обход графа сети);
- $\forall j (e^5_{j,t} = ver_i) \wedge \exists k (l^6_{k,t} = yes) \quad \Longrightarrow$
  $e^5_{j,t+1} := clear_i, \forall j; e^6_{j,t+1} := \Delta, \forall j$
  (если вхождение правила r_i включено в некоторый запрещенный контекст среди $F_{i,l}, \ldots, F_{i,m}$, то дать команду всем смежным автоматам обнулить вспомогательную информацию);
- $\exists k (l^5_{k,t} = clear_i) \quad \Longrightarrow \quad e^6_{k,t+1} := \Delta$
  (получив от соседа команду, занулить вспомогательную информацию);
- $\forall j (e^5_{j,t} = clear_i) \quad \Longrightarrow \quad e^5_{j,t+1} := c_{i+1}$
  (после зануления вспомогательной информации перейти к поиску вхождения следующего правила).

• Если левая часть правила r_i содержит $q(\lambda D_i) = 2$ вершины (т.е.

D_i представляет собой ребро), то произвести следующие преобразования. Пусть $F_{i,1}, \ldots, F_{i,m}$ — запрещенные контексты для r_i, в которых центральная вершина правила сопоставлена вершине, отличной от центра звезды; $F_{i,m+1}, \ldots, F_{i,k_i}$ — запрещенные контексты, в которых центральная вершина правила сопоставлена центру звезды. Занумеруем вершины графов $F_{i,l}, \ldots, F_{i,m}$ таким образом, чтобы центр звезды в каждом контексте получил номер 1, а вершина, которой сопоставлена центральная вершина правила, получила номер 2; вершины графов $F_{i,m+1}, \ldots, F_{i,k_i}$ нумеруются таким образом, чтобы номер 1 получила вершина, которой сопоставлена центральная вершина правила, а номер 2 — вершина, которой сопоставлена вторая вершина из левой части правила. В определении ψ_2 добавляются следующие правила:

- $\diamond$ $(e^5_{j,t} = ci, \forall j) \wedge (\neg P(\lambda_i(D_i)) \vee P(F_{i,m+1} \vee \ldots \vee P(F_{i,k_i})) \quad \Longrightarrow$
  $e^5_{j,t+1} := c(i+1), \forall j$
  (если нет вхождения левой части i-го правила или оно включено в запрещенный контекст, то переходим к поиску вхождения следующего правила);
- $\diamond$ $(e^5_{j,t} = ci, \forall j) \wedge P(\lambda_i(D_i)) \wedge \neg P(F_{i,m+1}) \wedge \ldots \wedge \neg P(F_{i,k_i}) \quad \Longrightarrow$
  $e^6_{j,t+1} := 1, \forall j; e^5_{j,t+1} := ver_i, \forall j$
  (если существует вхождение левой части правила r_i и оно не включено ни в один из запрещенных контекстов среди $F_{i,m+1}, \ldots, F_{i,k_i}$, то переходим к проверке, не включено ли это вхождение в некоторый запрещенный контекст среди $F_{i,1}, \ldots, F_{i,m}$);
- $\diamond$ $\exists j \quad ((l^6_{j,t} = 1) \wedge (l^5_{j,t} = ver_i) \wedge ((P(F_{i,1}) \wedge (l^3_{j,t} = \lambda_{F_{i,1}}(v_2))$
  $\wedge(l^4_{j,t} = \lambda_{F_{i,1}}((v_1, v_2)))) \vee \ldots \vee (P(F_{i,m}) \wedge (l^3_{j,t} = \lambda_{F_{i,m}}(v_2))$
  $\wedge(l^4_{j,t} = \lambda_{F_{i,m}}((v_1, v_2))))) \quad \Longrightarrow$
  $e^6_{j,t+1} := yes$
  (если сосед-автомат запросил проверить, не включен ли он в некоторый запрещенный контекст среди $F_{i,1}, \ldots, F_{i,m}$, и такое включение удалось обнаружить, то сообщить ему об этом);
- $\diamond$ $\exists j \quad ((l^6_{j,t} = 1) \wedge (l^5_{j,t} = ver_i) \wedge ((\neg P(F_{i,1}) \vee \neg(l^3_{j,t} = \lambda_{F_{i,1}}(v_2))$
  $\vee\neg(l^4_{j,t} = \lambda_{F_{i,1}}((v_1, v_2)))) \wedge \ldots \wedge (\neg P(F_{i,m}) \vee (l^3_{j,t} = \lambda_{F_{i,m}}(v_2))$
  $\vee\neg(l^4_{j,t} = \lambda_{F_{i,m}}((v_1, v_2)))) \quad \Longrightarrow$
  $e^6_{j,t+1} := no$
  (если сосед-автомат запросил проверить, не включен ли он в некоторый запрещенный контекст среди $F_{i,1}, \ldots, F_{i,m}$, и такое включение не удалось обнаружить, то сообщить ему об этом);

◇ $\forall j \quad (e^5_{j,t} = ver_i) \wedge \forall k(l^3_{k,t} \neq \lambda'_i(v_2))$
$\vee(l^4_{k,t} \neq \lambda'_i((v_1, v_2))) \vee (l^6_{k,t} = yes) \quad \Longrightarrow$
$e^5_{j,t+1} := clear_i, \forall j; e^6_{j,t+1} := \Delta, \forall j$
(если любое вхождение правила r_i включено в некоторый запрещенный контекст среди $F_{i,1}, \ldots, F_{i,m}$, то дать команду всем смежным автоматам обнулить вспомогательную информацию);

◇ $\exists k(l^5_{k,t} = clear_i) \quad \Longrightarrow \quad e^6_{k,t+1} := \Delta$
(получив от соседа команду, занулить вспомогательную информацию);

◇ $\forall j(e^5_{j,t} = clear_i) \quad \Longrightarrow \quad e^5_{j,t+1} := c_{(i+1)}$
(после зануления вспомогательной информации перейти к поиску вхождения следующего правила);

◇ $\forall j(e^5_{j,t} = ver_i) \wedge \exists k(l^6_{k,t} = no) \wedge (l^3_{k,t} = \lambda_i(v_2)) \wedge (l^4_{k,t} = \lambda_i((v_1, v_2))) \Longrightarrow$
$e^5_{j,t+1} := draw_i, \forall j; e^6_{k,t+1} := 2, e^6_{j,t+1} := \Delta, \forall j \neq k$
(если некоторое вхождение правила r_i не включено ни в один запрещенный контекст среди $F_{i,1}, \ldots, F_{i,k_i}$, то перейти к переписыванию этого вхождения);

◇ $\exists k((l^5_{k,t} = draw_i) \wedge (l^6_{k,t} = \Delta)) \quad \Longrightarrow \quad e^6_{k,t+1} := \Delta$
(если соседний автомат перешел к переписыванию вхождения, не содержащего данный автомат, то занулить вспомогательную иннформацию);

◇ $\exists k((l^5_{k,t} = draw_i) \wedge (l^6_{k,t} = 2)) \quad \Longrightarrow$
$e^6_{k,t+1} := \Delta; e^3_{j,t+1} := \lambda'_i(v_2), \forall j; e^4_{k,t+1} := \lambda'_i((v_1, v_2))$
(если соседний автомат перешел к переписыванию вхождения, содержащего данный автомат, то занулить вспомогательную информацию и изменить состояние автомата согласно правилу переписывания r_i);

◇ $\forall j(e^5_{j,t} = draw_i) \wedge \exists k(e^6_{k,t} = 2) \quad \Longrightarrow$
$e^3_{j,t+1} := \lambda'_i(v_1), \forall j; \quad e^4_{k,t} = \lambda'_i((v_1, v_2)); \quad e^5_{j,t+1} := \Delta, \forall j;$
$e^7_{m,t+1} := yes, \forall m; \quad e^2_{m,t+1} := A, \forall m$
(переписав вхождение правила, разрешить продолжить обход графа сети).

• Построение правил изменения состояния автомата по правилу переписывания r_N производится аналогично тому, как описано выше, за единственным исключением: вместо перехода к поиску вхождений следующего правила возобновляется обход графа сети.

Сеть конечных автоматов R, граф которой $G(R)$ лежит в $I(P)$, и каждой вершине которой сопоставлена копия автомата $\Omega_0 = (E_2, L_2, \psi_2)$,

всегда переходит в стационарное состояние, поскольку моделирует допустимую цепь переписывания системы переписывания с запрещенными контекстами S_0 на графе из $I(P)$.

Поскольку на один обход графа сети требуется $O(V(G))$ тактов работы сети, а во время каждого обхода моделируется применение как минимум одного правила переписывания, то сеть автоматов перейдет в стационарное состояние не более чем через $O(V(G) \cdot N_0)$ тактов работы.

Предложение 2. Пусть проблема K разрешима над семейством графов $I(P)$ с помощью локального параллельного алгоритма Журавлева A_0 за время T_0. Тогда существует конечный автомат Ω_0 такой, что проблема K разрешима над семейством графов $I(P)$ с помощью конечного автомата Ω_0 за время $O(T_0)$.

Доказательство. Пусть $P_1^v, P_2^v, \ldots, P_m^v$ — предикаты, определенные на вершинах графа алгоритмом A_0; $P_1^e, P_2^e, \ldots, P_n^e$ — предикаты, определенные на ребрах графа алгоритмом A_0.

Построим по алгоритму A_0 эквивалентный ему конечный автомат $\Omega_0 = (E_0, \psi_0)$ следующим образом.

Заметим, что между множеством инцидентров графа и множеством ветвей конечного автомата существует естественное взаимооднозначное соответствие; обозначим его через ϕ.

В качестве множества состояний ветвей автомата возьмем множество $E_0 = \{(a_1, \ldots, a_m, b_1, \ldots, b_n) \mid a_i, b_j \in \{1, 0, \Delta\}\}$. Другими словами, состояние ветви автомата есть вектор, являющийся конкатенацией информационных векторов для соответствующей вершины и инцидентного ей инцидентора графа.

Функция изменения состояний автомата Ω_0 строится по функциям изменения значений предикатов алгоритма A_0 следующим образом:

- $e_{(v,b),t+1} = (\varphi_1^v(\phi(S_1((v,b))) \mid_{\phi(v)}), \varphi_2^v(\phi(S_1((v,b))) \mid_{\phi(v)}), \ldots,$
  $\varphi_m^v(\phi(S_1((v,b))) \mid_{\phi(v)}), \varphi_1^e(\phi(S_1((v,b))) \mid_{\phi(v,b)}),$
  $\varphi_2^e(\phi(S_1((v,b))) \mid_{\phi(v,b)}), \ldots, \varphi_n^e(\phi(S_1((v,b))) \mid_{\phi(v,b)}))$.

Пусть (v,b) — ветвь конечного автомата, $e = phi((v,b))$ — соответствующий ей инцидентор графа. В качестве начального состояния ветви (v,b) возьмем следующее:

$$(P_{0,1}^v(v), P_{0,2}^v(v), \ldots, P_{0,m}^v(v), P_{0,1}^e(e), P_{0,2}^e(e), \ldots, P_{0,n}^e(e)).$$

Другими словами, начальное состояние ветви автомата есть конкатенация начальных информационных векторов для соответствующей вершины и инцидентного ей инцидентора графа.

Построенный автомат $\Omega_0 = (E_0, \psi_0)$ является искомым, поскольку его действие на графе идентично действию локального параллельного алгоритма Журавлева A_0.

Предложение 3. Пусть проблема K разрешима над семейством графов $I(P)$ в модели сетей конечных автоматов за время T_0. Тогда существует локальный параллельный алгоритм Журавлева, дающий решение проблемы K над семейством графов $I(P)$ за время $O(T_0)$.

Доказательство. Пусть конечный автомат $\Omega_0 = (E_0, \phi_0)$ дает решение проблемы K над семейством графов $I(P)$ за время $O(T_0)$. Построим по нему локальный параллельный алгоритм Журавлева A_0, также дающий решение этой проблемы над семейством графов $I(P)$ за время $O(T_0)$.

Заметим, что между множеством инцидеторов графа и множеством ветвей конечного автомата существует естественное взаимооднозначное соответствие; обозначим его через ϕ.

Определим на инциденторах графа следующие предикаты:

$$P_1^0, P_2^0, \ldots, P_{|E_0|}^0.$$

При этом значение i-го предиката на инциденторе графа e равно истине в том и только том случае, если соответствующая ему ветвь $\phi(e)$ имеет метку $a_i \in E_0$. В любой момент времени работы локального алгоритма Журавлева A_0 и на любом инциденторе графа будет выполняться следующее условие: ровно один предикат, определенный на данном инциденторе, имеет значение равное **истина**. Значение всех остальных предикатов на данном инциденторе равно **ложь**. Таким образом, по информационному вектору $I(e)$ инцидентора e однозначно восстанавливается метка ветви $\phi(e)$.

Пусть v — некоторая вершина графа, e — инцидентор этой вершины. Определим функции $\varphi_1^0, \ldots, \varphi_{|E_0|}^0$ изменения значений предикатов следующим образом:

• Если $\psi_0(\phi(S(v)))\ |_{\phi(e)} = Id$, то

$$\begin{array}{lll} P_{t+1,1}^0(e) & = \varphi_{t+1,1}^0(e) & := P_{t,1}^0(e), \\ P_{t+1,2}^0(e) & = \varphi_{t+1,2}^0(e) & := P_{t,2}^0(e), \ldots, \\ P_{t+1,|E_0|}^0(e) & = \varphi_{t+1,|E_0|}^0(e) & := P_{t,|E_0|}^0(e). \end{array}$$

• Если $\psi_0(\phi(S(v)))\ |_{\phi(e)} \neq Id, \psi_0\ |_{\phi(e)}$ предписывает изменить состояние ветви $\phi(e)$ на $a_i \in E_0$ и $P_{t,k}^0(e) = true$, то произвести следующие изменения значений предикатов:

$$P^0_{t+1,k}(e) := false, \qquad P^0_{t+1,i}(e) := true;$$
$$P^0_{t+1,j}(e) := P^0_{t,j}(e), \qquad \forall j \neq i, k.$$

Пусть v — некоторая вершина графа, e — инцидентор этой вершины, $\phi(e)$ — ветвь конечного автомата, соответствующая инциденторy e, которая имеет начальную метку $a_i \in E_0$. Тогда начальные значения предикатов на инциденторе e устанавливаются следующим образом: $P^0_{0,i}(e) := true, P^0_{0,j}(e) := false, \forall j \neq i$.

Построенный алгоритм Журавлева является искомым автоматом, решающим проблему K, так как его поведение на графе неотличимо от поведения сети конечных автоматов Ω_0.

Предложение 4. Пусть проблема K разрешима над семейством графов $I(P)$ с помощью конечного автомата Ω_0 за время T_0. Тогда существует система переписывания графов с запрещенными контекстами S_0 такая, что проблема K разрешима над семейством графов $I(P)$ с помощью S_0 за время $O(|V(G)| \cdot T_0)$.

Доказательство. Идея доказательства заключается в следующем. С помощью системы переписывания графов с запрещенными контекстами производится повторяющийся обход графа в ширину. Во время каждого такого обхода моделируется один такт работы сети конечных автоматов Ω_0. Алгоритм закончит свою работу, когда произойдет первый обход графа, в течение которого метка ни одного инцидентора графа не была изменена.

Каждый раз, когда вершина графа затронута новым обходом в ширину, в ней инициализируется процесс моделирования одного такта работы конечного автомата Ω_0 в данной вершине. Он заключается в изменении меток всех инцидентoров данной вершины согласно предписаниям функции изменения состояний ψ_0 для Ω_0.

Метка каждого инцидентора будет состоять из двух частей. Первая часть метки предназначается для результатов вычислений, проделанных во время последнего обхода графа в ширину; вторая предназначается для хранения результатов вычислений, проделанных на предыдущем обходе. При окончании очередного обхода информация из первой части метки переписывается во вторую, после чего первая часть зануляется.

Осталось только показать способ моделирования одного такта работы автомата Ω_0 в данной вершине.

Рассмотрим общий вид правила из функции изменения состояний

автомата ψ_0 (в квадратных скобках указана необязательная компонента).

Если

$$(\exists i_1, i_2, \ldots, i_k, i_{k+1}, \ldots, i_s |\ e_{(i_1,t)} = a_{i_1}, \ldots, e_{(i_k,t)} = a_{i_{k+1},t} = a_{i_{k+1}}, \ldots, l_{i_s,t} = a_{i_s}) \wedge$$
$$\Big[\nexists j_1, j_2, \ldots, j_r \mid (e_{(j_1,t)} = b^1_{j_1}, \ldots, e_{(j_k,t)} = b^1_{j_k}, l_{(j_{k+1},t)} = b^1_{j_{k+1}}, \ldots, l_{(j_r,t)} = b^1_{j_r}) \bigvee \ldots \bigvee (e_{(j_1,t)} = b^d_{j_1}, \ldots, e_{(j_k,t)} = b^d_{j_k}, l_{(j_{k+1},t)} = b^d_{J_{k+1}}, \ldots, l_{(j_r,t)} = b^d_{j_r})\Big],$$

то $e_{i_1,t+1} := c_{i_1}, \ldots e_{i_s,t+1} := c_{i_s} (a_l, b_l, c_l \in E_0)$.

Для каждого такого правила из ψ_0 добавляется одно правило переписывания с d запрещенными контекстами (указаны только вторые части меток).

Так как число тактов работы сети конечных автоматов равно T_0, а на моделирование одного такта требуется не более чем $O(V_G)$, то длина максимально возможной цепи переписывания для системы переписывания графов S_0 не превышает $O(V_G \cdot T_0)$.

СПИСОК ЛИТЕРАТУРЫ

1. **Billaud M., Lafon P., Metivier Y., Sopena E.** Graph rewriting systems with priority: definitions and applications. — Univ. Bordeaux 1: Intern. Rep. N 8909, 1989.
2. **Litowsky I., Metivier Y., Sopena E.** Different local controls for graph relabelling systems // Math. Syst. Theory. — 1995. — Vol. 28. — P. 41–65.
3. **Журавлев Ю.И.** Локальные алгоритмы вычисления информации. 1 // Кибернетика. — 1965. — N 1. — С. 12–19.
4. **Журавлев Ю.И.** Оценки сложности локальных алгоритмов для некоторых экстремальных задач на конечных множествах // ДАН СССР. — 1964. — Т. 158, N 5. — С. 1018–1021.
5. **Евстигнеев В.А.** О некоторых свойствах локальных алгоритмов на графах // Комбинаторно-алгебраические методы в прикладной математике. — Горький, 1983. — С. 72–105.
6. **Евстигнеев В.А.** Локальные алгоритмы и проблема установления конца вычислений в распределенных системах // Комбинаторно-алгебраические и вероятностные методы дискретного анализа. — Горький, 1989. — С. 55–63.
7. **Rosenstiehl P., Fiksel J.R., Holliger A.** Intelligent graphs: networks of finite automata capable of solving graph problems // J. Graph Theory and Computing, 1972.
8. **Евстигнеев В.А., Бояршинов В.А.** Распознавание хордальных графов в рамках распределения модели вычислений // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 107—129.
9. **Бояршинов В.А.** Распознавание интервального порядка на базе локальных вычислений // Там же. — С. 130–145.

Л. С. Мельников, И. В. Петренко

НЕКОТОРЫЕ ИНВАРИАНТЫ КУБОПОДОБНОГО ГРАФА[1]

1. ВВЕДЕНИЕ И ВСПОМОГАТЕЛЬНЫЕ РЕЗУЛЬТАТЫ

Пусть $U = \{1, \ldots, n\}$ — конечное n–элементное множество; $\mathcal{P}(U)$ — множество всех подмножеств множества U; $S \subseteq \mathcal{P}(U)$ — некоторое семейство подмножеств U.

Определение 1. Кубоподобным графом $Q_n(S)$ назовем граф с множеством вершин $\mathcal{P}(U)$, пара которых смежна тогда и только тогда, когда их симметрическая разность $(x\Delta y)$ лежит в S.

Понятие кубоподобного графа введено Ловасом (Lovasz). Определенный круг авторов проявлял интерес к этому классу графов, в частности, к их хроматическим характеристикам (см. [9]). Результаты, полученные ранее, сформулированы в основном для дистанционных графов $Q_n[D]$, которые являются собственным подклассом класса кубоподобных графов. Множество вершин $Q_n[D]$ идентично множеству вершин кубоподобного графа, а множество D — множество целых неотрицательных чисел. Пара вершин x и y в дистанционном графе $Q_n[D]$ смежна в том и только в том случае, если $|x\Delta y| \in D$.

Жежер (Jaeger) [7] доказал, что для случая, когда S — семейство двухэлементных подмножеств U, которое может быть рассмотрено в качестве множества ребер некоторого графа, имеет место неравенство:

$$\chi(Q_n(S)) \leq 2^{\lceil \log_2 \gamma \rceil},$$

где γ — хроматическое число графа $G = (U, S)$. Он также полагал, что равенство достижимо.

В работе [10] было доказано, что $\chi(Q_n[2]) = 2^{[\log_2 n]}$, если n представимо в виде $2^i, 2^i - 1, 2^i - 2, 2^i - 3$.

Дворжак (Dvorjak) высказывал предположение, что хроматическое число дистанционного графа всегда является степенью 2.

Соколова (Sokolova) [12] доказала, что хроматическое число $Q_{2k}[1, 2k]$ так называемого "расширенного нечетного графа" равно четырем.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Наконец, Пэйян (Payan) [11] опроверг предположение Дворжака контрпримером — им была построена минимальная семицветная раскраска графа $Q_6[4]$. Помимо этого в статье была доказана следующая

Теорема 1. (C. Payan). Для всякого недвудольного кубоподобного графа имеет место неравенство $\chi(Q_n(S)) \geq 4$.

Для некоторого $x \in \mathcal{P}(U)$ кубоподобного графа $Q_n(S)$ и некоторого множества $S_1 \subseteq \mathcal{P}(U)$ под записью вида $x\Delta S_1$ будем понимать множество вида $\{x\Delta s_1, \ldots, x\Delta s_l\}$, где $s_1, \ldots, s_l$ — элементы множества S_1.

В дальнейшем будем полагать, что граф $Q_n(S)$ не содержит петель, т. е. ребер вида (x, x).

Легко видеть, что операция Δ обладает следующими свойствами:

1) $(x\Delta y)\Delta z = x\Delta(y\Delta z)$;
2) $x\Delta y = y\Delta x$;
3) $x\Delta y = \emptyset \Leftrightarrow x = y$;
4) $x\Delta z = y\Delta z \Leftrightarrow x = y$.

Исходя из этих свойств симметрической разности, легко видеть, что если $x\Delta y = z$, то $x = y\Delta z$.

Лемма 1.1. Для произвольного $U \subset \mathcal{N}$ множество $\mathcal{P}(U)$ является группой относительно операции симметрической разности Δ.

