МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. М.В. Ломоносова

ТРУДЫ ФАКУЛЬТЕТА
ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

№ 2

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

Под редакцией
чл.- корр. РАН Л.Н. Королева

Москва
2001

УДК 519.6+517.958
ББК 22.19
П75

Печатается по решению Ученого Совета факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова

П75 **Программные системы и инструменты**: Тематический сборник факультета ВМиК МГУ/ Под. ред. Л.Н. Королева. – М: Издательский отдел факультета ВМиК МГУ, 2001. – Вып. 2. – 214 с.

ISBN 5-89407-115-1

В данный сборник включены научные работы и сообщения по следующим темам:

- общие вопросы программирования и информатики;
- исследование алгоритмов построения решений;
- применение генетических алгоритмов и теории нечетких множеств;
- вопросы троичной логики.

В этих публикациях нашли отражение исследования и разработки в области создания программных систем, выполненных учеными, аспирантами и студентами факультета. Большая часть результатов доложена на Ломоносовских Чтениях 2001 года.

Для студентов, аспирантов и специалистов в области разработки прикладных программных систем с использованием новых информационных технологий.

УДК 519.6+517.958
ББК 22.19

ISBN 5-89407-115-1

СОДЕРЖАНИЕ

СБОРНИК

" Программные системы и инструменты"

От редактора

В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные в основном описаниям инструментальных программных систем, разработанных авторами публикаций.

В нем также публикуются статьи и заметки общего характера, касающиеся разделов информатики, тесно связанных с программированием.

Большинство статей представляют собой тексты докладов, прочитанных на Ломоносовских Чтениях 2001 на факультете ВМиК.

Перечень разделов сборника таков:

- общие вопросы программирования и информатики;
- нечеткие множества, генетические алгоритмы и троичная логика;
- прикладные программные системы;
- сообщения.

Статьи сборника будут интересны специалистам, разрабатывающим прикладные программные системы и исползующим новые информационные технологии.

Л.Н. Королев

Раздел I
Общие вопросы программирования и информатики

Королев Л. Н.

Архитектура и программирование (размышления)

CISC, RISC, VLIW, EPIC, DFC архитектуры и их связь с программированием.

Проектировщики вычислительных систем сейчас и раньше стремились достичь одновременно двух целей:

1. из имеющейся в наличии элементной базы построить вычислитель, с наилучшим соотношением «стоимость/производительность», для данного класса потенциальных пользователей.
2. сделать машину удобной для программирования.

Первые экземпляры вычислительной техники у нас и за рубежом ориентировались на узкий класс пользователей, в основном физиков и ракетчиков, создававших атомное оружие и средства его доставки. Главное было достичь, не обращая внимание на затраты, по возможности высокой производительности и приемлемой надежности.

К этому типу по существу уникальных машин у нас следует отнести «Стрелу» и «БЭСМ-1».

Роль математиков, обслуживавших физиков и ракетчиков, на этапе проектирования этих машин, состояла в определении минимально необходимой разрядности машинного слова для хранения операндов, с учетом требуемой точности результатов вычислений, и в предложениях по кодировке полей машинных команд, что можно назвать разработкой машинного языка.

Как сейчас представляется, 50 годы явились началом новой ветви математики, получившей название прикладной математики, тесно связанной с понятием построения алгоритмов, доступных для реализации на вычислительной машине, работающей, вообще говоря, как бездумный автомат - исполнитель чужой воли.

Потребовались новые подходы к теоретическим исследованиям в области численных методов в условиях ограничений разрядной сетки реальных машин, потребовалось исследование феномена программы, совершенно нового объекта в математике, со своими правилами их взаимодействия, их формального преобразования, изучения их эквивалентности и сложности. Потребовалась разработка новых эффективных алгоритмов и доказательства их работоспособности,

правильности полученных результатов за приемлемое время. Именно, на основе этих исследований математиками предлагались и обосновывались требуемые характеристики проектируемых вычислительных машин – ЭВМ.

В начале 50 годов под грифом секретно вышла замечательная книга, в которой описаны многое оригинальные вычислительные алгоритмы, пригодные для реализации на ЭВМ. Среди ее авторов были такие выдающиеся математики как М. Р. Шура-Бура, А. А. Абрамов, М. А Люстерник и др. [4]. Эта книга оказала большое влияние на развитие программирования в СССР, сформировав взгляд на эту отрасль знаний, как на новую часть математики вообще, и прикладной математики в частности.

«Удобство» программирования, которое учитывается при проектировании ЭВМ, понятие достаточно расплывчатое, смысл которого значительно изменялся со временем, так же как и само понятие программирования.

В начале на этапе программирования в машинных кодах под удобством программирования понималась хорошая мнемоника машинного языка в 8-ричном или 16-ричном цифровом представлении, в зависимости от перфорационного оборудования.

С появлением алфавитно-цифровых перфораторов и первых языков символьного кодирования (автокодов, ассемблеров) роль внутренней кодировки перестала играть существенную роль. Приобрели значение запоминающаяся мнемоника записи смысла операций, символика задания имен (символических адресов) переменных. Синтаксис первых машинно-независимых языков типа Фортран и Кобол явно несет отпечатки синтаксиса автокодов.

Первым машинно-независимым языком с синтаксисом и семантикой, приближенным к математической нотации, по-видимому, следует считать Алгол-60, изначально предназначавшийся для записи и публикации «математических» алгоритмов. В Советском Союзе он в свое время был возведен в ранг стандарта.

Первые трансляторы с Алгола-60 были созданы в СССР, что явилось крупным достижением того времени. Эти две конкурировавших разработки были выполнены под руководством и непосредственном участии выдающихся математиков, М. Р. Шуры-Буры и С. С. Лаврова Какое-то время Алгола-60 соперничал с прагматическим языком Фортран, облюбованным физиками.

Свойства блочно и процедурно ориентированных языков типа Алгол нашли свое отражение в архитектурах некоторых машин, а именно в машинах фирмы Барроуз (Burrous) и в отечественных машинах серии Эльбрус. Это пример влияния языкотворчества на архитектуру ЭВМ, выразившейся в аппаратной поддержке таких удобных средств программирования, как рекурсии и процедуры.

Хотя архитектура типа Барроуз не получила дальнейшего развития в коммерческих разработках, идеи, реализованные в Барроуз, нашли свое отражение в современных процессорах, в которых хорошо развит стековый аппарат и тегирование, составляющие основу машин, ориентированных на реализацию быстрой трансляции с языков типа Алгол.

ЭВМ начала 50 годов, как уже было сказано, были созданы на потребу военно-промышленных комплексов и на поддержку научных исследований так или иначе связанных с ними. Не смотря на это, на этих машинах проводились экспериментальные работы инициативного характера, носившие чисто научный, академический характер, никак не связанный с решением серьезных «государственных» задач.

В частности, на БЭМС-1 проводились эксперименты по автоматическому переводу с английского языка на русский. В составе коллектива, работавшего под руководством Д. Ю. Панова и при кураторстве зам. директора ИТМиВТ И. С. Мухина над проблемами машинного перевода довелось работать и автору доклада. Автором был предложен и реализован механизм, получивший потом название «хеширования», для ускорения поиска по словарю [1].

Чисто академический интерес представляли первые работы по символьному дифференцированию, выполненные при участии и руководстве автора на той же машине БЭСМ-1 [2], совсем не предназначенной для решения задач подобного типа. В БЭСМ-1 отсутствовали как таковые операции символьной обработки, была только арифметика с плавающей запятой и минимальный набор поразрядных логических операций.

Подобного рода эксперименты проводились и на других машинах у нас и за рубежом. Они послужили основанием создания вычислительных машин со сложной системой команд, машин CISC-архитектуры, в которых аппаратно поддерживается большое разнообразие типов данных и операций над ними, включая манипуляции с цепочками кодов символов, десятичную, с плавающей запятой и целочисленную арифметику. Благо, достижения в электроники 60-70 годов позволяли это сделать.

Обусловленная коммерческими и научными интересами погоня за универсальностью ЭВМ, в смысле обработки как можно более широкого спектра типов данных, естественно привела к резкому усложнению системы команд и логики работы аппаратуры, и вошло в противоречие с «инженерной» целью – добиться максимальной производительности аппаратуры при решении любых задач. Логическую сложность машины характеризует число различных команд. В системе IBM 360/370, например, это число приближалось к 300.

В конце 50, начале 60 годов прошлого столетия число различных программно несовместимых типов машин разных больших и малых фирм исчислялось сотнями.

Возникла и, в некоторых проектах была реализована, идея использовать микропрограммирование для создания универсальной машины, способной настраиваться на любой лад, т. е. способной имитировать работу в любой системе команд.

Однако, наиболее мощная фирма IBM приступила к выпуску своего ряда программно-преемственных машин, который, как казалось, снимал проблему медленной и дорогой разработки системного программного обеспечения для новых поколений машин и не требовала от уже многочисленных пользователей переобучения в работе на новых установках.

Сама по себе благородная идея как можно меньше тревожить массового пользователя в значительной мере задержала развитие новых идей в развитии архитектур систем обработки данных. Если посмотреть на процесс развития архитектуры микропроцессоров, на примере фирмы Intel, то станет очевидным ее стремление оставаться на позициях архитектуры со сложной системой команд, несмотря на то, что в RISC-архитектуре проще и эффективнее реализовать суперскалярную (параллельную) обработку данных, проще организовать конвейер потока команд.

В семействе Pentium-6 и процессоров AMD нашли выход в многоступенчатой дешифрации исходных сложных команд, в результате которой получаются команды, свойственные RISC-архитектуре. По существу, микропрограммными средствами производится преобразование исходной системы команд во внутренний язык RISC машины.

Это привело к необычайному усложнению процессора, достаточно сказать, что число логически активных элементов (диодов, транзисторов) на одном чипе перевалило за 20 миллионов. Вспомним, что в БЭСМ-6 их число не превышало миллиона.

Это немыслимое усложнение оправдывается тем, что при переходе на более производительные высокочастотные процессоры не надо переделывать системные программы. Так ли это?

Следует отметить, что консервативная идея сохранить стиль работы пользователя, не выдержала испытание временем. Пользователю непрерывно приходится доучиваться или переучиваться, если он хочет воспользоваться новыми мультимедийными возможностями, сетевым взаимодействием, компьютерной телефонией и другими новейшими услугами компьютерных систем.

Разработка программного обеспечения для машин CISC-архитектуры, не смотря на богатство средств машинного языка, среди команд которого можно выбрать наиболее подходящую для

соответствующей цели, на самом деле усложнило работу системных программистов, т. к. потребовала от них глубокого знания многочисленных архитектурных тонкостей.

Многими программистами была высоко оценена прозрачная и логически простая система команд отечественной машины БЭСМ-6, архитектура которой отражала в какой-то степени гениальный стиль мышления ее главного конструктора, выдающегося ученого, патриарха отечественной вычислительной техники академика Сергея Алексеевича Лебедева.

Огромный успех этой машины среди пользователей, обязан простоте ее логического построения, простоте ее архитектуры. Автор гордится тем, что принимал непосредственное участие в разработке машины БЭСМ-6, и считает себя одним из многих учеников Сергея Алексеевича Лебедева.

Но, усложнение работы программистов, в большей степени связано с другими обстоятельствами, с лавинообразным ростом требований приложений вычислительной техники. Соответственно, программное обеспечение становиться все более сложным и ресурсоемким.

Это резкое усложнение логики работы системного программного обеспечения, прежде всего операционных систем, вызванное необходимостью добавления новых средств взаимодействия с пользователем и прикладными системами, приводит к резкому повышению накладных расходов. В результате компьютер все большую часть времени работает, как бы сам на себя и все меньшую часть времени занят решением задач. Здесь в полную силу работает Закон Паркинсона.

Не получается ли так, что усложнение процессора, с целью добиться повышения производительности, съедается непомерно раздутой сферой обслуживания пользовательского интерфейса в операционных системах?

При этом следует отметить, современные операционные системы стараются строить по принципу конструктора, что в принципе позволяет их настраивать под нужды конкретного пользователя, не вводя в их состав не нужных ему прибамбасов, и тем самым уменьшать накладные расходы на самообслуживание. Однако, исследований проблемы накладных расходов, как кажется, толком никто не проводил. А надо бы!

Из вышесказанного следует заключить, что «внешняя оболочка» CISC-архитектуры будет долго здравствовать особенно в сфере компьютеров массового производства и «бытового» использования.

Процессоры RISC-архитектуры, такие как Alpha-DEC, POWER PC, SUN-SPARC, HP-PA, хотя и используются в компьютерах

массового применения, оказались в большей степени ориентированы на сферу применения в научных исследованиях, решения академических задач, в качестве серверов в компьютерных сетях.

Это разделение сфер влияния RISC- и CISC-архитектуры повлияло на разделение усилий разработчиков программного обеспечения. Огрубляя, можно говорить о UNIX (RISC) - ориентированных и WINDOWS (CISC)-ориентированных программистах.

Остается заманчивой перспектива использования VLIW и EPIC архитектур для процессоров со многими функциональными устройствами. На самом деле, в современных суперскалярах на внутренних стадиях конвейера потока команд что-то подобное делается. В динамике работы из пула выбирается группа независимых по данным команд и одновременно передается на обработку соответствующим функциональным устройствам. В трансляторы вводятся дополнения, которые повышают эффективность данного процесса. Первоначальная идея разработчиков полностью переложить VLIW-распараллеливание на плечи анализатора статического текста программы, написанной на языке высокого уровня, вряд ли даст желаемый результат получения максимальной производительности VLIW-систем.

Успеха следует ожидать, комбинируя статический анализ и динамическую корректировку в ходе исполнения программы.

Появление процессоров VLIW и EPIC типов, ставит перед системными программистами новые сложные задачи по разработке высоко интеллектуальных трансляторов нового поколения. В нашей стране разработкой процессоров типа VLIW архитектуры успешно занимается коллектив под руководством чл.-корр. РАН Б. А. Бабаяна.

Однако, мало вероятно, что в обозримом будущем процессоры для машин массового спроса будут VLIW-системами, в этом нет настоятельной необходимости.

Объектно-ориентированная парадигма программирования – существо или мода.

В настоящее время обо всех новых программных системах почти обязательно говорят, что они объектно-ориентированы, просто по той причине, что система написана с использованием языка C++. Тем самым девальвируется смысл этого нового знаменательного подхода к созданию программного продукта.

При разработке систем управления в реальном масштабе времени, при создании систем моделирования, создании операционных систем и программ машинной анимационной графики, их разработчикам стало ясно, что вычислительная машина должна уметь

оперировать с объектами совсем иной природы, не похожими на числовые массивы и логические переменные.

Думается, что это понимание возникло гораздо раньше красивого термина объектно-ориентированной парадигмы. Достаточно вспомнить систему MULTICS MIT, язык Simula-67, да и язык PL, принятый в США в качестве стандарта для разработки программ военного назначения.

В конце 60 годов автором доклада в препринтном издании ИТМиВТ была опубликована статья [3], в которой классифицировались объекты операционной системы, такие как память, внешние устройства и т. д., рассматривались операции над этими объектами, и высказывалась пожелание создать специальный язык описания работы с этими объектами. Это, конечно, нельзя считать провозглашением объектно-ориентированного подхода к проектированию операционных систем, но это характеризует то, что в недрах практических разработок требования к такому подходу возникли.

