

ТЕОРЕТИКО-ГРАФОВЫЕ МЕТОДЫ И СИСТЕМЫ ПРОГРАММИРОВАНИЯ

В. Н. Касьянов, Е. В. Касьянова

Институт систем информатики им. А. П. Ершова СО РАН,
630090, Новосибирск, Россия
Новосибирский государственный университет,
630090, Новосибирск, Россия

УДК 004

Статья посвящена теоретико-графовым методам и системам программирования, работа над которыми ведется в лаборатории конструирования и оптимизации программ ИСИ СО РАН при финансовой поддержке Российского фонда фундаментальных исследований.

Ключевые слова: визуализация, графы, графовые алгоритмы, системы программирования.

The paper is devoted to graph-theory methods and programming systems being under development at Laboratory for Program Construction and Optimization of IIS SB RAS with support of the Russian Foundation for Basic Research.

Key words: visualization, graphs, graph algorithms, programming systems.

Введение. Современное состояние программирования нельзя представить себе без теоретико-графовых методов и алгоритмов. Широкая применимость графов в программировании связана с тем, что они являются очень естественным средством объяснения сложных ситуаций на интуитивном уровне.

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.), граф-схемной модели для императивных программ в виде операторных алгоритмов (1958–1962 гг.), теории схем Янова с использованием их графового представления и концепции так называемой разметки (1966–1968 гг.) и граф-схемной теории экономии памяти (1961–1966, 1972 гг.). Первой книгой, посвященной применению графов в программировании, была изданная в 1977 г. монография А. П. Ершова [1], в которой он рассмотрел две классические задачи теоретического программирования, решения которых и развитые на этих решениях методы привели к созданию теоретического программирования как самостоятельной математической дисциплины. Это задача экономии памяти в операторных схемах Лаврова и задача построения полной системы преобразований в схемах Янова. В данной книге, написанной в виде беседы с читателем, Андрей Петрович продемонстрировал применение графовых методов к решению задач программирования в действии, начиная с элементарных постановок решаемых задач и кончая полным решением проблем во всей их сложности.

Авторы благодарны всем, кто принимает участие в выполнении проектов, рассмотренных в данной статье. Работа выполнена при финансовой поддержке РФФИ (грант № 15-07-02029).

Программирование, по словам А. П. Ершова, это новый вид универсальной деятельности, при которой человек должен вложить в ЭВМ все, что видит, слышит, знает, и научить ее всему, что делает сам. Важнейшим свойством информационной модели или управляющей системы является ее структура, или, говоря математическим языком, совокупность бинарных отношений на наборах элементарных единиц данных и действий. Эти структуры данных и структуры действий являются единственными ипостасями программ и обрабатываемой ими информации, в которых они могут существовать в воображении программиста во чреве компьютера. Вот почему, утверждал Андрей Петрович, графы являются основной конструкцией для программиста. Он считал, что „графы обладают огромной, неисчерпаемой изобразительной силой, соразмерной масштабу задачи программирования“, и говорил, что „программисту о графах нужно много знать, при этом с большим запасом по отношению к любой конкретной задаче“.

Поэтому не случайно, что в отличие от Москвы, где, начиная с работ Юрия Ивановича Янова, в большей степени развивался логический подход к программированию, или Киева, где в работах Виктора Михайловича Глушкова и его учеников явно прослеживается приоритет алгебраических методов, Новосибирск стал центром применения теоретико-графовых моделей и методов в программировании. Созданная в Новосибирске академиком А. П. Ершовым и его учениками авторитетная школа программирования, пользующаяся мировой известностью, внесла значительный вклад в становление и развитие теоретического и системного программирования с использованием теоретико-графовых методов [2].

Это направление по-прежнему продолжает активно развиваться и в наши дни, теперь уже в работах сотрудников Института систем информатики СО РАН, носящего имя академика А. П. Ершова. В статье мы остановимся на некоторых из этих работ, выполняемых в лаборатории конструирования и оптимизации программ ИСИ СО РАН при финансовой поддержке Российского фонда фундаментальных исследований.

1. Вики-словарь WikiGRAPP. Проблема терминологии, без сомнения, является одной из основных проблем в применении теоретико-графовых методов в программировании и информатике. Терминология в прикладной теории графов далеко не устоялась, при написании статей требуется терминологическая привязка к одной из существующих на русском языке монографий, что становится все более трудным делом из-за сокращения числа издающихся книг, в том числе переводных, и резкого сокращения их тиража.

В 1999 году в издательстве „Наука“ вышел в свет толковый словарь по теории графов и ее применению [3], который охватывал основные связанные с графами термины из монографий, вышедших на русском языке. Это был первый словарь по графам в информатике, и он вызвал большой интерес среди читателей. Электронная версия словаря получила название GRAPP (GRaphs and their APplications).

Новое исправленное и дополненное издание словаря [4], работа над которым была завершена авторами в 2009 году, представляет собой расширение словаря 1999 года и включает в себя более 1000 новых терминов из статей, рефераты которых публиковались в РЖ „Математика“ в разделе „Теория графов“, а также из томов ежегодных конференций „Graph-Theoretic Concepts in Computer Science“ и книг серии „Graph Theory Notes of New York“.