Доказательство. Действительно, поскольку, как показано выше, операция Δ удовлетворяет всем аксиомам групповой операции, для доказательства достаточно продемонстрировать, что $\mathcal{P}(U)$ замкнуто относительно операции Δ. Для двух произвольных x и y из $\mathcal{P}(U)$ имеет место $x, y \subseteq U$. Соответственно $x \setminus y$ и $y \setminus x$ также являются подмножествами U, равно как и их объединение, откуда следует, что $x\Delta y$ является элементом $\mathcal{P}(U)$. Замкнутость операции доказана.

Следующие утверждения будут полезны в ходе дальнейшего изложения.

Утверждение 1.2. $Q_n(\emptyset)$ — пустой (петельный) граф.

Из определения кубоподобного графа и доказанной леммы следует: поскольку $S = \emptyset$, то для двух смежных вершин x и y имеем: $x\Delta y = \emptyset$ тогда и только тогда, когда $x = y$, что означает, что множество ребер данного графа $E(Q_n(S)) = \{(x, x); x \in \mathcal{P}(U)\}$, т. е. состоит из петель, которые мы игнорируем.

Утверждение 1.3. $Q_n(\mathcal{P}(U))$ — полный граф.

Согласно определению смежности в кубоподобном графе $(x, y) \in E(Q_n(\mathcal{P}(U)))$ тогда и только тогда, когда $(x\Delta y) \in \mathcal{P}(U)$. Легко видеть, что в силу такого определения смежности любая пара вершин в данном графе является смежной, а следовательно, $Q_n(\mathcal{P}(U))$ — полный граф.

Утверждение 1.4. $Q_n(s)$ — паросочетание.

Покажем для случая $S = \{s\}$, что каждая вершина имеет в точности одного соседа. Пусть это не так. Тогда для некоторой вершины x существует по меньшей мере пара вершин y_1, y_2 таких, что $x\Delta y_1 = x\Delta y_2 = s$, это, в свою очередь, согласно лемме 1.1 означает, что $y_1 = y_2 = y$, т. е. для любой вершины x существует в точности одна вершина y, смежная с ней.

Утверждение 1.5. Пусть $|S| = 2$. Тогда кубоподобный граф $Q_n(S)$ двудольный.

Согласно известной теореме Кенига граф является двудольным в том и только в том случае, если он не содержит циклов нечетной длины. Предположим, что наш граф $Q_n(S)$, где $S = \{s_1, s_2\}$, содержит цикл нечетной длины. Легко видеть, что длина такого цикла не превосходит трех. Допустим, существует три такие вершины — x, y, z, — которые образуют такой цикл. Это означает, что множество S содержит элементы $x\Delta y$, $y\Delta z$, $z\Delta x$, причем все они различны, что противоречит условию утверждения. Таким образом, утверждение доказано.

2. КЛИКИ, НЕЗАВИСИМЫЕ МНОЖЕСТВА, ПЛОТНОСТЬ, НЕПЛОТНОСТЬ, КЛИКОМАТИЧЕСКОЕ ЧИСЛО КУБОПОДОБНОГО ГРАФА

Лемма 2.1. Если $Q_n(S)$ содержит полный подграф, то найдется $S_1 \subseteq S$ такое, что S_1 индуцирует ребра полного подграфа и является группой относительно операции симметрической разности Δ.

Доказательство. Пусть $\{x_1, \ldots, x_l\}$ — полный подграф. Применим метод математической индукции.

$k = 2$, т. е. полный подграф состоит из двух вершин $\{x_1, x_2\}$, множеству ребер данного подграфа соответствует множество $S_1 = \{x_1\Delta x_2, \emptyset\}$, которое, очевидно, является группой относительно симметрической разности Δ.

Полагаем, что для некоторого l предположение индукции доказано, и для полного подграфа $K_l = \{x_1, \ldots, x_l\}$ множество S_l, индуцирующее его ребра, является группой относительно Δ.

Исходя из сформулированнной индукционной гипотезы, докажем наше утверждение для $l+1$. Для этого необходимо рассмотреть множество

$$S_{l+1} = S_l \cup \{x_{l+1}\Delta x_1, \ldots, x_{l+1}\Delta x_l\}.$$

Очевидно, что множество S_{l+1} индуцирует множество ребер полного подграфа K_{l+1}. Докажем, что S_{l+1} — группа. Для чего покажем замкнутость S_{l+1} относительно симметрической разности.

Для этого рассмотрим следующие три варианта:

1) $s_1, s_2 \in S_l$ — в этом случае в силу того, что S_l — группа, очевидно, что $s_1\Delta s_2 \in S_l \subseteq S_{l+1}$;

2) $s_1, s_2 \in S_{l+1}\backslash S_l$ — в этом случае имеем $s_1\Delta s_2 = x_{l+1}\Delta x_i\Delta x_{l+1}\Delta x_j$, что в силу выше доказанных утверждений равно $x_i\Delta x_j \in S_l \subseteq S_{l+1}$;

3) $s_1 \in S_l, s_2 \in S_{l+1} \setminus S_l$ — в этом случае $s_1\Delta s_2 = x_i\Delta x_j\Delta x_{l+1}\Delta x_k$. Ясно, что $x_i\Delta x_j = s_m$ для некоторого m, причем $s_m \in S_l$ в силу индукционной гипотезы, т. е. $x_k\Delta s_m \in K_l = \{x_1, \ldots, x_l\}$. Откуда в силу того, что вершина x_{l+1} является смежной со всеми вершинами K_l, имеем

$$x_{l+1}\Delta x_i\Delta x_j\Delta x_k \in S_{l+1},$$

что доказывает, что S_{l+1} замкнуто относительно операции Δ, а это означает, что S_{l+1} — группа, что и требовалось доказать.

Лемма 2.2. Если $S_1 \subseteq S \subseteq \mathcal{P}(U)$ таково, что S_1 — группа относительно Δ, то $Q_n(S)$ содержит полный подграф мощности $|S_1|$.

Доказательство. Для произвольной вершины x графа $Q_n(S)$ рассмотрим множество вершин $K = x\Delta S_1$. Поскольку все элементы S_1 попарно различны $|K| = |x\Delta S_1| = |S_1|$ в силу вышедоказанных свойств Δ. Докажем, что K — полный подграф.

Для $x_1, x_2 \in K$ имеем $x_1 = x\Delta s_1$, $x_2 = x\Delta s_2$ для некоторых $s_1, s_2 \in S_1$. Следовательно, $x_1\Delta x_2 = x\Delta s_1\Delta x\Delta s_2 = s_1\Delta s_2$, где $s_1, s_2 \in S_1$, а поскольку S_1 — группа, имеем $s_1\Delta s_2 \in S_1$, что означает, что рассмотренные вершины смежны. Тем самым лемма доказана.

Таким образом, мы доказали следующую теорему.

Теорема 2.3. $Q_n(S)$ содержит полный подграф K тогда и только тогда, когда существует $S_1 \subseteq S$ такое, что S_1 — группа относительно Δ, при этом $|K| = |S_1|$.

Поскольку независимое множество есть не что иное, как полный подграф в дополнении исходного графа, а дополнение графа $Q_n(S)$ есть граф $Q_n(\mathcal{P}(U) \setminus S)$, может быть сформулирована следующая теорема, двойственная к предыдущей, и являющаяся ее очевидным следствием.

Теорема 2.4. $Q_n(S)$ содержит независимое множество вершин I тогда и только тогда, когда существует $S_1 \subseteq \mathcal{P}(U) \setminus S$ такое, что S_1 — группа, при этом $|I| = |S_1|$.

Замечание. Отметим, что клике, т. е. максимальному полному подграфу в $Q_n(S)$, соответствует максимальная по включению группа $S_1 \subseteq S$ и наоборот. Аналогичное утверждение верно и для максимального независимого множества с заменой индуцирующего ребра множества S на его дополнение.

Пусть $S \subseteq \mathcal{P}(U)$ является группой относительно Δ. Элементы $s_1, \ldots, s_m$ будем называть *индуцирующими группу,* если для любых $i, j, k \in \{1, \ldots, m\}$ выполняется $s_i \neq s_j \Delta s_k$.

Лемма 2.5. Если $S \subseteq \mathcal{P}(U)$ — группа относительно Δ, то $|S| = 2^m$, где $m \leq n = |U|$ — некоторое натуральное число.

Доказательство. Пусть $X = \{x_1, \ldots, x_l\}$ — максимальный набор индуцирующих элементов группы S. Через X_Δ обозначим множество вида

$$\{x_1, \ldots, x_l, x_1 \Delta x_2, \ldots, x_{l-1} \Delta x_l, x_1 \Delta x_2 \Delta x_3, \ldots \\ \ldots, x_{l-2} \Delta x_{l-1} \Delta x_l, \ldots, x_1 \Delta x_2 \Delta \ldots \Delta x_l\}.$$

Очевидно, что $X_\Delta \subseteq S$.

Докажем обратное. Предположим, что $S \setminus X_\Delta \neq \emptyset$, т. е. существует $s \in S \setminus X_\Delta$. Поскольку S — группа, а следовательно, S замкнуто относительно операции симметрической разности Δ, то существует пара $\{s_1, s_2 : s_1 \Delta s_2 = s\}$. Если при этом хотя бы один из этих трех элементов, например s_1, принадлежит множеству X_Δ, то получаем противоречие с предположением о максимальности множества индуцирующих группу S элементов. Если же ни один из указанных элементов не принадлежит X_Δ, то $S \setminus X_\Delta$ содержит группу (поскольку пустое множество содержится в любом из множеств) относительно Δ, для которой может быть

построен набор индуцирующих элементов. Обозначим этот набор через X_1. При этом $X \cup X_1$ образует новый индуцирующий набор для группы S, который превосходит исходный максимальный. Снова получено противоречие с предположением относительно максимальности набора индуцирующих элементов X, что доказывает требуемое включение $S \subseteq X_\Delta$. Таким образом, доказано, что $S = X_\Delta$.

Далее, поскольку все элементы X_Δ попарно различны (в силу свойств симметрической разности рассмотренных выше), то $|X_\Delta| = 2^l$, где l — длина индуцирующего набора, т. е. некоторое натуральное число, например m. Очевидно, $|S| = |X_\Delta| = 2^m$.

Доказанная лемма в сочетании с теоремами, сформулированными выше, позволяет сформулировать следующие теоремы.

Теорема 2.6. Плотность кубоподобного графа $\omega(Q_n(S)) = 2^m$, где m — целое неотрицательное число.

Теорема 2.7. Неплотность кубоподобного графа $\varepsilon(Q_n(S)) = 2^l$, где l — целое неотрицательное число.

Теорема 2.8. Кликоматическое число k кубоподобного графа $Q_n(S)$ есть 2^l, где l — некоторое целое неотрицательное число, причем если выполняется равенство $\omega(Q_n(S)) = 2^m$, то для l имеет место равенство $l = n - m$.

Доказательство. Допустим, граф $Q_n(S)$ содержит клику K_1, индуцируемую подмножеством S_1 множества S. Предположим, что вершина $\emptyset \in V(K_1)$, а следовательно, любая вершина K_1 смежна с $\emptyset$. Иными словами, $x \in K$, если и только если $x \in S_1$. Выберем произвольным образом вершину x_1 из множества $V(Q_n(S)) \setminus S_1$. Отметим, что множество вершин $\{x_1 \Delta S_1\}$, безусловно, также является кликой. Обозначим ее K_2. После этого произведем удаление вершин двух выявленных клик из множества вершин исходного графа и будем повторять описанную операцию до тех пор, пока множество вершин $Q_n(S)$ не будет исчерпано. Наконец, получим последовательность клик $K_1, K_2, \ldots, K_p$. Заметим, что $V(K_i) \cap V(K_j) = \emptyset$.

Действительно, предположим, что некоторая вершина $y \in K_i \cap K_j$ для некоторых различных $i, j \in \{1, \ldots, p\}$. Из этого следует, что $y = x_i \Delta s_1$, где $s_1 \in S_1$, а $x_i \in K_i$, при этом, с другой стороны, $y = x_j \Delta s_2$, где также $s_2 \in S_1, x_j \in K_j$, т. е. имеет место равенство

$$x_i \Delta s_1 = x_j \Delta s_2,$$

эквивалентное равенству

$$x_i \Delta x_j = s_1 \Delta s_2,$$

в котором $s_1 \Delta s_2 = s$, где $s \in S_1$ в силу того, что S_1 — группа относительно Δ, а s_1, s_2 — ее элементы. Откуда следует, что $x_i \Delta x_j \in S_1$, т. е. обе вершины x_i и x_j смежны в $Q_n(S_1)$, — в таком суграфе $Q_n(S)$, который содержит только ребра клики, что противоречит условию выбора вершин x_i и x_j. Тем самым доказано, что $K_i \cap K_j = \emptyset$ для всех i, j.

Ясно, что описанный алгоритм на некотором шаге завершает работу. При этом на каждом шаге работы из множества вершин исходного графа удаляется 2^m элементов. В силу этого имеет место неравенство $p \leq 2^{n-m}$. Докажем, что

$$V(K_1) \cup \ldots \cup V(K_p) = V(Q_n(S)).$$

Для этого достаточно показать, что

$$V(Q_n(S)) \subseteq V(K_1) \cup \ldots \cup V(K_p).$$

Действительно, если найдется такая вершина x, которая не принадлежит ни одной из клик $K_1, \ldots, K_p$, то она должна быть либо несмежна ни с одной другой вершиной, чего не может быть в силу того, что индуцирующее множество S непусто, либо она (вершина) в совокупности с множеством соседей $x \Delta S_1$ не образует клику, чего также не может быть в силу того, что S_1 — группа, а также в силу вышедоказанных лемм.

Таким образом, имеем

$$V(K_1) \cup \ldots \cup V(K_p) = V(Q_n(S)).$$

Теперь заметим, что $|V(K_i)| = |V(K_j)|$ для всех i, j. Или, иными словами, $|V(K_1)| \cdot p = |V(Q_n(S_1))| = 2^n$, с другой стороны, поскольку нам известно, что $|V(K_1)| = 2^m$, имеем уравнение $2^m p = 2^n$ относительно p, из которого ясно, что $p = 2^l$, где $l = n - m$. Таким образом, теорема доказана.

Замечание. Отметим, что в случае, когда $Q_n(S)$ таков, что S_1 индуцирует клику, а множество $S \setminus S_1$ не индуцирует клику мощности больше двух, граф $Q_n(S)$ является совершенным, поскольку имеет место равенство $k(Q_n(S)) = \varepsilon(Q_n(S))$.

Действительно, если плотность кубоподобного графа $Q_n(S)$ есть 2^m, то его неплотность $\varepsilon(Q_n(S)) = 2^{n-m}$, с другой стороны, $k(Q_n(S)) =$

2^{n-m} в силу вышедоказанной теоремы, если выполнено оговоренное выше условие. Отсюда может быть сделан следующий вывод: *при оговоренных в посылке замечания условиях для хроматического числа кубоподобного графа* $Q_n(S)$ *имеет место равенство*

$$\chi(Q_n(S)) = \omega(Q_n(S)) = 2^m,$$

где m *— некоторое целое неотрицательное число, не превышающее размерности* n*.*

3. РЕБЕРНОЕ, РЕБЕРНОЕ ПРЕДПИСАННОЕ И ТОТАЛЬНОЕ ХРОМАТИЧЕСКИЕ ЧИСЛА КУБОПОДОБНОГО ГРАФА

Теорема 3.1. Для реберного хроматического числа кубоподобного графа $Q_n(S)$ имеет место равенство $\chi_e(Q_n(S)) = \Delta(Q_n(S))$.

Доказательство. Как сказано выше, в случае, если S содержит в точности один элемент, граф $Q_n(S)$ — это паросочетание, которое есть монохроматический класс множества ребер исходного графа. Можно выделить в точности |S| таких монохроматических классов, т. е. $\chi_e(Q_n(S)) = |S|$. С другой стороны, $|S| = \Delta(Q_n(S))$, тем самым доказано, что

$$\chi_e(Q_n(S)) = \Delta(Q_n(S)).$$

Лемма 3.2. Кубоподобный граф $Q_n(S)$ может быть разложен на конечное число двудольных суграфов.

Доказательство. В силу выше доказанного утверждения, если множество S состоит из двух элементов ($|S| = 2$), то $Q_n(S)$ — двудольный. Тогда S произвольной длины может быть разложено на $\lceil \frac{|S|}{2} \rceil$ непересекающихся подсемейств семейства S. Иными словами, произвольный кубоподобный граф может быть разложен на $\lceil \frac{|S|}{2} \rceil$ двудольных графов. Лемма доказана.

Теорема 3.3. Для реберного предписанного хроматического числа кубоподобного графа $Q_n(S)$ имеет место равенство

$$\chi_{el}(Q_n(S)) = \Delta(Q_n(S)).$$

Доказательство. Из предыдущей леммы для некоторого l имеем

$$Q_n(S) \cong \bigcup_{i=1}^{l} Q_n(S_i),$$

где $\forall i : S_i \subseteq S$ и $\bigcup_{i=1}^{l} S_i = S$, причем полагаем $\forall i, j : S_i \cap S_j = \emptyset$. Положим

$$G_i = Q_n(S_i).$$

Каждый G_i является двудольным. Воспользовавшись теоремой Гэлвина (Galvin) [8], получим, что ребра каждого G_i могут быть окрашены списками цветов длины

$$\Delta(G_i) = \Delta_i = |S_i|.$$

Множество элементов таких списков обозначим через Λ_i. Предположив, что

$$\Lambda_i \cap \Lambda_j = \emptyset$$

при $i \neq j$, получим возможность окрасить каждое ребро графа $Q_n(S)$ списком длины

$$\Delta(Q_n(S)) = \sum_{i=1}^{l} \Delta_i.$$

Доказано требуемое равенство.

Подмножество элементов графа будем называть *тотально независимым*, если никакие два его элемента не являются смежными и/или инцидентными никаким другим объектам этого же множества. *Тотальным хроматическим числом* $\chi_t(G)$ называется наименьшее число цветов нужное для раскраски элементов $V(G) \cup E(G)$ так, чтобы любое одноцветное множество элементов было тотально независимым. Визинг [1, 2] и Бехзад (Behzad) [5] более 35 лет назад предположили, что для обыкновенных графов справедливо

$$\chi_t(G) \leq \Delta(G) + 2.$$

Позднее эта гипотеза получила название гипотезы о тотальной раскраске и была обобщена для мультиграфов (см. [13, 3, 4, 9]). Следущая теорема подтверждает не только справедливость этой гипотезы для кубоподобных графов, но и демонстрирует достижимость ее оценок для нетривиальных графов этого класса.

Теорема 3.4. Для тотального хроматического числа кубоподобного графа $Q_n(S)$ имеют место следующие неравенства

$$\Delta(Q_n(S)) + 1 \leq \chi_t(Q_n(S)) \leq \Delta(Q_n(S)) + 2.$$

Более того, обе оценки точные на нетривиальных кубоподобных графах для $n \geq 3$.

Доказательство. Докажем верхнюю оценку

$$\chi_t(Q_n(S)) \leq \Delta(Q_n(S)) + 2.$$

Пусть C — множество цветов, причем $|C| = \Delta(Q_n(S)) + 2$. Произведем раскраску вершин данного графа цветами из C. Каждому ребру данного графа припишем список длины $\Delta(Q_n(S))$, полученный выбрасыванием из множества C элементов, использованных при раскраске концевых вершин ребра, т. е. для некоторого ребра (x, y) список цветов будет иметь вид

$$\Lambda_{(x,y)} = C \setminus \{c(x), c(y)\},$$

где $c(x), c(y)$ — цвета вершин x и y. Таким образом, каждое ребро будет окрашено списком длины $\Delta(Q_n(S))$.

Рассмотрим замкнутую окрестность $N[x]$ вершины x. В раскраске вершин $N[x]$ участвует не более чем $\Delta + 1$ цветов, т. е. по крайней мере один из цветов множества C, скажем c_1, входит в каждый список $\Lambda_{(x,y)}$, где $y \in N(x)$. Этот цвет допустим для раскраски любого из ребер вида $\{(x, y),\ y \in N(x)\}$. Выберем произвольно $z_1 \in N(x)$, а затем припишем ребру (x, z_1) цвет c_1 и удалим его из списков цветов всех ребер, инцидентных вершине x. Рассмотрим теперь подграф, порожденный множеством вершин $N[x] \setminus \{z_1\}$. Ясно, что в списках цветов его ребер найдется цвет c_2 такой, что

$$c_2 \in \bigcap_{y \in N(x) \setminus \{z_1\}} \Lambda_{(x,y)} \setminus \{c_1\}.$$

Повторяя описанную процедуру в точности Δ раз, получим раскраску всех ребер, инцидентных вершине x. После этого перейдем к произвольной вершине $y \in N(x)$ и повторим всю последовательность действий для нее с учетом того, что одно или несколько ребер уже раскрашено. Это значит, что цвета, уже использованные для окраски этих ребер, не могут рассматриваться в качестве допустимых. Продолжая двигаться некоторым образом по вершинам графа, получим его правильную тотальную раскраску не более чем в $\Delta(Q_n(S)) + 2$ цветов.

Доказательство нижней оценки основано на том простом факте, что вершина максимальной степени $\Delta(Q_n(S))$ и инцидентные ей ребра должны быть окрашены разными цветами.

Тотальная раскраска пустого и полного $2k$–вершинного графов (см. утверждения 1.2 и 1.3) возможна в один и в $2k+1$ цветов соответственно.

Эти тривиальные графы показывают достижимость нижней и верхней оценок.

В качестве нетривиальных кубоподобных графов возьмем полный двудольный граф $K_{n,n}$, n–мерный куб Q_n и декартово произведение $K_n \times K_2$. Для первого из графов $K_{n,n}$ в качестве S можно взять $\mathcal{P}_o(U)$ все подмножества из U нечетной мощности, для последнего из графов $K_n \times K_2$ в случае $n = 2^k$ в качестве S можно взять объединение $\mathcal{P}(U)$ всех подмножеств из $U = \{1, 2, ...k\}$ и $\{k+1\}$.

Утверждение 3.5. Для тотального хроматического числа графа $K_{n,n}$ имеет место следующее равенство:

$$\chi_t(K_{n,n}) = \Delta(K_{n,n}) + 2 = n + 2.$$

Доказательство. Докажем нижнюю оценку

$$\chi_t(K_{n,n}) \geq \Delta(K_{n,n}) + 2.$$

Максимальное тотально независимое множество графа $K_{n,n}$ имеет мощность не более чем n. Подсчитаем количество элементов в $K_{n,n}$:

$$|V(K_{n,n})|+|E(K_{n,n})| = 2n+\Delta(K_{n,n})\cdot\frac{|V|}{2} = 2n+n^2 = n(n+2) = n(\Delta+2).$$

Поскольку максимальное тотально независимое множество есть максимальный монохроматический класс при правильной тотальной раскраске данного графа, справедливо отношение

$$\chi_t(K_{n,n}) \geq \frac{n(\Delta+2)}{n}$$

или

$$\chi_t(K_{n,n}) \geq \Delta + 2.$$

Следующее неравенство получается из правильной тотальной раскраски, где каждую долю красим своим цветом и затем ребра красим в Δ цветов.

$$\chi_t(K_{n,n}) \leq \Delta(K_{n,n}) + 2.$$

Откуда, объединяя два полученных неравенства, имеем

$$\chi_t(K_{n,n}) = \Delta + 2.$$

Предложение доказано.