Представляется, что суть объектно-ориентированного подхода в программировании состоит в построении иерархии уровней описания взаимодействия сущностей, отражающих реальный мир. Частным случаем этих сущностей можно считать программные процессы в их взаимодействии в процессе их жизненного цикла.

Филосовски рассуждая, можно также сказать, что все сущности окружающей нас природы, выражаются иерархией понятий и их взаимосвязью. В этом смысле объектно-ориентированный подход есть попытка прямого отражения пнятийной структуры восприятия окружающей действительности на уровень возможностей компьютера.

Практически же объектно-ориентированный подход (ООП) к конструированию программ просто полезен, т. к. упрощает процесс программирования, при условии, что создана соответствующая хорошая библиотека классов. Правда, процесс создания соответствующих библиотек трудоемкая вещь, требующая высокой квалификации.

С прагматической точки зрения ООП можно рассматривать, как развитие старой идеи создания универсального инструмента для разработки платформенно-независимого программного обеспечения, реализация которой началась с создания машинно-независимых алгоритмических языков. Эти языки всегда были ориентированы на вполне определенный круг задач: либо вычислительных, либо логических, либо информационно- поисковых. У всех этих языков был предопределенный класс объектов, с которыми они могли оперировать. Круг задач, подлежащих решению на ЭВМ, непрерывно расширялся, и возникла прямая необходимость задавать средствами одного и того же языка новые объекты и операции над ними. Так возникли языки, допускающие формирование абстрактных типов данных (АТП), и, наконец, объектно-ориентированные языки типа C_{++}.

Смысл вышесказанного сводится к тому, что не следует злоупотреблять термином объектно-ориентированного подхода в отношении даже очень современных и полезных программных систем. Некоторые разработчики прикладных программ утверждают к тому же, что программы, написанные просто на языке С или Фортране, часто работают заметно эффективнее чем аналогичные программы , написанные с использованием языка C_{++} или JAVA.

В нашей стране несомненным лидером в развитии объектно-ориентированного подхода в конструировании программ является Институт Системного Программирования, возглавляемый членом-корреспондентом РАН В. П. Иванниковым, который также заведует кафедрой Системного Программирования (СП) нашего факультета ВМиК.

Что такое программирование сегодня и что будет завтра?

Как бы ни определять понятие «программирование» смысл этого рода занятия состоит в преобразовании алгоритма в форму, принимаемую машиной для исполнения последовательности действий (или одного действия), указанных в алгоритме.

В общем определении алгоритма значатся свойства массовости, однозначности понимания указанных в нем действий исполнителем, гарантированности получения результата за конечное время (реализуемости).

Когда под исполнителем понимается вычислительная машина, оснащенная программным обеспечением, то перечень действий, которые она способна выполнять по объему и содержанию многократно превосходит перечень действий, выполняемых аппаратурой, выполняемых машинными командами.

В самом начале появления ЭВМ программирование сводилось к искусству преобразования алгоритма, сформулированного как подробная схема решения той или иной задачи, в последовательность машинных команд. Такой этап программирования в машинных кодах продолжался очень не долго, до половины 50 годов. Программирование на языках символьного кодирования, на автокодах и ассемблерах, заменившее его, упростило процесс преобразования алгоритма в машинный код, оставив существо дела без изменения. Самой машине удалось передать значительный объем технической работы по получению двоичного кода программы. Ассемблеры сами по себе не увеличили перечень действий выполняемых аппаратурой.

Бурное расширение перечня действий, воспринимаемых машиной как исполнителем алгоритмов, началось с появлением

библиотечных систем, поддержанных аппаратными средствами обращения к подпрограммам. В арсенал «неделимых» действий машины были включены такие операции, как вычисления элементарных функций, обращение матриц, решение систем алгебраических и дифференциальных уравнений и т. д. Соответственно, алгоритмы можно было формулировать в более емких по семантике терминах, в более емких операторах.

В нашей стране одним из пионеров создания библиотечных систем являлся профессор факультета Е. А. Жоголев.

Огромен вклад в развитие программирования и в обучение этой новой специальности внесен профессором Н. П. Трифоновым, по книгам которого обучались тысячи студентов СССР и России.

Процесс расширения перечня действий, которые способна выполнять машина, продолжается, с еще большей интенсивностью, и по сей день.

Принято делить программирование на теоретическое, системное и прикладное. В нашей стране работы мирового класса по теоретическому программированию проводились, начиная с 50 годов, и первопроходцами в этом направлении, смыкающимся с теорией алгоритмов, были такие математики, как А. А. Ляпунов, Ю. И. Янов и многие другие. На нашем факультете эти работы продолжаются под руководством профессора Р. И. Подловченко.

Что касается системного программирования его основная задача как раз и состоит в расширении перечня действий, которые можно заставить выполнять машину. Иными словами, системное программирование призвано непрерывно увеличивать возможности машин, как исполнителей алгоритмов.

Прикладное программирование призвано использовать эти все возрастающие возможности исполнителя для решения новых практических задач, непрерывно возникающих в необъятной сфере применения компьютеров охватывающей науку, производство, коммерцию и быт. Если прикладной программист создал хорошую программу, которую могут использовать другие специалисты, то и она может поступить в копилку средств, увеличивающих возможности компьютера, увеличивающих его интеллектуальные возможности (иногда это называют и так). Поэтому, четкой грани между прикладным и системным программирование не существует.

Процесс непрерывного увеличения интеллектуальных возможностей компьютеров приводит, и в значительной степени уже привел, к тому, что программирование с использованием языков высокого уровне для большинства людей, работающих на компьютере, становиться ненужным. В самом деле, лица, сидящие в офисах, операторы в банках, бухгалтера на предприятиях, менеджеры, брокеры и т. д., не должны ничего программировать. Если такая необходимость

возникает, то это просто означает, что поставленная для них программная система не годится для использования и надо купить или заказать иную.

Для абсолютного большинства людей, использующих компьютер на работе или дома, первоначальное понятие алгоритма, как описания последовательности действий для достижения результата, превратилось в описание цели задачи, превратилось в последовательность нажатия кнопок на клавиатуре и манипуляции мышкой. Эти правила игры на клавиатуре и правила передвижения стрелки указателя, конечно можно также назвать алгоритмом, но его смысл состоит в формулировке задания на выполнение работы, без указания как ее надо выполнить.

В 70 годы появился лозунг «всеобщей компьютерной грамотности», предполагавший компьютерный всеобуч, начиная со школьной скамьи. Неотъемлемой частью этой грамотности предполагалось освоение учащимися какого либо языка программирования, например Бейсика.

Но многим сотням миллионов людей, пользующихся компьютерами сегодня, совсем не надо осваивать языки программирования. Им надо нечто другое, уметь пользоваться электронной почтой, отыскивать нужную информацию в базах данных, пользоваться средствами дистанционного обучения и т. п. Каково должно быть наполнение понятия компьютерной грамотности сегодня, тема особого разговора.

Упрощение жизни обычным пользователям приводит в свою очередь к усложнению задач, решаемых усилиями системных программистов.

Для того, чтобы создать хорошую полезную вещь необходим надлежащий набор инструментов. Инструментами для системных программистов являются прежде всего языки программирования высокого и более чем высокого уровня.

Градацию высот уровней языков программирования точно определить трудно. Но если сам язык рассматривать как исполнитель алгоритмов, как некую виртуальную машину, то по уровню семантики, обозначенных в нем неделимых действий, как-то можно судить о его ранге по высоте.

В процессе создания инструментов для системных программистов, их же усилиями, стремятся достичь нескольких целей, для которых нужны разные языки и инструментальные системы.

Необходимы языки спецификаций, позволяющие описывать требования к проекту программной системы и допускающие формальную проверку выполнения этих требований. Этот языки должны быть обращены в сторону заказчика системы, и по этому их должно быть много и они должны быть проблемно ориентированны на

область интересов заказчика. Их синтаксис и семантика должны быть понятны как заказчику, так и разработчику программной системы.

Примерами таких языков у нас можно считать «КАНТ», созданный под руководством профессора М. Р. Шуры-Буры.

Интенсивные работы в направлении создания языков спецификаций ведутся на кафедре СП, под руководством чл.-корр. РАН, профессора П. Иванникова.

В лаборатории ЛВК кафедры АСВК под руководством проф. Р. Л. Смелянского ведутся интересные исследования по разработке языка спецификаций, описывающего требования к поведению программ, работающих в системах реального времени. Предложенный язык спецификаций включает в свой состав операторы модальной логики, характеризующие временные зависимости между отдельными процессами исследуемой (или разрабатываемой) программной системы, реализующей управление сложными техническими комплексами типа бортовой аппаратуры современного самолета.

Отличительной особенностью этого языка является, принципиальная возможность транслировать программные модули, в тех случаях, когда степень детализации спецификаций это допускает. Разработаны инструментальные средства, автоматически проверяющие программную систему на непротиворечивость в отношении сформулированных спецификаций временной синхронизации.

Представляют интерес, проводимые в этой лаборатории, исследования по моделированию поведения аппаратуры и программного обеспечения в системах реального времени, что при проектировании позволяет обоснованно вносить коррективы, как в программы, так и в архитектуру аппаратного комплекса.

Необходимы языки моделирования поведения проектируемой программной системы, позволяющие описывать взаимодействия объектов, включая такие объекты как программные модули. Таких языков создано много, начиная с Симулы –67, HVDL, и других проблемно ориентированных языков. Делаются попытки создания универсальных проблемно независимых языков моделирования. К этой категории можно отнести язык UML с высоко развитой системой визуализации.

В направлении методов имитационного моделирования пионерскими работами, значение которых в анализе правильности и эффективности работы сложных объектов, как архитектура машины во взаимодействии с программами, бесспорно следует считать исследования, выполненные профессором кафедры АСВК профессором А. Н. Томилиным, на этапе проектирования машины БЭСМ-6. Эти работы были инициированы академиком С. А. Лебедевым.

Особое значение приобрела необходимость изготовления программных систем из уже готовых, хорошо отлаженных заготовок.

Большинство средств современных CASE-технологий по своему существу нацелены именно на решение данной задачи. Язык Java с его «апплетами» и «аппликейшинами» можно рассматривать, как попытку отказа от физической компоновки программной системы в единое целое, а превращение ее в распределенную, работающую на разных платформах сети, систему. Существенно при этом, что Java-объекты при этом не должны претерпевать перекодирования.

В части исследований и разработок инструментов создания программных систем, в особенности для компьютеров, встраиваемых в системы управления, интенсивные работы ведутся под руководством профессора И. В. Машечкина в лаборатории технологий программирования.

Основными работами этой лаборатории являются:

-исследование и построение системы моделирования вычислительной системы, позволяющей производить "настройку" архитектуры вычислительной системы под смесь решаемых задач;

-исследование и разработка инструментальных средств программирования специализированных ЭВМ (коллективом лаборатории разработана и внедрена система программирования для одной из специализированных ЭВМ, применяющейся в космической промышленности);

- практическое применение математического аппарата теории нечетких множеств в системах поддержки принятия решений;

- использование современных технологий программирования в построении Интернет систем в частности, сотрудниками лаборатории, с использованием передовых технологий фирм Informix и HP разработана одна из наиболее популярных в России Интернет-систем "МОЛОТОК".

Не потеряли своего значения исследования механизмов кросс программирования, выполняемые в этой лаборатории.

Системное и, главным образом, прикладное программирование все более превращаются в сборочный процесс конструирования из ранее созданных заготовок. Очевидно, что для такой конструкторской работы особое значение приобретает система стандартов, на собираемые в единое целое программные заготовки.

В деле разработки стандартов возник целый ряд научных проблем, связанных с определением методов получения новых проблемно-ориентированных стандартов, удовлетворяющих целому ряду критериев, главные из которых состоят в достижении непротиворечивости в отношении уже принятых, надежности и защищенности программных модулей, которые будут изготовляться по этим стандартам. В этой области, получившей название теории открытых систем, работает профессор кафедры АСВК В. А. Сухомлин.

Процесс создания новых заготовок, с учетом принятых стандартов, силами программистской элиты интенсивно продолжается по сей день.

Для этого необходимы «старые», испытанные языковые средства, дополненные, разумеется, новыми возможностями, такими, например, как визуализация или графические версии известных языков.

Неотъемлемой частью всех вновь разрабатываемых прикладных программных систем, являются блоки визуализации результатов обработки информации. Машинная графика, анимация, и вообще обработка изображений в этой связи приобрела характер необходимого инструмента в конструировании программного продукта. Это направление на факультете ВМиК успешно развивается в части системного программирования под руководством признанного специалиста в этой области доцента (пока еще) кафедры АСВК Ю. М. Баяковского.

В связи с вышесказанным, возникает непростой вопрос, как и чему учить системных программистов нового поколения?

Раньше ответ на вопрос чему учить программистов самый общий ответ состоял в том, что их прежде всего надо обучить алгоритмизации. Сейчас надо при обучении во главу угла ставить системное мышление. Но каким содержанием надо наполнить этот красивый лозунг? Это тема серьезных размышлений и дискуссий.

Параллельные вычислительные системы и параллельные (распределенные) программы.

Одной из актуальнейших задач системного программирования в настоящее время стала задача создания средств эффективного использования массивно-параллельных вычислительных систем и организации распределенных по сети вычислений.

Следует отметить, что теоретические рассмотрения возможности параллельных вычислений начались достаточно давно. В конце 60 годов этими исследованиями в нашей стране занималась группа в СОАН под руководством Н. Косарева и Э. Евреинова, изучавших идею создания мультипроцессорной системы, состоящей из многих сотен процессоров, каждый из которых должен был быть не больше «спичечного коробка». Правда, подтверждающие идеи эксперименты ограничились объединением двух машин Минск 22 на общую память и получением «сверх аддитивного» эффекта.

В 1975 г. вступила в эксплуатацию система ILLAC-IV в 64 процессорном варианте, и исследования в области распараллеливания стали практически необходимыми.

В настоящее время мультипроцессорные системы стали более широко доступны, как для проведения академических экспериментов, так и для решения насущных задач науки и техники.

В нашей стране появились несколько мультипроцессорных систем, пиковая производительность которых достигла терафлопного уровня. Это система МВС 1000 (главный конструктор В. А. Левин), установленная в суперкомпьютерном центре РАН, доступная для дистанционного использования институтами РАН и крупными университетами России, это и ряд других систем, предназначенных для специального использования.

В институте ИТМиВТ под руководством чл. корр РАН, профессора Г. Г. Рябова, который также преподает на кафедре АСВК, разрабатываются оригинальные по архитектуре мультипроцессорные системы высокой производительности, встраиваемые в системы управления сложными техническими объектами.

Во многих институтах и университетах устанавливаются многопроцессорные кластеры, достигающие пиковой производительности в десятки миллиардов операций в секунду. На сегодняшний день самый высокий по производительности и числу процессорных элементов университетский кластер установлен НИВЦ МГУ. При этом число процессоров в нем перманентно наращивается.

Статистика, однако, показывает, что в системах параллельной обработки при массовом счете используется меньше 20% от их пиковой производительности.

Основной вклад в это явление вносит закон Амдала, но не малая часть потерь происходит от неумения программировать, точнее, от несовершенства программного обеспечения параллельных вычислений.

С законом Амдала, по-видимому, можно бороться, изобретая новые численные методы или новые модели физических и информационных процессов.