Авторы словарей отдавали себе отчет в постоянно развивающемся теоретико-графовом лексиконе в информатике и поэтому одновременно с завершением работ по подготовке второго словаря к изданию инициировали работы по созданию на базе изданных словарей новой версии электронного словаря, которая была бы расширяемой. Новый словарь, полу-

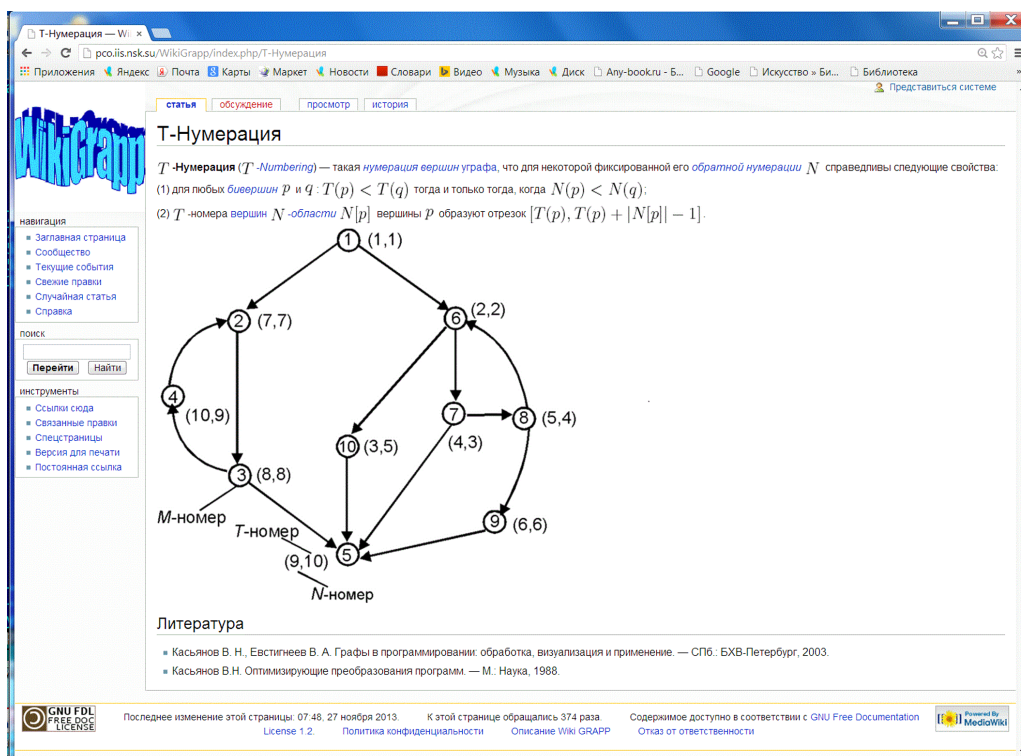


Рис. 1. Вики-словарь WikiGRAPP

чивший название WikiGRAPP, обладает интерактивностью и поддерживает коллективную сетевую работу по его пополнению и развитию (рис. 1).

Для его создания мы использовали систему MediaWiki — написанное на препроцессоре гипертекста (PHP) свободно распространяемое программное обеспечение, предназначенное для создания гипертекстовых „вики“-систем таких вебсайтов, структуру и содержимое которых пользователи могут сообща изменять с помощью инструментов, предоставляемых самими сайтами. И хотя MediaWiki изначально создавалась в качестве „движка“ всем известной „Википедии“, она имеет гораздо более широкое применение в многочисленных и весьма разнообразных сайтах, включая сайт о самой системе MediaWiki.

К настоящему времени завершена работа по наполнению словаря WikiGRAPP до уровня, покрывающего печатные издания, и ведется активная работа по улучшению статей словаря и пополнению его новыми терминами. Осуществлено расширение словаря (и энциклопедии WEGA — см. ниже) средствами автоматизации подготовки оригинал-макетов изданий произвольных подборок его статей, выбранных пользователем.

2. Вики-энциклопедия WEGA. Теория графов из академической дисциплины все более превращается в средство, владение которым становится решающим для успешного применения компьютеров во многих прикладных областях. Несмотря на наличие обширной специальной литературы по решению задач на графах, широкое применение в практике программирования полученных математических результатов затруднено в силу отсутствия систематического их 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.

Выполнен цикл работ по изучению и систематизации применения теоретико-графовых методов в программировании. Впервые издана книга [5], которая содержит систематическое и полное изложение фундаментальных основ современных компьютерных технологий, связанных с применением теории графов. Даны основные модели, методы и алгоритмы прикладной теории графов. Подробно описаны такие основные области приложения теории графов в программировании, как хранение и поиск информации, трансляция и оптимизация программ, анализ, преобразование и распараллеливание программ, параллельная и распределенная обработка информации.

Ведется работа по созданию на базе этой книги вики-системы WEGA, являющейся расширяемой интерактивной электронной энциклопедией теоретико-графовых алгоритмов решения задач информатики и программирования (рис. 2).

В отличие от Д. Кнута, при создании данной энциклопедии мы, как и в изданной книге, ориентируемся на абстрактную модель современных ЭВМ (равнодоступная адресная машина РАМ) и высокоуровневое описание алгоритмов в терминах специального языка высокого уровня (ВУ-язык). Этот язык является псевдоязыком (лексиконом) программирования и содержит в качестве базовых традиционные конструкции математики и языков программирования (рис. 3). Наряду с обычными для современных языков типами простых и составных данных, он допускает такие более сложные структуры данных, как, например, деревья, графы и т. д. Для каждой базовой конструкции ВУ-языка фиксируется класс ее допустимых реализаций на РАМ. Предполагается, что ВУ-язык позволяет наряду с базовыми использовать любые необходимые конструкции, если очевидны или заранее зафиксированы оценки их сложности, а также те реализации этих конструкций на РАМ, которые допускают такие оценки. Такой подход позволяет формулировать алгоритмы в естественной форме, допускающей прямой анализ их корректности и сложности, а также простой перенос алгоритмов на реальные языки программирования и ЭВМ с сохранением полученных оценок сложности. Подобный стиль описания алгоритмов является также базой для доказательного стиля программирования: он позволяет понять алгоритм на содержательном уровне, оценить его пригодность для решения конкретной задачи и осуществить модификацию алгоритма, не снижая степень математической достоверности окончательного варианта программы.