Утверждение 3.6. Для тотального хроматического числа графа n–мерного куба Q_n и $n \geq 3$ имеет место следующее равенство:

$$\chi_t(Q_n) = \Delta(Q_n) + 1 = n + 1.$$

Причем только первые четыре цвета используются для раскраски всех вершин Q_n.

Доказательство. Отметим, что при $n = 1, 2$ справедливо противоположное равенство $\chi_t(Q_n) = \Delta(Q_n) + 2 = n + 2$.

Доказательство индукцией по n.

База индукции: для $n = 3$ тотальная раскраска Q_3 в четыре цвета следующая:
1 — вершины $\{\emptyset\}, \{1,2,3\}$ и ребра $(\{1\},\{1,2\}), (\{2\},\{2,3\}), (\{3\},\{1,3\})$;
2 — вершины $\{2\}, \{1,3\}$ и ребра $(\{3\},\{2,3\}), (\{\emptyset\},\{1\}), (\{1,2\},\{1,2,3\})$;
3 — вершины $\{3\}, \{1,2\}$ и ребра $(\{\emptyset\},\{2\}), (\{1\},\{1,3\}), (\{2,3\},\{1,2,3\})$;
4 — вершины $\{1\}, \{2,3\}$ и ребра $(\{\emptyset\},\{3\}), (\{2\},\{1,2\}), (\{1,3\},\{1,2,3\})$.

Допустим, справедливо индукционное предположение, сформулированное в предложении. Для тотальной раскраски Q_{n+1}, состоящего из двух копий Q_n и совершенного паросочетания, соответствующего подмножеству $\{n+1\}$, в первой копии используем тотальную раскраску индукционного предположения, а во второй — тотальную раскраску с переименованием цветов согласно перестановке

$$\pi = (1,2,3,4)(5)...(n)(n+1),$$

цвет $n+2$ использован для раскраски совершенного паросочетания, соответствующего подмножеству $\{n+1\}$. Предложение доказано.

Утверждение 3.7. Для тотального хроматического числа графа $K_n \times K_2$ и $n \geq 3$ имеет место следующее равенство:

$$\chi_t(K_n \times K_2) = \Delta(K_n \times K_2) + 1 = n + 1.$$

Доказательство. В силу того, что нижняя оценка тривиальна, остановимся на построении тотальной раскраски, лежащей в основе верхней оценки. Рассмотрим два случая: (a) n — нечетно, (b) n — четно.

Случай (a). $n = 2k + 1$ — нечетно.
Представим вершины первой и второй копий K_n как вершины правильного n–угольника соответственно $\{x_0, x_1, ..., x_{2k}\}$ и $\{y_0, y_1, ..., y_{2k}\}$. Индексы в описанной ниже раскраске рассматриваются по модулю $2k+1$.

Тотальная раскраска $K_{2k+1} \times K_2$ в $2k+2$ цвета следующая:
цвет $2k+1$ — вершин нет, а ребра $(x_0, y_0), (x_1, y_1), ..., (x_{2k}, y_{2k})$;
цвет $i, i \in \{0, 1, ..., 2k\}$ — вершины x_i, y_{i+1} и ребра (x_{i-1}, x_{i+1}), $(x_{i-2}, x_{i+2}), ..., (x_{i-k}, x_{i+k}), (y_i, y_{i+2}), (y_{i-1}, y_{i+3}), ..., (y_{i-k+1}, y_{i+k+1})$.

Случай (b). $n = 2k$ — четно.

Представим вершины первой и второй копий K_n как вершины правильного n–угольника соответственно $\{x_0, x_1, ..., x_{2k-1}\}$ и $\{y_0, y_1, ..., y_{2k-1}\}$. Индексы в описанной ниже раскраске рассматриваются по модулю $2k$.

Тотальная раскраска $K_{2k} \times K_2$ в $2k+1$ цвета следующая:
цвет $2k$ — вершин нет, а ребра $(x_0, x_k), (x_1, x_{k+1}), ..., (x_{k-1}, x_{2k-1})$, $(y_0, y_k), (y_1, y_{k+1}), ..., (y_{k-1}, y_{2k-1})$.

Если $k = 2t$ — четно, то множество элементов в монохроматическом классе следующее:
цвет $i, i \in \{0, 1, ..., 2k-1\}$ — вершины x_{i+t}, y_{i-t} и ребра (x_{i+2t-1}, x_{i+1}), $(x_{i+2t-2}, x_{i+2}), ..., (x_{i+t+1}, x_{i+t-1}), (x_{i+2t}, x_{i-1}), (x_{i+2t+1}, x_{i-2}), ...$
$..., (x_{i+3t-1}, x_{i-t}), (y_{i+2t+1}, y_{i-1}), (y_{i+2t+2}, y_{i-2}), ..., (y_{i+3t-1}, y_{i-t+1})$, $(y_{i+2t}, y_{i+1}), (y_{i+2t-1}, y_{i+2}), ..., (y_{i+t+1}, y_{i+t}), (x_i, y_i)$.

Если $k = 2t+1$ — нечетно, то множество элементов в монохроматическом классе следующее:
цвет $i, i \in \{0, 1, ..., 2k-1\}$ — вершины x_{i+t+1}, y_{i-t-1} и ребра (x_{i+2t+1}, x_{i+1}), $(x_{i+2t}, x_{i+2}), ..., (x_{i+t+2}, x_{i+t}), (x_{i+2t+2}, x_{i-1}), (x_{i+2t+3}, x_{i-2}), ...$
$..., (x_{i+3t+1}, x_{i-t}), (y_{i+2t+1}, y_{i-1}), (y_{i+2t+2}, y_{i-2}), ..., (y_{i+3t}, y_{i-t})$, $(y_{i+2t}, y_{i+1}), (y_{i+2t-1}, y_{i+2}), ..., (y_{i+t+1}, y_{i+t}), (x_i, y_i)$.

Предложение доказано.

Утверждение 3.5 подтверждает достижимость верхней оценки теоремы 3.4, любое из следующих утверждений 3.6 и 3.7 подтверждает достижимость нижней оценки теоремы 3.4. Теорема доказана.

Замечание. Отметим, что утверждение 3.5 и случай (a) утверждения 3.7 появились ранее в [6], а здесь приведены для полноты.

4. ЗАКЛЮЧЕНИЕ

На основании некоторых алгебраических свойств симметрической разности удалось получить необходимое и достаточное условие существования клики и/или независимого множества, а также подсчитать величину характеристик кубоподобного графа, таких как плотность, неплотность, кликоматическое число. Помимо того найдено точное значение реберного хроматического и реберного предписанного хромати-

ческого чисел. Для тотального хроматического числа подтверждена гипотеза Визинга–Бехзада (Behzad) и показана точность верхней и нижней оценок как для тривиальных, так и для нетривиальных кубоподобных графов. Представляется перспективным дальнейшее исследование кубоподобного графа на предмет получения оценки или, что лучше, точного значения хроматического числа кубоподобного графа, а также отыскание критерия, когда кубоподобный граф имеет тотальное хроматическое число равное его максимальной степени плюс единица.

СПИСОК ЛИТЕРАТУРЫ

1. **Визинг В. Г.** Об оценке хроматического класса p–графа // Дискретный анализ: Сб. науч. тр. — Новосибирск: Ин-т математики СО АН СССР, 1964. — Вып. 3. — С. 25–30.
2. **Визинг В. Г.** Хроматический класс мультиграфа // Кибернетика. — 1965. — № 3. — С. 29–39.
3. **Визинг В. Г.** Некоторые нерешенные задачи в теории графов // Успехи мат. наук. — 1968. — Т. 23, вып. 6. — С. 125–141.
4. **Зыков А. А.** Теория конечных графов. — Новосибирск: Наука, Сибирское отделение, 1969. — 544 с.
5. **Behzad M.** Graphs and their chromatic number. — Ph. D. Thesis, Michigan State University, 1965.
6. **Behzad M., Chartrand G., Cooper J. K.** The color number of complete graphs // J. London Math. Soc. — 1967. — Vol. 42. — P. 226–228.
7. **Dvorjak T., Havel I., Labordee J. M., Leibl P.** Generalized hypercubes and graph embedding with dilation // Rostock Math. Colloq. — 1990. — Vol. 39. — P. 13–20.
8. **Galvin F.** The list chromatic index of a bipartite multigraph // J. Comb. Theory. Ser. B. — 1995. — Vol. 63, № 1. — P. 153–158.
9. **Jensen Tommy R., Toft B.** Graph Coloring Problems. — N.-Y.: J. Wiley & Sons, 1995. — 295 p. — (Wiley–Interscience Ser. in Discrete Mathematics and Optimization; Vol. XIX).
10. **Linial N., Meshulam R., Tarsi M.** Mathroidal biections between graphs // J. Combin. Theory. Ser. B. — 1988. — Vol. 45, № 1. — P. 31–44.
11. **Payan C.** On the chromatic number of cube–like graphs // Discrete Math. — 1992. — Vol. 103, № 3. — P. 271–277.
12. **Sokolova M.** The chromatic number of extended odd graph is four // Časopis Pešt. Mat. — 1987. — Vol. 112, № 3. — P. 308–311.
13. **Zykov A. A.** Problem 12. // **Sachs H., Voss H.–J., Walther H.** Beiträge zur Graphentheory vorgetragen auf dem Internationalen Kolloquium in Manebach DDR. — 1967. — 228 p.

И.А. Лисицын

ОРГАНИЗАЦИЯ ГРАФИЧЕСКОГО ВЫВОДА В СИСТЕМЕ ВИЗУАЛИЗАЦИИ ИЕРАРХИЧЕСКИХ ГРАФОВЫХ МОДЕЛЕЙ[1]

1. ВВЕДЕНИЕ

Графы широко применяются в различных областях технических и естественных наук, так как позволяют строить наглядные и удобные для обработки модели сложных структур. При создании таких моделей граф наделяется некоторой семантикой, которая обычно выражается набором атрибутов, приписываемых его вершинам и дугам.

На практике преимущества графовых моделей во многих случаях становятся ощутимыми только при наличии хороших инструментальных средств их визуализации и обработки. Анализ существующих на сегодняшний день средств визуализации графов [1] показывает, что немаловажным параметром инструмента визуализации, определяющим круг его пользователей, является обеспечиваемое качество графического вывода. При этом можно выделить следующие основные критерии:

- качество получаемого изображения, т. е. точность соблюдения правил, заданных геометрическими атрибутами в представлении графа; иначе говоря, аккуратность построения рисунка;
- спектр средств (графических примитивов и их комбинаций, шрифтов, цветов и т.д.), используемых для создания изображения графа;
- гибкость задания графических параметров; например, система, в которой метку дуги можно разместить только у ее конца или начала, является менее гибкой, чем та, в которой эту метку можно разместить в любом месте вдоль линии дуги;
- набор возможностей пользователя по манипулированию изображением графа в процессе работы над ним; удобство этих манипуляций как следствие “интеллектуальности” системы.

Стоит отметить, что приведенные критерии часто противоречат друг другу, например: чем более широкий спектр средств используется для получения изображения и чем более аккуратное изображение мы пытаемся

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 00-07-90296) и Министерства образования РФ.

получить, тем больше машинного времени требуется на его построение, что может отрицательно сказаться на быстродействии системы и, соответственно, на возможностях пользователя быстро манипулировать изображением графа. Следовательно, нужно, с одной стороны, искать компромисс между удовлетворением данных критериев, с другой стороны, применять методики, позволяющие устранить противоречие между критериями. Основными параметрами системы, влияющими на достижение указанных целей, являются:

- схема задания геометрических атрибутов элементов графа;
- набор графических параметров элементов графа;
- методы построения графического представления по геометрическим атрибутам;
- оптимизация графического вывода;
- метод визуализации процесса редактирования графа.

При визуализации более сложных структур, таких как иерархические графы, появляются дополнительные проблемы, связанные с увеличением количества типов элементов графа и отношений между ними. Однако основные критерии качества остаются те же.

Иерархический граф состоит из вершин, дуг и фрагментов. Вершины и дуги представляют собой обычный граф, который может быть ориентированным или неориентированным. Каждый фрагмент ассоциируется с набором вершин, которые ему принадлежат. Фрагменты могут быть вложенными, если набор вершин одного является подмножеством набора вершин другого. Иначе пересекаться они не могут. Набор всех вершин определяет *главный* фрагмент иерархического графа. Более точное определение иерархических графов и других понятий, связанных с ними, можно найти в [2]. Различным вопросам визуализации иерархических графов посвящены работы [4–7].

Настоящая статья посвящена методам организации эффективного графического вывода в системах визуализации графов. Мы рассмотрим эти методы на примере системы Higres, являющейся визуализатором и редактором иерархических графовых моделей. В следующем разделе дается краткое описаие системы, ее основных возможностей и пользовательского интерфейса. Разд. 3 посвящен описанию схемы задания геометрических атрибутов элементов графа, использованной в системе. Указываются некоторые альтернативы и преимущества выбранной схемы. Построение изображения графа по данной схеме описано в разд. 4. В разд. 5 и 6 дается

описание методов оптимизации графического вывода и визуализации процесса редактирования графа, реализованных в системе.

2. СИСТЕМА HIGRES

Система Higres [8,9] предназначена для создания и визуализации иерархических графовых моделей, представляющих собой иерархические графы с заданной семантикой. Пример иерархического графа, созданного в системе Higres, приведен на рис. 1. Такие графы используются во внутреннем представлении IF-1 языка SISAL [3].

Семантика иерархического графа представляется в системе с помощью типов объектов. Каждый объект в графе принадлежит к какому-нибудь типу. Для каждого типа определяется набор меток. Каждая метка имеет тип данных, имя и несколько других параметров. Набор значений ассоциируется с каждым объектом графа в соответствии с набором меток, определенным для типа этого объекта. Вместе с разделением на типы эти значения задают семантику графа. Таким образом, пользователь может определять семантику, добавляя новые типы и их метки.

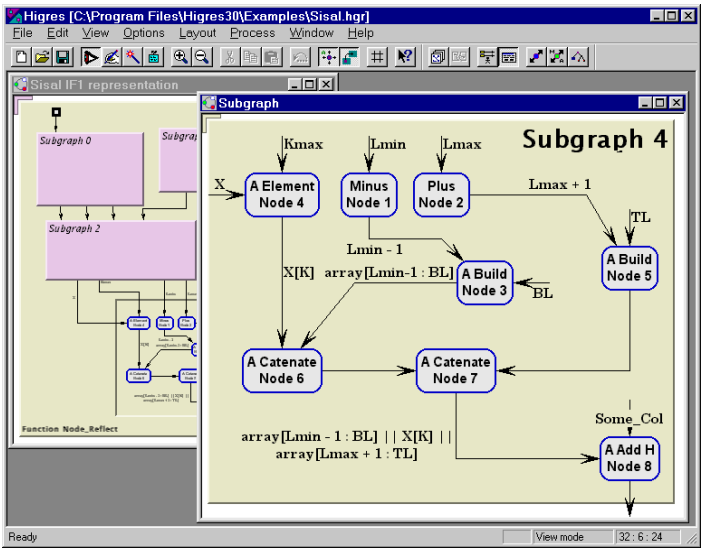


Рис. 1. Иерархический граф в системе Higres

В системе Higres каждый фрагмент визуально задается прямоугольником. Все вершины, принадлежащие фрагменту, располагаются внутри этого прямоугольника. Фрагменты так же, как и вершины, никогда не накладываются друг на друга. Каждый фрагмент может быть либо *закрытым*, либо *открытым*. В первом случае содержимое фрагмента скрыто от пользователя, во втором — видно внутри прямоугольника, представляющего данный фрагмент. Для каждого фрагмента можно открыть отдельное окно, в котором будет видно содержимое только этого фрагмента и его открытых подфрагментов.

Большая часть атрибутов объекта определяется его типом. Следовательно, семантически близкие объекты имеют сходное визуальное представление. Для визуализации меток объекта пользователь задает некоторый шаблон текста, в который вставляются значения меток каждого конкретного объекта.

Возможности визуализации, реализованные в системе, включают:

- различные формы и стили вершин;
- дуги в виде ломаных линий и гладких кривых;
- различные стили линий и стрелок дуг;
- выбор цвета для всех элементов графа;
- возможность задавать произвольный масштаб изображения;
- возможность перемещать текст дуги вдоль ее линии;
- позиционирование внешнего текста вершины в любом месте около ее границы;
- выбор шрифта для всех элементов графа;
- два файловых формата графического вывода;
- широкий набор опций визуализации.

Более полное описание пользовательского интерфейса системы и ее дополнительных возможностей можно найти в [8,9].

3. СХЕМА ЗАДАНИЯ ГЕОМЕТРИЧЕСКИХ АТРИБУТОВ И ВИЗУАЛЬНЫХ ПАРАМЕТРОВ ГРАФА

Все элементы графа в системе располагаются на некоторой абстрактной плоскости, части которой, соответствующие фрагментам графа, можно просматривать в окнах этих фрагментов.

3.1. Вершины. Основными параметрами вершины являются координаты ее центра, высота, ширина и форма. Следует отметить, что в некоторых случаях бывает удобно, чтобы центр вершины имел целые координаты, поэтому задавать вершины координатами углов окаймляющего прямоугольника не вседа правильно.

Каждая вершина может иметь две метки (вернее, два текста, каждый из которых может содержать несколько значений меток, но мы будем для простоты называть эти тексты метками): внутреннюю и внешнюю. Внутренняя метка располагается внутри вершины либо в ее центре, либо в верхнем правом углу, в зависимости от ее параметров. Внешняя метка располагается снаружи вблизи вершины следующим образом. Можно себе представить окаймляющий прямоугольник метки. Его размеры будут зависеть от длины текста и шрифта, следовательно, они будут разными для различных вершин. Метка располагается таким образом, чтобы данный прямоугольник касался каемки вершины. Направление вектора, соединяющего центры вершины и прямоугольника, задается отдельно для каждой вершины. При редактировании графа пользователь может изменять это направление, просто перемещая метку вокруг вершины. Для каждой формы вершины расположение внешней метки вычисляется по-разному, но фактически для задания этого расположения нам нужно знать только параметры вершины, текст метки, шрифт, которым она отображается, и направление соответствующего вектора. На рис. 2 приведены примеры позиционирования внешних меток для вершин различных форм.

Задание расположения внешней метки с помощью указанного вектора удобно потому, что при изменении размеров вершины и метки их приблизительное расположение относительно друг друга остается неизменным. В большинстве случаев при незначительных коррекциях метки или размеров вершины дополнительно подстраивать координаты метки не требуется. В системах, где метки позиционируются отдельно от вершин, это не так.

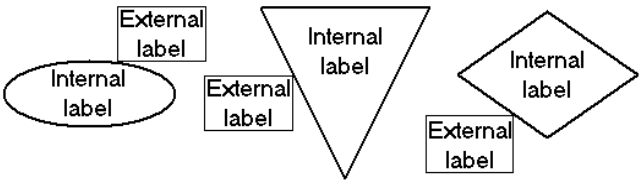


Рис.2. Позиционирование внешних меток вершин

3.2. Фрагменты. Фрагменты задаются координатами левого верхнего и правого нижнего углов. Предполагается, что все вершины и фрагменты, топологически лежащие внутри данного, имеют соответствующие координаты.

Как и вершины, фрагменты имеют две метки, однако отображаются несколько по-другому. Одна из меток отображается в том случае, когда фрагмент закрыт, другая — когда открыт. “Закрытая” метка фрагмента аналогична внутренней метке вершины и располагается либо в верхнем левом углу фрагмента, либо в его центре. “Открытая” метка тоже располагается внутри фрагмента, но ее можно позиционировать в любом месте около границы фрагмента. Как и в случае с внешней меткой вершины, позиционирование производится с помощью задания вектора, соединяющего центры прямоугольника фрагмента и окаймляющего прямоугольника метки.

3.3. Дуги. В системе Higres дуги изображаются как ломаными линиями, так и гладкими кривыми. Рассмотрим способ задания геометрического расположения дуги, имеющей форму ломаной линии. Нужно определить крайние точки линии, лежащие на границах изображений вершин, инцидентных данной дуге, и координаты всех точек сгиба дуги. Так как таких точек может быть произвольное количество, множество точек сгиба представляется списком. Граничные точки (т.е. точки входа в вершины) определяются одним из трех способов, задаваемым для каждой дуги отдельно:

1) ориентацией по центру вершины; при таком способе граничная точка выбирается так, чтобы крайнее звено дуги было направлено к центру вершины (см. фрагменты A и B на рис. 3);
2) ориентацией с помощью дополнительного вектора, задающего направление от центра вершины в точку входа дуги в вершину; вектор задается для двух концов дуги отдельно;
3) ортогональной ориентацией; данный способ используется для построения ортогональных изображений графов, т.е. таких, в которых все дуги представляются ломаными линиями, состоящими из вертикальных и горизонтальных звеньев; при редактировании графа строгую ортогонализацию внутренних звеньев легко произвести с помощью прямоугольной сетки, однако для крайних звеньев лучше всего подходит данный способ ориентации дуги (см. нижнее звено дуги в фрагменте C на рис. 3).

Важное свойство всех приведенных способов задания граничных точек состоит в том, что наиболее важные характеристики взаиморасположения вершины и входящей в нее дуги не изменяются при незначительных передвижениях объектов.

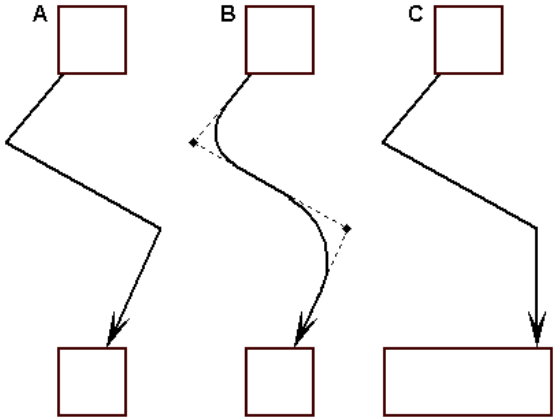


Рис.3. Задание расположения дуг с помощью точек сгиба.
Варианты прикрепления дуг к вершинам

Геометрическое расположение гладких дуг задается точно так же, как и для дуг, — в виде ломаных линий. Гладкая форма получается из ломаной путем “скругления” углов, т.е. замены дугами окружностей частей звеньев ломаной (см. фрагмент B на рис. 3).

Многие аналоги системы используют для изображения гладких дуг сплайны, составленные из кривых 2-го или 3-го порядков. Опыт показывает, что получаемые результаты в этом случае не имеют эстетических преимуществ над описанным выше способом. В то же время можно выделить два основных преимущества схемы, реализованной в системе Higres:

- более очевидный способ задания геометрического расположения (с помощью ориентирующей ломаной линии);
- более высокое быстродействие, связанное с использованием графических примитивов, отображаемых низкоуровневыми функциями операционной системы (в MS Windows нет функций для изображения произвольных кривых 2-го порядка).