Что касается программного обеспечения параллельных вычислений, то здесь все упирается в проблему распределения программных процессов (включая распределение данных) по процессорам вычислительной системы. При этом необходим тонкий учет архитектурных особенностей конкретных вычислительных систем, таких как отношение скоростей обмена данными и производительности процессоров, принятую топологию, способы синхронизации, кэширование и др. Эта проблема получила название отображения алгоритма на вычислительную систему.

Достаточно хорошо решение этой проблемы отображения продвинуто для машин с векторно-конвейерным параллелизмом, в том числе в результате работ отечественных ученых – академика В. В. Воеводина и выпускника кафедры АСВК, профессора Вл. В. Воеводина.

Для машин с массовым параллелизмом проблемой эффективного отображения вычислений занимаются многие коллективы у нас и за рубежом, но пока что достигнутые результаты оставляют желать лучшего.

Параллельные версии языков Фортран, C++ и др. возлагают в основном ответственность на качество распараллеливания на разработчиков метода и программы. Отображением языковых средств распараллеливания средствами PVM и MPI должно делаться операционной системой.

Заманчивые попытки распределения процессов и данных, основанные на анализе текста программы (статическое планирование), когда речь идет о настоящих, больших программах, пока не перешли на уровень практической реализации. Но эти работы продолжаются. В частности, на кафедре АСВК такие работы ведутся в рамках развития системы ПАРАЛАКС (под руководством доцента Н. Н. Попвой).

Динамическое назначение процессоров на выполнение работ (динамическое планирование), для своей реализации требует дополнительных накладных расходов и, что наиболее сложно осуществимо, выделения некоторого процессора, осуществляющего функции слежения за загруженностью всех процессоров системы. Динамическое планирование может потребовать перераспределения данных в ходе вычислений. Существует также идея использовать статическое и динамическое планирование совместно, при этом статическое планирование играет роль своего рода начального приближения. Исследования в этом направлении так же ведутся на кафедре АСВК.

Представляет интерес подход к отысканию оптимального расписания вычислений на мультипроцессорах с использованием генетических алгоритмов, развиваемый в лаб. ЛВК кафедры АСВК.

Современные машины с массовым параллелизмом и многопроцессорные кластеры используются в основном для решения, так называемых, суперзадач, связанных с моделированием сложных процессов квантовой механики, ядерной физики, химии и т. пр.

Модели этих процессов можно разделить на три больших класса.

Модели, в которых прослеживаются траектории объектов в ходе развития процесса во времени. На мультипроцессорные системы эти модели отображаются с помощью закрепления за каждым элементарным процессором объекта (или группы объектов). Элементарный процессор тем самым как бы моделирует движение объекта во времени и пространстве.

Следующий класс - это модели, в которых рассматриваются области (точки) пространства и анализируется состояние этих областей

в ходе развития процесса во времени. Элементарный процессор имитирует область пространства, через которую движутся объекты.

Третий класс - это модели, в которых определяется либо объект, либо область, либо состояние процесса, удовлетворяющие неким критериям.

Эти последние модели можно обобщенно назвать задачами информационного поиска. К ним, в частности, можно отнести задачи ассоциативного поиска, задачи на отыскание экстремумов, задачи распознавания образов и классификации объектов.

Для решения задач информационного поиска в настоящее время широко используется нейросетевой подход и эволюционные (генетические) вычисления.

Суть нейросетевого подхода состоит в принятии решений на основе накопленного опыта, по прецеденту.

Суть эволюционных (генетических) вычислений состоит в попытке взять на вооружение механизм естественного отбора, придуманный природой, для нахождения хорошего, в каком-то смысле, решения (или образца). Эти алгоритмы ведут свое начало от известных методов монтекарло отыскания экстремумов функций.

Существенно то, что эволюционные вычисления могут быть применены к широкому кругу задач, не сводящихся к строгой математической постановке, не сводящихся к прозрачной числовой интерпретации.

И нейросетевые и эволюционные (генетические) вычисления привлекательны тем, что они параллельны по своему существу и на первый взгляд прямо «ложатся» на мультипроцессорные системы. Однако это только на первый взгляд.

Большинство типов нейросетей требуют наличия очень большого числа связей между нейронами, в пределе «каждый с каждым». Если отобразить каждый нейрон на каждый элементарный процессор вычислительной системы, то из-за огромного числа транзитных передач в реальных системах при такой схеме практически никакого эффекта от распараллеливания получить невозможно. Приблизительно такая же ситуация возникает при использовании генетических вычислений

Одной из важных задач системного программирования является создание программных систем, эффективно реализующих эти алгоритмы на конкретных установках хотя бы для основных типов нейросетей: персептронов, сетей Хопфилда, рекуррентных сетей.

Эти работы ведутся на кафедрах АСВК и АНИ в рамках гранта РФФИ по созданию системы анализа динамических процессов с использованием нейросетевых и генетических алгоритмов (Н. Н. Попова, А. М. Попов и Л. Н. Королев).

Актуальные задачи по нейросетевой тематике и эволюционным алгоритмам, которые исследуются на кафедре АСВК могут быть сформулированы следующим образом:

1. Анализ архитектурных решений, позволяющих поддерживать широкий класс эволюционных и нейросетевых алгоритмов, включая обучение, дообучение (в случае необходимости) и функционирование нейросетей.
2. Исследование поведения различных нейросетевых алгоритмов на различных реальных базах данных и конкретных многопроцессорных системах.
3. Реализация общесистемного подхода к организации функционирования нейросетей в условиях распределенной обработки данных и удаленного доступа пользователей.
4. «Интеллектуализация» обработки данных с использованием новых технологий: автоматическая настройка параметров генетических алгоритмов, «самоорганизация» нейросетей и т.д.
5. Сравнительный анализ применения новых информационных подходов к решению известных задач с классическими математическими методами, с целью определения того алгоритма, который более всего подходит для решения данной конкретной задачи.

Представленные задачи решаются аспирантами и студентами спецсеминара и поддерживаются грантом РФФИ.

Еще одной целью данных исследований и разработок является обучение студентов методам построения и применения нейросетей, с использованием параллельных вычислительных систем.

Из приведенных рассуждений общего характера следует, что насущной необходимостью стала разработка такой межпроцессорной системы обмена информации, такой коммутационной среды, которая обеспечит темпы обмена, не зависящие от числа процессоров и их взаимного расположения в вычислительной системе, которая обеспечит моделирование связей типа «каждый с каждым» без существенных задержек при транзитах.

В этом случае резко упроститься задача эффективного распараллеливания и тем самым задача повышения эффективности использования машин с массовым параллелизмом. Есть определенные надежды на успехи в оптоэлектронике, если из нее изъять как можно больше чисто электронных компонент. Как известно, ведутся интенсивные работы по исследованию возможности создания чисто оптических машин.

Возможно, в не столь отдаленном будущем реально появятся квантовые компьютеры, в которых, как представляется, проблемы задержек в коммутационной среде не должно быть. Квантовое

взаимодействие одновременно всех атомов системы исключает эту проблему.

Вообще, квантовые компьютеры для многих программистов представляют вещь в себе, пока что не очень хорошо понимаемую. Например, не очень ясно, какие алгоритмы потребуются для управления взаимодействием обычного компьютера с квантовым сопроцессором для возбуждения его начального состояния и необходимых начальных данных для его моментального срабатывания. Эта проблема уже возникала при создании аналого-цифровых гибридных систем. Говорят, что иногда программная настройка универсальных аналого-цифровых систем занимала так много времени, что быстрее было бы решить всю задачу на традиционном компьютере. Это, правда, не касается систем специального назначения.

В заключение можно сказать, что грандиозная пирамида программного обеспечения, основание которой держится на очень маленьких по габаритам микропроцессорах, продолжает неуклонно расти, открывая все новые горизонты для исследований и разработок в области программирования и информационных технологий. Сфера деятельности системных программистов непрерывно растет. Их работа похожа на движение в гору, которая становится все круче и круче. До вершины еще далеко.

Литература

1. L. Korolev. Coding and code compression J. ACM, v.5, *N* 4,1958.
2. Л. Н. Королев и др. Программа автоматического дифференцирования. Из. ИТМиВТ АН СССР, 1959.
3. Л. Н. Королев. Архитектура операционной системы. Сб. «Труды семинара № 8» Из. ИТМиВТ АН СССР, 1968.
4. Люстерник М. Р. Шура-Бура, А. А. Абрамов, Шестаков. Из. АН СССР, 1952 г.

Машечкин И.В, Петровский М.И, Попов И.С., Шляховая Е.М.

"Web-система моделирования средств планирования операционных систем mainframe компьютеров"

Введение.

Вычислительные системы на основе мэйнфреймов[1] обычно ориентированы на способность поддержки одновременной работы сотен или даже тысяч пользователей. Мэйнфреймы и до сегодняшнего дня остаются одними из наиболее мощных вычислительных систем общего назначения, обеспечивающими непрерывный режим эксплуатации. Пример классического использования мэйнфрейма – это хорошо управляемая и безопасная среда вычислений предприятия, построенная на базе такой машины с подключёнными к ней терминалами.

Нужно заметить, что огромный рост производительности персональных компьютеров, рабочих станций и серверов создал тенденцию перехода с мэйнфреймов на компьютеры менее дорогих классов, в частности, решения основанные на архитектуре клиент-сервер.

Однако в самое последнее время этот процесс перехода несколько замедлился. К основным причинам возрождения интереса к мэйнфреймам нужно отнести:

- невыгодность отказа от наработанного за десятилетия программного обеспечения для мэйнфреймов;
- переход к распределённой архитектуре связан с рядом проблем, затраты на преодоление которых могут быть неоправданны;
- распределенная среда не обладает достаточной надежностью для наиболее ответственных приложений, которой обладают мэйнфреймы;

Таким образом, в настоящее время сохраняется актуальность применения мэйнфреймов в качестве средств централизованной обработки информации в крупных организациях.

В данной статье рассматриваются проблемы построения экспериментальной системы моделирования, предназначенной для анализа эффективности настройки подобных вычислительных комплексов.

1. Описание и постановка проблемы

1.1 Структура вычислительной системы

В архитектурном плане мэйнфрейм представляет собой многопроцессорную систему, предназначенную для работы с большими объёмами данных. Несколько мэйнфреймов могут объединяться в сисплекс[1] - многомашинный кластер, выглядящий с точки зрения пользователя единым компьютером.

На сегодняшний день значительное число мэйнфреймов в мире представлено машинами фирмы IBM, работающими под управлением операционной системы MVS[2] и её клонов.

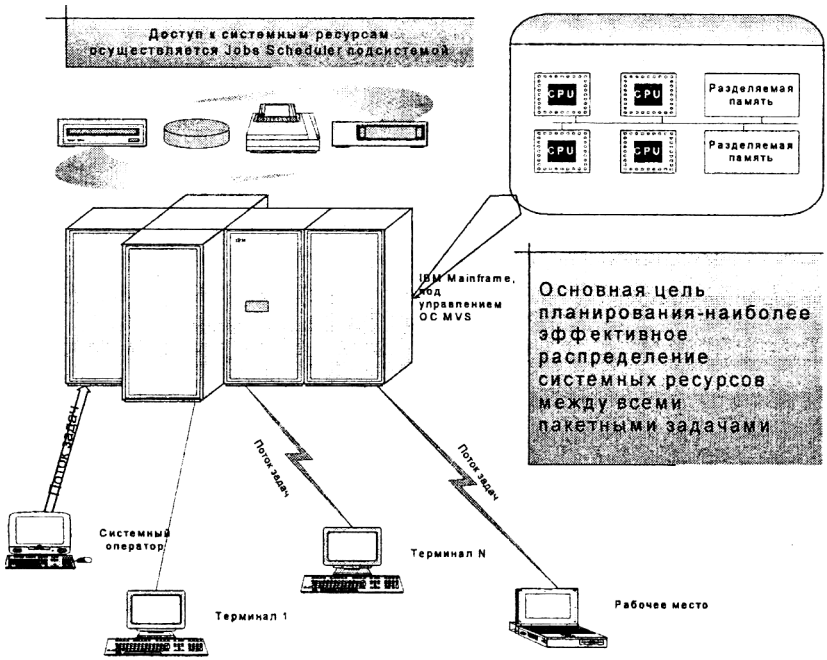
В статье рассматривается данная платформа, как наиболее распространённая и популярная.

Основным режимом выполнения пользовательских задач в ОС MVS является пакетный режим[1,3].

Существуют различные программные средства планирования пакетных задач: от встроенных в ОС компонент с фиксированными стратегиями выбора задач до достаточно сложных, динамически настраиваемых подсистем планирования.

Различные планировщики, как правило, работают одновременно на одной вычислительной системе, и потому разные задачи могут обслуживаться разными планировщиками, что увеличивает гибкость системы, но усложняет выбор наиболее эффективной схемы настройки планировщиков.

Изменения в настройках планировщиков осуществляются администратором системы, например, на основании статистики по обработанным задачам, или в связи с изменением аппаратной конфигурации вычислительной системы.



Вычислительный комплекс на основе IBM мэйнфрейма

Комплект поставки ОС MVS, как правило, включает в себя следующие средства планирования:

- Job Entry Subsystem2 - базовое средство планирования, основанное на системе приоритетов и классов и использующее Resource Affinity Subsystem – систему управления разделяемыми ресурсами. [4]
- Workload Manager – планировщик, позволяющий подняться над уровнем непосредственного управления ресурсами и использовать так называемые *цели выполнения задачи – Service Goals*. [5]

1.2 Формулировка проблемы

Очевидно, что эффективность функционирования подобной вычислительной системы в конкретном случае зависит от конфигурации аппаратной части системы; состава и настроек планировщиков операционной системы; и, в некоторой степени, от характеристик задач, обслуживаемых системой.

Поэтому возникающая проблема формулируется как проблема выбора и оценки эффективности настройки подсистем планирования, в проекции на конфигурацию вычислительной системы и характеристики входного потока задач, то есть возможность стратегической оценки эффективности применения вычислительной системы конкретной конфигурации для решения заданного класса задач.

Типичные затруднения, с которыми сталкивается администратор комплекса при настройке системы, выглядят следующим образом:

- эксплуатация подобных вычислительных систем довольно дорогостояща, из-за высокой стоимости аппаратной платформы и соответствующего программного обеспечения; поэтому эксперименты по оценке эффективности на реальной системе довольно накладны.
- существует необходимость предварительной оценки эффективности ещё проектируемого вычислительного комплекса.

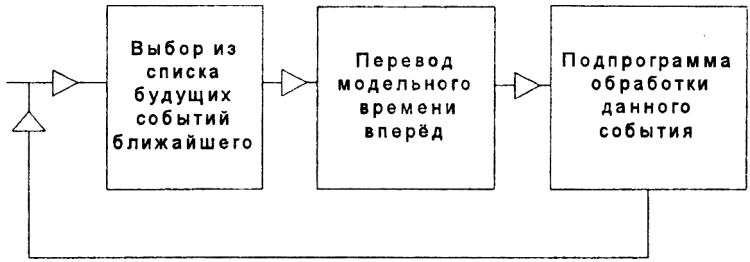
1.3 Выбор методов решения.

Для решения обозначенных проблем были применены методы дискретно-событийного имитационного моделирования[6]. Это вызвано тем, что построение эффективной математической модели исследуемой вычислительной системы оказалось достаточно проблематичным из-за общей сложности описания всех настроек системы и присутствия в системе взаимосвязанных случайных процессов (например, активность пользователей), которые не могут быть описаны адекватно. Естественное решение подобных проблем – это использование алгоритмов имитационного моделирования .