Еще одной важной особенностью создаваемой энциклопедии является возможность анимационного исполнения представленных в ней графовых алгоритмов. На наш взгляд, анимация является удобным средством демонстрации работы любых алгоритмов, в том числе алгоритмов на графах. Она помогает человеку понять на конкретных примерах смысл и последовательность работы алгоритма, делая это гораздо нагляднее, чем любые текстовые описания, пояснения и отдельные рисунки. Поэтому создаваемая энциклопедия является интерактивной и ориентирована на поддержку полноценной динамической визуализации (анимации) работы графовых алгоритмов, представленных в ней.

3. Системы визуализации графов и графовых алгоритмов. Визуализация информации — это процесс преобразования больших и сложных видов абстрактной информации в интуитивно понятную визуальную форму. Универсальным средством такого представления структурированной информации являются графы. Графы применяются для представления любой информации, которую можно промоделировать в виде объектов и связей между объектами. Поэтому визуализация теоретико-графовых моделей является

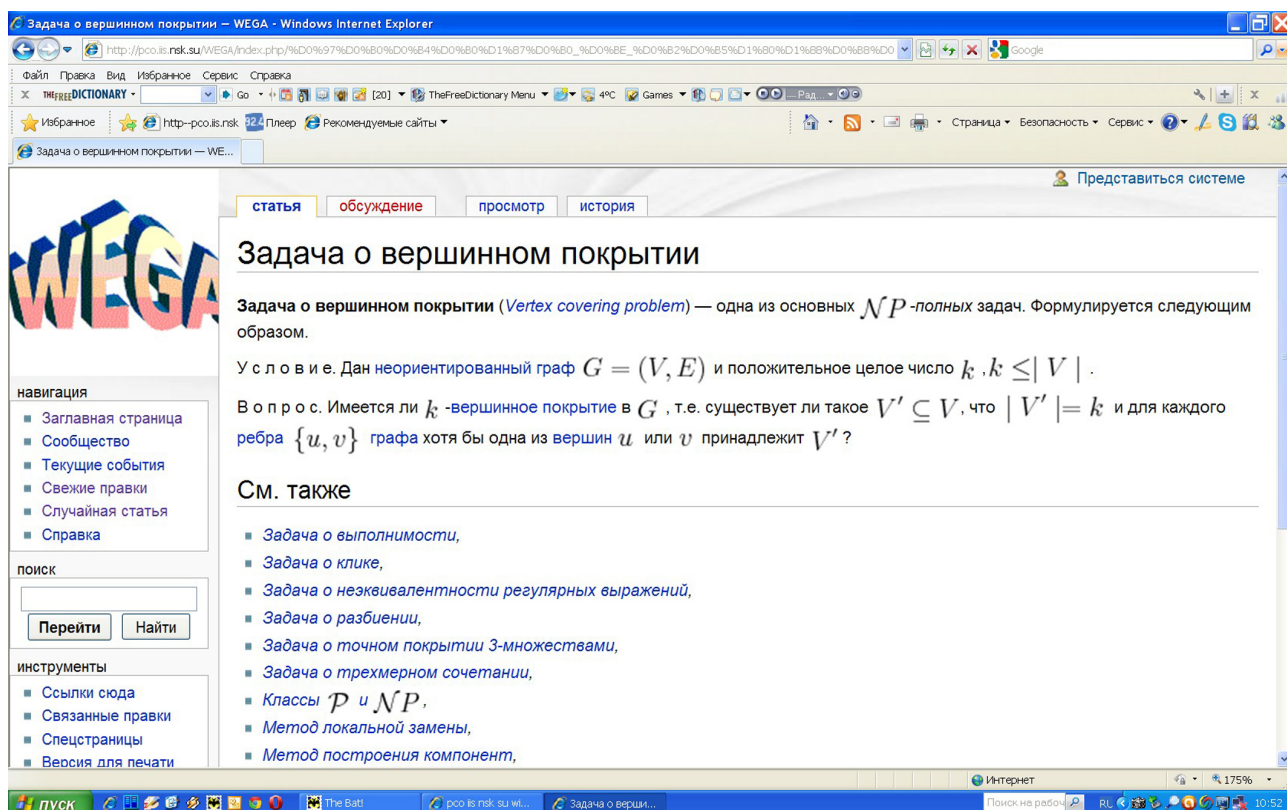


Рис. 2. Вики-энциклопедия WEGA

ключевой компонентой во многих приложениях в науке и технике, а методы визуализации графовых моделей представляют собой теоретическую основу методов визуализации абстрактной информации [6, 7].

Выполнен цикл исследований в области визуализации графов и графовых моделей, в котором наряду с более традиционными вопросами качества и эффективности при автоматическом размещении графов на плоскости большое внимание было уделено вопросам визуализации больших графов, интерактивности и навигации, характерным для большинства современных приложений, использующих визуализацию структурированной информации [8].

Предложена и исследована модель иерархических графов, ориентированная на моделирование сложноорганизованных систем и охватывающая классы составных и кластерных графов, традиционно используемые при визуализации сложных систем, обладающих иерархической структурой [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 орграфом (ориентированным графом), мультиграфом (с кратными ребрами) или псевдографом (с петлями).

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.