Одной из наиболее сложных задач при проектировании схемы задания геометрических атрибутов графа является задание расположения меток дуг. Это связано с тем, что дуги геометрически могут представляться более разнообразно, чем вершины и фрагменты. В системе Higres эта задача решена следующим образом.

Так как расположение дуги задается координатами точек сгиба и точек крепления дуги к инцидентным вершинам, разумно именно к этим точкам привязывать координаты меток. Это обеспечит нам основное полезное свойство: при перемещении дуги метка перемещается вместе с ней, причем расположение ее будет сохраняться при незначительном изменении расположения дуги.

В то же время хочется достичь определенной гибкости в задании расположения метки, т. е. иметь возможность позиционировать ее лишь в месте одной из точек сгиба (или точки крепления) представляется не достаточно гибким способом. В системе Higres введено понятие точек привязки меток. Точками привязки метки для дуги являются:

а) точки сгиба;
б) точки входа дуги в инцидентные вершины;
в) точки, лежащие на серединах звеньев ломаной, представляющей дугу (для гладких дуг имеется в виду воображаемая ломаная линия, из которой получается линия дуги путем скругления углов).

Расположение метки дуги задается двумя параметрами: номером точки привязки и способом привязки. Существует 5 способов привязки:

а) точка привязки совпадает с серединой окаймляющего прямоугольника метки;
б) точка привязки совпадает с левым верхним углом окаймляющего прямоугольника метки;
в) точка привязки совпадает с левым нижним углом окаймляющего прямоугольника метки;
г) точка привязки совпадает с правым верхним углом окаймляющего прямоугольника метки;
д) точка привязки совпадает с правым нижним углом окаймляющего прямоугольника метки.

4. ПОСТРОЕНИЕ ИЗОБРАЖЕНИЯ ГРАФА

4.1. Вершины. На рис. 4 приведены различные варианты форм и стилей вершин, используемые в системе. Визуальные атрибуты, определяемые для каждого типа вершин, включают:

- форму (6 вариантов);
- стиль каемки (6 вариантов);
- цвет внутренней части (задается произвольно);
- цвет каемки (задается произвольно);
- шрифт, которым отображается внутренняя и внешняя метки.

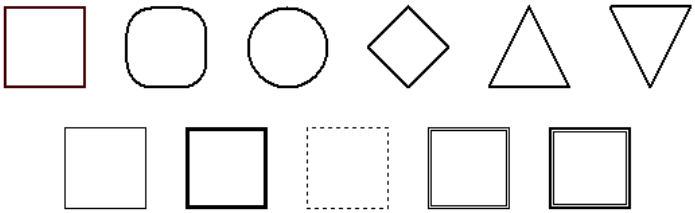


Рис. 4. Варианты форм и стилей вершин

4.2. Фрагменты. Изображение фрагмента имеет несколько меньше графических атрибутов, чем изображение вершины, а именно: для каждого типа фрагментов задается

- цвет фрагмента в закрытом состоянии;
- цвет фрагмента в открытом состоянии;
- шрифт, которым отображаются метки.

Закрытые и открытые фрагменты изображаются не только разными цветами, но и с каемками разного вида. Прямоугольники открытых фрагментов визуально представляются углублениями, а закрытых — выступами. Можно сказать, что закрытые фрагменты выглядят как участки плоскости, закрытые крышками (рис. 5).

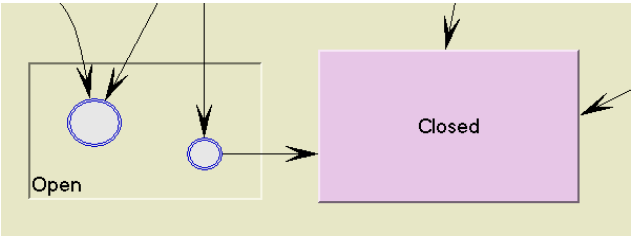


Рис .5. Изображение открытого и закрытого фрагментов

4.3. Дуги. Изображение дуг — наиболее сложная задача при выводе изображения графа. Функции, использующиеся исключительно для этой цели, в системе Higres в совокупности представляют собой около 500 строк кода на языке C++. Сложность изображения дуг обусловлена в первую очередь особенностями графического вывода дуг окружности в системе Windows. Во-первых, все параметры функции вывода дуги являются целочисленными. Следовательно, происходит двойное округление. Первый раз точные параметры дуги округляются при вызове функции, второй — исходя из уже округленных параметров производятся расчеты внутри этой функции, результаты которых опять округляются при выводе изображения. Во-вторых, вывод отрезков и дуг окружности существенно зависит от толщины линии. Обе проблемы фактически приводят к тому, что линии, из которых состоит дуга, плохо стыкуются друг с другом, что существенно ухудшает качество изображения. Чтобы избежать этого эффекта, необходимо при выводе корректировать координаты всех графических примитивов в зависимости от их относительного расположения и толщины линии.

Общая схема скругления углов ломаной линии при выводе гладких дуг приведена на рис. 6. Точки A, B и C — три последовательные точки сгиба дуги (A и C могут быть точками крепления дуги к вершине). Чтобы скруглить угол ABC, выбираем наибольший из отрезков AB и BC. Пусть это будет BC. Откладываем на нем точку N так, чтобы AB = BN. Координаты точки N легко вычисляются исходя из координат точек A, B и C. Делим отрезки AB и BN пополам. Получаем точки M и P. Их координаты также легко вычисляются. Наша цель — заменить дугой MP угол MBP. Для этого нужно вычислить центр окружности, вписанной в данный угол и касающейся его сторон в точках M и P. Этот центр будет лежать на пересечении перпендикуляров к AB и BC, проведенных через точки M и P соответственно. Уравнения прямых AB и BC легко получаются по координатам исходных

точек. Уравнения перпендикуляров к ним выводятся известным из аналитической геометрии способом.

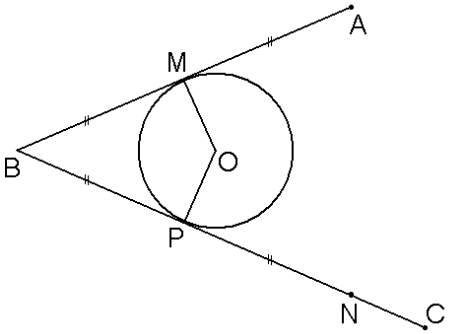


Рис. 6. Скругление угла при рисовании гладких дуг

Естественно, расчеты по данной схеме вычислений при реализации требуют отдельного рассмотрения всех экстремальных случаев (когда какие-либо из использующихся прямых параллельны осям координат).

Для ориентированных графов на концах дуг необходимо изображать стрелки, указывающие направление дуги. На рис. 7 приведены три варианта наконечников ориентированных дуг, используемых в системе. При рисовании наконечников производятся достаточно простые геометрические расчеты. Как и в случае с линией дуги, эти расчеты требуют корректирующей поправки для адаптации к функциям системы Windows и учета ширины линии.

Рис. 7. Варианты наконечников дуг

Заметим также, что для линий максимальной ширины (для дуг 3 пикселя) при рисовании крайнего звена основной линии дуги необходимо учитывать наличие наконечника, поскольку он имеет заостренную форму, сужающуюся до одного пикселя.

При изображении иерархических графов необходимо уметь изображать дуги, идущие внутрь закрытых фрагментов. Так как в данном случае вер-

шина, в которую идет дуга, скрыта внутри фрагмента, дуга должна заканчиваться на границе фрагмента. При этом если граф ориентированный, то дуга должна заканчиваться наконечником, указывающим на фрагмент (рис. 8). Естественно, изображение таких дуг должно быть как можно ближе к изображению тех же дуг при открытом состоянии фрагмента. В системе Higres используется следующий алгоритм для решения данной задачи.

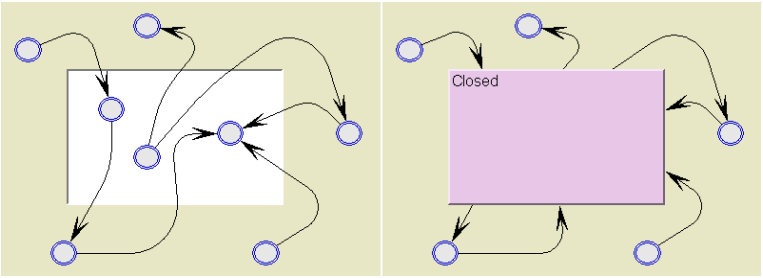


Рис. 8. Изображение дуг, идущих внутрь закрытых фрагментов

Если требуется изобразить дугу, одна из инцидентных вершин которой лежит внутри закрытого фрагмента, а другая за его пределами, перебираем отрезки, соединяющие последовательные точки сгиба данной дуги, начиная с конца, идущего от внешней вершины. Под точками сгиба будем понимать как непосредственные точки сгиба, так и точки крепления дуги к вершинам.

Как только при переборе дойдем до отрезка, пересекающего границу фрагмента, вычисляем координаты точки пересечения и заменяем последнюю из рассмотренных точек сгиба дуги на данную точку. Далее рисуем дугу по полученному набору точек сгиба.

Аналогичным способом решается задача изображения дуги, соединяющей вершины, принадлежащие двум различным закрытым фрагментам.

4.4. Метки. При изображении внешних меток вершин и меток дуг возникает проблема цвета фона. Для каждого типа вершин и дуг используется параметр, который указывает, заполнять ли фон при отображении меток данного типа объектов или нет. Отображение фона необходимо использовать, например, в случае, если текст метки накладывается непосредственно на линию дуги. Тогда текст может “слиться” с этой линией, если выводить его с выключенной опцией фона. С другой стороны, если эта опция включена, то возникает следующая проблема. Метка может накладываться на произ-

вольный участок плоскости. Теоретически он может содержать произвольные объекты произвольного цвета. Следовательно выбор цвета фона неоднозначен. Для наиболее правильного выбора этого цвета используется следующий метод. Просчитывается окаймляющий прямоугольник метки и берется 5 образцов цвета фона из точек, лежащих под углами данного прямоугольника и под его центром. Цвет, присутствующий в большинстве перечисленных точек, используется в качестве цвета фона. В большинстве случаев данный подход дает оптимальный результат.

4.5. Масштабирование изображения. Важным параметром системы визуализации является ее способность создавать качественные изображения графа при любом масштабе. Основная сложность при масштабировании состоит в том, что практически невозможно и в большинстве случаев нерационально пытаться масштабировать все элементы изображения одинаковым образом. В системе Higres масштабирование производится по следующей схеме.

- Размеры вершин и фрагментов изменяются при изменении масштаба.
- Толщина линий, включая линии дуг, каемок вершин и фрагментов, не изменяется.
- Шрифты масштабируются средствами Windows, что дает не полностью корректный, но приемлемый результат.
- Размеры наконечников дуг либо изменяются пропорционально масштабу, либо остаются неизменными в зависимости от соответствующего параметра типа дуги.

Для получения более качественного изображения координаты всех объектов при масштабировании переводятся в целые числа только при получении итогового расположения элементов графа.

5. ОПТИМИЗАЦИЯ ГРАФИЧЕСКОГО ВЫВОДА

Правильно выбранная схема задания геометрических характеристик элементов графа и применение адекватных методов построения изображения по этой схеме создают основу для получения быстродействующей системы визуализации. Однако необходимо также уделить внимание последнему этапу получения изображения — непосредственно графическому вы-

воду. Так как задача сводится к отображению на экране (или другом подобном устройстве вывода) уже просчитанного с точностью до пикселя изображения, решение существенно зависит от особенностей операционной системы.

Метод графического вывода, реализованный в системах MS Windows, можно кратко описать следующим образом. В каждом окне есть клиентская часть, за перерисовку которой отвечает приложение, создавшее данное окно, т. е. приложение предоставляет операционной системе функцию, которую нужно вызывать каждый раз, когда требуется перерисовать клиентскую область окна или какую-либо ее часть. Это происходит либо в случае, когда другое окно перемещается поверх данного и требуется перерисовать открывшийся участок, либо по инициативе самого приложения в случае, когда нужно обновить содержимое окна.

В обоих случаях перерисовываемый участок имеет прямоугольную форму. Таким образом, задача, решаемая функцией перерисовки, сводится к выводу всех элементов изображения, находящихся внутри некоторого прямоугольника с заданными координатами.

Самое элементарное решение данной задачи (к сожалению, часто используемое на практике) состоит в том, чтобы перерисовать все изображение. Механизм перерисовки Windows устроен так, что при этом фактически на экране будет отображаться только нужная часть. Оптимизированный вариант функции должен попытаться не тратить время на рисование элементов, лежащих за пределами области перерисовки. Именно такой метод применен в системе Higres.

Другая важная оптимизация состоит в следующем. Далеко не все действия пользователя, требующие перерисовки изображения графа, требуют также изменения графического представления элементов графа. Например, если пользователь просто прокручивает изображение в окне, в принципе нет необходимости пересчитывать расположение графических примитивов, из которых состоят изображения элементов графа. Этот пересчет нужен только при изменении графа (включая перемещение элементов и модификацию любых графических параметров) и масштаба изображения.

В системе Higres реализована оптимизированная трехэтапная схема графического вывода. Для построения изображения фрагмента графа (напомним, что нам всегда требуется нарисовать какой-либо фрагмент, так как именно фрагменты ассоциированы с окнами) на первом этапе строится множество графических примитивов (с координатами и прочими атрибутами), из которых состоит изображение. При этом рассматриваются только

вершины и фрагменты графа, лежащие внутри данного, и только дуги, инцидентные этим вершинам. Это множество запоминается в виде списка и пересчитывается только тогда, когда происходит изменение графа, способное повлиять на изображение данного фрагмента. Заметим, что такое изменение может быть произведено пользователем и внутри окна другого фрагмента. Для определения области влияния модификаций графа применяется специальный алгоритм, который позволяет избежать излишних пересчетов и перерисовок.

На втором этапе строится изображение перерисовываемого прямоугольника в памяти компьютера. Наконец, на третьем этапе сформированное в памяти изображение переносится в нужное место экрана. Необходимость промежуточного этапа рисования в памяти обусловлена двумя причинами. Во-первых, операционная система Windows устроена таким образом, что процесс рисования в памяти происходит быстрее, чем непосредственно на экране, даже если учесть время, потраченное на дополнительную операцию копирования на экран. Во-вторых, при таком подходе обновленное изображение появляется на экране мгновенно, даже если система потратила некоторое время на его построение.

Применение приведенной схемы оптимизации графического вывода позволяет прежде всего сделать максимально комфортной для пользователя процедуру просмотра графа, поскольку традиционная задержка при перерисовке прокручиваемого изображения графа сведена к минимуму. Хотя время этой задержки в любом случае линейно зависит от количества элементов в фрагменте, линейная составляющая начинает сказываться только на достаточно больших фрагментах (порядка нескольких сотен вершин).

6. ВИЗУАЛИЗАЦИЯ ПРОЦЕССА РЕДАКТИРОВАНИЯ ГРАФА

Система Higres является не только визуализатором графовых моделей, но и полноценным графовым редактором. Это означает, что с помощью нее можно вносить в модель произвольные изменения. Многие модификации графа должны производиться визуально. К таким модификациям относятся следующие:

- добавление и удаление вершин, фрагментов и дуг;
- перемещение вершин, фрагментов и дуг;
- позиционирование меток элементов графа.

Практически во всех системах редактирование осуществляется с помощью выделения одного или нескольких объектов и последующих манипуляций с выделенными объектами с помощью мыши, например, можно выделить вершину и далее переместить ее в другое место изображения. При реализации этого подхода возникают две основные проблемы.

1. Выделение должно производиться таким образом чтобы пользователь мог легко отличать выделенные объекты от невыделенных. Это естественное требование достаточно легко соблюсти, когда система позволяет выделять только один или несколько однотипных объектов. В нашем случае имеется несколько видов объектов (вершины, фрагменты и дуги), которые, кроме того, могут накладываться друг на друга. Для того чтобы можно было выделять несколько объектов разных видов одновременно, необходимо использовать различные способы выделения для различных видов объектов, а также производить выделение каждого объекта таким образом, чтобы оно как можно точнее указывало на его расположение.

2. В процессе изменения пользователем геометрических параметров редактируемого элемента крайне нежелательно производить полную перерисовку изображения графа, поскольку даже при всех возможных оптимизациях это очень дорогостоящий по времени процесс. В то же время пользователь должен иметь возможность отслеживать визуальные параметры объектов во время редактирования. Например: если пользователь перемещает группу вершин с помощью мыши, то, с одной стороны, он должен видеть их текущую позицию в каждый момент времени, с другой стороны, если для этого при каждом сдвиге перерисовывается все изображение фрагмента, интерфейс системы будет явно слишком медленным при достаточно большом размере графа.

На рис. 9 показаны различные варианты выделения объектов при редактировании в системе Higres: выделение нескольких объектов различных видов (A), выделение одного фрагмента (B), вершины (C) и дуги (D). Во всех случаях выделение производится пунктирной линией. В первом случае пунктиром показываются только окаймляющие прямоугольники фрагментов и вершин, а также линии дуг. В остальных случаях, когда выделен только один объект, выделение имеет дополнительные свойства. Для фрагмента прямоугольником выделяется его метка (пользователь может позиционировать ее с помощью мыши), а также в центре прямоугольника фрагмента изображается перекрестие для того, чтобы сделать выделение фрагмента отличным от выделения вершины. Для вершины дополнительно выделяется

текст ее внешней метки. Как и в случае с фрагментами, этот текст можно позиционировать мышью. Для дуги дополнительно выделяется текст метки и все ее точки сгиба.

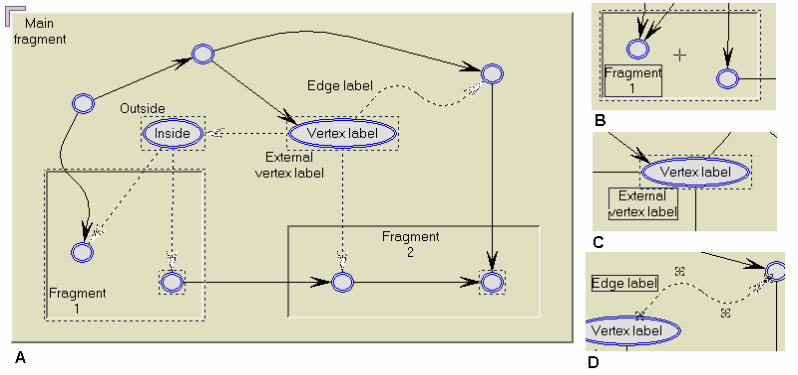


Рис. 9. Выделение элементов графа при редактировании

Вторая из приведенных выше проблем решается в системе Higres следующим образом. Изображение выделенного объекта получается из изображения невыделенного добавлением дополнительных пунктирных линий, которые рисуются методом инвертирования изображения, т. е. если дважды нарисовать одну и ту же линию, получим исходное изображение. Таким образом, чтобы убрать с изображения выделение, достаточно просто еще раз его отобразить. Если нам нужно переместить выделение (например, если пользователь перемещает выделенные объекты с помощью мыши), достаточно вывести его один раз со старыми координатами (что вернет нам изображение без выделения), а затем вывести его с новыми координатами.

Так как выделение состоит из набора графических примитивов, разумно применить к нему ту же оптимизацию, что и к основному изображению, а именно: запоминать отдельно множество примитивов, используемых для вывода выделения, и пересчитывать его только тогда, когда оно существенно изменяется. В отличие от основного изображения выделение выводится непосредственно на экран, поскольку в данном случае промежуточный этап вывода в память не может служить оптимизацией. Выделение, как правило, состоит из достаточно маленького числа линий, поэтому их вывод занимает намного меньше времени, чем копирование всего изображения из памяти на экран.

7. ЗАКЛЮЧЕНИЕ

Мы рассмотрели методы организации графического вывода в системах визуализации графовых объектов. Отмечены и классифицированы основные проблемы, возникающие при проектировании таких систем, даны общие методы их решения и описан конкретный вариант реализации данных подходов в системе Higres, являющейся визуализатором и редактором иерархических графовых моделей.

ЛИТЕРАТУРА

1. **Лисицын И.А.** Системы визуализации и редактирования графовых объектов. — Новосибирск: ИСИ СО РАН. — (В печати).
2. **Касьянов В.Н.** Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 7–32.
3. **Густокашина Ю.В., Евстигнеев В.А.** IF1 — промежуточное представление SISAL-программ // Проблемы конструирования эффективных и надежных программ. — Новосибирск, 1995. — С. 70–78.
4. **Eades P., Feng Q.W.** Drawing clustered graphs on an orthogonal grid // Proc. of Graph Drawing 97. — Berlin a.o.: Springer Verlag, 1997. — P. 182–193. — (Lect. Notes Comput. Sci.; Vol. 1353).
5. **Eades P., Feng Q.W.** Multilevel visualization of clustered graphs // Proc. of Graph Drawing 96. — Berlin a.o.: Springer Verlag, 1996. — P. 101–112. — (Lect. Notes Comput. Sci.; Vol. 1190).
6. **Eades P., Feng Q.W., Lin X**. Straight-line drawing algorithms for hierarchical graphs and clustered graphs // Proc. of Graph Drawing 96. — Berlin a.o.: Springer Verlag, 1996. — P. 113–128. — (Lect. Notes Comput. Sci.; Vol. 1190).
7. **Feng Q.W., Cohen R., Eades P**. How to draw a planar clustered graph // In COCOON '95. — Berlin a.o.: Springer Verlag, 1995. — P. 21–31. — (Lect. Notes Comput. Sci.; Vol. 959).
8. **Lisitsyn I.A. Kasyanov V.N**. Higres — Visualization system for clustered graphs and graph algorithms // Proc. of Graph Drawing 99. — Berlin a.o.: Springer Verlag, 1999. — P. 82–89. — (Lect. Notes Comput. Sci.; Vol. 1731).
9. **Лисицын И.А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С.64–77.

И.А. Лисицын

ОРГАНИЗАЦИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА В СИСТЕМЕ ВИЗУАЛИЗАЦИИ ИЕРАРХИЧЕСКИХ ГРАФОВЫХ МОДЕЛЕЙ[1]

1. ВВЕДЕНИЕ

Графовые модели традиционно считаются наиболее адекватным и наглядным способом представления структурированной информации. Для эффективного применения этих моделей на практике необходимо наличие хороших инструментальных средств их визуализации и обработки. Анализ существующих на сегодняшний день средств визуализации графов [1] показывает, что важными характеристиками системы визуализации являются удобный пользовательский интерфейс и наличие функций, позволяющих пользователю легко ориентироваться в структуре визуализируемого объекта, а также функций, автоматизирующих процесс редактирования.