Идея дискретно-событийного имитационного моделирования формулируется следующим образом:

Исследователь описывает события, которые могут изменять состояния системы и определяет логические взаимосвязи между ними. Имитация происходит путём выбора из списка будущих событий ближайшего по времени и его выполнении. Выполнение события приводит к изменению состояния системы, и генерации будущих событий, логически связанных с выполняемым. Эти события заносятся в список будущих событий и упорядочиваются в нём по времени

наступления. Имитационное время продвигается от события к событию и между событиями состояние модели не меняется.



Выполнение событий в дискретно-событийной имитационной модели

1.4 Постановка задачи. Этапы разработки.

Постановка задачи формулируется следующим образом:

Построение эффективной системы моделирования средств планирования - Job Entry Subsystem2 и Workload Manager (WLM), контролируемых подсистемой управления ресурсами Resource Affinity Subsystem.

Процесс решения задачи разделился на два этапа.

1) Разработка имитационной модели, реализация desktop системы моделирования (ThruPut Simulator), позволяющей оценить производительность и загрузку вычислительной системы, в зависимости от физической конфигурации, настроек планировщиков и характеристик входного потока задач.

2) Реализация Web-системы моделирования, необходимость в которой была вызвана проблемами, возникшими в процессе работ над desktop системой. Она получила название Web Thruput Simulator.

2. Реализация – проект Web ThruPut Simulator

2.1 Выбор средств реализации

Средства разработок для имитационного моделирования делятся на две группы:

- Языки моделирования: Simula, Sim++, C++ Sim и другие. Их достоинства – удобство, быстрота программирования и

концептуальная выразительность. В настоящее время насчитывается около 700 таких языков[7,8]. Однако они обеспечивают меньше возможностей по сравнению с универсальными языками программирования.

- Проблемно-ориентированные средства и системы имитационного моделирования: ПОДСИМ, Ithink, PowerSim, АСИМПТОТА, ProModel и др. [7,8]. Имитационная модель генерируется самой системой в процессе диалога с пользователем. Такие системы не требуют от пользователя знания программирования, но позволяют моделировать лишь относительно узкие классы сложных систем.

Предварительный анализ существующих средств выявил ряд неудобств, возникающих из-за специфики характеристик разрабатываемого продукта. В частности, при использовании языков моделирования это недостаточная производительность получаемой системы моделирования, и, как правило, платформенная зависимость построенных систем. При использовании существующих систем моделирования довольно трудозатратно настроить систему, то есть описать все процессы и алгоритмы происходящие в моделируемой вычислительной системе.

Поэтому имитационная модель разработанной системы была реализована на обычном языке программирования – C++.

2.2 Проект Sysplex ThruPut Sumulator – первый этап разработки.

Разработанная desktop версия системы моделирования получила название Sysplex ThruPut Simulator (TS).

Это продукт, основанный на использовании методов дискретного событийно-ориентированного моделирования, задачей которого является априорная оценка характеристик MVS сисплекса.

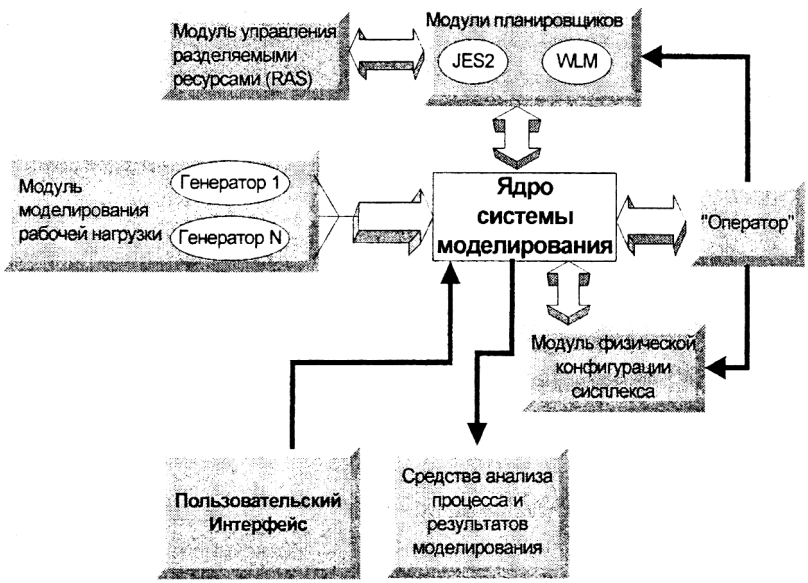
TS моделирует прибытие, обработку и выполнение потоков пакетных задач в подобной вычислительной системе, управляемой базовым планировщиком JES2 и стандартной подсистемой управления ресурсами. Система дополнительно может контролироваться планировщиком задач Workload Manager (WLM).

TS поддерживает следующие возможности

- Визуальное задание конфигурации сисплекса.
- Моделирование прихода, конвертирования, анализа и исполнения задачи.
- Функционирование планировщика JES2 и его подсистемы исполнения.

- Функционирование планировщика WLM и его подсистемы исполнения.
- Функционирование подсистемы управления ресурсами RAS.
- Визуальное задание параметров потока задач (характеристики задач, требуемые ресурсы, распределения времени прихода, конвертирования, анализа и выполнения).
- Импорт параметров входного потока задач из информационных файлов реальной системы.
- Имитация действий "оператора/администратора" (динамическое изменение состояний системных ресурсов, процессорной мощности и состояний программных компонент).
- Графическое и текстуальное отображение различных характеристик и собранной во время моделирования статистики.
- Экспортирование результатов моделирования в базы данных MS Access и Lotus Approach.
- Генерация отчётов в форматах MS Office и Lotus Notes 1-2-3
- Сохранение параметров моделирования и конфигураций всех объектов в формате XML [9].

Проект функционирует на платформе операционных систем семейства Windows.



Архитектура системы моделирования TS

2.3 Web ThruPut Sumulator – второй этап разработки.

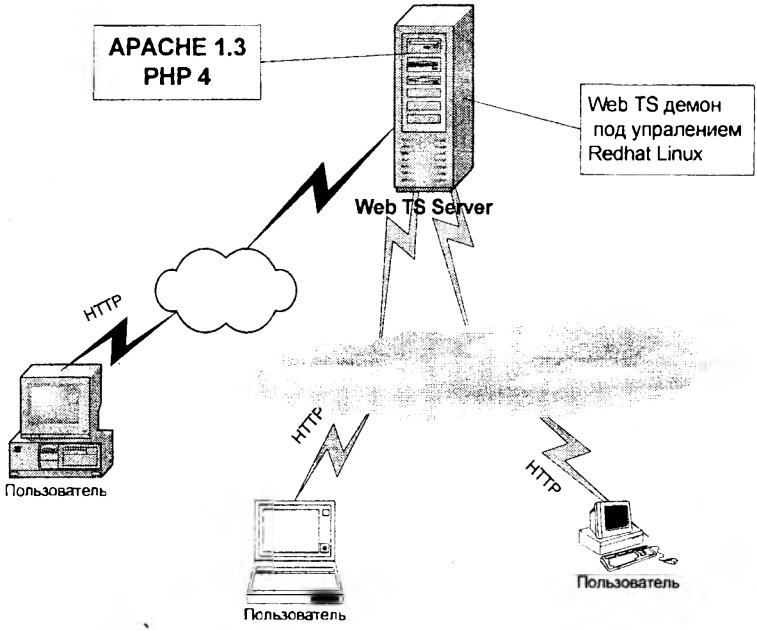
В процессе разработки, тестирования и использования системы моделирования TS выявились следующие неудобства и проблемы:

- Пользователь должен обладать определёнными навыками, чтобы правильно инсталлировать, настроить (и использовать) систему моделирования.
- С увеличением степени детализации имитационной модели и с увеличением сложности самой модели вычислительной системы компьютер пользователя должен удовлетворять всё более высоким и высоким требованиям по производительности, для того, чтобы обеспечить приемлемую скорость процесса моделирования для этой имитационной модели.
- Появилась проблема платформо-зависимости. То есть некоторые пользователи оказались неспособны использовать продукт ввиду его ориентированности на платформу Windows.

В результате, подверглось переработке ядро системы TS и были разработаны соответствующие приложения и Web-интерфейсы к этому ядру.

В итоге, Web-система, унаследовав большинство возможностей desktop версии, приобрела новые свойства:

- Возросшая производительность системы моделирования (связана с переносом ядра Web-системы под платформу Unix).
- Минимальные требования по квалификации пользователя.
- Минимальные требования по ресурсам компьютера клиента – достаточно наличие Web-броузера; вся вычислительная нагрузка ложится на Web-сервер достаточно большой мощности.
- Проблема платформо-зависимости естественным образом исчезает.
- Невидимая для пользователя эволюция программного обеспечения системы моделирования (а также облегчённая процедура тестирования и отладки этого программного обеспечения).
- Взаимодействие пользователей Web-системы моделирования между собой (например, можно собирать рейтинги оптимальных настроек для тех или иных конфигураций, и сразу после загрузки конфигурации вычислительной системы на сервер, по возможности, предлагать пользователю подходящие для его случая настройки).



Структура Web TS: Платформа – Win32 или Linux Redhat 6.2 Apache-server. Основное ядро – Win32 сервис или, соответственно, демон для Linux версии. Дополнительные модули – Php4.

2.4 Анализ результатов работы системы моделирования

Априорная оценка определённых характеристик моделируемого сисплекса достигается путём анализа результатов, полученных в ходе моделирования.

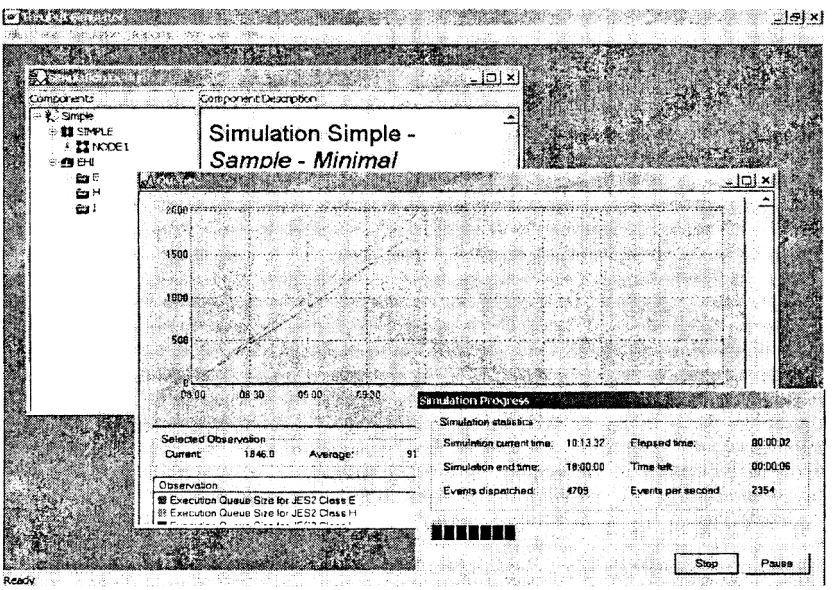
К ним относятся:

- Широкий спектр статистик, показывающих состояния программных и физических компонент на протяжении всего процесса моделирования (или каких-то его ключевых этапов). К примеру, могут собираться статистики о загруженности того или иного физического процессора, использования разделяемых ресурсов, состояния очередей задач на каждом из этапов их обработки. Для удобства пользователя, состояния нужных статистик могут отображаться визуально, в виде динамически меняющихся графиков. Следует отметить, что отключая

сбор ненужных в конкретном случае статистик, можно добиться значительного прироста производительности системы моделирования.

- Информационные файлы, созданные системой моделирования, отображающие последовательность событий, генерируемых моделью и последовательность состояний модели во времени, соответственно.

По окончании процесса моделирования, на основании полученных информационных файлов система может осуществить конвертирование ключевых данных в формат баз данных Access и Approach для их последующего более подробного и тщательного анализа. Это вызвано тем, что среднее число задач проходящих через систему моделирования составляет десятки-сотни тысяч.



Процесс прогонки модели в системе моделирования TS (desktop-версия)

3. Результаты

Результаты проделанной работы:

- Предложена имитационная модель, адекватно представляющая структуру рассматриваемой вычислительной системы.

- Разработана (Web) система моделирования средств планирования JES2 и WLM, позволяющая получать различные качественные оценки эффективности вычислительной системы.
- Система обладает широкими возможностями визуальной настройки и конфигурирования. Разработаны развитые средства отображения процесса и результатов моделирования.
- Система моделирования удовлетворяет жёстким требованиям по скоростным характеристикам и обеспечивает обработку порядка сотен тысяч событий в час для нетривиальной конфигурации вычислительной системы.

Существующая версия Web-системы доступна по адресу http://ts.mlab.cs.msu.su

4. Заключение

Основной аспект использования созданной системы - разработанное программное средство позволяет оценить какая из схем планирования и при каких настройках наиболее эффективна на данном мэйнфрейме для данного класса задач.

Дополнительно предоставляется возможность предварительно подобрать конфигурацию мэйнфрейма, оптимально подходящую для определённого заданного класса задач.

В настоящее время ведутся работы по унификации и параметризации свойств аппаратных и программных подмоделей, с целью расширения применимости разработанного средства для более широкого класса вычислительных систем.

Проект Web Thruput Simulator был реализован в сотрудничестве с рядом канадских фирм, работающих в данной области: MVS Solutions и Oxbow Technologies.

В настоящее время разработанный продукт используется в среде пользователей операционных систем семейства MVS.

Литература

1. Jim Hoskins, Goerge Coleman "Exploring IBM S/390 Computers", 6-th edition. Maximum Press, 1999
2. "MVS/ESA Version 5. System Management Facilities", IBM, 1995
3. Дейтел Г. "Введение в операционные системы". том 2. Москва, "Мир", 1987

4. “MVS/ESA Version 5. Planning: Workload Management”, IBM, 1995

5. “ MVS/ESA Version 5. WLM: Resource Affinity Scheduling”, IBM, 1997

6. Moshe A. Pollatschek “Programming Discrete Simulation”, Prentice Hall , 1995

7. Емельянов В.В., Ясиновский С. И . "Введение в интеллектуальное имитационные моделирование сложных дискретных систем и процессов", Москва, "АНВИК", 1998

8. “Методологические основы и математические методы” Москва, “Мир”, 1981

9. David Lewis, Matthew Fuchs. “Designing XML Internet Application”, Prentice Hall, 1998

Брусенцов Н.П.

Трехзначная диалектическая логика

Диалектика - это наука о наиболее общих законах развития природы, общества и мышления, характеризуемая вкратце как учение о единстве противоположностей [1, с. 79]. Диалектическая логика по сути должна быть методом этой науки, однако, суть ее все еще остается дискуссионной проблемой [1, с. 80]. Ее противопоставляют формальной (традиционной) и современной (математической) логике как неформальную, содержательную и вместе с тем как логику движения, развития в духе гераклитова "все течет, все изменяется". Поскольку формальность понимается также как отвлечение от действительности и априорность принципов, то диалектической должна быть логика, принципы которой реальны, а критерий истинности состоит в соответствии опыту, практике. Суть этой логики в том, что в ней адекватно отображены взаимосвязи вещей, и что по вещему замыслу Аристотеля "будучи способом исследования, она прокладывает путь началам всех учений" [2, "Топика", 101b3].