З а д а ч а

О б ъ е к т ы. Ориентированный граф G , каждая вершина которого может находиться в одном из двух состояний: „помечена“, „непомечена“.

О п е р а ц и и. Для любой вершины p графа G операция ПОМЕТИТЬ(p) выполнима, если p находится в состоянии „непомечена“, и при своем выполнении переводит p в состояние „помечена“, а предикат НЕПОМЕЧЕНА(p) ложен, если p находится в состоянии „помечена“.

Д а н о. Задана вершина p_0 , такая, что каждая вершина графа G , достижимая из p_0 , находится в состоянии „непомечена“.

Т р е б у е т с я. Перевести в состояние „помечена“ все вершины графа G , достижимые из p_0 .

З а м е ч а н и е. Предполагается, что процедура ПОМЕТИТЬ(p) при своем выполнении осуществляет определенные действия с информационным содержимым вершины p , для реализации которых и производится обход графа.

М е т о д

```

проц ЛЕС( $p_0$ : вершина) =
   $S$ : семейство дуг =  $\emptyset$ ;
   $q$ : вершина =  $p_0$ ;
   $L$ : начало ПОМЕТИТЬ ( $q$ );
   $S \leftarrow \exists$  ИСХОД( $q$ );
  пока  $S \neq \emptyset$  цикл
     $q :=$  КОНЕЦ( $\exists S$ );
  если НЕПОМЕЧЕНА( $q$ ) то начать  $L$  все
все
конец
все

```

все**С в о й с т в а**

1. Временная сложность алгоритма составляет $O(k)$ времени, где k — число дуг графа G , достижимых из вершины p_0 .
2. Алгоритм изменяет состояние непомеченной вершины q тогда и только тогда, когда q достижима из p_0 .
3. Если S является стеком, то процедура помечает вершины графа G , достижимые из p_0 , в порядке поиска в глубину, а если S — очередь, то вершины помечаются в порядке поиска в ширину.

Рис. 3. Общий алгоритм обхода графа с запоминанием дуг

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 .

Важный частный случай иерархических графов образуют так называемые простые иерархические графы, в которых все фрагменты являются подграфами основного графа. Поскольку подграфы однозначно определяются множествами своих вершин, есть возможность определять простой иерархический граф H как пару (G, T) , состоящую из основного графа G и вершинного дерева вложенности T , удовлетворяющего следующим условиям. Вершины дерева T соответствуют некоторым подмножествам вершин основного графа G таким образом, что подмножества вершин, соответствующие листьям T , образуют разбиение множества всех вершин графа G на одноэлементные подмножества. Дуги дерева T отражают непосредственную вложенность соответствующих подмножеств. Такое представление иерархического графа называется вершинным (рис. 4).

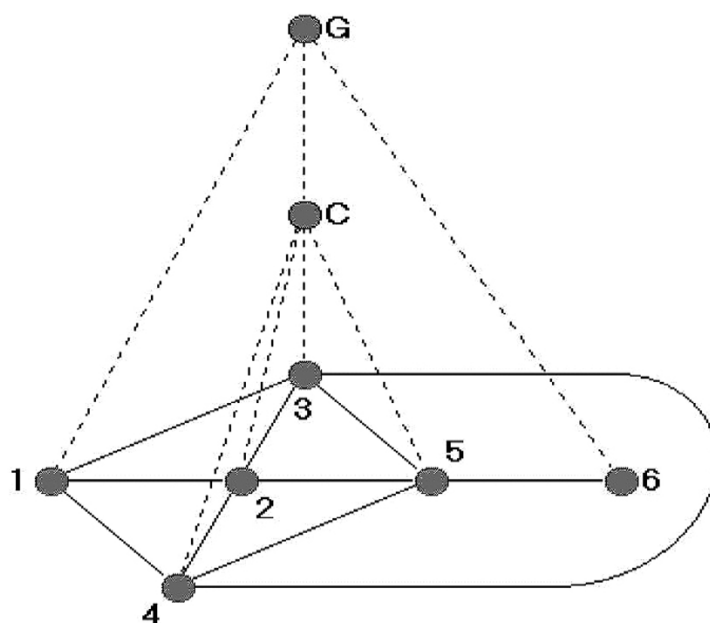


Рис. 4. Вершинное представление простого иерархического графа с двумя нетривиальными фрагментами: сплошные линии — это ребра основного графа, штриховые линии — ребра вершинного дерева вложенности

Разработаны эффективные методы и алгоритмы построения наглядных изображений иерархических графовых моделей на плоскости и их редактирования, и создан графовый редактор *Higres*, поддерживающий многооконную работу с иерархическими графами [10]. Важным его отличием от других графовых редакторов является способность *Higres* сохранять во внутреннем представлении и визуализировать не только сам граф, но и его семантику, представленную в виде системы типов атрибутированных вершин, фрагментов и дуг графа и библиотеки алгоритмов обработки графов так называемых внешних модулей. При этом пользователь может корректировать и доопределять семантику графовой модели с помощью введения новых типов и внешних модулей, а также управлять методами ее визуализации. Такой подход обеспечивает с одной стороны универсальность системы *Higres*, с другой — возможность ее специализации.

Несомненным достоинством системы *Higres* является также то, что она является не только редактором иерархических графовых моделей, но и платформой для исполнения и анимации алгоритмов работы с графами (рис. 5). Пользователь с помощью системы может выбрать нужный ему алгоритм из расширяемой библиотеки внешних модулей (в частности, алгоритм автоматического размещения графа на плоскости) и запустить его. Система передает текущий граф обрабатываемому 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. Анимация алгоритмов, поддерживаемая системой *Higres*, может быть использована для их тестирования и отладки, для образовательных целей, а также для изучения итеративных процессов, возникающих, например, в некоторых методах рисования графов.