Настоящая статья посвящена организации пользовательского интерфейса в системе Higres, являющейся визуализатором и редактором иерархических графов. Точное определение иерархических графов и других понятий, связанных с ними, можно найти в [2]. Различным вопросам визуализации иерархических графов посвящены работы [4–7]. Подробное описание системы Higres содержится в [8,9], а также в [3], где также описана организация графического вывода в системе.

2. ВИЗУАЛИЗАЦИЯ ИЕРАРХИЧЕСКОЙ СТРУКТУРЫ ГРАФА

Для визуализации иерархической структуры графа в системе применяется следующий подход. Каждый фрагмент графа связан с отдельным окном, которое можно открыть внутри гравного окна системы. Окна фрагментов можно открывать и закрывать по мере необходимости. Обозревая фрагмент в окне родительского фрагмента, можно с помощью двойного щелчка левой кнопки мыши открыть окно данного фрагмента.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 00-07-90296) и Министерства образования РФ.

На рис. 1 показано окно системы, содержащее несколько окон фрагментов. Среди этих фрагментов есть как открытые, так и закрытые. Масштаб изображения в каждом окне выбирается произвольно. Таким образом, с помощью открытия дополнительных окон можно обозревать структуру графа на нескольких уровнях абстракции.

3. РЕЖИМЫ ПРОСМОТРА И РЕДАКТИРОВАНИЯ ГРАФА

Просмотр и редактирование графа в системе Higres происходит в двух различных режимах, что позволяет предоставить пользователю максимально удобный интерфейс для того вида работы с системой, который интересует его в данный момент.

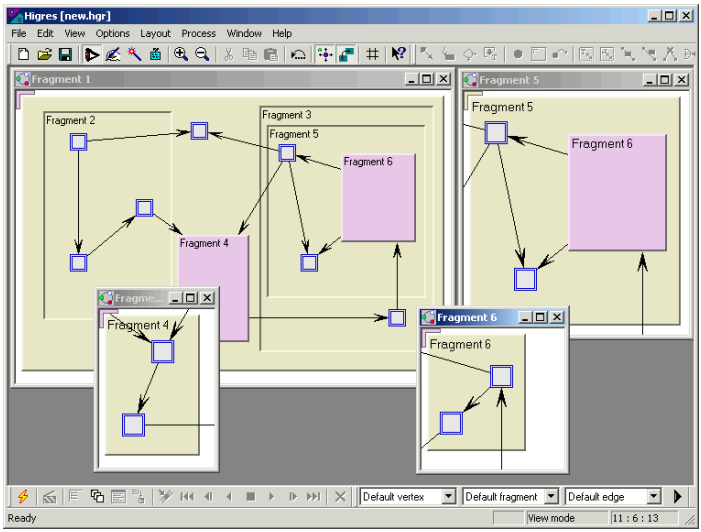


Рис. 1. Визуализация иерархической структуры графа

В режиме просмотра пользователю доступны следующие функции:

- просмотр графа в окнах фрагментов;
- прокручивание изображения фрагмента с помощью линеек прокрутки окна фрагмента, клавиш со стрелочками или “перетаскивания” мышью;

– изменение масштаба изображения с помощью клавиш “+” и “-” или соответствующих пунктов главного меню;
– открытие и закрытие фрагментов с помощью выбора соответствующих команд из меню, всплывающего при нажатии правой кнопки мыши на изображении фрагмента;
– открытие окна фрагмента с помощью двойного щелчка мышью на изображении данного фрагмента в окне внешнего фрагмента;
– все прочие операции, доступные через меню и панели инструментов системы.

В режиме редактирования графа пользователь может производить любые изменения структуры графа и его расположения на плоскости. В данном режиме также доступны почти все функции просмотра, кроме перетаскивания изображения мышью. Данная функция мыши используется для выделения элементов графа и перемещения выделенных частей.

Для того чтобы выделить один объект или группу объектов, пользователь нажимает левую кнопку мыши и, удерживая ее в нажатом состоянии, перемещает курсор. Точка, в которой произошло нажатие кнопки, и текущее положение курсора задают два противоположных угла прямоугольника, с помощью которого происходит выделение. Когда пользователь отпускает кнопку мыши и координаты данного прямоугольника фиксируются, выделяются все вершины, дуги и закрытые фрагменты, части изображений которых оказываются внутри прямоугольника.

Если при выполнении данной процедуры удерживать в нажатом состоянии клавишу “Shift”, то вновь выделенные объекты добавятся к уже имеющемуся выделению. Если удерживать клавишу “Control”, то произойдет инвертирование выделенного состояния данных объектов.

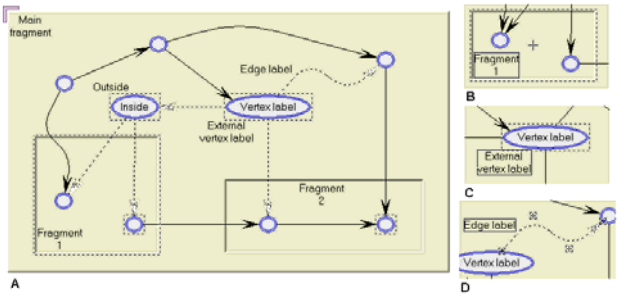


Рис. 2. Режим редактирования

Можно выделить (или добавить к выделению) один объект, для чего достаточно щелкнуть левой кнопкой мыши на его изображении. Заметим, что это единственный способ выделения открытых фрагментов, которые нельзя выделить вместе с группой других объектов. Для выделения одной дуги нужно щелкнуть левой кнопкой мыши рядом с линией дуги.

Набор операций редактирования, которые пользователь может производить с выделенными объектами, зависит от того, выделен ли один объект или несколько, и если один, то какой именно (вершина, фрагмент или дуга). На рис. 2 показаны случаи выделения нескольких различных объектов (A), одного фрагмента (B), одной вершины (C) и одной дуги (D).

Если выделена одна вершина, пользователь может производить следующие операции:

- перемещать вершины с помощью перетаскивания мышью;
- изменять размеры вершины с помощью перетаскивания мышью границы вершины;
- перемещать внешний текст метки. При выделении одной вершины внешний текст метки этой вершины окаймляется прямоугольником. Этот прямоугольник можно вращать вокруг вершины с помощью мыши;
- удалять вершину (с помощью всплывающего меню);
- добавлять в граф дугу, начинающуюся в данной вершине. Для выполнения этой операции нужно сначала выбрать соответствующий пункт всплывающего меню, а затем щелчками правой кнопки мыши последовательно отметить на экране все точки сгиба добавляемой дуги и ее конечную вершину.

Если выделен один фрагмент, пользователь может производить следующие операции:

- перемещать фрагмент с помощью перетаскивания мышью; при этом все вершины, фрагменты и сгибы дуг, лежащие внутри фрагмента, перемещаются вместе с ним; таким образом, изображение графа внутри фрагмента не изменится;
- изменять размер прямоугольника фрагмента с помощью перетаскивания границы фрагмента (только для открытых фрагментов);
- перемещать метку открытого фрагмента вдоль границы фрагмента (аналогично перемещению внешней метки вершины);

- разворачивать фрагмент (удалять с сохранением содержимого фрагмента, которое автоматически переходит во внешний по отношению к данному фрагмент);
- удалять фрагмент вместе со всем его содержимым (рекурсивное удаление);

Если выделена одна дуга, пользователь может производить следующие операции:

- перемещать точки сгиба дуги с помощью мыши; точки сгиба при выделении одной дуги помечаются перекрестиями;
- перемещать точки входа дуги в инцидентные вершины вокруг каемки вершины;
- перемещать текст метки дуги вдоль линии дуги;
- удалять точку сгиба дуги; данная команда добавляется в сплывающее меню, если щелкнуть правой кнопкой мыши около точки сгиба;
- вставлять в дугу новую точку сгиба; вставка происходит между теми точками сгиба, где произошло нажатие правой кнопки для вызова всплывающего меню;
- ориентировать дугу по центру инцидентной вершины;
- ортогонально ориентировать дугу по отношению к инцидентной вершине;
- удалять дугу.

При выделении одного объекта независимо от его вида доступны также следующие операции:

- редактирование значений меток с помощью специального блока диалога;
- изменение типа; при этом, естественно, значения меток объекта утрачиваются.

При выделении произвольной группы объектов пользователь также может:

- перемещать выделенную группу с помощью мыши;
- производить операции Cut/Copy/Paste;
- пользоваться рядом дополнительных функций.

Кроме того, независимо от наличия выделенных объектов в режиме редактирования пользователь может выбрать из всплывающего меню следующие команды.

- Создание вершины. После выбора данной команды нужно выделить мышью окаймляющий прямоугольник новой вершины, и она будет добавлена в граф.
- Создание фрагмента. Как и в случае с добавлением вершины, после выбора данной команды необходимо выделить окаймляющий прямоугольник фрагмента. Если в этот прямоугольник попадут некоторые вершины и фрагменты, они будут принадлежать созданному фрагменту.
- Создание вершины и идущей из нее дуги. Выполнение данной команды эквивалентно последовательному выполнению добавления вершины и, далее, добавления дуги, выходящей из этой вершины.

При создании, перемещении и изменении размеров объектов пользователь может применять прямоугольную сетку для выравнивания положения объектов. Для активирования сетки следует в процессе редактирования удерживать клавишу “Control” в нажатом состоянии.

Перед созданием новых объектов необходимо выбрать тип объекта с помощью панели инструментов “Types”. Данная панель содержит три элемента списка для выбора текущего типа вершин, фрагментов и дуг. При выделении какого-либо объекта текущий тип также переключается соответствующим образом.

В процессе редактирования некоторая полезная информация о выделенной части графа, связанная с текущей операцией редактирования, отображается в панели состояния в нижней части главного окна системы.

Для удобства пользователя в системе предусмотрено два дополнительных режима, в которых можно более эффективно выполнять отдельные операции редактирования, – режимы создания объектов и режим редактирования меток.

4. СОЗДАНИЕ И РЕДАКТИРОВАНИЕ ТИПОВ ОБЪЕКТОВ

Семантика графа передается в системе с помощью типов объектов. Создание, удаление и редактирование параметров типов происходит с помощью одного блока диалога, содержащего слева список типов, а справа три страницы свойств: первая страница содержит основные свойства, вторая – свойства, связанные с выводом текста меток, третья задает сами метки. На рис. 3 и 4 соответственно показаны первая и последняя страницы.

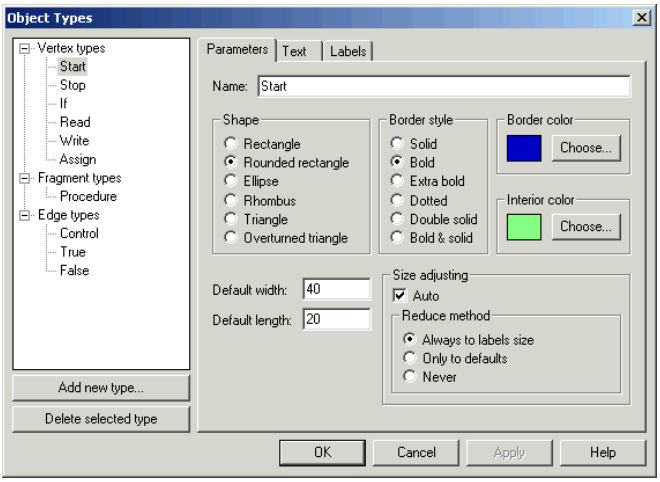


Рис. 3. Блок диалога типов объектов (страница основных свойств)

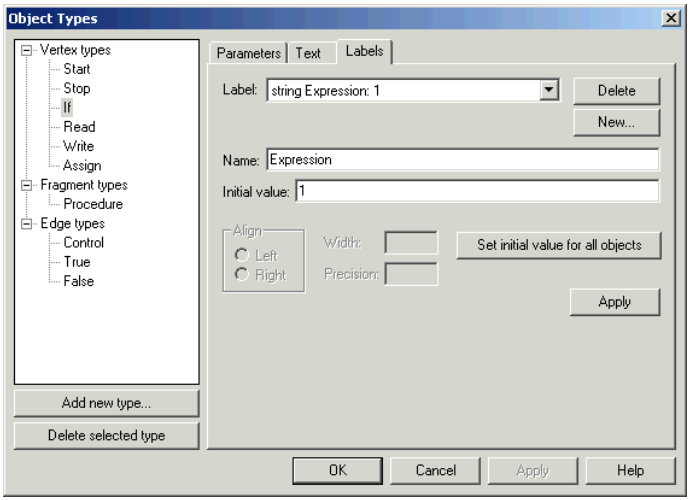


Рис. 4. Блок диалога типов объектов (страница меток)

Под списком типов расположены кнопки, позволяющие создавать и удалять типы. В системе есть опция, позволяющая создавать новые типы либо

с минимальным набором меток, либо совсем без меток. Новые метки можно добавить, нажав кнопку “New…” на последней странице блока диалога.

5. АВТОМАТИЧЕСКАЯ КОРРЕКЦИЯ ИЗОБРАЖЕНИЯ

Одним из важных параметров любого редактора является способность автоматически подстраивать изображение редактруемого документа или объекта для приведения его в соответствие с некоторыми заранее определенными требованиями. В системе Higres реализовано две возможности такого рода. Во-первых – подстраивание размеров вершин под размер текста их внутренних меток. Заметим, что для вершин различных форм данная коррекция размеров происходит различным образом. Во-вторых – устранение пересечений вершин и фрагментов. Данная коррекция применяется после каждой операции редактирования, которая могла привести к изменению размеров или координат вершин и фрагментов.

Изображение графа в системе всегда должно удовлетворять следующим ограничениям:

1. Изображения вершин не пересекаются.
2. Прямоугольник фрагмента целиком включает в себя изображения тех и только тех вершин и фрагментов, которые ему принадлежат.

При перемещении элементов графа или изменении их размеров пользователь может легко нарушить любое из этих ограничений, поэтому необходима дополнительная коррекция изображения, которая устранит нарушение.

Предусмотрены две опции, определяющие поведение системы при корректировке операций редактирования, приводящих к нарушению приведенных правил: “Allow auto move” и “Allow F to F move”.

Если первая из этих опций выключена, операция игнорируется и подстраивание не производится. Вторая опция разрешает перемещать вершины и подфрагменты из одного фрагмента в другой. Если эта опция выключена, попытка переместить вершину или подфрагмент за пределы фрагмента приведет к подстраиванию размеров фрагмента таким образом, чтобы перемещаемые объекты снова оказались внутри него.

6. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ СИСТЕМЫ

В системе предусмотрен ряд дополнительных возможностей, облегчающих редактирование графа и делающих использование системы более удобным.

1. Глобальные операции:
   - подстраивание размеров всех вершин графа;
   - оптимизация размеров всех фрагментов графа;
   - разворачивание всех фрагментов (устранение иерархии);
   - унификация расположения меток дуг (2 способа);
   - устранение сгибов всех дуг;
   - приведение всех дуг к центральному и к ортогональному входу в вершины;
   - отбрасывание семантики графа (устранение типов и меток).
2. Операция Undo. Количество операций редактирования, которые можно обратить, определяется пользователем и может быть произвольным (чем оно больше, тем больше памяти расходуется).
3. Опция автоматической проверки одновходовости фрагментов. Если эта опция включена, система не позволит модифицировать граф таким образом, чтобы в нем возникли фрагменты, имеющие более одной начальной вершины.
4. Запоминание списка открывавшихся файлов и возможность редактировать его и использовать для повтороного открытия.
5. Использование упрощенных блоков диалога при редактировании меток объектов в случае, если у объекта всего одна или две метки. Данная опция позволяет в большинстве случаев существенно ускорить процесс редактирования меток.
6. Обзорное окно (рис. 5). Данное окно применяется для просматривания достаточно больших графов. В нем всегда показан целиком фрагмент, окно которого активно в настоящий момент. Кроме того, выделен прямоугольник, соответствующий части фрагмента, видимой в его окне. Этот прямоугольник можно перемещать по обзорному окну с помощью мыши. Изображение в окне фрагмента будет перемещаться соответствующим образом.

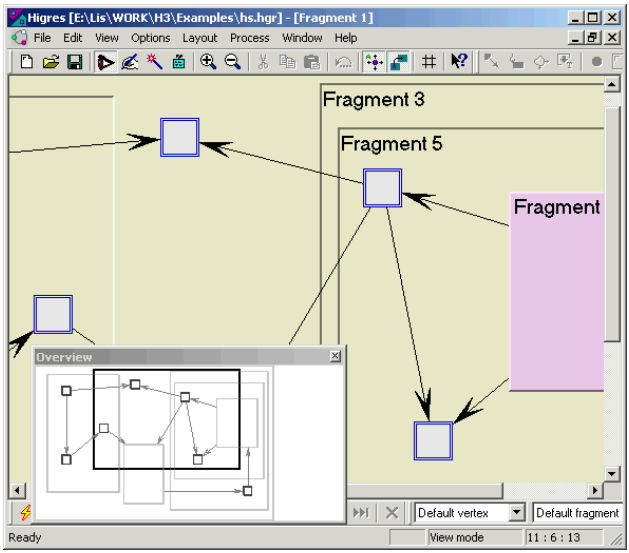


Рис. 5. Обзорное окно.

7. ЗАКЛЮЧЕНИЕ

Рассмотрена организация пользовательского интерфейса в системе Higres, являющейся визуализатором и редактором иерархических графовых моделей. Описаны особенности визуализации иерархической структуры графа, методы, позволяющие предоставить пользователю возможность быстро и удобно редактировать граф и его изображение, алгоритмы автоматического достраивания изображения, а также некоторые дополнительные возможности, реализованные в системе.

ЛИТЕРАТУРА

1. **Лисицын И.А.** Системы визуализации и редактирования графовых объектов. — Новосибирск: ИСИ СО РАН. — (В печати).
2. **Касьянов В.Н.** Иерархические графы и графовые модели: вопросы визуальной обработки // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 7–32.

3. **Лисицын И.А.** Организация графического вывода в системе визуализации иерархических графовых моделей // Настоящий сб. — С. 194—211.
4. **Eades P., Feng Q.W.** Drawing clustered graphs on an orthogonal grid // Proc. of Graph Drawing 97. — Berlin a.o.: Springer Verlag, 1997. — P. 182–193. — (Lect. Notes Comput. Sci.; Vol. 1353).
5. **Eades P., Feng Q.W.** Multilevel visualization of clustered graphs // Proc. of Graph Drawing 96. — Berlin a.o.: Springer Verlag, 1996. — P. 101–112. — (Lect. Notes Comput. Sci.; Vol. 1190).
6. **Eades P., Feng Q.W., Lin X**. Straight-line drawing algorithms for hierarchical graphs and clustered graphs // Proc. of Graph Drawing 96. — Berlin a.o.: Springer Verlag, 1996. — P. 113–128. — (Lect. Notes Comput. Sci.; Vol. 1190).
7. **Feng Q.W., Cohen R., Eades P.** How to draw a planar clustered graph // In COCOON '95. — Berlin a.o.: Springer Verlag, 1995. — P. 21–31. — (Lect. Notes Comput. Sci.; Vol. 959).
8. **Lisitsyn I.A. Kasyanov V.N.** Higres – Visualization system for clustered graphs and graph algorithms // Proc. of Graph Drawing 99. — Berlin a.o.: Springer Verlag, 1999. — P. 82–89. — (Lect. Notes Comput. Sci.; Vol. 1731).
9. **Лисицын И.А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С.64–77.

Т. С. Мердишева, Е. С. Мердишева

ПОДГОТОВКА ГРАФОВЫХ ИЛЛЮСТРАЦИЙ С ПОМОЩЬЮ СИСТЕМЫ VEGRAS[1]

1.ВВЕДЕНИЕ

Графы являются удобным инструментом для представления различных сложных структур. В настоящее время растет интерес к методам и системам визуализации графов. В частности, появилось достаточно много программных систем, обеспечивающих работу с графами. Не так давно в лаборатории конструирования и оптимизации программ была разработана система визуальной обработки иерархических графовых моделей HIGRES [1,2]. Однако все подобные средства ориентируются на какую-то конкретную прикладную область, и применение их в какой-либо другой области затруднительно.

Как показал анализ большинства систем для работы с графами [1–8], вопросу подготовки графовых иллюстраций не уделялось должного внимания. Рассмотренные программные продукты тем или иным способом позволяют создавать изображения графов, но либо ограничивают возможности пользователя по подготовке этих изображений, либо не отличаются простотой их подготовки. Но именно этот вопрос достаточно важен, поскольку ни одна публикация по теории графов не может обойтись без иллюстративного материала.

С учетом вышесказанного было принято решение разработать специальную систему для подготовки графовых иллюстраций, которая бы отличалась простотой в использовании и позволяла создавать достаточно качественные изображения.

2. ИНТЕРФЕЙС СИСТЕМЫ VEGRAS

Интерфейс системы придерживается основных общепринятых стандартов для Windows-приложений. Система имеет главное окно, внутри которого расположены окна документов (изображений графов), которые пользователь может открывать и закрывать по мере надобности.

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 00-07-90296) и Министерства образования РФ.

Главное окно системы содержит меню и панели инструментов, обеспечивающие быстрый доступ к часто используемым операциям. Место расположения панелей инструментов можно изменять, при этом каждая панель может либо прикрепляться к границе окна, либо находиться в произвольном месте. В нижней части окна расположена строка состояний, в которой отображается служебная информация.

Каждое окно документа располагается внутри главного окна и имеет полосы прокрутки, с помощью которых можно передвигаться по изображению. При нажатии правой кнопки мыши в этом окне появится всплывающее меню, из которого можно выбрать нужные операции. На рис.1 приведен пример работы системы VEGRAS.

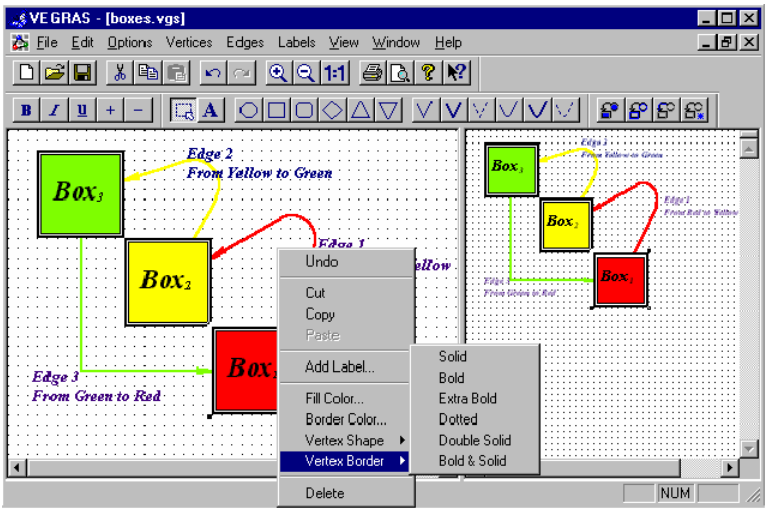


Рис. 1. Система VEGRAS в работе

Опишем процесс создания графов с помощью данной системы, при этом объектами графа будем называть вершины, дуги и метки графа. При запуске системы создается новое пустое окно графа, в котором можно начать рисовать новый граф. Далее, изменяя с помощью панели инструментов или меню форму, цвет и другие параметры объекта, к графу можно добавить новые вершины, дуги и метки.