Практическая неадекватность формальной логики обусловлена искусственностью ее основополагающего принципа - закона исключенного третьего, в силу которого важнейшее из отношений несовместимости - контрарность - не является базисным. Поскольку "третьего не дано", то логика двухзначна, и операцией отрицания в ней естественно оказывается дополнение, а отношением несовместимости - комплементарность (контрадикторность).

В отличие от контрарности, предполагающей наличие между членами отношения противоположности третьего-среднего, промежуточного, комплементарность дихотомична - третье-промежуточное, представляющее собой переход из одной крайности в другую, в ней исключено. Но как раз в этом переходе предмет диалектической логики - изменчивость, развитие, становление. И надо заметить, что недиалектичность обусловлена не комплементарностью отрицания, а именно игнорированием переходного третьего. Комплементарность не противопоказана ни диалектике, ни трехзначности: в трехзначной логике "не-первое" есть "второе либо третье", а "третье" есть "не-первое и не-второе". Формалисты полагают, что "не-первое и не-второе" ("не-да и не-нет") невозможно, не существует, поэтому ни аристотелева силлогистика, ни модальная, ни тем более диалектическая логика адекватного отображения у них не находят и оттого подозреваются в некорректности.

Очевидным свидетельством неестественности двухзначной логики служит общество "рыцарей" и "лжецов" [3], устроенное и функционирующее по ее законам. Все его члены либо "рыцари", либо "лжецы", "Рыцарь" никогда не лжет, "лжец" лжет всегда, не может не лгать. "Рыцарь" есть "не-лжец", "лжец" есть "не-рыцарь". Таков дискретный мир двухзначности - крайне грубая карикатура реального мира.

В реальности нет ни идеальных рыцарей, ни абсолютных лжецов, причем если рыцарями признают избегающих лжи, то лжецом называют и единожды солгавшего. Но в подавляющем большинстве люди - не рыцари и не лжецы, а нечто третье- промежуточное, то лгут, то не лгут, в зависимости о обстоятельств, и как показал Сократ, ложь тоже бывает добродетелью. Реальность не так проста, как полагают формалисты-двухзначники, напрочь устранившие ее диалектическое содержание своим априорным "законом исключенного третьего".

Конечно, логика в результате стала простой, но осваивают люди ее (вернее - выучивают) с большим трудом, потому что она не соответствует здравому смыслу, выхолащивает сущность взаимосвязей, обнаруживаемых в действительности. Это не тот Органон, в котором Аристотель усматривал начала и общий метод всех наук, универсальный инструмент познания. Не удивительно, что науки развиваются сами по себе, независимо от формалистского "Органона", а настойчивое стремление положить его в основание математики привело к кризису [4,5]. В практических делах формальную логику не применяют, полагаясь на интуицию и здравомыслие. Исключение составляют, пожалуй, двоичные компьютеры - материализация логики "рыцарей" и "лжецов". Но даже в условиях триумфа цифровых технологий безуспешность "интеллектуализации обработки информации" настораживает.

Закон исключенного третьего незаслуженно приписывают Аристотелю, и таким образом авторитетнейший философ оказывается отцом двухзначной логики. На самом деле принцип двухзначности изобрели, вопреки аристотелизму, стоики (в частности, Хрисипп). Аристотель же неотступно настаивает на существенности третьего (среднего, промежуточного, привходящего), квалифицируемого им как предмет диалектики [2, "Метафизика, 996b27]. В силлогистике наряду с дихотомией (контрадикторностью) имеет место трихотомия (контрарность), и "не быть" не равнозначно "быть не-", чего формалистам, от стоиков до ультрамодерных, постигнуть, видимо, не дано.

У Аристотеля третье исключено только в "началах доказательства" применительно к первичным терминам, присущность/антиприсущность которых индивидным объектам должна быть однозначной, иначе ничего определенного доказать нельзя [2,

"Вторая аналитика", 74b5]. Действительно, в силлогистике заключение о необходимой присущности термина *x* термину *z* возможно только из посылок, выражающих также необходимые взаимосвязи терминов *x* с *y* и *y* с *z*, т. е. только из общеутвердительных либо общеотрицательных суждений, в которых термины дискретны и двухзначны, как в булевой алгебре. Однако отношение присущности у Аристотеля трехзначно: общеутвердительное суждение, выражающее присущность, контрарно общеотрицательному (суждению антиприсущности), совместное же отрицание обоих (не-присущность и не-антиприсущность) составляет третье, промежуточное-привходящее.

В традиционной трактовке силлогистики истолковывать общие суждения как выражающие присущность (она же необходимое следование) не принято. Поэтому третье-среднее остается невыявленным, треугольник присущность/антиприсущность/не-антиприсущность подменен "логическим квадратом Псела", и нет возможности преодолеть парадоксы материальной (равно как и "строгой", "сильной" и пр.) импликации, чтобы прийти к безупречному определению необходимого следования, уже сформулированному Аристотелем [2, "Первая аналитика", 57b1] две с лишним тысячи лет тому назад.

Впрочем, "квадрат Псела" вполне корректен и естественно обобщается, если не ограничиваться пределами "позитивной" силлогистики, в "логический куб" с четырьмя частными суждениями. Затруднение составляет несовместимость утверждаемых в нем законов подчинения частных суждений общим с законом исключенного третьего [6]. Однако именно последний принципиально несовместим с силлогистикой в силу ее диалектической сущности.

Аристотель употребил в силлогистике термины-буквы в качестве символов абстрактных особенностей, попарно различных и соединимых в различные по составу совокупности (в булевой алгебре элементарные конъюнкции) - всевозможные виды составных понятий 1-й ступени. Конкретные понятия порождаются приданием терминам тех или иных смысловых значений, подобно тому как в числовой алгебре входящие в выражение буквы-переменные заменяются числовыми значениями либо выражениями, принимающими числовые значения. Двухзначность терминов, т.е. подчиненность закону исключенного третьего, понимается в духе аристотелевых "начал доказательства" как четкая попарная различимость: сопоставляемый с термином *x* произвольно взятый термин есть либо *x*, либо не-*x*, что равнозначно анти-*x*, ибо третье исключено. Практически исключенность выражается в недопустимости такого начертания термина-буквы, при котором однозначное опознание ее не гарантировано. Короче, исключение третьего обеспечивает дискретность идентификации терминов.

Далее проблема возникает в связи со значением, а лучше сказать, со статусом, термина в составе совокупности (элементарной конъюнкции), определяющей сущность рассматриваемого (данного) понятия. По Аристотелю, термин может быть либо необходимо присущим понятию (принадлежащим совокупности), либо антиприсущим (антипринадлежащим), либо привходящим (не принадлежащим с необходимостью и не антипринадлежащим). В булевой алгебре совокупность-конъюнкция содержит присущие термины неинвертированными, антиприсущие - с символом инверсии-отрицания, а привходящие не содержит (умалчивает). Например, сущность понятия, охарактеризованного в терминах *x, y, z, u* так, что *x* и *u* присущи, *y* антиприсуще, *z* привходяще, выражается конъюнкцией $xy'u$, где y' символизирует диаметральную инверсию (контрарное отрицание) термина *y*, а привходящее *z* умалчивается.

Как видно, отношение присущности у Аристотеля трехзначно (трихотомично), и эта трихотомия вполне отобразима посредством надлежащим образом интерпретируемой булевой алгебры. Сам Дж. Буль, выявивший третье-привходящее в результате решения логических уравнений, называл его *неопределенностью*, а умалчиваемые термины - *элиминированными* [7,8]. Сущность привходящего, которое по Аристотелю не обусловлено однозначно, но может быть "либо так, либо не так - как попало", выявляет процедура, формирующая выражение $xy'u$ "склеиванием" двух противоположных относительно *z* индивидных конъюнкций:

$$xy'zu \vee xy'z'u \equiv xy'u(z \vee z') \equiv xy'u$$

Исходная дизъюнкция индивидных конъюнкций - совершенная дизъюнктивная нормальная форма (СДНФ) выражения $xy'u$ - свидетельствует о том, что это выражение нечетко, огрубленно определяет рассматриваемое понятие в заданном наборе терминов, идентифицируя не индивид, а *класс*, включающий два индивида. Тождественное преобразование СДНФ в минимальные формы осуществимо благодаря тому, что в последних термины обретают трехзначность. В приведенном примере атрибут $xy'u$, присущий каждому индивиду, выражает сущность класса в целом, тогда как умалчиваемый атрибут *z* является привходящим.

СДНФ произвольного *n*-терминного выражения включает до 2^n индивидных конъюнкций, в частности, не склеиваемые ни по одному термину, либо попарно склеиваемые по нескольким терминам, так что все они оказываются привходящими по отношению к той или иной неиндивидной конъюнкции в минимальной форме. Присущность термина СДНФ-выражению в целом характеризуется в общем случае отношением числа членов СДНФ, содержащих этот термин неинвертированным, к числу всех ее членов. Например, для выражения материальной импликации $x \rightarrow y \equiv xy \vee x'y \vee x'y'$ присущность

термина x равна 1/3, присущность термина y равна 2/3. Минимальная форма этого выражения - $x' \vee y$ - преобразуется в СДНФ восстановлением умалчиваемых в ней терминов:

$$x' \vee y \equiv x'(y \vee y') \vee (x \vee x')y \equiv$$
$$\equiv x'y \vee x'y' \vee xy \vee x'y \equiv xy \vee x'y \vee x'y'$$

Диалектика, или логика познания, определяемая Аристотелем как учение о привходящем, не ограничивается выявлением сущности последнего и восстановлением умалчиваемых в булевой алгебре привходящих терминов. Главная ее задача - устранение неопределенности, установление условий, при которых привходящий термин обретает статус непривходящего.

В случае материальной импликации эта задача сводится к решению уравнения $x' \vee y = 1$ относительно соответствующего термина. Так, для термина y получим: $y = x \vee x'y$, т.е. если дано x, то необходимо дано y, а если x исключено, то y привходяще. Аналогично для x: если y исключено, то и x необходимо -исключено, иначе x привходяще. И никаких парадоксов.

В случае с привходящим z в предыдущем примере решение уравнения $x'yu = 1$ относительно z невозможно, поскольку термина z это уравнение не содержит, он элиминирован как не связанный с прочими терминами. Требуется эмпирически уточнить или конкретизировать взаимосвязи, возможно, с привлечением новых терминов. Так, может оказаться, что наряду с двумя учтенными в примере альтернативами - индивидными членами дизъюнкции - имеются другие либо обнаружится хотя бы один еще не совпадающий с рассматриваемыми термин, что неизбежно увеличит число альтернатив.

Надо сказать, что невозможность исследовать данный пример путем решения булева уравнения не случайна. С диалектической точки зрения этот пример некорректен. Один лишь термин z представлен в нем противоположностью своих статусов, позволяющей заключить, что же есть z в отличие от прочих терминов.

Ведь только из сопоставления того, чему термин присущ, с тем, чему он антиприсущ, извлекается представление о том, что им обозначено и чем он отличается (если отличается) от других терминов. Это именно тот принцип диалектики, который Гегель неудачно определил как тождество, или единство, противоположностей. При общепринятом понимании "тождества" в смысле "одно и то же", а "единства" в смысле совместности-конъюнкции тождество, как и единство, противоположностей, немыслимо, невозможно. Это фундаментальный принцип логики - установленный Аристотелем закон недопустимости противоречия: противоречащее одно другому не может сказываться вместе" ["Метафизика", 1007b18].

В диалектике нет необходимости отвергать этот общепризнанный принцип, неприятие которого исключает возможность рассуждать. Реанимация логики достижима при не столь катастрофическом и вполне отвечающем реальности постулате *сосуществования противоположностей*.

Противоположности - это члены отношения необходимой несовместимости (контрарности), которое также называют противоположностью. Противоположны присущность и антиприсущность, наличие и отсутствие, утверждение и отрицание, бытие и небытие. Безусловная несовместимость противоположностей алгебраически выражается тождеством: $x \wedge x' \equiv 0$ - конъюнкция (т.е. единство) утверждения и отрицания одного и того же (термина x) тождественна невозможности, есть "ничто". Это аристотелево требование непротиворечивости. Оно исключает существование вещи, которой были бы присущи совместно противоположные определенности, или иначе говоря, присуще и антиприсуще одно и то же.

Но это требование не препятствует тому, чтобы противоположности одновременно были присущи разным вещам. Более того, вещи тем и различаются, что каждой из них присуще нечто противоположное присущему другим. Таким образом, противоположности необходимо *сосуществуют* в различных вещах и соответственно в понятиях об этих вещах. Рассмотренная выше пара индивидных конъюнкций, различавшихся статусом термина z, является примером такого сосуществования.

В общем виде принцип сосуществования противоположностей выразим тождеством $\mathbf{V}x\mathbf{V}x' \equiv 1$, в котором символ "интегральной дизъюнкции" $\mathbf{V}$ означает дизъюнкцию значений, принимаемых термином x и его инверсией x', распространенную на все члены рассматриваемого выражения. Требуется, чтобы данному тождеству удовлетворял любой из входящих в выражение терминов, т.е. все без исключения термины должны быть привходящими. Это и есть постулат диалектической логики: ничто не остается неизменным - "все течет", "покой нам только снится".

Замечательно, что сформулированный постулат находится в основании аристотелевой силлогистики. Бесчисленные попытки алгебраизации этой древнейшей логической системы, кстати, единственной не конфликтующей со здравым смыслом и не порождающей парадоксов, неизменно оказывались безуспешными, вследствие чего сложилось мнение, будто бы у Аристотеля нелады с понятием пустого множества [9,10]. Необходимым и достаточным условием положительного решения этой проблемы является аксиома сосуществования противоположностей [11].

В n-терминном Универсуме Аристотеля (УА) аподиктически выполняется тождество

$$Vx_1Vx'_1Vx_2Vx'_2 \ldots Vx_nVx'_n \equiv 1$$

Это значит, что у каждого выражения в УА подразумевается, может быть, по умолчанию, конъюнктивно связанная с ним левая часть данного тождества. Например, выражение "строгой импликации" Льюиса [12] $V'xy'$ становится в УА общеутвердительным суждением силлогистики:

$$Axy \equiv V'xy'VxVx'VyVy' \equiv V'xy' VxVy' \equiv VxyV'xy'Vx'y'$$

- необходимо существуют объекты, идентифицируемые как xy и объекты, идентифицируемые как $x'y'$, исключено существование объектов класса xy'. В таком понимании это суждение избавлено от парадоксов и строго соответствует аристотелеву определению необходимого следования, равнозначного присущности сказуемого y подлежащему x.

Конъюнкция $VxyV'xy'Vx'y'$ существований/антисуществований индивидных в наборе терминов x, y объектов определяет нечеткое по Л.Заде [13] множество объектов этого рода, декларируя принадлежность ему xy и $x'y'$, антипринадлежность xy' и умалчивая $x'y$, придав тем самым ему статус привходящего, не присущего и не антиприсущего и потому вольного при устранении нечеткости стать либо присущим, либо антиприсущим. В последнем случае четкое множество $VxyV'xy'$ $V'x'yVx'y'$ определяет отношение тождественности: $x \equiv y$.