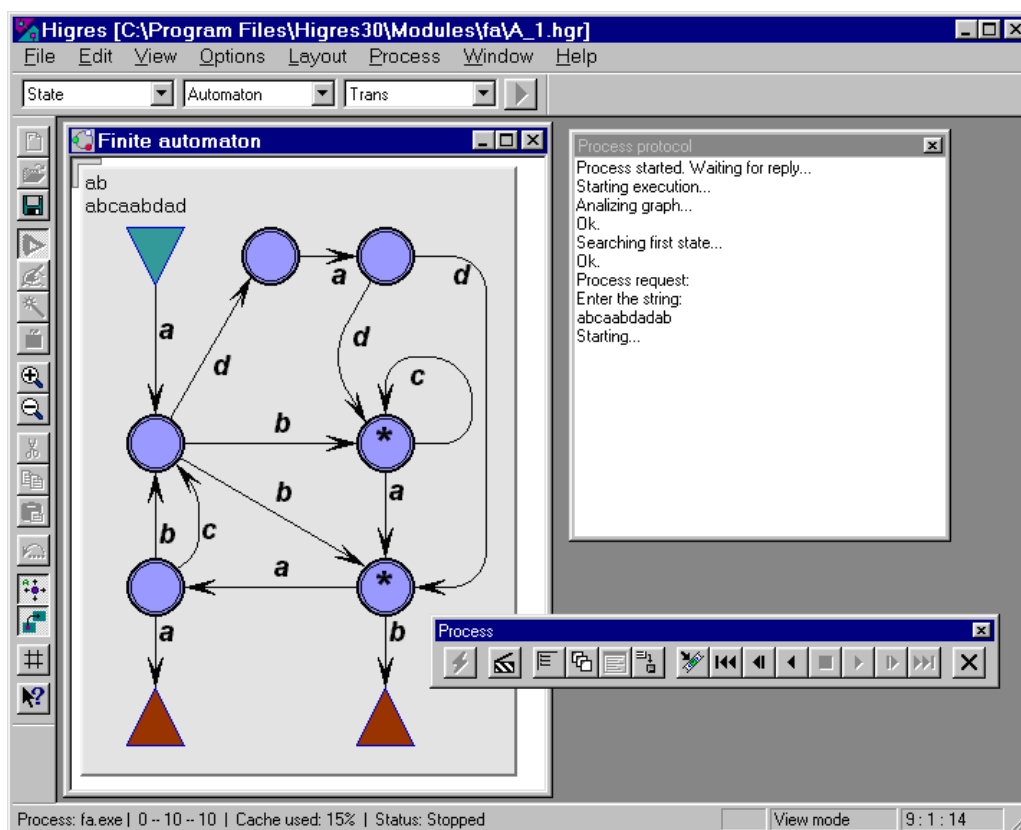


Рис. 5. Система Higes: анимация работы конечного автомата

Разработаны новые методы и алгоритмы визуализации информации на основе графовых моделей, в том числе сравнения сложноорганизованных изображений, и разработана экспериментальная версия системы визуализации атрибутированных иерархических графов Visual Graph [11] (рис. 6). Система работает под управлением ОС Windows, Linux и MacOS, поддерживает обработку произвольных атрибутированных иерархических графов (в том числе, составных и кластерных графов), ориентирована на визуализацию структур данных, возникающих в компиляторах, и позволяет одновременно работать с ними как в графовой, так и в текстовой форме. Она предоставляет богатые возможности для навигации по графовой модели, работы с атрибутами ее элементов, а также настройки системы на нужды конкретного пользователя, использует для спецификации входного (визуализируемого) графа стандартный язык описания графов GraphML и обеспечивает плавность выполнения основных операций над графами, содержащими до 100 000 элементов (вершин и дуг).

4. Система минимизации компиляторных тестов. Тенденция развития программирования состоит в том, что все более разнообразные процессы обработки программ и данных и все в большей степени поддерживаются машиной. Большинство из этих процессов обработки программ и данных реализуется в существующих инструментах как текстовые или языковые, но является семантическими. В них, как правило, требуется сохранение некоторого инварианта, определенным образом связанного с семантикой обрабатываемых объектов (например, трансляция и другие функционально эквивалентные преобразования программ сохраняют функцию, реализуемую программой). Поэтому без

