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*)

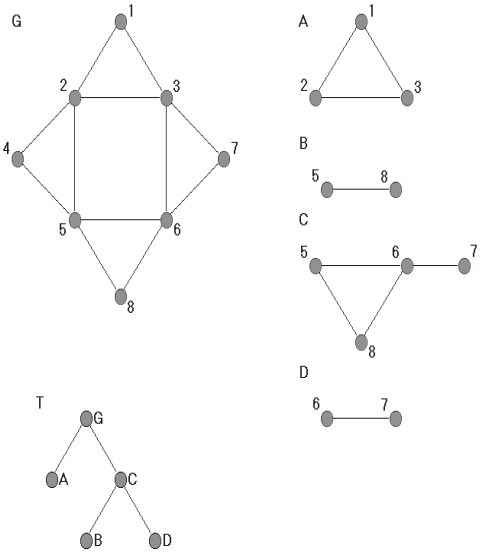

Рис. 1. Пример связного иерархического графа

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.

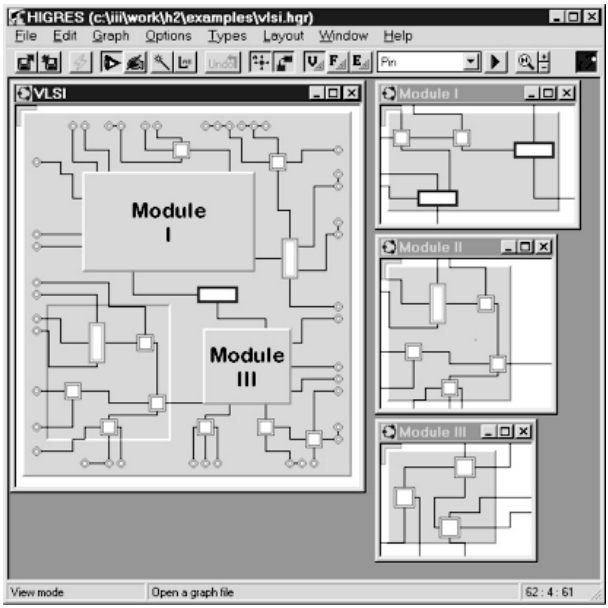

Рис. 2. Модель СБИС, представленная иерархическим графом

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.

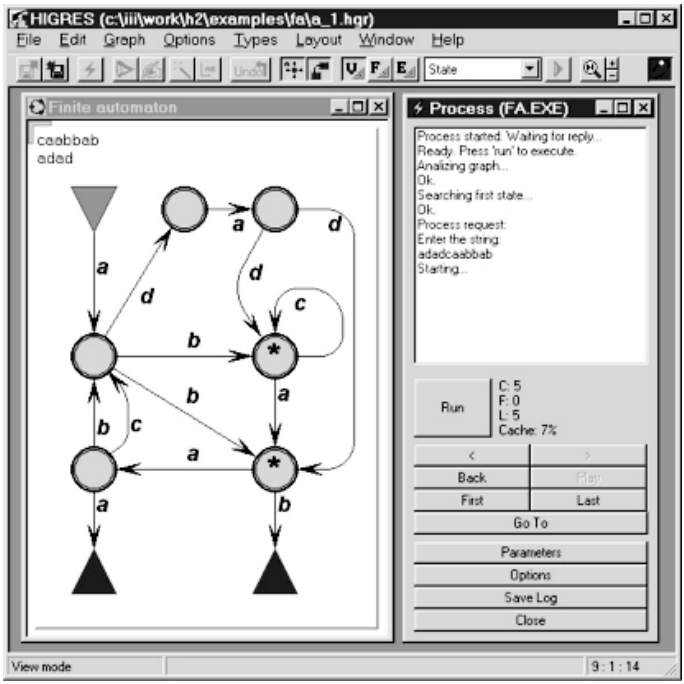

Рис. 3. Моделирование работы конечного автомата

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.

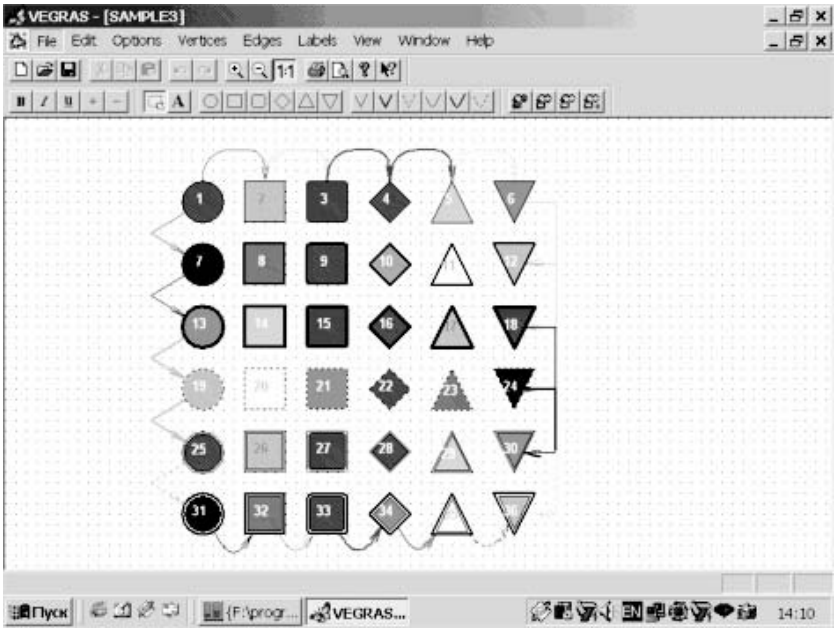

Рис. 4. Система VEGRAS

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.

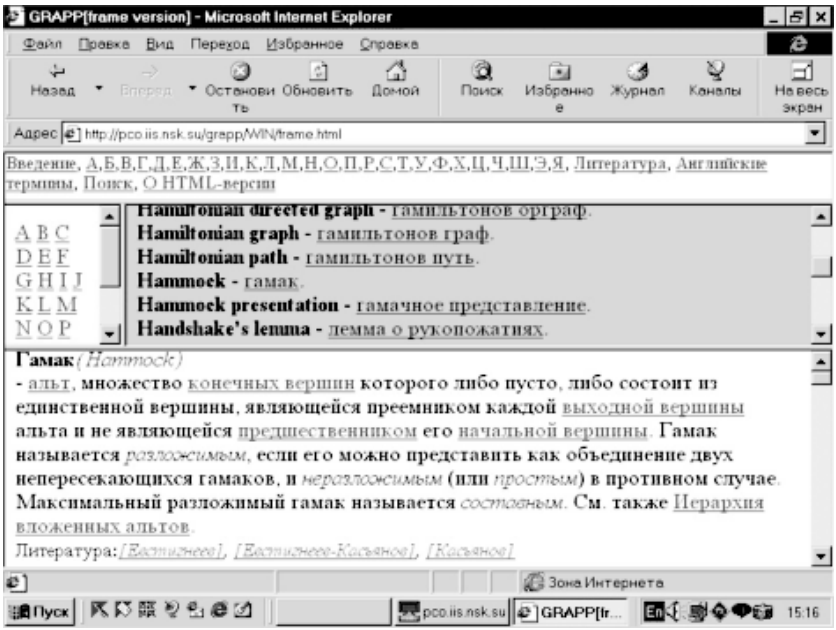

Рис. 5. Система GRAPP

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.