Представленные определения сводят силлогистику к теории нечетких множеств в УА. В сущности это трехзначная логика 2-й ступени - логика дизъюнктов (интегральных дизъюнкций), изоморфная трехзначной логике терминов 1-й ступени, однако в условиях сосуществования противоположностей, т.е. в УА. Например, общеотрицательное суждение Exy получается из общеутвердительного инвертированием термина-сказуемого y:

$$Exy = inv.y(Axy) = inv.y(VxyV'xy'Vx'y') =$$
$$= Vxy'V'xyVx'y = V'xyVxVy$$

Другое общеотрицательное суждение - несовместимость антиподов $Ex'y'$, упущенная традиционной силлогистикой, получается инвертированием в Axy термина x:

$$Ex'y' = inv.x(Axy) = Vxy'Vx'yV'x'y' = Vx'Vy'V'x'y'$$

Частноутвердительные суждения Ixy и $Ix'y'$ суть отрицания-дополнения в УА соответствующих общеотрицательных:

$$Ixy = \underset{УА}{\neg} Exy = \underset{УА}{\neg}(V'xyVxy'Vx'y) = VxyVx'Vy'$$

$$Ix'y' = \underset{УА}{\neg} Ex'y' = Vx'y'VxVy$$

Законы подчинения частных суждений общим, обращения и контрапозиции, упускаемой практически во всех формализациях силлогистики, с полной очевидностью обнаруживаются при сопоставлении выражений, представляющих связанные этими законами суждения. Все правильные модусы силлогизма, в том числе традиционно принимаемые в качестве аксиом модусы 1-й фигуры, теперь доказуемы в УА посредством дизъюнктов воспроизведением с некоторой корректировкой невостребованных на протяжении уже более ста лет результатов, достигнутых Льюисом Кэрролом при помощи его замечательной диаграммы [13, 14].

Ослабленной версией сосуществования противоположностей является неисключенность сосуществования их:

$$(Vx_1 \vee Vx'_1)(Vx_2 \vee Vx'_2) \ldots (Vx_n \vee Vx_n) \equiv 1$$

Это тождество определяет n-терминный Универсум интуиционистов (УИ), выражая принцип квазидиалектической интуиционистской, или модальной, логики. Первичные термины x, y, z, ... в ней не необходимо привходящи, т.е. могут принимать статус как привходящего, так и общезначимого, что выразимо при помощи "модальных функторов" L, M, Q, именующих двухзначные функции трехзначной переменной: Lx, Mx, Qx, определенные в УИ так:

$$Vx \vee Vx' \equiv VxV'x' \vee V'xVx' \vee VxVx' \equiv Lx' \vee Lx \vee L'x'L'x \equiv$$
$$\equiv M'x \vee M'x' \vee MxMx' \equiv Lx' \vee Lx \vee Qx \equiv 1$$

Короче, в УИ: $Mx \equiv Vx$, $Lx \equiv V'x'$, $Qx \equiv VxVx'$. Очевидно, что Lx, Lx', Qx несоисключимы и попарно несовместимы, т.е. представляют собой двухзначные компоненты трехзначной характеристики x, причем $Mx \equiv Lx \vee Qx$.

Интуиционисты не исключают привходящего в качестве третьего статуса, но сосуществование противоположностей у них не необходимо.

В классической двухзначной логике сосуществование противоположностей вовсе исключено. Ее универсум - Универсум Буля (УБ) - порождается исключением в УИ третьего: $VxVx' \equiv 0$, что равносильно необходимости условия $V'x \vee V'x' \equiv 1$, с принятием которого однотерминное определение УБ сводится к тождеству

$$(Vx \vee Vx')(V'x \vee V'x') \equiv VxV'x' \vee V'xVx' \equiv$$
$$\equiv Lx \vee Lx' \equiv Mx \vee Mx' \equiv x \vee x' \equiv 1.$$

А это и есть закон исключенного третьего.

Литература

1. Кондаков Н.И. Введение в логику. - М. : "Наука", 1967.

2. Аристотель. Сочинения в четырех томах. - М.: "Мысль", т.1 - 1975, т.2 - 1978.

3. Смолиан Р.М. Как же называется эта книга? - "Мир", 1981.

4. Арнольд В.И. Антинаучная революция и математика. // Вестник РАН, том 69, № 6, 1999, с. 533-558.

5. Традиционная логика и канторовская диагональная процедура / Е.В.Болдин, С.Н.Бычков, Д.И.Виннер, Л.О.Шашкин. - М.: "Янус-К", 1997.

6. Смирнов В.А. Силлогистика без закона исключенного третьего и ее погружение в исчисление предикатов // Исследование логических систем. - М.: "Наука", 1970, с. 68-77.

7. Boole G. An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities. - London, 1854.

8. Порецкий П.С. О способах решения логических равенств и об обратном способе математической логики. - Казань, 1884.

9. Бурбаки Н. Теория множеств. - М.: "Мир", 1965.

10. Колмогоров А.Н., Драгалин А.Г. Введение в математическую логику. - М.: Изд-во Моск. ун-та, 1982.

11. Брусенцов Н.П. Искусство достоверного рассуждения. Неформальная реконструкция аристотелевой силлогистики и булевой математики мысли. - М.: Фонд "Новое тысячелетие", 1998.

12. Слинин Я.А. Современная модальная логика. - Л.: Изд-во Ленингр. ун-та, 1976.

13. Кофман А. Введение в теорию нечетких множеств. Предисловие Л.А.Заде. - М.: "Радио и связь", 1982.

14 Кэррол Льюис. Символическая логика // Льюис Кэррол. История с узелками. - М.: "Мир", 1973.

15. Брусенцов Н.П. Диаграммы Льюиса Кэррола и аристотелева силлогистика // Вычислительная техника и вопросы кибернетики. Вып.13. - М.: Изд-во Моск. ун-та, 1977, с. 164-182

Раздел II
Нечеткие множества, генетические алгоритмы и троичная логика

Петровский М.И.

Применение методов теории нечетких множеств в системах поддержки принятия решений.

Введение

Системы поддержки принятия решений - это компьютерные системы, которые позволяют человеку - лицу, принимающему решения, использовать объективные данные наряду с субъективными моделями и оценками, для анализа и решения слабоструктурированных и неструктурированных задач [1]. Ключевыми в данном определении являются понятие степени структурированности задачи и понятие субъективности оценки, тесно связанные с понятием неопределенности.
По степени структурированности задачи обычно делят на три группы [2]. Это, во-первых, хорошо структурированные и количественно сформулированные, в которых существующие зависимости, условия, ограничения и цели выяснены настолько хорошо, что они могут быть выражены численно и сведены к классическим задачам оптимизации целевой функции на множестве альтернатив. Т.е. к хорошо структурированным задачам можно отнести те, для которых существуют адекватные математические модели. Второй тип - это неструктурированные или качественно выраженные задачи, содержащие лишь описание важнейших ресурсов, признаков и характеристик, количественные зависимости между которыми неизвестны или не могут быть выражены. И наконец, к третьей группе относятся проблемы смешанного типа или слабо структурированные задачи, которые содержат как известные количественные зависимости, так и качественные, субъективные оценки. В процессе принятия решения при выборе альтернатив как правило приходится учитывать достаточно большое число зачастую противоречивых требований и сравнивать варианты решений по многим критериям, что приводит к субъективности оценок и неопределенности. Принято выделять следующие типы неопределенности [1]: неопределенность первого типа, связанная с неполнотой или неточностью исходной информации о решаемой задаче, неопределенность второго типа, связанная с

невозможностью точной оценки результата принимаемого решения, а также неопределенность третьего типа, возникающей в результате неточного понимание лицом, принимающим решения, целей принятия решения, т.е. существенная субъективность оценки решений. Задачи обладающие какой-либо из перечисленных выше неопределенностей нельзя формально свести к задачам с точно поставленными целями [1,2].

1. Постановка задачи

Ряд прикладных проблем, решаемых с помощью систем поддержки принятия решений и относящихся к области информационного поиска, классификации, кластеризации данных и т.д., приводят к классу задач, обладающему следующими свойствами:

- исходное множество альтернатив конечно;
- сравнение альтернатив производится на основе экспертных оценок и субъективных предпочтений лица, принимающего решение;
- альтернативы имеют как числовые так и не числовые (качественные) характеристики;
- может присутствовать неопределенность, связанная с неполнотой или неточностью исходной информации;

Цель данной работы состояла, во-первых, в исследовании и разработке методов сравнения и оценки альтернатив для данного класса задач с использованием аппарата теории нечетких множеств, во-вторых, практическая реализация предложенных методов для одной из прикладных задач, относящихся к данному классу.

Формальная постановка задачи:

Задано $\langle A, X, V_x, S, P \rangle$

$A = \{a_1, a_2, ..., a_N\}$ - конечное непустое множество вариантов решения (альтернатив), полученное на этапе генерации или непосредственно заданное в задаче;

$X = \{x_1, x_2, ..., x_T\}$ - конечное непустое множество свойств альтернатив (атрибутов) такое, что

$x_i : A \to V_{x_i}$, где V_x - есть нечеткое множество, определенное на множестве допустимых значений свойства x, соответственно каждая альтернатива a, характеризуется набором $X(a) = \langle x(a) \rangle_{x \ni X}$, причем

$X(a) \subseteq \langle V_x \rangle_{x \ni X}$

S – базовая шкала для сравнения альтернатив

$P = \{P_1, P_2, ..., P_M\}$ - конечное непустое множество правил (или высказываний), задающих условия на свойства выбираемой альтернативы: $P_i : X \to S$

Необходимо построить функцию предпочтения: $F_P : A \to S$, такую, что $F_P(a_i) < F_P(a_j)$, тогда и только тогда, когда a_j «лучше» a_i в смысле условий *P*.

2. Метод построения функции предпочтения с использованием нечеткого вывода

Рассмотрим метод построения функции предпочтения для данной задачи с использованием нечеткого вывода [3].

Стандартный вид правила из *P*:

$$P_k : if\ (x_1 is V_1^k)\ and\ (x_2 is V_2^k) ... (x_T is V_T^k)\ then\ S_k$$

Алгоритм вычисления значения функции предпочтения

$S_a = F_P(a)$ для альтернативы $a = \{x_1(a), ..., x_T(a)\}$

Этап I.

Для всех m правил из P вычисляется степень истинности условия

$$P_k(a) = \min(Poss(x_1(a) \mid V_1^k), ..., Poss(x_T(a) \mid V_T^k))$$

$Poss(X \mid V)$ – мера соответствия нечеткого множества X множеству V и вычисляется по формуле

$$Poss(X \mid V) = \max_U \min(\mu_X, \mu_V),$$

где μ есть функция принадлежности соответствующего нечеткого множества, а U – базовое множество для X и V

После данного этапа получаем m пар вида $\langle S_k, P_k(a) \rangle_{r=1..M}$ - распределение степени истинности на шкале S с учетом всех правил из P.

Этап II.

Вычисляется значение функции предпочтения

$$S_a = f(\langle S_k, P_k(a) \rangle_{k=1..m})$$

f называется функцией интерпретации результата. основные подходы это интерпретация по максимальному значению степени истинности,

$$S_a = \arg\max_{S_k}(P_k(a))$$

и полная интерпретация

$$S_a = \sum_{k=1..m} S_k P_k(a) \Big/ \sum_{k=1..m} P_k(a)$$

Принципиальное различие данных подходов заключается в том, что в первом случае учитывается влияние только одного правила, дающего максимальную степень истинности, а во втором учитывается вклад всех правил из *P*.

Попытка применения в реальной прикладной системе данного метода сразу выявила следующие его недостатки: во-первых, метод работает только с условиями вида «x is V», а в прикладных задачах возникает необходимость использовать условия вида «x ⊃ V», «x ⊂ V», «x = V», причем степень истинности этих условий тоже может быть нечеткой, во-вторых, метод не позволяет задавать в условиях количественные зависимости и связи между характеристиками альтернатив.

Для устранения этих недостатков была предложена следующая модификация стандартного метода с полной интерпретацией:

- для численных свойств альтернатив используются нечеткие LR числа [3], и соответственно, появляется возможность задавать в условиях выражения с использованием нечеткой арифметики;
- для нечисловых (качественных) свойств предлагается использовать нечеткие подмножества, заданные на множестве допустимых значений этих свойств, в условиях появляется возможность использовать операции объединения, пересечения, дополнения [3];
- для вычисления условий вида «x ⊃ V», «x ⊂ V», «x = V» предлагается использовать меру включения *Inc* и меру совпадения *Eql* следующего вида:

для «x ⊂ V»

$$Inc(x,V) = \frac{\|x \text{ I } V\|}{\|V\|} = \frac{\|\min(\mu_x, \mu_V)\|}{\|\mu_V\|}$$

результатом является число в диапазоне [0,1], характеризующее степень того, «насколько *V* содержится в *x*»

«x ⊂ V» аналогично *Inc(V,x)*

для «x = V»

$$Eql(x,V) = \frac{\|x \text{ I } V\|}{\|x \text{ Y } V\|} = \frac{\|\min(\mu_x, \mu_V)\|}{\|\max(\mu_x, \mu_V)\|}$$

результатом является число в диапазоне [0,1], характеризующее степень того, «насколько *V* совпадает с *x*»

- для не числовых свойств альтернатив в условиях вида «x содержит V», «x содержится в V», «x совпадает с V» можно дополнительно

задавать степень важности (iw – importance weight) для элементов V. В этом случае индексы будут иметь вид:

$$Inc(x,V) = \frac{\sum_U iw(u)\min(\mu_x(u),\mu_V(u))}{\sum_U iw(u)\mu_x(u)}$$

$$Inc(V,x) = \frac{\sum_U iw(u)\min(\mu_x(u),\mu_V(u))}{\sum_U iw(u)\mu_V(u)}$$

$$Eq(x,V) = \frac{\sum_U iw(u)\min(\mu_x(u),\mu_V(u))}{\sum_U iw(u)\max(\mu_x(u),\mu_V(u))}$$

В результате предложенной модификации правила из P приобретают следующий вид:

$$P_k : if\ (f_1^k(x_{i11}, x_{i12},...)\ o V_1^k)\ and\ (f_2^k(x_{i21}, x_{i22},...)\ o V_2^k)...then\ S_k$$

где ° есть либо is, либо =, либо ⊃, либо ⊂

$f_j^k(x_{ij1}, x_{ij2},...)$ есть выражение с операциями *,/,+,- для числовых свойств, и с операциями ∩, ∪, ¬ для остальных свойств.