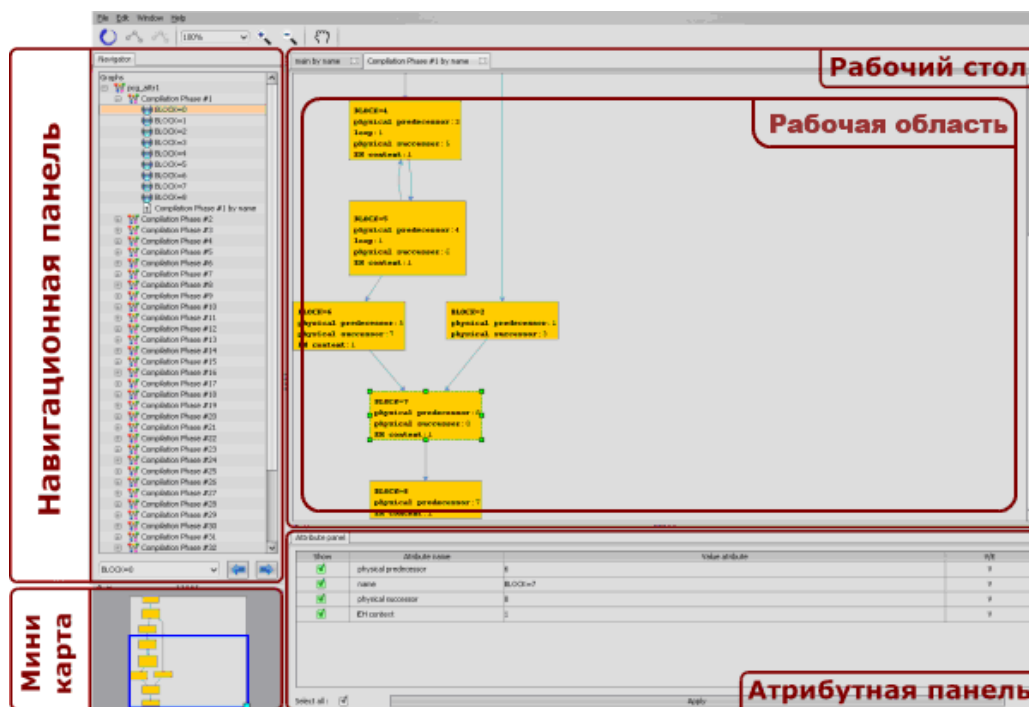


Рис. 6. Система VisualGraph

всестороннего изучения и глубокого использования в инструментальных системах семантических преобразований нельзя достичь ни надежного, ни эффективного решения задач автоматизации программирования, перейти от кустарного производства программ к технологии и массовому производству.

Трансформационный подход трактует программирование как систематическое применение фундаментальных процессов семантической обработки программ, сохраняющих при преобразовании программы определенный ее семантический инвариант и образующих в совокупности „сумму технологий“. Трансформационные методы используются в качестве основного средства для достижения эффективности при автоматизации программирования методами трансляции, особенно в связи с появлением вычислительных систем новых архитектур. Они являются перспективным направлением в создании новых, более мощных средств автоматизации конструирования эффективных и надежных программ [12].

В плане развития методов трансформационного программирования исследовалась задача применения редуцирующих преобразований для упрощения императивных программ с сохранением их семантических свойств, выявляемых компилятором в качестве ошибок. Эти ошибки могут проявляться как на стадии построения программы компилятором, так и во время исполнения построенной программы (например, как разница, наблюдаемая в программах, полученных компилятором с использованием оптимизаций и без).

Создана экспериментальная версия системы Reduce для минимизации компиляторных тестов, являющихся C/C++ и Fortran программами. Система Reduce поддерживает расширяемый набор упрощающих преобразований, ориентированных на минимизацию компиляторных тестов с сохранением воспроизводимости ошибок. Эти редуцирующие преобразования выполняются системой на внутреннем представлении программтестов в виде так называемого гибридного абстрактного синтаксического дерева (AST). В отличие от