3. ПРЕДСТАВЛЕНИЕ ГРАФОВ В СИСТЕМЕ

Вершины графа изображаются различными геометрическими фигурами, такими как эллипс, прямоугольник, скругленный прямоугольник, ромб, треугольник или перевернутый треугольник. Каждая вершина имеет каемку (простую, жирную, очень жирную, пунктирную или двойную). Цвет каемки выбирается отдельно.

Дуги графа изображаются либо ломаными линиями, задаваемыми точками сгиба, либо гладкими кривыми, для задания которых используется некоторый набор точек на плоскости (будем называть эти точки точками сгиба). Дуги бывают различной толщины и цвета. Для каждой дуги можно выбрать форму стрелки на конце или указать, что стрелка отсутствует, если граф не ориентированный. Число точек сгиба практически не ограничено.

Все вершины и дуги графа имеют произвольное количество меток, место расположения которых пользователь может изменять. Цвет и шрифт можно задавать отдельно для каждой метки. Метки могут иметь верхние и нижние индексы, что весьма полезно при создании иллюстраций к научным публикациям.

4. ВСПОМОГАТЕЛЬНЫЕ СРЕДСТВА

В процессе редактирования графа можно изменять размеры, форму, цвет вершин и дуг, перемещать и удалять вершины, дуги, метки или точки сгиба дуг, копировать выделенные объекты в буфер (операции Cut/Copy/Paste), добавлять и удалять точки сгиба дуги. Можно также выделять группу объектов графа и производить какие-либо действия сразу над всей группой (изменять цвета, форму, добавлять метки). При перемещении вершин, дуг или точек сгиба дуги соответствующие им метки тоже передвигаются так, чтобы их относительное месторасположение не изменялось.

Полезным является наличие операций Undo/Redo, позволяющих отменить или повторить до 32 последних действий (причем эти действия могут быть сложными, как, например, удаление группы объектов или изменение каких-либо параметров сразу для всей группы). В системе VEGRAS реализованы операции ZoomIn/ZoomOut, позволяющие менять масштаб изображения от 1 до 1000 %.

Если нужно, окно редактирования графа делится на несколько частей, в каждой из которых можно просматривать различные части графа, причем

можно указать разный масштаб для каждой из частей, и тогда в одном окне, например, можно видеть сразу весь граф, а в другом — только нужный фрагмент.

Для удобства пользователя в системе существует возможность применять прямоугольную сетку, внешний вид и параметры которой можно изменять.

В настоящий момент не существует общепринятого формата файлов для хранения графов, поэтому в системе VEGRAS используется новый специальный формат. Однако в систему включена возможность чтения графов формата hgr (созданных с помощью системы Higres). При реализации данной возможности использовалась библиотека HGL, поставляемая вместе с указанной системой.

Система позволяет создавать качественные изображения графов и записывать их в файлы форматов Windows BMP и PCX. Последний формат может служить для создания иллюстративного материала для книг и статей, подготовленных в системе TEX.

Система VEGRAS имеет встроенную help-поддержку, включающую описание всех пунктов меню, панелей инструментов и блоков диалога. В справочную систему включены статьи, посвященные описанию общих положений и освещающие все основные возможности системы.

Ниже приводятся несколько примеров изображений графов, полученных с помощью системы VEGRAS.

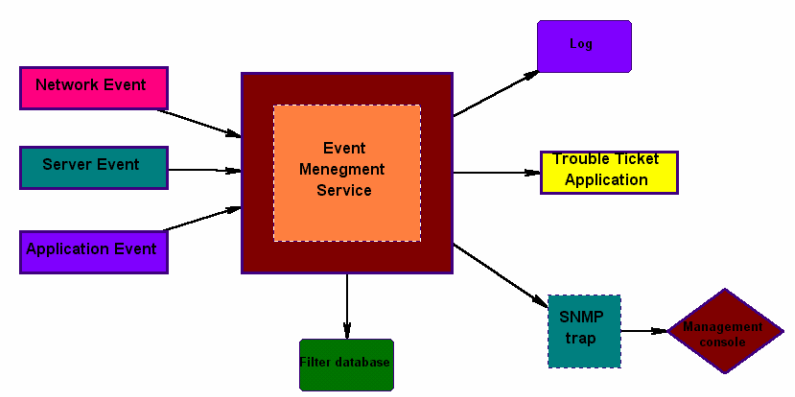


Рис. 2. Граф, описывающий работу некоторой программы

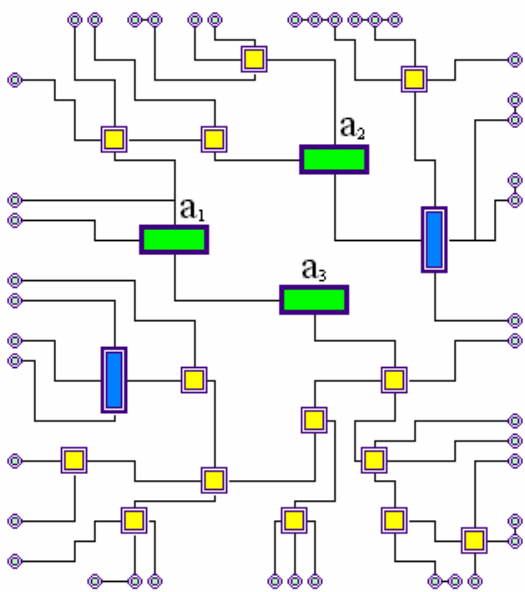


Рис. 3. Ортогональный атрибутированный граф

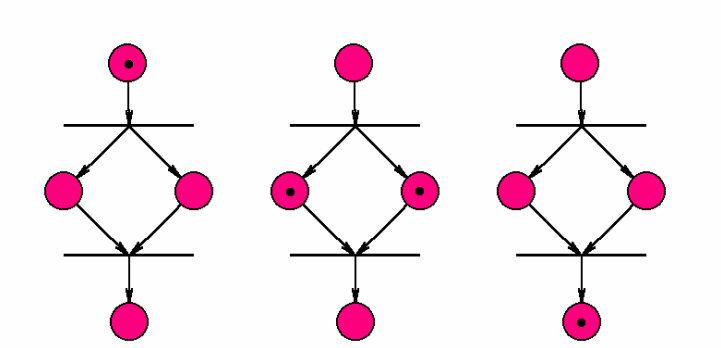


Рис. 4. Работа сети Петри

5. РЕАЛИЗАЦИЯ

Система VEGRAS написана на языке C++ с использованием компилятора Microsoft Visual C++ Version 6.0 и библиотеки MFC (Microsoft Foundation Class Library) и представляет собой универсальный инструмент для подготовки иллюстраций атрибутированных графовых моделей. Программа работает под операционными системами Windows 9x/NT.

Система VEGRAS использует механизм ActiveX для обеспечения возможности вставки изображений графов в документы других приложений, поддерживающих механизм ActiveX, таких как Microsoft Word, Excel и многие другие.

6. ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана система VEGRAS, являющаяся универсальным инструментом для визуализации и редактирования атрибутированных графов. Основным ее отличием является то, что она ориентирована именно на подготовку иллюстраций и позволяет создавать их более быстро и качественно.

Дальнейшее развитие системы может быть связано с

1) увеличением числа читаемых форматов графов,
2) добавлением алгоритмов размещения графов,
3) расширением интерфейсных возможностей.

Кроме того, в ближайшее время планируется разместить систему VEGRAS в сети Интернет.

Авторы будут признательны всем высказавшим свои замечания и пожелания по поводу улучшения данной системы по e-mail: m_tanya@mail.ru.

СПИСОК ЛИТЕРАТУРЫ

1. **Lisitsyn I.A., Kasyanov V.N.** HIGRES — Visualization System for Clustered Graphs and Graph Algorithms. // Proc. of Graph Drawing. — Berlin a.o.: Springer-Verlag, 1999. — P. 82—89. — (Lect. Notes Comput. Sci.; Vol. 1731).

2. **Лисицын И.А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей // Проблемы систем информатики и программирования. — Новосибирск, 1999. — С. 64—77.
3. **Система** HIGRES. — Доступна по адресу: http://lis.iis.nsk.su/higres.
4. **Frohlich M., Werner M.** Demonstration of the interactive graph visualization system daVinci // Proc. of Graph Drawing. — Berlin a.o.: Springer-Verlag, 1994. — P. 266—269. — (Lect. Notes Comput. Sci.; Vol. 894).
5. **Система** daVinci. — Доступна по адресу: http://www.informatik.uni-bremen.de/~inform/forschung/daVinci/daVinci.html.
6. **SmartDraw** Software Incorporated, 9974, Scripps Ranch Blvd, 35 San Diego, CA 92131. — Доступна по адресу http://www.smartdraw.com.
7. **Демонстрационная** система компании Tom Sawyer Software, 804 Hearst Avenue, Berkeley, CA 94710. — Доступна по адресу http://www.tomsawyer.com
8. **Himsolt M.** The Graphlet system (system demonstration) // Proc. of Graph Drawing. — Berlin a.o.: Springer-Verlag, 1994. — P. 194—205. — (Lect. Notes Comput. Sci.; Vol. 894).

Э. В. Харитонов

РЕАЛИЗАЦИЯ СОПОСТАВЛЕНИЯ С ОБРАЗЦОМ В ЯЗЫКЕ LISP НА ОСНОВЕ АНАЛОГИЧНЫХ СРЕДСТВ В ЯЗЫКАХ REFAL И HASKELL[1]

ВВЕДЕНИЕ

В языках Refal [1], ML, Haskell [2] разбор структурированных данных удачно поддержан механизмом сопоставления с образцом. Lisp [3, 4] — язык, хорошо приспособленный к обработке структурных данных, но не содержащий в наборе стандартных возможностей средств сопоставления с образцом. Тем не менее базовые средства языка Lisp вполне достаточны для реализации подобных возможностей. Более того, было бы естественно расширить Lisp несколькими модификациями новых средств одновременно и оценить разные варианты расширения языка в общей программной среде.

В данной работе рассматриваются некоторые способы введения в язык Lisp средств сопоставления с образцом и оцениваются удобства применения этих средств для обработки структурированных данных.

Lisp, расширенный средствами сопоставления с образцом, обозначим Lisp*.

1. ФОРМА FUNC — ОПРЕДЕЛЕНИЕ ФУНКЦИИ ПРИ ПОМОЩИ СОПОСТАВЛЕНИЯ С ОБРАЗЦАМИ

При традиционном программировании на языке Lisp, с разбором вариантов и рекурсией, функции, обрабатывающие структурные выражения (S-выражения), выделяют элементы структуры параметров и в зависимости от их значений возвращают разные результаты. Например, функция вычисления длины списка имеет следующий вид:

```
(defun length1 (L)
 (cond
  ((null L) 0)
  (t (+ 1 (length1 (cdr L))))
))
```

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

Аналогичное определение на языке Haskell выглядит так:

```
length2 []       -> 0
length2 (x:xs)   -> 1 + length2 xs
```

Здесь первая строка — длина пустого списка, вторая — длина списка с головой `x` и хвостом `xs`.

Для языка Lisp можно было бы определить вместо `defun` новую форму — `func`, позволяющую оформить определение функции `length` в стиле приведенного Haskell-фрагмента. Определение на языке Lisp с использованием сопоставления с образцом может выглядеть, например, так:

```
(func length3
 (nil)          0
 ((x . xs))     (+ 1 (length3 xs))
)
```

Это означает, что если список параметров функции `length3` совпадает с `(nil)`, то результат равен `0` (длина пустого списка). Если же параметр `length3` возможно представить в виде `(cons x xs)`, то значение вычисляется рекурсивным вызовом `(+ 1 (length3 xs))`.

Пусть форма `func` определяет новую функцию и имеет вид:

```
(func ИМЯ_ФУНКЦИИ
 ОБРАЗЕЦ1       ВЫРАЖЕНИЕ1
 ...
 ОБРАЗЕЦN       ВЫРАЖЕНИЕN
)
```

Здесь `ИМЯ_ФУНКЦИИ` — символьный атом, пополняющий глобальную среду имен, `ОБРАЗЕЦi` — S-выражение, задающее вариант структуры списка параметров, `ВЫРАЖЕНИЕi` — выражение, вычисляющее результат функции при успешном сопоставлении списка параметров с `i`-м образцом. При этом если в образце содержатся переменные, то при успешном сопоставлении эти переменные получают значения соответствующих фрагментов списка параметров. Например: при вызове `(length3 '(1 2 3))` список параметров, очевидно, `((1 2 3))`, и сопоставление с образцом `(nil)` неуспешно, поскольку `(1 2 3)` и `nil` — разные значения, а сопоставление с образцом `((x . xs))` успешно при x = 1, `xs` = `(2 3)`, поэтому будет возвращен результат `(+ 1 (length3 '(2 3)))`, из которого в конце концов получится число `3`.

Заметим, что переменная `x` не используется в выражении `(+ 1 (length3 xs))`. Если переменная, входящая в образец, не используется

в выражении, ее предпочтительнее обозначать подчерком:

```
(func length4
 (nil)          0
 ((_ . xs))     (+ 1 (length4 xs))
)
```

Рассмотрим несколько примеров применения введенной формы `func`. Тестовые вызовы определяемых ниже функций будут обозначаться `ВЫРАЖЕНИЕ => ЗНАЧЕНИЕ`, например, "`(length nil) => 0`".

Пример. Функция, возвращающая подсписок из первых `N` элементов данного списка `L`.

Lisp:

```
(defun take1 (L N)
 (cond
  ((null L) nil)
  ((= N 0) nil)
  (t (cons (car L) (take1 (cdr L) (- N 1))))
))
```

Haskell:

```
take2 []     _   -> []
take2 _      0   -> []
take2 (x:xs) N   -> x : take2 xs (N-1)
```

Lisp*:

```
(func take3
 (nil      _)    nil
 (_        0)    nil
 ((x . xs) N)    (cons x (take3 xs (- N 1)))
)

(take3 '(A B C (1 2) 3 4) 4)   =>   (A B C (1 2))
```

Пример. Функционал, выполняющий данную функцию `F` над каждым элементом списка `L` и выдающий список из результатов

Lisp:

```
(defun map1 (F L)
 (cond
  ((null L) nil)
  (t (cons (funcall F (car L)) (map1 F (cdr L))))
))
```

Haskell:

```
map2 _ []       -> []
map2 f (x:xs)   -> (f x) : map2 f xs
```

Lisp*:

```
(func map3
 (_ nil)        nil
 (F (x . xs))   (cons (funcall F x) (map3 F xs))
)

(map3 'sqrt '(1 4 9 16))  =>  (1 2 3 4)
```

2. ПОИСК ОБЩИХ ПОДВЫРАЖЕНИЙ В ОБРАЗЦАХ

В языке Refal, в отличие от Haskell'a, в образце разрешается несколько вхождений одной переменной, что означает требование совпадения данных, соответствующих разным вхождениям переменной при сопоставлении с таким образцом.

Пример. Поиск в ассоциативном списке `((E1 . V1) ... (En . Vn))`.

Lisp:

```
(defun sass1 (E L)
 (cond
  ((null L) nil)
  ((eq E (caar L)) (cdr L))
  (t (sass1 E (cdr L)))
))
```

Lisp*:

```
(func sass2
 (_ nil)                nil
 (E ((E . V) . _))      V
 (E (_ . L))            (sass2 E L)
)

(sass2 y '((x . 1) (y . -3) (z . A)))  =>  (y . -3)
```

Пример. Свойство из списка `(E1 V1 ... EN VN)`.

Lisp:

```
(defun get1 (E L)
 (cond
  ((null L) nil)
  ((eq E (car L)) (cadr L))
  (t (get1 E (cddr L)))
))
```

Lisp*:

```
(func get2
 (_ nil)          nil
 (E (E V . _))    V
 (E (_ _ . L))    (get2 E L)
)

(get2 'name '(apval 5 name A))  =>  A
```

3. ФУНКЦИИ С ПЕРЕМЕННЫМ КОЛИЧЕСТВОМ ПАРАМЕТРОВ

Количество параметров функции может быть переменным, поскольку в образцах фигурирует весь список параметров.

Пример. Вычисление суммы квадратов от переменного количества аргументов.

Lisp:

```
(defun sumsqr1 (&rest Args)
 (cond
  ((null Args) 0)
  (t (+ (expt (car Args) 2) (apply 'sumsqr1 (cdr Args))))
))
```

Lisp*:

```
(func sumsqr2
 nil      0
 (a . b)  (+ (* a a) (apply 'sumsqr2 b))
)
```

Здесь строка "`nil 0`" означает, что если функцию `sumsqr2` вызвали без параметров, то она возвращает `0`. Если же первый параметр связать

с переменной a, а список всех остальных параметров — с переменной b, то результат — сумма от (* a a) и вызова sumsqr2 со списком параметров из b.

```
(sumsqr2 1 -2 3)  =>  14
```

Пример. Объединение переменного количества списков.

Lisp:

```
(defun append1 (&rest l)
 (cond
  ((null l) nil)
  ((null (cdr l)) (car l))
  ((null (car l)) (apply 'append1 (cdr l)))
  (t (cons (caar l) (apply 'append1 (cons (cdar l) (cdr l)))))
))
```

Lisp*:

```
(func append2
 nil                    nil
 (a)                    a
 (nil . a)              (apply 'append2 a)
 ((a . b) . c)          (cons a (apply 'append2 (cons b c)))
)

(append2 '(1 2) '(a b) '((x))  =>  (1 2 a b (x))
```

Из приведенных примеров видно, что форма func синтаксически и семантически сходна с комбинацией форм defun и cond. Применение формы func наиболее удачно в примерах, описывающих функции sassoc2, get2 и append2. Это связано с большей сложностью структуры образцов в указанных примерах, и, следовательно, с более заметным переносом на образцы действий по разбору структурных выражений. Таким образом, форма func, реализующая сопоставление с образцами, наиболее адекватно применима при создании функций обработки сложных структур.

4. ФУНКЦИОНАЛЬНЫЕ ВЫРАЖЕНИЯ С ОБРАЗЦАМИ

Форма func пополняет глобальную среду новым именем функции. Для локального определения функций нужен аналог лямбда-выражения. Форма "→" определяет новую функцию без имени и возвращает ее в качестве значения:

```
(mapcar
 (-> (x) (+ 1 x))
'(1 2 3 4)
)
; Результат: (2 3 4 5)

(let*
 ((length5
   (->
    (nil)       nil
    ((x . xs))  (+ 1 (funcall length5 xs))
 )))
 ; Функция ‘‘длина списка’’ связана с временной
 ; переменной length5
 (mapcar length5 '(nil (q) (3 4 5)))
)
; Результат: (0 1 3), действие локального значения length5
; прекратилось
```

5. МОДИФИКАЦИИ СИНТАКСИСА

Формы `func` и “→” должны быть более подвержены ошибкам при программировании, чем, например, форма `cond`, потому что ветви “образец выражение” не обособлены дополнительными скобками. Отчасти это сделано для того, чтобы не перегружать функциональные определения дополнительными скобками. Поскольку в языке Lisp есть возможность вводить синтаксические расширения, можно было бы определить более наглядный вид конструкций сопоставления с образцами, например:

```
(func length6
 {nil         -> nil}
 {(x . xs)    -> (+ 1 (funcall length6 xs))}
)

(func append3
 {              -> nil}
 {a             -> a}
 {nil . a       -> (apply 'append3 a)}
 {(a . b) . c   -> (cons a (apply 'append3 (cons b c)))}
)
```

6. НЕКОТОРОЕ ОБОБЩЕНИЕ ПОНЯТИЯ СТРУКТУРЫ И ОБРАЗЦА

Фактически образец содержит утверждение, при помощи каких средств сконструированы сопоставляемые данные, а для сопоставления с образцом требуется информация о том, как "разобрать" данные.

Функции для конструирования данных назовем конструкторами, а функции для доступа к частям, из которых данные сконструированы, — реструкторами. Реструкторы являются обратными функциями по отношению к конструкторам.

Другой подход мог бы состоять в том, что реструктор (пусть он в этом случае называется полным реструктором) — функция, возвращающая весь список параметров конструктора.

Если F — конструктор, и $Y = F(X_1, ..., X_n)$ — сконструированные данные, то обозначим реструкторы как $F'_1(Y), ..., F'_n(Y)$. Полный реструктор будет иметь вид $(F'_1(Y)...F'_n(Y))$. В случае неоднозначности конструктора неоднозначными могут быть и $F'_i(Y)$. Тогда может потребоваться вычисление всех возможных вариантов значений. Например, если в качестве конструктора используется операция умножения целых чисел, для разбора полученной "структуры" понадобится нахождение делителей числа. Для однозначности будем предполагать, что образец `(* x y)` обозначает: `x` — наименьший простой делитель в сопоставляемом этому образцу числе.

```
; Список простых делителей числа
(func primes-list
 { 1             -> nil }
 { (* x y)       -> (cons x (primes-list y)) }
)


(func length6
 { nil          -> 0 }
 { (cons _ xs)  -> (+ 1 (length6 xs)) }
)
```

Итак, чтобы функцию можно было применять в качестве конструктора, необходимо задать для нее полный реструктор.

Реструктор для функции `CONS`:

```
(lambda (x) (list (car x) (cdr x)))
```

Реструктор для функции `LIST`:

```
(lambda (x) x)
```

Реструктор для функции `VECTOR`:

```
(lambda (v) (coerce v 'list))
```

Реструктор для функции "`*`" (вариант):

```
; (lsd x) --- наименьший простой делитель от x
(lambda (x) (let ((y (lsd x))) (list y (/ x y))))
```

Поскольку имя функции-конструктора — символьный атом, то реструктор может быть задан, например, как свойство `RESTR` для символьного атома — имени конструктора:

```
(setf (get 'cons 'restr)
 (lambda (x) (list (car x) (cdr x)))
)
(setf (get 'list 'restr)
 (lambda (x) x)
)
(setf (get 'vector 'restr)
 (lambda (v) (coerce v 'list))
)
(setf (get '* 'restr)
 (lambda (x) (let ((y (lsd x))) (list y (/ x y))))
)
```

Рассмотренный материал позволяет заключить, что расширение языка Lisp средствами сопоставления с образцом может способствовать повышению компактности программ, а также некоторому упрощению программирования.

СПИСОК ЛИТЕРАТУРЫ

1. **Turchin V.F.** Refal-5, Programming Guide and Reference Manual — Holyoke: New England Publishing Co., 1989.
2. **The Haskell 98** Report. — 1999. — http://haskell.org
3. **McCarthy J.** Lisp 1.5 programming manual. — Cambridge: The MIT Press., 1963.
4. **Henderson P.** Functional Programming Application and Implementation. — London: Prentice-Hall International Inc., 1980.