Функция предпочтения в модифицированном методе будет вычисляться следующим образом:

$$F_P(a) = \sum_{k=1..m} S_k P_k(a) \Big/ \sum_{k=1..m} P_k(a)$$

где $a = \{x_1(a),...,x_T(a)\}$

$$P_k(a) = \min(f_1^k(x_{i11}, x_{i12},...)\ o V_1^k, f_2^k(x_{i21}, x_{i22},...)\ o V_2^k,...)$$

$f_j^k(x_{ij1}, x_{ij2},...)\ o V_j^k = Poss(f_j^k(x_{ij1}, x_{ij2},...) | V_j^k)$, если ° есть is

$f_j^k(x_{ij1}, x_{ij2},...)\ o V_j^k = Eq(f_j^k(x_{ij1}, x_{ij2},...), V_j^k)$, если ° есть =

$f_j^k(x_{ij1}, x_{ij2},...)\ o V_j^k = Inc(f_j^k(x_{ij1}, x_{ij2},...), V_j^k)$, если ° есть ⊃

$f_j^k(x_{ij1}, x_{ij2},...)\ o V_j^k = Inc(V_j^k, f_j^k(x_{ij1}, x_{ij2},...))$, если ° есть ⊂

3. Экспериментальная реализация метода

Предложенный метод был реализован в прикладной информационной системе управления персоналом [4,5]. Данная система

предназначена для управления персоналом в крупной компании, занимающейся реализацией проектов в области информационных технологий и электронной коммерции. В компании есть несколько территориально распределенных подразделений, и помимо основного штата сотрудников, есть еще штат «внешних» консультантов, разработчиков и аналитиков, которых компания нанимает время от времени под конкретные проекты. Каждый проект состоит из множества заданий, для которых заданы расписание, набор ролей, которые выполняют специалисты в рамках этого задания. Каждая роль имеет индивидуальное расписание и набор профессиональных требований, которым должен соответствовать специалист, выполняющий эту роль. Каждый специалист имеет набор профессиональных знаний и навыков, которыми он обладает, и индивидуальное расписание, сформированное как совокупность расписаний ролей, на которых уже работает данный специалист. На любую роль в проекте может быть определен один специалист, но любой специалист может выполнять несколько ролей, в том числе одновременно и в разных проектах.

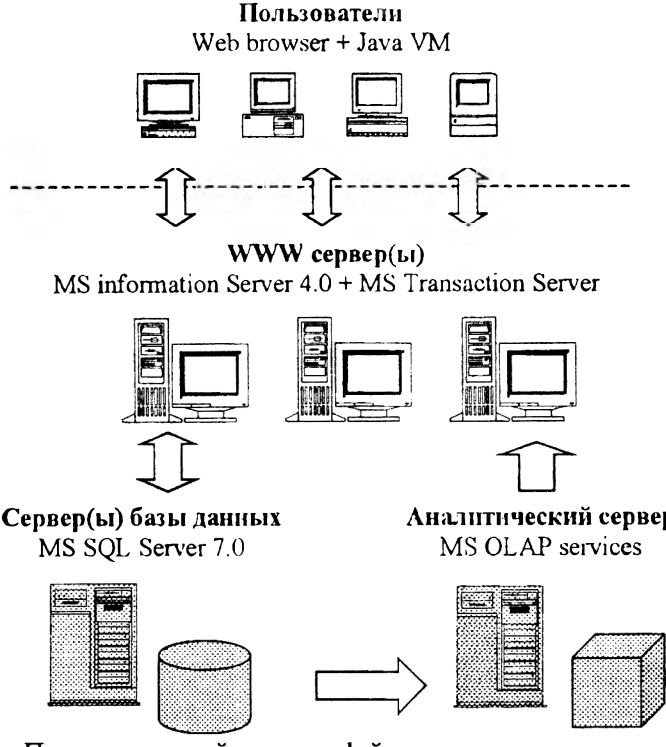
Информационная система выполняет следующие функции:

- поддерживает базу данных персонала, «внешних» специалистов и проектов;
- предоставляет возможность для руководителей проекта осуществлять автоматизированный поиск и подбор персонала на новые проекты с учетом квалификации, расписания, уровня оплаты и места выполнения проекта;
- предоставляет возможность для «внештатных» специалистов осуществлять поиск подходящих открытых позиций в проектах компании и подавать запрос на участие;
- руководитель проекта может отслеживать степень завершенности текущих проектов, уровень занятости персонала, оценивать прибыль и расходы по проектам за любой промежуток времени.

Данная информационная система обладает функциональностью системы поддержки принятия решений: перед руководителем проекта – лицом принимающим решение - стоит задача о выборе специалиста на роль в проекте, в соответствии с объективными требованиями, предъявляемыми для этой роли, и субъективными предпочтениями самого руководителя. Причем выбор происходит по нескольким критериям, в частности, по расписанию, квалификации, уровню оплаты и т.д. Множество всех специалистов можно представить как множество альтернатив с числовыми, такими как уровень оплаты, и не числовыми, такими как квалификация, характеристиками. Руководитель формулирует свои требования в виде набора утверждений (правил), после чего для всех специалистов вычисляется степень их соответствия заданным условиям, используя предложенный метод построения

функции предпочтения и руководителю выдается выборка наиболее подходящих специалистов. Следует заметить, что за счет использования предложенного модифицированного метода, основанного на нечетком выводе, информативная выборка будет получена даже в случае противоречивости заданных условий или отсутствия среди альтернатив в точности соответствующей заданным условиям, в этих случаях будут найдены наиболее «близкие» варианты. Кроме этого предложенный метод позволяет использовать нечеткость в условиях, и в описании свойств. Логическая структура данных, типы задаваемых условий и функциональность системы подробно рассматриваются в работах [4,5].

Данная экспериментальная система является Internet системой, построенной на с использованием современных инструментальных средств и технологий, таких как язык программирования Java, СУБД Microsoft SQL Server 7.0, Web сервер Internet Information Server 4.0 и средств аналитической обработки данных Microsoft OLAP Services.



Пользовательский интерфейс реализован на Java, взаимодействие с сервером происходит через HTTP. Логика функционирования системы реализована в виде набора COM компонент. База данных используется для хранения информации о специалистах и проектах, помимо этого, в ней хранятся некоторые

предварительно обработанные данные, используемые затем OLAP сервером. Аналитический сервер на базе Microsoft OLAP Server поддерживает многомерное представление хранящихся данных, на нем производится вычисление степени соответствия и выбор альтернатив в соответствии с заданными критериями. Часть функциональности реализована непосредственно средствами OLAP сервера, с помощью MDX-запросов, часть пришлось реализовывать на C++ и встраивать в OLAP сервер в виде «внешних» функций.

Заключение

В результате проведенного исследования методов теории нечетких множеств для данного класса задач, решаемых с помощью систем поддержки принятия решений, была предложена модификация метода построения функции предпочтения на основе нечеткого вывода. За счет использования рассмотренных в работе мер включения и совпадения, а также нечеткой арифметики предложенная модификация метода позволяет работать с более широким и удобным в прикладном плане набором условий. Предложенный метод был реализован в информационной системе управления персоналом на базе средств OLAP и подтвердил свою адекватность на реальных данных.

Литература

1. Трахтенгерц Э.А. Компьютерная поддержка принятия решений. -СИНТЕГ, Москва 1998г.

2. Ларичев О.И., Мошкович Е.М. Качественные методы принятия решений - М. Наука. Физматлит. 1996г.

3. А.Н. Аверкин, И.З. Батыршин, А.Ф. Блишун, В.Б. Силов, В.Б. Тарасов Нечеткие множества в моделях управления и искусственного интеллекта - М., изд-во "Наука", 1986 г.

4. Машечкин И.В., Петровский М.И. Применение элементов теории Нечетких Множеств и OLAP технологий для построения систем календарного планирования - Тезисы докладов на XIV международной научно-технической конференции "Интеллектуальные САПР-99". Таганрогский Государственный Радиотехнический университет. с. 347-349

5. Машечкин И.В., Петровский М.И. Об одном подходе к построению информационных систем подбора персонала - Журнал "Проблемы программирования" УКРПРОГ 2000г. Институт программных систем Национальной Академии Наук Украины. С. 51-62.

Костенко В.А.

Способы представления и преобразования расписаний в итерационных алгоритмах

Введение

Общая задача построения статических (априорных) расписаний заключается в распределении и упорядочивании множества фиксированных работ на заданное число ресурсов таким образом, чтобы оптимизировать желаемую меру эффективности и выполнить заданные ограничения.

Расписание является комбинаторной структурой - заданы два множества: множество ресурсов (процессоров) и работ (процессов), потребляющих ресурсы; на множестве процессов всюду определена функция привязки, которая задает распределение процессов по процессорам (в дальнейшем привязка); на множестве процессов определено отношение частичного порядка, удовлетворяющее условиям ацикличности и транзитивности (в дальнейшем порядок). Отношение порядка на множестве процессов, распределенных на один и тот же процессор, является отношением полного порядка. Различные варианты расписания могут быть получены изменением привязки и порядка в заданных ограничениями пределах. Таким образом, расписание задано, если определены: множества процессов и процессоров, привязка и порядок.

Задача составления расписаний относится к классу оптимизационных комбинаторных задач с ограничениями. Расписание может быть представлено набором целочисленных или бинарных переменных (параметров), задающими привязку и порядок. Если задача построения расписаний является подзадачей некоторой болеее общей задачи (например, требуется определить тип процессоров из допустимого набора, число процессоров каждого типа и построить расписание для определенного множества процессоров), то расписание может рассматриваться как один из параметров задачи оптимизации. В этом случае мы будем говорить, что расписание является комбинаторным параметром общей задачи.

В работах [1,2] показано, что почти все задачи построения расписаний являются *NP*-полными в строгом смысле. В данной работе на вариантах постановки задач построения расписаний, для которых существуют полиномиальные алгоритмы, останавливаться не будем. Отметим, что большинство таких задач являются практически вырожденными, поскольку на исходные условия накладываются жесткие ограничения: тип исходного отношения частичного порядка на

множестве процессов – лес или пустое множество, число процессоров менее трех, одинаковая вычислительная сложность процессов, дополнительные ресурсы не допустимы, допустимость прерывания выполнения любого процесса и возобновление его выполнения на любом из процессоров ВС. Данным ограничениям большинство реальных ВС и программ не удовлетворяют.

NP-полнота задач построения расписаний обусловила широкое применение для их решения эвристических алгоритмов, основанных на жадных стратегиях, и итерационных алгоритмов (задается начальное расписание, которое затем итерационно корректируется с целью нахождения более эффективного расписания): генетические и эволюционные алгоритмы, алгоритмы имитации отжига, алгоритмы случайного поиска (ненаправленного, направленного, направленного с самообучением), алгоритмы детерминированной коррекции расписания.

При использовании итерационных алгоритмов комбинаторный характер задачи (а именно, транзитивность отношения порядка на множестве процессов) создает проблему согласованного изменения задающих расписание переменных для получения на очередной итерации алгоритма допустимого варианта расписания (изменение привязки и порядка в заданных ограничениями пределах). В данной работе предлагается решение проблемы согласованного изменения переменных, позволяющее избежать получения недопустимых вариантов решения на всех итерациях алгоритма. Это решение построено на основе понятия инварианта поведения программы, введенного в работе [4], и модели поведения программ, предложенной в работе [3].

Для непосредственного способа представления расписания (привязка и порядок задаются явно) получена функционально полная система операций преобразования расписаний. Данные операции используются в алгоритмах случайного поиска, имитации отжига и детерминированной коррекции расписания.

Для параметрического представления расписания (привязка и порядок задаются значениями некоторых параметров, по которым, с использованием алгоритма восстановления, они могут быть восстановлены) доказана возможность представления любого допустимого варианта расписания и однозначность его восстановления. Данная форма представления используется в генетических и эволюционных алгоритмах. Параметрическое представление допускает независимое изменение значений всех переменных, задающих расписание, в заданных интервалах, что позволяет гарантированно получать допустимые варианты расписания при выполнении операций генетического алгоритма. Данная форма представления может быть использована также в алгоритмах случайного поиска и имитации отжига.

1.Инвариант поведения программы, расписание как комбинаторный параметр

Инвариант поведения программы

Следуя [3], обозначим поведение программы *Bh(PR)* и определим *Bh(PR)* следующим образом: $Bh(PR)=<S,\{R—(PR)\},\{R\rightarrow(PR)\}>$, где S – множество всех возможных шагов процессов в допустимом диапазоне входных данных программы, $\{R\rightarrow(PR)\}$ - отношения, определяющие частичный порядок на множестве шагов каждого процесса, $\{R—(PR)\}$ - отношения взаимодействия между процессами.

Шаги процесса определяются последовательностью взаимодействий с другими процессами[3]. Назовем рабочим интервалом процесса внутренние действия процесса между двумя последовательными взаимодействиями с другими процессами. Каждый рабочий интервал процесса по существу является реализацией соответствующего шага процесса.

Для задачи синтеза архитектур будем использовать одну историю поведения программы $H(PR)\in Bh(PR)$ [3]. Для *H(PR)* отношение $\{R\rightarrow (PR)\}$ является отношением полного порядка, а множество S сужается до множества шагов, которые делают процессы для конкретного набора входных данных. *Bh(PR)* может быть сведено к *H(PR)*: для программ поведение которых не зависит от исходных данных; программ в которых любой процесс имеет не более одного рабочего интервала; или использована наиболее "сложная" история или искусственно построенная история на основе *Bh(PR)*.

H(PR) можно представить ациклическим ориентированным размеченным графом. Вершинам $P=\{p_i\}_{i=1}^{N_1}\cup\{p_i\}_{i=1}^{N_2}\cup K\cup\{p_i\}_{i=1}^{N_K}$, соответствуют рабочие интервалы процессов, дугам $\pi=\{\pi_{ik}=(p_i,p_k)\}_{(i,k)\in(1...N)}$ - связи, определяющие взаимодействия между рабочими интервалами из множества P (определяются объединением отношений $\{R--(PR)\}$, $\{R\rightarrow(PR)\}$). Где N_i - число рабочих интервалов в i-ом процессе, K - число процессов в программе PR. $N=N_1+N_2+...+N_K$ - мощность множества P. Чередование рабочих интервалов различных процессов, назначенных на один и тот же процессор, допустимо, если не нарушается частичный порядок, заданный π. Отношение π_{ik} представляется следующим образом: если $p_i\,\pi_{ik}\,p_k$, то рабочий интервал p_i, необходимо выполнить до начала выполнения рабочего интервала p_k. На π накладываются условия ацикличности и транзитивности. Каждая вершина имеет свой уникальный номер и метки: принадлежности рабочего интервала к процессу и вычислительной сложности рабочего интервала. Вычислительная

сложность рабочего интервала позволяет оценить время выполнения рабочего интервала на процессоре. Дуга определяется номерами смежных вершин и имеет метку, соответствующую объему данных обмена. Объем данных обмена для каждой связи из π позволяет оценить затраты времени на выполнение внешнего взаимодействия при условии, что разделяемые ресурсы, требуемые для выполнения взаимодействия, в момент обращения свободны.

При сделанных выше допущениях используемый инвариант поведения программы будет частным случаем (одна история) инварианта поведения программы, определенном в [4]. Используемый инвариант поведения определим:

1. Множеством рабочих интервалов процессов, составляющих программу *PR*:

$$P = \{p_i\}_{i=1}^{N_1} \cup \{p_i\}_{i=1}^{N_2} \cup \mathrm{K} \cup \{p_i\}_{i=1}^{N_r}$$

Нумерация рабочих интервалов является сквозной и удовлетворяет условиям полной топологической сортировки. Каждый рабочий интервал имеет метку принадлежности к процессу.

2. Частичным порядком на *P*: $\pi = \{\pi_{ik} = (p_i, p_k)\}_{(i,k) \in (1...N)}$;
3. Вычислительной сложностью рабочих интервалов: $\{w_i\}_{i=1}^{N}$;
4. Объемом данных обмена для каждой связи из π : $\{v_{ik}\}_{(i,k) \in (1...N)}$;.