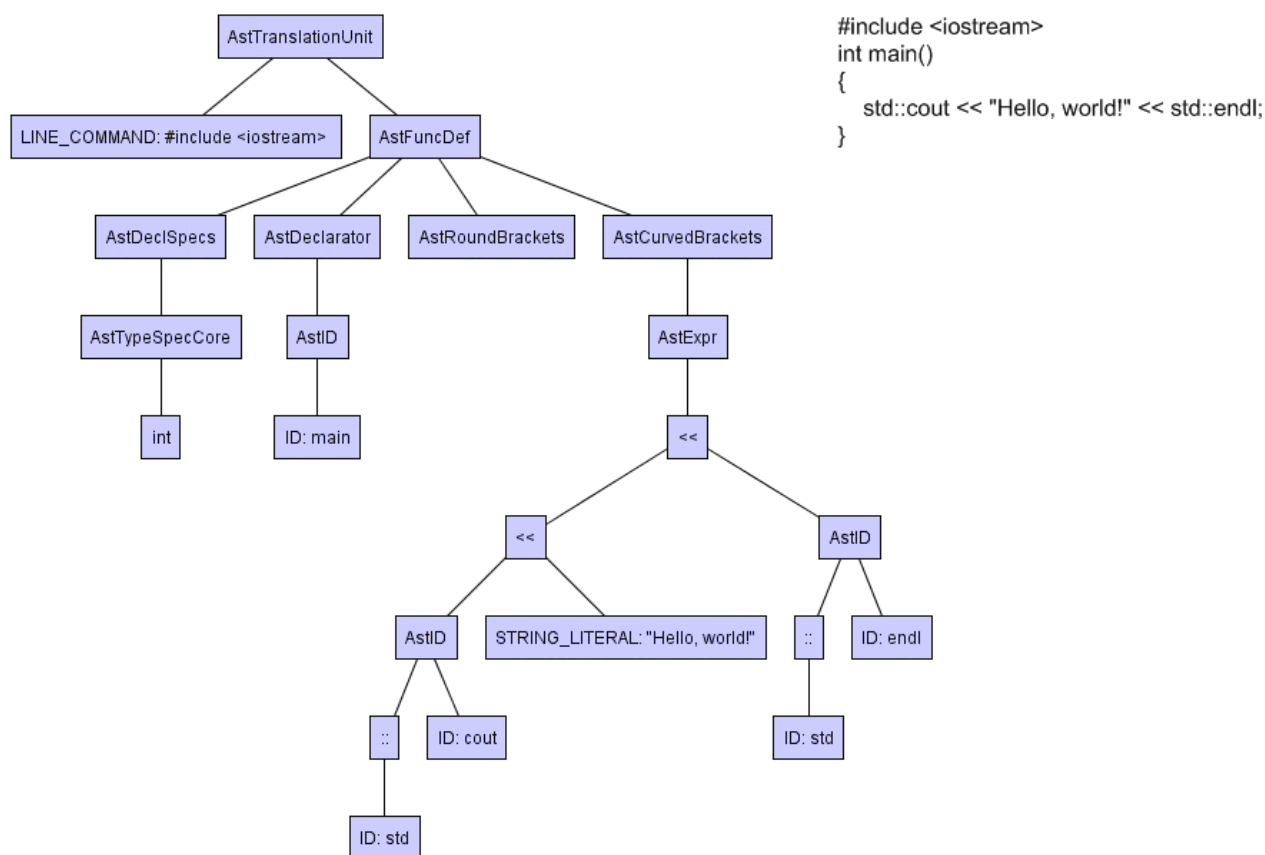


Рис. 7. Пример AST-представления программы

обычного синтаксического дерева, в гибридном дереве те части программы, которые заведомо не будут преобразовываться, могут не раскрываться в виде поддеревьев, а оставаться в виде текстовых вершин (рис. 7).

Разработан язык описания редуцирующих преобразований и стратегий их применения, на котором записана используемая системой Reduce стандартная система редуций. У пользователя системы Reduce есть возможность самому как задавать свою систему редуций, так и настраивать существующую, в том числе как за счет расширения набора редуцирующих преобразований, составляющих систему редуций, так и путем изменения стратегии их применения. Высокий уровень языка описания системы редуций и возможность визуализации AST-представления преобразуемых программ облегчают этот процесс.

5. Система функционального программирования для поддержки облачных супервычислений. Параллельные вычисления в настоящее время являются одной из главных парадигм современного программирования и охватывают чрезвычайно широкий круг вопросов разработки программ. Ввиду значительно более сложной природы параллельных вычислений по сравнению с последовательными большое значение приобретают методы автоматизации разработки параллельного программного обеспечения, основанные на применении техники формальных моделей, спецификаций и преобразований параллельных программ.

Используя традиционные языки и методы, очень трудно разработать высококачественное, переносимое программное обеспечение для параллельных компьютеров. В частности,

параллельное программное обеспечение не может быть разработано с малыми затратами на последовательных компьютерах и потом перенесено на параллельные вычислительные системы без существенного переписывания и отладки. Поэтому высококачественное параллельное программное обеспечение может разрабатываться только небольшим кругом специалистов, имеющих прямой доступ к дорогостоящему оборудованию.

Однако, используя языки программирования с неявным параллелизмом, такие как функциональный язык Sisal [13, 14], можно преодолеть этот барьер и предоставить широкому кругу прикладных программистов, не имеющих достаточного доступа к параллельным вычислительным системам, но являющихся специалистами в своих прикладных областях, возможность быстрой разработки высококачественных переносимых параллельных алгоритмов на своем рабочем месте. Функциональная семантика языков программирования с неявным параллелизмом гарантирует детерминированные результаты для параллельной и последовательной реализации: то, что невозможно гарантировать для традиционных языков, подобных языку Фортран. Пропадает необходимость переписывания исходного кода при переносе его с одного компьютера на другой. Гарантировано, что программа с неявным параллелизмом, правильно исполняющаяся на персональном компьютере, будет давать те же результаты при ее исполнении на высокоскоростном параллельном или распределенном вычислителе. Более того, по сравнению с императивными языками (подобными языку Фортран), функциональные языки, такие как Sisal, упрощают работу программисту. В функциональной программе программист должен только специфицировать результаты вычислений и может переложить большую часть работ по их организации на компилятор, который отвечает за отображение алгоритма на определенную архитектуру вычислителя (включая планирование команд, передачу данных, синхронизацию вычислений, управление памятью и т. д.). По сравнению с другими функциональными языками, язык Sisal [13, 14] поддерживает типы данных и операторы, присущие научным вычислениям, такие как циклы и массивы.

Создаваемая облачная интегрированная визуальная среда параллельного программирования базируется на языке Sisal и использует при своей работе внутреннее теоретико-графовое представление функциональных и параллельных программ. Цель проекта — дать возможность широкому кругу лиц, находящихся в удаленных населенных пунктах или в местах с недостаточными вычислительными средствами, разрабатывать параллельные программы на доступных персональных компьютерах, имеющих выход в Интернет, и оперативно дистанционно использовать для их исполнения вычислительные мощности, сосредоточенные в крупных вычислительных центрах. Создаваемая среда должна любому пользователю, имеющему выход в Интернет, позволить без установки дополнительного программного обеспечения на своем рабочем месте в визуальном стиле создавать и отлаживать переносимые параллельные программы на языке Cloud Sisal, а также в облаке осуществлять эффективное решение своих задач, исполняя на некотором супервычислителе, доступном ему по сети, созданные и отлаженные переносимые Cloud-Sisal-программы, предварительно адаптировав их под используемый супервычислитель с помощью облачного оптимизирующего кросс-компилятора, предоставляемого средой.