Ю.В. Малинина

ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ ПРИ РАЗРАБОТКЕ WIS[1]

1. ВВЕДЕНИЕ

В настоящее время WEB является широко распространенным информационным хранилищем. Возрастающая популярность Интернета и быстрое развитие программного обеспечения от любительских Web-сайтов до корпоративных систем заставляют гипермедиа-технику совершенствоваться быстрее, чем когда-либо прежде. Это новое поколение приложений, включающее гипермедиа-функции в сложные системы транзакций, развивается непрерывно; новые версии нуждаются в строгих спецификациях, так как могут повлиять на основу жизнедеятельности компании (как, например, в приложениях е-торговли). Все это приводит к тому, что комплекс гипермедиа-приложений должен разрабатываться с использованием последних достижений технологии программирования. Тем не менее гипермедиа-приложения обладают специфическими характеристиками, отличающими их от стандартного программного обеспечения. Во-первых, поскольку построение их навигационной архитектуры является для них критическим, разработка навигации должна быть выделена как самостоятельная задача и явно специфицирована. Кроме того, гипермедиа-приложения являются диалоговыми по определению и аспекты интерфейса пользователя должны тоже быть описаны при помощи проектных примитивов. Наконец, поскольку рынок навязывает циклы короткой разработки, время становится важным аспектом создания эффективных приложений и, следовательно, многократное использование проектов предпочтительнее, так как позволяет избежать разработки с нуля. Проектный опыт — существенный аспект сокращения времени разработки, и поэтому необходимы средства для представления проектного опыта в виде некоторых формализмов, которые облегчат его повторное использование и распространение среди разработчиков.

Существующая потребность в новых методах, решающих упомянутые выше аспекты в проектировании, не отменяет того, что эти методы должны

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

соответствовать существующим стандартам проектирования приложений. К несчастью, состояние искусства проектирования программного обеспечения и некоторых методов, как, например, Unified Modeling Language (UML), терпят неудачу при применении к проектированию гипермедиа-приложений из-за их специфических свойств.

В предлагаемой статье рассматривается использование проектных шаблонов для решения перечисленных проблем. Фактически, целью является рассмотрение процесса применения шаблонов для улучшения проектирования архитектуры Web-информационных систем (WIS) .

2. ОСОБЕННОСТИ ПРОЕКТИРОВАНИЯ WIS

Проектирование WIS — информационных систем, которые созданы с использованием Web-технологии, — включает множество проблем: определение общей архитектуры приложения, например тип интерфейса (связи, форматы и т.п.) с базами данных или другими системами; организацию информационных узлов таким образом, чтобы пользователь мог легко управлять ими; представление навигации и функционала и т.д.

Тем не менее требуется использовать опробованную технологию программирования для проектирования. Существует много методов, которые могли бы применяться для моделирования некоторых аспектов WIS, как, например, OOHDM [7, 8], RMM [4] и т.п. Они обеспечивают поддержку ссылок и платформенно-независимое проектирование, которые помогают описывать проблемную область так, чтобы гипермедиа-компоненты (узлы, связи и индексы) могли быть затем реализованы при помощи инструментальных средств, запускаемых во время установки. В этих методах есть ясное разделение на методы, представляющие интерес для архитектурного (концептуального) проектирования, методы для проектирования навигации и методы для проектирования интерфейса пользователя. Выделение проектирования навигации в отдельную проблему позволяет разработчикам сконцентрироваться на отличительной черте WIS-гипермедиа-систем.

Однако существуют программные системы, нацеленные на поддержку проектирования WIS, которые игнорируют выделение проектирования навигации как отдельной задачи, обращаясь с ней, как с некоторой функцией интерфейса пользователя. Например, в VisualWave описывается разработка Smalltalk-приложения, где интерфейс содержит элементы управления, допускающие открытие нового окна. Но если воспользоваться этим элементом для операции навигации, то мощность навигации будет потеряна не из-

за характеристики окружения, а главным образом потому, что разработчики не принимают во внимание навигационное измерение приложений. Естественно, они просто разрабатывают прикладную модель с объектами, а затем формируют интерфейс пользователя. Тем не менее существует много задач проектирования, которые можно решить, если только рассматривать навигацию как отдельную задачу проектирования.

Методология Relationship Management Methodology (RMM) первоначально была предложена Isakowitz в [4]. Она подходит для гипермедиа-приложений, имеющих неизменную структуру и динамические данные, требующие частого обновления, например каталоги продуктов. RMM плохо подходит для гипермедиа-приложений, которые являются хорошо структурированными, но с неизменными в течение продолжительных периодов данными, или гипермедиа-приложений с динамической структурой и динамическими данные. Основу этой проектной методологии можно условно разделить на три стадии.

Первая стадия — это представление информационной области гипермедиа-приложения посредством ER-диаграммы. Выделенные сущности и их отношения формируют основу гипермедиа-приложения, реализуемую как узлы и связи.

Вторая стадия — это определение информации для представления в узлах и уровней доступа к ней пользователей, что подразумевает разделение сущностей на части для последующей группировки. Связи между частями называются структурными. Структурные связи отличаются от ассоциативных отношений в структурных связях, существующих между одной и той же сущностью, тогда как ассоциативное отношение ссылается на другие сущности. Это различие важно для навигации.

Третья стадия — это разработка навигации. Чтобы иметь возможность обновлять информацию, эта стадия описывается в терминах общих условий, а не жестко определяет сценарий навигации, что достигается ссылкой на свойства сущностей и их связей.

Общим во всех этих методах является то, что проектирование гипермедиа-приложений имеет такую же сложность, как и проектирование программного приложения общего назначении. OOHDM [8], например, разделяет проектное пространство на четыре части: концептуальное проектирование, разработку навигации, абстрактную разработку интерфейса и реализацию. В каждой части должна быть построена проектная модель, что увеличивает время проектирования. Вообще OOHDM обеспечивает известные объектно-ориентированные конструкции (например, классы, объекты, свя-

зи, сценарии использования, и т.п.) для определения моделей, что позволяет во время концептуального проектирования строить (объектно-ориентированную) модель проблемной области на основе этих конструкций и, таким образом, в принципе допускает модификацию разработанной концептуальной модели для другого потребителя, чем сокращает время проектирования. Однако это требует от разработчика дополнительных усилий.

Описанный в литературе опыт использования OOHDM для проектирования дает возможность сделать некоторые выводы: метод позволяет лучше понимать проектные решения и получать подтверждение от пользователя, облегчая использование и развитие приложения, и одновременно улучшает связь между участниками проекта (менеджерами и конечными пользователями, проектировщиками и заказчиком). Тем не менее, если мы хотим улучшить разработку в смысле многократного использования, нам нужны абстракции более высокого уровня, чтобы подтверждать сложные проектные решения и записывать успешные решения для последующего использования.

3. ПРИМЕНЕНИЕ ШАБЛОНОВ В ПРОЕКТИРОВАНИИ WIS

Проектные шаблоны являются мощным средством для фиксирования и передачи проектного опыта. Они введены Alexander C. [1] в области градостроительства и широко используются в объектно-ориентированном проектировании программ [2]. Проектный шаблон описывает проблему и ее абстрактное решение таким образом, чтобы это решение можно было использовать для других экземпляров этой же самой проблемы, а также описывает разумное обоснование и выбор оптимального решения.

Гипермедиа — богатая область для обнаружения и применения шаблонов. Гипермедиа-шаблоны можно условно разделить на архитектурные, навигационные и шаблоны интерфейса. Архитектурные шаблоны решают общие проблемы (связь навигации с базами данных или программным обеспечением). Навигация и шаблоны интерфейса [6] так же, как образцы в [1], показывают возможности формирования информационного пространства. Они хранят коллективный опыт гипертекст-общества, использующего шаблоны как по одному [2], так и соединяя их в группы и в виде каталогов.

Навигационные шаблоны показывают, как расширить простую интерпретацию узлов и связей и перейти от обычных интуитивных решений к проблеме проектирования. Например, пока неопытные разработчики рассматривают ссылки как прочную семантическую связь, “Set-based

Navigation"-шаблон показывает, когда связь полезна для соединения узлов (например, когда они принадлежат одному множеству благодаря некоторым свойствам стоящей перед пользователем задачи). "Node in Context"-шаблон показывает, как разрешить ситуацию, когда один и тот же узел появляется в многочисленных множествах, допуская различные представления в зависимости от множества, в пределах которого он доступен.

Шаблоны интерфейса имеют дело с построением более "причудливых" интерфейсов. Например, "Information on Demand" описывает, как организовать объекты интерфейса, когда проектировщик хочет избежать слишком большой познавательной нагрузки, если узел имеет слишком много информационных пунктов для восприятия. Обнаружение новых шаблонов является полезной деятельностью, так как помогает увеличить свой опыт, повысить уровень рассуждений и помочь в оценке своего (и чужого) проекта. Языки шаблонов для пользовательских интерфейсов и проектного взаимодействия (множество шаблонов, которое покрывают целиком проектное пространство) только сейчас разрабатываются [9].

Шаблоны не привязаны к конкретному методу проектирования: можно использовать свой каталог шаблонов при разработке гипермедиа-приложения. Кроме того, шаблоны являются очень мощным средством, даже если не используется никакой метод проектирования, чтобы специфицировать гипермедиа-приложение.

Проектные шаблоны дополняют процесс проектирования, так как показывают решения, превосходящие полученные примитивными методами. Например, наивный объектно-ориентированный разработчик стремится строго следовать основным понятиям объектной парадигмы, изолируя структуру и алгоритмы в одном объекте. Тем не менее сложные проблемы требуют более гибкого решения, подобно шаблону Bridge, Strategy or State [2]. В этих случаях представление, алгоритмы или состояние объекта переносятся в объектное поле, определенное за пределами объекта в контексте отдельной иерархии. Мы можем определить шаблоны на уровнях другой абстракции; они действуют как микроархитектура в системе, улучшая системную модульность и снижая дополнения.

Когда мы создаем шаблон, то описываем, какие типы проблем порождают микроархитектуру, в которой шаблон должен быть использован, и что такое выбор оптимального решения при его использовании. Как объяснено в [2], шаблоны не являются ни исходными, ни новыми решениями программных проблем, но они хорошо показали себя в успешных проектах. Фактически, используя стандартные проектные методы (подобно OOHDM

или RMM) для работы со сложным проектом, хорошие разработчики применяют свой опыт, чтобы выделять повторяющиеся проблемы и использовать уже найденные решения. Это и есть момент появления шаблонов. Как таковые шаблоны не изобретаются отдельно от проблемы, а обнаруживаются в процессе проектирования. Часто опытным разработчикам эти шаблоны могут казаться очевидными, поскольку отражают обычно применяемые проектные решения. Но для менее опытных разработчиков эти шаблоны составят ценный источник экспертных знаний в области проектирования для создания нового проекта.

Проектные шаблоны имеют неоценимое значение с тех пор, как проектные знания не являются решением, ориентированным на настоящее, а являются проблемно-ориентированным решением. Отметим уникальное свойство шаблонов: они описывают проблемы, которые часто встречаются в процессе разработки, вместе с усилиями, связанными с решением этих проблем. Шаблоны имеют превосходные обучающие принципы, поскольку содержат контекст и описание выбора оптимального решения.

Проектные шаблоны полезны для записи повторяющихся проектных задач в программных системах. Изучение успешных приложений и абстракция проблем и решений помогают обнаруживать новые шаблоны, что увеличивает наше понимание процесса проектирования и, следовательно, мы можем находить общее у аналогичных приложений, устанавливая контекст, в котором эти шаблоны появляются.

Проектные методы должны развиваться в сторону включения новых шаблонов в наборы своих примитивов или создания абстракций.

ЗАКЛЮЧЕНИЕ

Здесь была кратко описана проблема проектирования гипермедиа с точки зрения создания программного обеспечения. Гипермедиа-проектирование является важным вопросом в новых областях, таких как электронная торговля, цифровые библиотеки и т.п. Проектные методы сами по себе не являются панацеей. Взаимодействие гипермедиа и каталогов проектных шаблонов растет. Эти каталоги могут использоваться проектными методами для улучшения процесса разработки, обеспечивая удобный путь многократного использования проектов и/или отдельных компонент. Тем не менее взаимодействие между шаблонами и методами далеко от тривиального. Требуют исследования как процесс внедрения шаблонов в качестве примитивов новых методов, так и процесс определения систематиче-

ской стратегии понимания шаблонов и формализации однозначных нотаций для их записи.

СПИСОК ЛИТЕРАТУРЫ

1. **Alexander C et all.** A Pattern Language. — N.-Y.: Oxford University Press, 1977.
2. **Gamma E. et all.** Design Patterns: Elements of Reusable Object-Oriented Software. — Addison-Wesley, 1995.
3. **Garzotto F. et all.** HDM — a model-based approach to hypertext application design // ACM Transactions on Information Systems (TOIS). — 1993. — Vol. 11(1). — P. 1—26.
4. **Isakowitz T. et all.** RMM: a methodology for structuring hypermedia design // Communications of the ACM (CACM). — 1995. — Vol.38, N 8. — P. 34—44.
5. ACM Hypertext '99 / Proc. Second Intern. Workshop on Hypermedia Development. Hypermedia Patterns / Ed. by D**.** Lowe et all. — Darmstadt, 1999.
6. **Rossi G. et all.** Design reuse in hypermedia applications development // Proc. of ACM Hypertext97. — Southampton, UK. — 1997. — P. 57—66.
7. **Schwabe D., Rossi G., Barbosa S.** Systematic hypermedia design with OOHDM // Proc. of the ACM Intern.l Conf. on Hypertext (Hypertext'96) . — Washington, 1996.
8. **Schwabe D., Rossi D.** An object oriented approach to Web-based application design //TAPOS (Theory and Practice of Object Systems) . — Wiley and Sons, 1998.
9. **Tidwell J.** Interaction design patterns //Patterns Languages of Programs (PLoP '98), Allerton Conference Center. — Urbana Champaign, 1998

**Ж.Л.-Д. Дылыков, Н.Б. Занаева, Е.В. Марьясов,
Ф.А. Мурзин, Д.Ф. Семич**

ЛОГИЧЕСКАЯ СТРУКТУРА ПРОЦЕССА ГЕНЕРАЦИИ И ОТГАДЫВАНИЯ ЗАГАДОК[1]

ВВЕДЕНИЕ

Теория решения изобретательских задач (ТРИЗ) была предложена в 1956 г. Г.С. Альтшуллером [1,2].

В настоящее время различные направления ТРИЗ завоевывают все более широкое признание и находят все новые и новые применения [3].

Очевидная алгоритмическая направленность ТРИЗ выводит на первый план задачу формализации и последующей автоматизации ее методов.

В рамках ТРИЗ рассматриваются самые разные задачи различной сложности. Цель данной работы — исследовать методы принятия решений в концепции ТРИЗ на примере загадок. Авторы выражают глубокую благодарность А.Г. Марчуку за поддержку проводимых исследований.

1. ПРОЦЕСС ГЕНЕРАЦИИ И ОТГАДЫВАНИЯ ЗАГАДОК

Обычно загадки формулируются на естественном языке. Однако лексика, используемая в их текстах, весьма ограниченна. Поэтому на основе анализа текста загадки довольно легко может быть сгенерирована логическая формула, как правило, на узком исчислении предикатов (языке первой ступени), которая представляет собой формальную запись данной загадки.

Обозначим эту формулу φ. Процесс отгадывания состоит из двух этапов.

Первый этап. По формуле φ строится некоторая новая формула $\varphi^*(x)$ с одной свободной переменной v_0 .

Второй этап. Предположим, что в нашем распоряжении имеется конечная модель $\mathbf{M} = \langle M, \sigma \rangle$. Процесс отгадывания состоит в нахождении

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

элемента модели $m \in M$ такого, что $\mathbf{M}|=\varphi^*(m)$, т.е. формула φ^* должна быть истинна в модели **M** на данном элементе.

Дополнительно в нашем распоряжении может быть некоторый информационный фонд, содержащий набор высказываний типа "*если белый, то не черный*" и т. д.

Каждое такое высказывание представляет собой формулу УИП, например

$$\forall x\big(White(x) \to \neg Black(x)\big).$$

Совокупность всех рассматриваемых формул образует теорию, т.е. информационный фонд есть некоторая теория *T*.

Очевидно, что имеется в виду ситуация $\mathbf{M} \models T$. С другой стороны, работая с Т, применяя средства логического вывода, мы облегчаем поиск требуемого элемента $m \in M$. Отметим также, что *m* может быть не единственным.

Процесс генерации загадок обратный к процессу отгадывания. Отправляясь от $m \in M$, формулируем логические условия на него, которые потом могут быть преобразованы в предложения на естественном языке, т.е. в текст загадки.

Часто могут быть сформулированы несколько различных загадок об одном и том же объекте.

2. ОПРЕДЕЛЕНИЯ И ОБОЗНАЧЕНИЯ

Пусть $\mathbf{M} = \langle M, \sigma \rangle$ — конечная модель сигнатуры σ,

$$\sigma = \langle \{P_i^{ni}\}_{1 \le i \le n}, \{c_k\}_{1 \le k \le m} \rangle.$$

Не уменьшая общности, можно считать, что $M = \{1,...,m\}$ — отрезок натурального ряда, где n_i — местность предиката P_i (для краткости опускается). Константа c_k интерпретируется элементом $k \in M$.

Можно также рассматривать случаи, когда область истинности логики состоит более чем из двух элементов, например из трех $\{0,1,\perp\}$, где знак $\perp$ обозначает *"неопределенность"* или является вещественным числом из отрезка [0,1], как в логике Заде.

Задание истинности значений в модели в этом случае означает определение функций вида

$$P_i : M^{ni} \to X,$$

где X — область истинности.

Определение. Если $k \in M$, то $d_k = \{P_i(x) : M \models P_i(c_k)\}$ называется диаграммой элемента k; $P_x = \{\varphi(x) : M \models \varphi(x)\}$ — типом элемента k. Здесь φ — произвольная формула сигнатуры σ узкого исчисления предикатов.

3. ОСНОВНЫЕ ТИПЫ ЗАГАДОК

3.1. Загадки, базирующиеся на описании свойств предметов

Загадки данного вида наиболее простые из известных. Приведем пример:

Легкий, но не пух.
Белый, но не снег.
Круглый, но не мяч.
Стучит, но не сердце.
...

(*Шарик для пинг-понга*)

Формальная запись любого из высказываний такого рода имеет вид

$$P_i(x) \,\&\, x \neq c_k .$$

Загадка может включать в себя более сложные высказывания:

Как снег, но не белый.
Как жемчуг, но дешевый.
Как бисер, но крупный.

В этом случае берется элемент модели ("*снег*", "*жемчуг*", "*бисер*") и рассматривается дизъюнкция всех формул из диаграммы этого элемента, за исключением одной, а именно отбрасывается $P_i(x)$, про которую в тексте сказано, что имеет место $\neg P_i(x)$. В данном случае: "*не белый*", "*не дорогой*", "*не мелкий*". В информационном фонде должна храниться эквивалентность: "*дешевый*" $\leftrightarrow$ "*не дорогой*".

В итоге получаем формулу вида

$$\vee\left(\alpha_k - \{P_i(x)\}\right) = \vee\left\{P_j(x) : P_j(x) \in \alpha_k \,\&\, j \neq i\right\}.$$

Формула φ представляет собой конъюнкцию конечного множества формул, описанных выше видов. Формула φ^* в этом случае имеет тот же вид, т.е. выполнено $\varphi^* = \varphi$.

3.2. Загадки, основанные на описании частей предметов

Сначала рассмотрим простой пример:

Есть у ежика.
Есть у шприца.
Есть у швейной машинки.
(Иголка)

В формальном виде можно записать

$$\varepsilon(x, c_{m1}) \,\&\, \varepsilon(x, c_{m2}) \,\&\, \ldots \,\&\, \varepsilon(x, c_{m3}).$$

Отличие от случая 3.1. состоит в том, что появился двумерный предикат.

Такого рода загадки не вызывают трудностей.

3.3. Загадки, в которых идет речь о количестве частей предмета

Загадки данного вида являются некоторым развитием загадок, рассмотренных в предыдущем разделе. Например:

Двое дужек.
Два стекла.
Одна оправа.

В более завуалированном виде загадка звучит так:

Два рыболовных крючка.
Два озера.
Одна буква "В".

Переходим к более подробному анализу. Введем предикат *Part*(*x ,y, n*), который содержательно обозначает, что "*x является частью y*" и "*y содержит x в n экземплярах*". Можно считать, что некоторый набор натуральных чисел (как объектов) входит в основное множество модели *M*.

Более того, можно предполагать, что туда же входят объекты типа "*мало*", "*много*", "*куча*", "*несколько*", "*неизвестно сколько*" и т. д.

При записи формул могут использоваться специальные символы, типа знака $\perp$, который обозначает неизвестно сколько.

Это, в частности, означает, что

$$M \models \forall x \forall y\, (Part(x, y, \perp) \rightarrow Part(x, y, 1)).$$

В случае, когда загадка задана в явном виде, формализовать ее в описанных выше терминах не представляет труда.

Рассмотрим теперь случай, когда загадка сформулирована в завуалированном виде.

Введем предикат α (*x, y*), который обозначает, что *x* аналогично *y,* например: "*рыболовный крючок аналогичен дужкам*".

Формальная запись загадки имеет вид

$$\varphi(x) = \underset{j}{\&}\, Part\ (c_{ij}, x, n_j).$$

Формула $\varphi^*(x)$ в данном случае отличается от $\varphi(x)$ и имеет вид

$$\varphi^*(x) = \exists \bar{u}\left(\underset{j}{\&} \alpha(u_j, c_{ij}) \,\&\, Part(u_j, x, n_j) \right).$$

Процесс отгадывания состоит в том, чтобы найти объекты $\bar{c}'$ в модели **M** аналогичные указанным в списке $\bar{c}$, т.е.

$$M \models \underset{j}{\&} \alpha(c_j', c_j);$$

а потом найти объект $c_k \in M$, который делает формулу $\varphi^*(x)$ истинной в **M**.

3.4. Загадки, содержащие описание мест

$$M \models \underset{j}{\&} Part(c_j', x, n_j).$$

В некотором смысле загадки данного вида являются дальнейшим усложнением рассмотренных в предыдущем параграфе загадок. Разница состоит в том, что более широко используются аналогии, ассоциации и т. д., характерные для естественного языка.

Переходим к рассмотрению конкретных примеров:

На черном плаще разбросаны:

$$\exists y(Part(x,y),\infty)\,\&\,\alpha(y,c_{ik}\ )),$$

где c_{ik} *— черный плащ.*

В блестящем зеркале отражается:

$$\exists y(\mathrm{Re}\,f(x,y)\,\&\,\alpha(y,c_{is}\ ))\,,$$

где $\mathrm{Re}\,f\,(\mathrm{x},\,\mathrm{y})$ *— x отражается в y,* c_{is} *— зеркало.*

В воздушном доме живут:

$$Part(x,c_{it},\perp)\,,$$

где c_{it} *– воздушный дом.*

В мокром видно:

$$\exists y(Vis(x,y)\,\&\,W(y))\,,$$

где *Vis (x, y) — x видно в y, W(y) — y — мокрый.*

Как и раньше, появляется множество различных предметов. Начнем уточнять первую фразу:

$$Part(x,c_{it},\infty)\,\&\,Bl(c_{it})\,,$$

где c_{it} *– плащ,* $Bl(x)$ *— x - черный.*

Но плащ может быть не только черным, но и других цветов.