Используемый средой язык IR1 для внутреннего представления Cloud-Sisal-программ является языком атрибутированных иерархических графов, построенных из простых и составных (вычислительных) вершин, дуг, портов и типов (рис. 8). Вершины соответствуют вычислениям. Простые вычислительные вершины являются вершинами основного графа и обозначают литералы или операции, такие как сложение или деление. Составные вер-

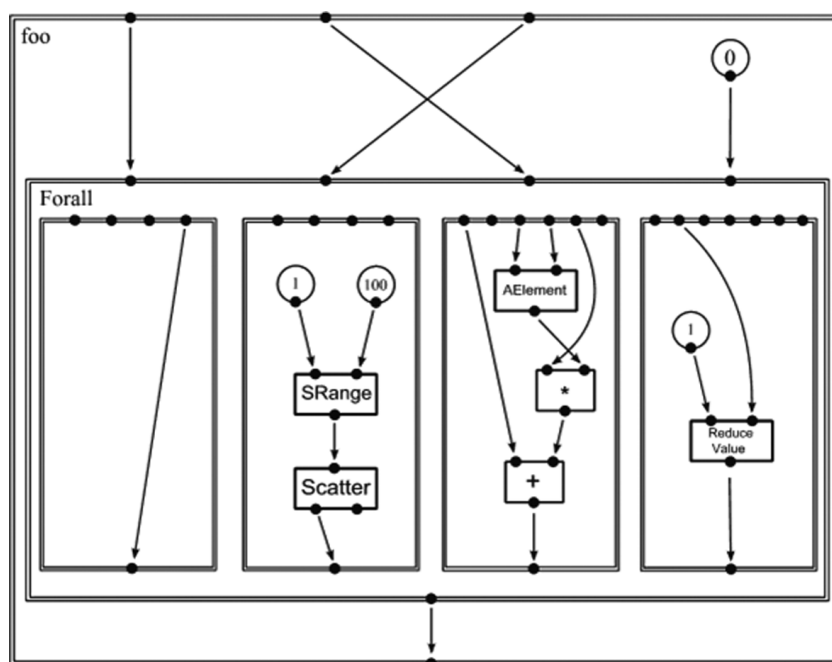


Рис. 8. IR1-представление функции foo

шины являются фрагментами (или подграфами) основного графа и представляют составные конструкции, такие как структурные выражения и циклы. Порты — это другой тип вершин основного графа, которые используются для изображения операндов вычислений. Они делятся на входные порты (входы) и выходные порты (выходы). Дуги соединяют порты и изображают передачу значений от одного операнда к другому. Они идут от выходов к входам вершин, содержащихся в одном и том же фрагменте, либо соединяют выходы вершин или фрагментов с выходами фрагментов, непосредственно их содержащих, или входы фрагментов с входами фрагментов или вершин, в них непосредственно вложенных. Типы — это атрибуты дуг, которые представляют типы значений, передаваемых по этим дугам.

Внутреннее представление Cloud-Sisal-программ используется как интерпретатором (и отладчиком) среды на этапе конструирования (и отладки) переносимой параллельной программы, так и компилятором на этапе настройки построенной переносимой программы на конкретный параллельный вычислитель. Средства визуализации внутреннего представления программ позволяют пользователю получать изображение структуры программы и отслеживать процессы вычисления и преобразования программы с помощью графических образов. Такой подход позволяет повысить эффективность процессов отладки и оптимизирующей компиляции Cloud-Sisal-программ, а также степень их понимания пользователями сервиса.

Список литературы

1. ЕРШОВ А. П. Введение в теоретическое программирование (беседы о методе). М.: Наука, 1977.
2. КАСЬЯНОВ В. Н. Ершов и графы в программировании // Андрей Петрович Ершов: ученый и человек. Новосибирск: Изд-во СО РАН, 2006. С. 150–157.

3. Евстигнеев В. А., Касьянов В. Н. Толковый словарь по теории графов в информатике и программировании. Новосибирск: Наука, 1999.
4. Евстигнеев В. А., Касьянов В. Н. Словарь по графам в информатике. Новосибирск: Сибирское Научное Издательство, 2009.
5. Касьянов В. Н., Евстигнеев В. А. Графы в программировании: обработка, визуализация и применение. СПб.: БХВ-Петербург, 2003.
6. DI BATTISTA G., EADES P., TAMASSIA R., TOLLIS I. G. Graph Drawing: Algorithms for Visualization of Graphs. Prentice Hall, 1999.
7. HERMAN I., MELANCON G., MARSHALL M. S. Graph visualization and navigation in information visualization: a survey // IEEE Trans. on Visualization and Computer Graphics. 2000. Vol. 6. P. 24–43.
8. Касьянов В. Н., Касьянова Е. В. Визуализация информации на основе графовых моделей // Научная визуализация. 2014. Т. 6. № 1. С. 31–50.
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. LISITSYN I. A., KASYANOV V. N. Higraphs visualization system for clustered graphs and graph algorithms // Lecture Notices in Computer Science. 1999. Vol. 1731. P. 82–89.
11. KASYANOV V. N., KASYANOVA E. V., ZOLOTUHIN T. A. Information visualization based on hierarchical graph models // International conference „Advanced mathematics, computations and applications 2014“. Abstracts. Novosibirsk: Academizdat, 2014. P. 48.
12. KASYANOV V. N. Transformational approach to program concretization // Theoretical Computer Science. 1991. Vol. 90. N 1. P. 37–46.
13. GAUDIOT J.-L., DEBONI T., FEO J., et al. The Sisal project: real world functional programming // Lecture Notices in Computer Science. 2001. Vol. 1808. P. 84–72.
14. Касьянов В. Н., Стасенко А. П. Язык программирования Sisal 3.2 // Методы и инструменты конструирования программ. Новосибирск: ИСИ СО РАН, 2007. С. 56–134.

*Виктор Николаевич Касьянов — д-р физ.-мат. наук, профессор,
главн. науч. сотрудник ИСИ СО РАН,
профессор НГУ, Новосибирск, Россия,
E-mail: kvn@iis.nsk.su*

*Елена Викторовна Касьянова — канд. физ.-мат. наук, доцент,
старш. науч. сотрудник ИСИ СО РАН, доцент НГУ,
E-mail: kev@iis.nsk.su*

Дата поступления — 22.10.2015