Введем предикат $Col(x,y)$ *— x обладает цветом y*. Считаем, что существует конечное множество цветов $y_1,...,y_r$, которые фактически являются элементами нашей модели $y_1 = c_{i1},...,y_r = c_{ir}$, т.е. цвета одновременно являются объектами.

В информационном фонде могут быть импликации типа: *если белый, то не черный* и т.д.

Решение данной загадки — *звезды*.

Рассмотрим еще один пример.

На суке — высокий крюк:

$$\varphi_1 = On(c_{i1},c_{i2})\,\&\,H(c_{i2})\,,$$

где c_{i1} *— сук,* c_{i2} *— крюк,* $On(x,y)$ *— x на y,* $H(x)$ *— x - высокий.*

На крюке висит сундук:

$$\varphi_2 = Han(c_{i2}, c_{i3}),$$

где c_{i2} — *крюк*, по-прежнему, c_{i3} — *сундук*, $Han(x, y)$ — *висеть*.

В сундуке том 5 ребят:

$$\varphi_3 = In(c_{i3}, c_{i4}, 5),$$

где c_{i4} — *ребенок*, $In(x, y, n)$ — *находиться внутри в количестве n штук* — предикат, аналогичный *Part*.

Загадка формулируется в виде конъюнкции формул $\varphi = \varphi_1 \& \varphi_2 \& \varphi_3$.

Формуле φ сопоставляется формула

$$\varphi^*(x) = \exists \bar{y} (\&_{k=1}^{4} Part(y_k, x) \& [\varphi]_{\bar{c}}^{\bar{y}} \& \&_{k=1}^{4} \alpha(\bar{y}_k, c_{ik})).$$

Здесь $[\varphi]_{\bar{c}}^{\bar{y}}$ обозначает формулу, возникающую из φ подстановкой переменных $\bar{y}$ вместо соответствующих констант $\bar{c}$.

Далее, как обычно, ищем объект, удовлетворяющий данной формуле, т.е. такой объект x, который состоит из частей $y_1, ..., y_4$, аналогичных $c_{i1}, ..., c_{i4}$ и находящихся в тех же самых отношениях.

4. О ПРОГРАММНОЙ РЕАЛИЗАЦИИ

4.1. Средства программирования

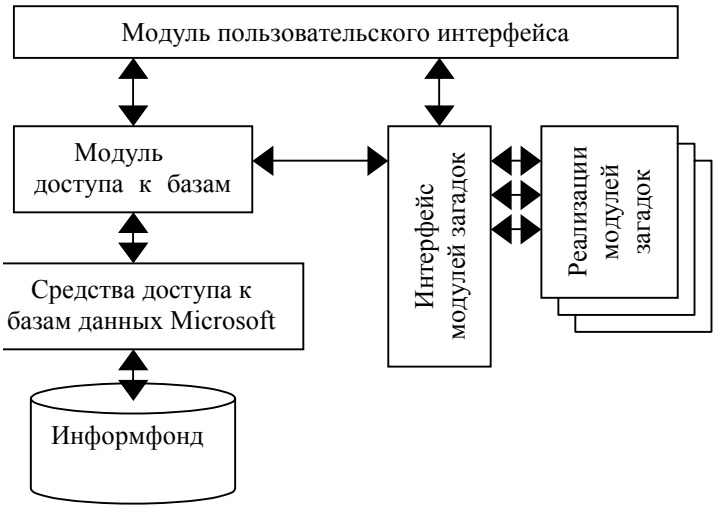
Для создания программы моделирования методов принятия решений в системе ТРИЗ на примере загадок была выбрана система Microsoft Visual C++, благодаря следующим ее особенностям:

- легкости прототипирования и создания программ с помощью интеллектуальных шаблонов (wizards) на базе библиотеки классов MFC;
- хорошо развитым и стандартизованным средствам работы с базами данных (ODBC, DAO);
- наличию объектно-ориентированной среды, повышающей скорость и качество разработки программ.

Кроме Microsoft Visual C++ в данной работе использовалась СУБД Microsoft Access из пакета Microsoft Office для моделирования и наполнения базы данных информационного фонда. Microsoft Access была выбрана из-за наличия развитых средств моделирования данных и простоты программирования.

4.2. Структура программы

Программа имеет модульную структуру (общая ее схема показана на приведенном ниже рисунке).



Структура программы

Модуль пользовательского интерфейса создан на базе библиотеки классов MFC. В его функции входит обеспечение связи с пользователем: выбор конкретного класса моделируемых загадок и возможность задания параметров генерации загадки. Модуль пользовательского интерфейса позволяет также выбрать базу данных для использования ее в качестве информационного фонда. Данная программа не решает задачи литературного и ритмического оформления результата работы. В пользовательском интерфейсе

для окончательного оформления загадки предусмотрена возможность ее редактирования.

5. ЗАКЛЮЧЕНИЕ

В данной работе исследованы методы принятия решений в системе ТРИЗ и продемонстрирована возможность их формализации. Формализация достигнута за счет классификации модельного материала и описания его на языке предикатов. Переход от модельного материала (загадок) к реальным задачам — предмет дальнейшего интересного исследования.

Создан программный продукт, который может найти применение в педагогике для генерации учебного материала, например для курса развития творческого мышления.

СПИСОК ЛИТЕРАТУРЫ

1. **Альтшуллер Г.С., Шапиро Р.Б.** О психологии изобретательского творчества //Вопросы психологии. — 1956. — N 6. — С. 37—49.
2. **Альтшуллер Г.С.** Найти идею // Новосибирск: Наука, 1986; 1991.
3. **Slocum M.S.** TRIZ: infinite in all directions //TRIZ J. — 1998.

СОДЕРЖАНИЕ

CONTENTS

УДК 519.6

О работе 16 Всемирного компьютерного конгресса ИФИП / Касьянов В. Н. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 9–30.

Представлен обзор работ 16-го Всемирного компьютерного конгресса, проходившего в августе 2000-го года в Пекине под лозунгом "Обработка информации. За рубежом 2000 года". Компьютерные конгрессы, проводимые ИФИП, являются главным мировым научным форумом в области информационных технологий, на котором рассматриваются основные проблемы и наиболее важные результаты в современной информатике. — Библиогр.: 0 назв.

At the 16-th IFIP World Computer Congress / Kasyanov V. N. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 9–30.

This is a review of the 16-th IFIP World Computer Congress held in Beijing, China, in August, 2000 under the title "Information Processing Beyond Year 2000". The IFIP Computer congress is the most important world scientific forum on information technologies where the main problems and the most significant results in modern informatics are considered. — Refs: 0 titles.

УДК 519.6 + 681.3.06

Анализ зависимостей по данным на базе алгоритма Шостака / Логачева С. А. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 31–43.

При распараллеливании программ одной из основных проблем является выявление зависимостей по данным. В настоящей статье строится точный тест для анализа зависимостей по данным, основанный на исследовании разрешимости системы линейных неравенств с двумя переменными на базе алгоритма Шостака. В данной работе приводится модель входной программы, описываются алгоритм Шостака и его модификация, а также программная реализация проекта. — Библиогр.: 10 назв.

Data dependence analysis based on the Shostak algorithm / Logacheva S. A. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 31–43.

The main problem of program parallelizing is to find out data depedences. This article presents a precise test for data dependence analysis based on analysis of solvability of a system of linear inequalities in two variables on the basis of Shostak algorithm. The model of a source program, as well as the Shostak algorithm and its modification, is given and implementation of the project is described. — Refs: 10 titles.

УДК 519.6 + 681.3.06

NUMA-архитектура: некоторые особенности компиляции и генерации кода / Евстигнеев В. А. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 44–53.

Обсуждаются некоторые особенности компиляции и генерации кода для NUMA-архитектур, дается определение понятия NUMA-архитектуры, приводятся различия между мультипроцессорами и мультикомпьютерами, перечисляются машины с NUMA-архитектурой, описываются особенности про-

граммного обеспечения NUMA-машин, затрагиваются вопросы реструктуризации циклов, порогового планирования графа заданий, унифицирующих преобразований данных и управления, а также представлен обзор работ, имеющих непосредственное отношение к реструктуризации циклов и преобразованиям данных и управления. — Библиогр.: 13 назв.

NUMA-architecture: specific features of compilation and code generation / Evstigneev V. A. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 44–53.

This paper is about some features of compilation and code generation for NUMA-architectures. The definition of a NUMA- architecture is given and the difference between multiprocessors and multicomputers is shown. Computers with NUMA-architectures and the peculiarities of software for them are described. A review of the papers related to cycle restructurization and data transformation is also given. — Refs: 13 titles.

УДК 519.6 + 681.3.06

Функциональный язык Sisal 3.0 / Касьянов В. Н., Бирюкова Ю. В., Евстигнеев В. А. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 54–67.

Представлен язык функционального программирования Sisal 3.0, выбранный в качестве начальной версии входного языка системы функционального программирования СФП, разрабатываемой в ИСИ СО РАН при финансовой поддержке РФФИ (грант N 98-01- 00748). В работе кратко описаны семантические и синтаксические характеристики этого языка, а также сделан обзор материалов, связанных с проблематикой языка Sisal 3.0. — Библиогр.: 31 назв.

A functional language SISAL 3.0 / Kasyanov V. N., Biryukova Yu. V., Evstigneev V. A. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 54–67.

The paper describes the Sisal 3.0 language which is the initial version of the source language of the functional programming system (FPS). Semantic and syntactical features of this language have been briefly described and a review of the papers related to the Sisal 3.0 problems have also been done. — Refs: 31 titles.

УДК 519.6 + 681.3.06

Параллельный алгоритм решения задачи о взаимодействии потоков разреженной плазмы / Вшивков В. А., Лобив И. В., Мурзин Ф. А. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 68–81.

Исследуется параллельный алгоритм решения задачи о взаимодействии потоков разреженной плазмы. Рассматривается простейшая параллельная архитектура. В рамках некоторых естественных предположений сделаны оценки времени выполнения алгоритма в параллельном и последовательном случаях, а также коэфициента ускорения. — Библиогр.: 4 назв.

A parallel algorithm for solving a problem of interaction between low-density plasma flows / Vshivkov V. A., Lobiv I. V., Murzin F. A. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 68–81.

The parallel algorithm for solving a problem of interaction between low-density plasma flows is studied. The most elementary parallel architecture is considered.

In the context of some natural assumptions, the run-time estimates have been made for the parallel and sequential cases; the acceleration coefficient has also been evaluated. — Refs: 4 titles.

УДК 519.6 + 681.3.06

О распараллеливании метода “МЕДУЗА” / Бурдонов И. В., Мурзин Ф. А. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 82–107.

Рассматриваются вопросы, возникающие при распараллеливании метода “МЕДУЗА”, который используется для решения ряда задач математической физики. — Библиогр.: 6 назв.

On parallelizing the MEDUZA technique / Burdonov I. V., Murzin F. A. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 82–107.

The problems connected with parallelizing the “MEDUZA” technique applied to a number of mathematical physics problems are considered. — Refs: 6 titles.

УДК 519.6 + 681.3.06

Программные средства для тестирования алгоритмов по обработке изображений / Мурзин Ф. А., Семич Д. Ф. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 108–113.

Кратко описываются алгоритмы, положенные в основу программы "испытательного стенда", который предназначен для разработки прикладного программного обеспечения в области обработки изображений.

Software for testing the image processing algorithms / Murzin F. A., Semich D. F. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 108–113.

The algorithms which are a base of "a test desk"program intended to develop applied software for image processing are briefly described.

УДК 519.6 + 681.3.06

Язык описания графовых моделей и алгоритмов GRAMAL / Маркин В. А. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 114–124.

При построении программных систем различных уровней сложности часто и широко используются графовые модели и различные методы их обработки. Графы, являясь очень удобным инструментом описания структур данных и информационных потоков, активно используются в различных математических задачах. Именно для этого создана система GRAMAL, преследующая четыре цели: предоставить инструмент для описания графовых моделей; графически представить эту модель; предоставить средства тестирования и отладки методов работы с графами; обеспечить возможность генерации программного кода для последующего применения. — Библиогр.: 5 назв.

A GRAMAL language for description of graph models and algorithms / Markin V. A. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 114—124.

Graph models and various methods of their processing are often and widely used when constructing software systems of different levels of complexity. Graphs (being a very convenient tool for description of data structure and information

flows) are actively used in a number of mathematical problems. The GRAMAL system here described is aimed at 4 points:

- to provide a tool for graph model description;
- to represent this model graphically;
- to provide testing and debugging tools for graph processing;
- to support code generation and its further application. — Refs: 5 titles.

УДК 681.3.016-681.3.06

Трансформ: интерфейс для ввода информации / Волянская Т. А., Малинина Ю. В. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 125–138.

Описывается интерфейс для ввода информации в информационно-поисковую систему Трансформ. Источник данных для этой системы - публикации по преобразованиям программ. Цель этой работы - создание программного комплекса, предоставляющего удаленному пользователю, связавшемуся с WWW-сервером ИС Трансформ, удобный интерфейс для регистрации в системе, ввода информации в базу данных, формирования запросов поиска, осуществления этого поиска и выдачи результатов в удобном виде. — Библиогр.: 3 назв.

Transform: a data input interface / Volyanskaya T. A., Malinina Yu. V. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 125–138.

A data input interface for the information retrieval system Transform is described. A source of data for this system is bibliography on program transformations. The aim of this work is to create a program complex which provides a remote user connected to IS Transform WWW-server with a convenient interface for registration in the system, input of data to the database, information retrieval, and output of results in a convenient form. — Refs: 3 titles.

УДК 519.6 + 681.3.06

Применение графов в программировании / Касьянов В. Н. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 139–167.

Теория графов из академической дисциплины все более превращается в средство, владение которым становится решающим для успешного применения ЭВМ во многих прикладных областях. Статья посвящена средствам поддержки применения графов в программировании, которые создаются сотрудниками лаборатории конструирования и оптимизации программ Института систем информатики СО РАН им. ак. А.П.Ершова при финансовой поддержке РФФИ. Дается обзор работ над "энциклопедией"алгоритмов на графах для программистов. Описываются методы и инструменты для визуальной обработки графов (графовых моделей). Рассматривается толковый словарь по теории графов в информатике и программировании, а также его электронная версия. — Библиогр.: 27 назв.

Application of graphs to programming / Kasyanov V. N. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 139–167.

This article describes a number of tools which support application of graphs to programming; they are under development in the Laboratory for program construction and optimization of the A.P. Ershov Institute of Informatics Systems.

A review of the papers on "encyclopedia"of graph algorithms for programmers is given. Methods and tools for graph (and graph model) visual processing are described. The dictionary on graph theory in programming and its electronic version are considered. — Refs: 27 titles.

УДК 519.6 + 681.3.06

Эквивалентность моделей локальных вычислений / Бояршинов В. А. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 168–178.

Рассматривается эквивалентность моделей локальных вычислений. В настоящее время существует несколько моделей локальных вычислений на графах: системы переписывания графов с приоритетом, системы переписывания графов с запрещенными контекстами, локальные алгоритмы Журавлева, сети конечных автоматов. Обсуждается вопрос сравнения классов задач, разрешимых за полиномиальное время в различных моделях локальных вычислений. Формулируется ряд предложений и приводятся их доказательства. — Библиогр.: 9 назв.

Equivalence of local computation models / Boyarshinov V. A. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 168–178.

This paper considers equivalence of local computation models. There exist several models of local computations on graphs, such as the systems of graph rewriting with priority, the systems of graph rewriting with supressed context, Zhuravlev's local algorithms, and the finite automata nets. The paper discusses the question of comparison between classes of problems solvable in polynomial time invarious models of local computations. Some propositions and their proofs are given. — Refs: 9 titles.

УДК 519.6 + 681.3.06

Некоторые инварианты кубоподобного графа / Мельников Л. С., Петренко И. В. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 179–192.

Кубоподобный граф введен в рассмотрение Ловасом. Это граф со специальным отношением смежности на своих вершинах. Статья посвящена изучению ряда инвариатов кубоподобного графа, в частности, хроматических. Найдено значение реберного предписанного хроматического числа, а также найден метод вычисления кликоматического числа и плотности кубоподобного графа. — Библиогр.: 13 назв.

Some invariants of a cube-like graph / Mel'nikov L. S., Petrenko I. V. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 179–192.

A cube-like graph was firstly introduced by Lovas. This is a graph with a special relation of vertex incidence. The article is devoted to a study of some cube-like graph invariants, in particular, chromatic ones. The value of an edge prescribed chromatic number is found, and the method for calculation of the cliquemathic number and the clique number of a cube-like graph is also described. — Refs: 13 titles.

УДК 519.6 + 681.3.06

Организация графического вывода в системе визуализации иерархических графовых моделей / Лисицын И. А. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 193–210.

Рассматриваются методы организации графического вывода в системах визуализации графовых объектов. Отмечаются и классифицируются основные проблемы, возникающие при проектированнии таких систем, даются общие методы их решения и описывается конкретный вариант реализации данных подходов в системе Higres, являющейся визуализатором и редактором иерархических графовых моделей. — Библиогр.: 9 назв.

A graphic output mechanism in the visualization system for hierarchical graph models / Lisitsyn I. A. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 193–210.

The mechanisms of graphic output in graph visualization systems are considered. The main problems of constructing such systems are presented and classified and the general solution methods are given for them. A particular implementation of these approaches in the Higres system, which is a visualizer and editor for hierarchical graph models, is described. — Refs: 9 titles.

УДК 519.6 + 681.3.06

Организация пользовательского интерфейса в системе визуализации иерархических графовых моделей / Лисицын И. А. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 211–221.

Описываются интерфейсные решения, примененные в системе Higres, являющейся визуализатором и редактором иерархических графовых моделей. Рассматриваются особенности визуализации иерархической структуры графа, методы, позволяющие предоставить пользователю возможность быстро и удобно редактировать граф и его изображение, алгоритмы автоматического достраивания изображения, а также дополнительные возможности, реализованные в системе. — Библиогр.: 9 назв.

User interface in the visualization system for hierarchical graph models / Lisitsyn I. A. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 211–221.

The user interface of the Higres system, a visualizer and editor for hierarchical graph models, are described. The methods for visualization of the hierarchical structure are presented which provide one with easy-to-use intuitive graph editor interface. The algorithms for automatic drawing refinement and some additional features implemented in the system are considered. — Refs: 9 titles.

УДК 519.6 + 681.3.06

Подготовка графовых иллюстраций с помощью системы VEGRAS / Мердишева Т. С., Мердишева Е. С. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 222–228.

В настоящее время растет интерес к методам и системам визуализации графов. Статья посвящена описанию системы VEGRAS для подготовки графовых иллюстраций, которая отличается простотой в использовании и позволяет создавать достаточно качественные изображения. Данная система является универсальным инструментом для визуализации и редактирования атрибутированных графов. — Библиогр.: 8 назв.

Making graph-based illustrations with the help of VEGRAS system / Merdisheva T. S., Merdisheva E. S. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 222–228.

The Vegras system described in this article is an editor tool for making graph-based illustrations. The motivation to develop such software is the lack of easy-to-use graph editor tools which allow users to make graph-based illustrations for scientific literature. With the help of our system it is possible to create nice looking graph-based illustration in a few minutes. In addition, our system provides a possibility to add different special mathematical symbols to graph's labels, as well as subscripts and superscripts. — Refs: 8 titles.

УДК 519.68

Реализация сопоставления с образцом в языке Lisp на основе аналогичных средств в языках Refal и Haskell / Харитонов Э. В. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 229–237.

Описано расширение языка Lisp средствами сопоставления с образцом на основе аналогичных средств в языках Refal и Haskell. Рассматриваются некоторые способы введения в язык Lisp средств сопоставления с образцом и оценивается удобство применения этих средств для обработки структурированных данных. — Библиогр.: 4 назв.

A LISP-implementation of a pattern-search on the basis of similar procedures of Refal and Haskell languages / Kharitonov E. V. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 229–237.

An extension of Lisp with a pattern-search procedure based on similar ones of Refal and Haskell languages is described. Some methods for extension of Lisp with pattern-search tools are considered and convenience of using them for structured data processing is estimated. — Refs.: 4 titles.

УДК 681.3.016-681.3.06

Использование шаблонов при разработке WIS / Малинина Ю. В. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 238–244.

В настоящее время WEB является самым популярным информационным хранилищем. Возрастающая популярность Интернета и быстрая эволюция программного обеспечения от любительских Web-сайтов до корпоративных систем заставляет развиваться гипермедиа-технику быстрее, чем когда-либо прежде. В предлагаемой статье рассматривается использование проектных шаблонов для решения различных проблем, возникающих на современном этапе проектирования гипермедиа-приложений, а именно процесс применения шаблонов для улучшения проектирования архитектуры Web- Информационных Систем (WIS). — Библиогр.: 9 назв.

Using patterns in WIS development / Malinina Yu. V. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 238–244.

Today WEB is the most popular information storage. This article discusses the use of project patterns when solving different problems of constructing hypermedia-applications, namely, the process of using patterns aimed to improve constructing the architecture of WEB-Information Systems (WIS). — Refs.: 9 titles.

УДК 519.6 + 681.3.06

Логическая структура процесса генерации и отгадывания загадок / Дылыков Ж. Л.-Д., Занаева Н. Б., Марьясов Е. А., Мурзин Ф. А., Семич Д. Ф. // Поддержка супервычислений и Интернет-ориентированные технологии. — Новосибирск, 2001. — С. 245–253.

Исследуются методы принятия решений в концепции ТРИЗ на примере процесса генерации и отгадывания загадок. — Библиогр.: 3 назв.

A logical structure of the process of puzzle generation and solution / Dylykov J. L.-D., Zanaeva N. B., Mar'yasov E. A., Murzin F. A., Semich D. F. // Supercomputing support and Internet-oriented technologies. — Novosibirsk, 2001. — P. 245–253.

The decision-making methods in the TRIZ concept are investigated for the process of puzzle generation and solution. — Refs: 3 titles.

ПОДДЕРЖКА СУПЕРВЫЧИСЛЕНИЙ И ИНТЕРНЕТ-ОРИЕНТИРОВАННЫЕ ТЕХНОЛОГИИ

Под редакцией
проф. Виктора Николаевича Касьянова

Рукопись поступила в редакцию 15. 12. 2000

Ответственный за выпуск Г. П. Несговорова
Редактор Л. А. Карева, З. В. Скок

Подписано в печать 30. 03. 2001
Формат бумаги 60 × 84 1/16 Объем 15,0 уч.-изд.л., 16,5 п.л.
Тираж 100 экз.

НФ ООО ИПО "Эмари" РИЦ, 630090, г. Новосибирск, пр. Акад. Лаврентьева, 6