Расписание

Модель расписания выполнения программы определим набором простых цепей и отношением частичного порядка π_{HP} на множестве *P*: $HP = (\{SP_i\}_{i=(1..M)}, \pi_{HP})$. Где $\{SP_i\}_{i=(1..M)}$ -набор простых цепей (ветвей параллельной программы). Они образуются рабочими интервалами процессов, распределенными на один и тот же процессор (*M* – число процессоров в ВС). Отношение частичного порядка π_{HP} на множестве *P* для *HP* определим как объединение отношений: $\pi_{HP} = \pi_c \cup \pi_1 \cup \ldots \cup \pi_M$, π_i - отношение полного порядка на SP_i, которое определяется порядковыми номерами рабочих интервалов в SP_i; π_c - набор секущих ребер, которые определяются связями рабочих интервалов, распределенных на разные процессоры. Если рабочие интервалы p_i и p_j распределены на разные процессоры и в π существует связь π_{ij}, то она определяет секущее ребро в *HP*.

Модель расписания можно рассматривать как граф *HP*, вершины которого имеют дополнительную метку "номер списка", а дуги определяются отношением π_{HP} или как граф *H*, вершины которого доразмечены двойками: {"номер списка", "порядковый номер вершины в соответствующем списке"}. В модели *HP* сохраняются нумерация вершин, дуг и их метки заданные в модели поведения программы *H*.

Таким образом, расписание выполнения программы определено, если для каждого рабочего интервала из множества P заданы: 1)привязка к одному из списков SP_i, $i \in (1...M)$; 2)порядковый номер в соответствующем списке.

Выделим два способа представления расписания:

1. непосредственное представление - для каждого рабочего интервала значения привязки и порядкового номера заданы явно;
2. параметрическое представление - для каждого рабочего интервала значения привязки и порядкового номера заданы значениями некоторых параметров, по которым, с использованием алгоритма восстановления, определяются значения привязки и порядкового номера.

В дальнейшем при рассмотрении свойств расписаний в некоторых разделах будет использоваться ярусная форма представления расписания. Любое допустимое расписание HP в силу ограничения "отношение π_{HP} ациклично" представляет собой ациклический ориентированный граф. Любой ациклический ориентированный граф всегда может быть представлен в ярусной форме [5]: на одном ярусе могут находиться лишь вершины не связанные транзитивным отношением частичного порядка; предшественники всегда находятся на более высоком ярусе (с меньшим номером), чем последователи. Верно также и обратное, если ориентированный граф представлен в ярусной форме, то он является ациклическим. Ярусная форма называется канонической, если вершины распределены по ярусам таким образом, что для любой вершины на предшествующем ярусе имеется непосредственный предшественник. Ярусной формой максимальной высоты будем называть такую ярусную форму, у которой на каждом ярусе находится не более одной вершины.

Ограничения на расписание

Расписание HP является допустимым (сохраняет инвариант поведения программы), если выполнены следующие ограничения:

1. Каждый рабочий интервал должен быть назначен на процессор (в SP_i): $\forall p_i \in P \Rightarrow \exists m : p_i \in SP_m$.
2. Каждый рабочий интервал должен быть назначен лишь на один процессор (в один SP_i): $\forall p_i \in SP_j \Rightarrow \neg \exists m \neq j : p_i \in SP_m$.
3. Частичный порядок, заданный в H должен быть сохранен в HP: $\pi \subset \pi^T_{HP}$, π^T_{HP} – *транзитивное замыкание* π_{HP}.
4. Расписание HP должно быть беступиковым. Условием беступиковости является отсутствие контуров в графе HP:

π_{HP} – *ациклично*.

5. Все рабочие интервалы одного процесса должны быть назначены на один и тот же процессор (в один и тот же SP_i): $\forall p_i, p_j \in \{p_l\}_{l=1}^{N_k} \Rightarrow \exists m : p_i, p_j \in SP_m$

Ограничения 1-4 являются обязательными для любого варианта постановки задачи и обеспечивают сохранение инварианта поведения программы, определяемого *H*. Ограничение 5 запрещает возобновление работы процесса после прерывания на другом процессоре, что соответствует способу организации параллельных вычислений в большинстве ВС. В дальнейшем будем говорить, что расписание допустимо $HP \in HP^{\bullet}_{1-5}$, если оно удовлетворяет ограничениям. Нижний индекс в $HP^{\bullet}_{1-5}$ указывает ограничения налагаемые на расписание.

2.Задача построения расписаний и методы ее решения

Задача построения расписаний

В задачах построения расписаний обычно минимизируется время его выполнения *T* при заданных ограничениях на аппаратные ресурсы или время выполнения ограничивается $T \le T^{dir}$ при условии использования необходимого минимума аппаратных ресурсов. В качестве меры эффективности расписания могут быть использованы и другие критерии [1]: среднее взвешенное время завершения, время ожидания обслуживания, среднее количество процессов, выполненное на заданном временном интервале.

В дальнейшем для определенности будем рассматривать время выполнения расписания: *T=f(HP,HW)*, где *HP*– модель расписания, *HW*-модель архитектуры вычислительной системы. Оценка значения *T* для конкретных вариантов решения задачи может быть получена совместной интерпретацией моделей *HP* и *HW*. Функция *T=f(HP,HW)* задана алгоритмически [6]: правилами/алгоритмами ее вычисления.

Выбор наиболее подходящего метода решения как правило определяется сложностью функции *T=f(HP,HW)*. Сложность функции *T=f(HP,HW)* будем характеризовать:

1) размером моделей *HP,HW*:
   - */HP/* - число рабочих интервалов (*N*), число процессов (*K*), мощность исходного отношения порядка (/ π /),
   - */HW/* - число процессоров (*M*), число разделяемых ресурсов (M^{EAS}), число уровней памяти (M^{MEM}), число вызываемых системных процессов (M^{SYS});
2) структурой модели *H*:
   - тип π ,

- дисперсия $\{w_i\}_{i=1}^{N}$;

3) набором временных задержек, учитываемых динамически:
   - t^{EP}, $t^{D\pi}$,
   - t^{EP}, $t^{D\pi}$, t^{EAS},
   - t^{EP}, $t^{D\pi}$, t^{EAS}, t^{SYS},

   где t^{EP}- временные задержки, обусловленные занятостью процессоров, $t^{D\pi}$ - ожиданием данных от других рабочих интервалов, t^{EAS}- ожиданием получения доступа к разделяемым ресурсам и t^{SYS}- задержки вносимые логической средой и доступом к памяти, которые не могут быть учтены статически;

4) законом распределения значений T: $\rho_i=\rho(T_i)$. Значение функции $\rho(T_i)$ равно относительному числу допустимых вариантов расписания для которых значение функции $T=f(HP,HW)$ равно T_i.

Учет возможных временных задержек при интерпретации также определяет уровень детализации моделей HP,HW и соответственно, точность вычисления функции $T=f(HP,HW)$ для различных классов архитектур вычислительных систем, то есть применимость функции $T=f(HP,HW)$. Например, для полносвязной архитектуры достаточно учитывать лишь задержки t^{EP}, $t^{D\pi}$, а для архитектур с коммутационной средой, в которой возможно ожидание получения канала связи, учет лишь t^{EP}, $t^{D\pi}$ будет приводить к неточным оценкам значения T.

Методы решения задачи построения расписаний

Алгоритмы, основанные на жадных стратегия подразумевают декомпозицию задачи на подзадачи (вложении задачи в семейство более простых задач) и предполагают выбор для подзадач локальных целевых функций, которые не имеют глобального характера. Следующие свойства ограничивают применение жадных алгоритмов лишь для архитектур или уровней рассмотрения архитектуры для которых целевая функция $T=f(HP,HW)$ достаточно точно задается лишь при учете задержек t^{EP}, $t^{D\pi}$.

Свойство 1. Для жадных алгоритмов решения NP-полных задач составления расписаний не существует единой для всех подзадач локальной целевой функции, приводящей к наилучшему варианту расписания, получаемого алгоритмом.

Свойство 2. Для жадных алгоритмов решения NP-полных задач составления расписаний влияние на качество расписания используемых локальных целевых функций возрастает с ростом сложности функции $T=f(HP,HW)$ по учитываемым временным задержкам.

Итерационные алгоритмы построения расписаний имеют по сравнению с жадными алгоритмами как правило более высокую вычислительную сложность, однако они свободны от основного

недостатка – ограниченности применения, присущего жадным алгоритмам. Схематично работу итерационных алгоритмов для решения задачи построения расписания можно представить следующим образом:

1. Задать начальное приближение HP^0, $k=0$.
2. Вычислить целевую функцию $f(HW,HP^k)$ (если в п.3 возможно получение недопустимых значений HP, то проверить выполнение ограничений 1-5).
3. Получить $HP^{k+1}=D(\{HP^i\},\{f(HW,HP^i)\},Pr^k: i\in(1,\ldots,k))$.
4. Если критерий останова не достигнут, то $k=k+1$ и перейти к п.2; в противном - завершить работу алгоритма.

Где, D – некоторая стратегия коррекции текущего расписания, Pr^k – параметры стратегии (для ряда стратегий, возможно, их изменение в ходе работы алгоритма).

В генетических алгоритмах [6,7,8] система операций преобразования расписаний определена: операции скрещивания, мутации и селекции. Требуется выбрать такую форму представления расписаний, чтобы операции генетического алгоритма не приводили к получению недопустимых вариантов расписаний и любое допустимое расписание могло быть представлено в выбранной форме представления расписаний. В алгоритмах имитации отжига, случайного поиска, детерминированной коррекции расписаний [7,9,10] система операций преобразования расписаний вводится при разработке алгоритма. Для этих алгоритмов способ представления расписаний и система операций преобразования расписаний должными обладать следующими свойствами: применение операции должно приводить к допустимому варианту расписания и система операций должна быть функционально полной.

Использование в итерационных алгоритмах системы операций преобразования расписаний, допускающей получение расписаний $HP \notin HP^*_{1-5}$, приводит к необходимости введения в функцию $T=f(HP,HW)$ слагаемых отвещающих за штрафы при нарушении ограничений 1-5. При этом возникают проблемы корректной оценки значений T для неинтерпретируемых расписаний и искажения пространства решений в случае введения штрафов для $HP \notin HP^*_{1-5}$.

3. Непосредственное представление и операции преобразования расписания

Бинарное непосредственное представление расписания

Расписание задается: матрицей привязки $Y(HP)_{N\times M}$ и матрицей смежности $X(HP)_{N\times N}$ графа HP, где элемент матрицы определяется:

$$y_{ij} = \begin{cases} 1, \text{если } p_i \in SP_j \\ 0, \text{если } p_i \notin SP_j \end{cases} \qquad x_{ij} = \begin{cases} 1, \text{если } \pi_{ij} \in \pi_{HP} \\ 0, \text{если } \pi_{ij} \notin \pi_{HP} \end{cases}$$

M – число процессоров в ВС, N – число рабочих интервалов в H. Недостатком данного представления является большое число бинарных переменных - $N^2 \cdot N \cdot M$.

Целочисленное непосредственное представление расписания

1.Расписание задается матрицей $Y(HP)_{N \times M}$, где элемент матрицы определяется: $y_{ij} = \begin{cases} c, \text{если } p_i \in SP_j \\ 0, \text{если } p_i \notin SP_j \end{cases}$, c – порядковый номер рабочего интервала p_i в SP_j. При данном способе представления расписания число целочисленных переменных равно $N \cdot M$.

2.Расписание задается: вектором привязки $Y(HP)_K$ и вектором порядка $X(HP)_N$, где i-й элемент вектора $Y(HP)_K$ равен номеру списка в который назначены рабочие интервалы i-го процесса, а i-й элемент вектора $X(HP)_N$ равен порядковому номеру рабочего интервала в соответствующем списке. При данном способе представления расписания число целочисленных переменных равно $K + N$.

Операции преобразования расписания

Возможны следующие варианты отличия расписаний HP и HP' друг от друга: 1)расписания HP и HP' отличаются лишь порядком рабочих интервалов как минимум в одном SP_j; 2)расписания HP и HP' отличаются привязкой рабочих интервалов.

Введем соответствующие операции преобразования расписаний, позволяющие устранить указанные варианты отличия: $O=\{O1,O2\}$. Операции $O1,O2$ определим для целочисленного непосредственного представления расписания. Операции будем определять при предположениях: каждый процесс имеет не более одного рабочего интервала или при условии, что на расписание не накладывается ограничение 5. После теоремы о функциональной полноте системы операций $O=\{O1,O2\}$ и получении условий их применимости покажем возможность снятия указанных предположений.

Операция изменения порядка рабочих интервалов в одном списке:

$$O1(p_i, SP_m, c) \equiv \begin{cases} \{c1 = y_{im},\ y_{im} = c \in (1,\mathrm{K}\ , c1-1); \forall j \mid\ y_{jm} \neq 0 \vee c \le y_{jm} < c1: \\ y_{jm} = y_{jm} + 1\} - \textit{уменьшение порядкового номера} \\ \{c1 = y_{im},\ y_{im} = c \in (c1+1,\mathrm{K}\ , NS_m); \forall j \mid\ y_{jm} \neq 0 \vee c1 < y_{jm} \le c: \\ y_{jm} = y_{jm} - 1\} - \textit{увеличение порядкового номера} \end{cases}$$

Операция изменяет порядковый номер рабочего интервала p_i в списке SP_m (порядковый номер рабочего интервала становится равным c) и корректирует порядковые номера соответствующих рабочих интервалов в данном списке. NS_m – число рабочих интервалов в списке SP_m.

Операция изменения привязки рабочих интервалов:

$$O2(p_i, SP_m \to SP_k, c) \equiv \{y_{ik} = c \in (1\mathrm{K}\ NS_k + 1);$$
$$\forall j \mid\ y_{jk} \neq 0 \vee c \le y_{jk} : y_{jk} = y_{jk} + 1,\ NS_k = NS_k + 1;$$
$$\forall j \mid\ y_{jm} \neq 0 \vee y_{im} < y_{jm} : y_{jm} = y_{jm} - 1,\ NS_m = NS_m - 1,\ y_{im} = 0\}$$

Операция переносит рабочий интервал p_i из списка SP_m в список SP_k (порядковый номер рабочего интервала становится равным c) и корректирует порядковые номера соответствующих рабочих интервалов в списках SP_m и SP_k.

Теорема 1. Если HP и HP' произвольные допустимые варианты расписания ($HP, HP' \in HP^{*}_{1-4}$), то существует конечная цепочка команд $\{O_i\}_{i=1}^{K}$, $O_i \in \{O1, O2\}$, переводящая расписание HP в HP', такая, что все K промежуточных расписаний являются допустимыми и $K \le 2N$.

Следствие 1. Существует стратегия перехода от произвольного допустимого расписания к оптимальному расписанию, такая, что длина цепочки команд $\{O_i\}_{i=1}^{K}$, $O_i \in \{O1, O2\}$, переводящей произвольное допустимое расписание в оптимальное, не превосходит значения $2N$ ($K \le 2N$) и все K промежуточных расписаний являются допустимыми.

Условия применимости операций не нарушающие ограничений на HP

Получим интервал значений параметра c при применении операции *O1/O2* к рабочему интервалу p_i^s. Расписания HP будем представлять в ярусной форме максимальной высоты. Обозначим через $IN_i = \{p_k^l : \pi_{ki} \neq 0\}$ - множество непосредственных предшественников рабочего интервала p_i^s (всегда выполняется $k<i$ и $l<s$), $OUT_i = \{p_k^l : \pi_{ik} \neq 0\}$ - множество непосредственных последователей рабочего интервала p_i^s (всегда выполняется $k>i$ и $l>s$). Операция $lin = \max_{IN_i}(l)$ - получает максимальный номер яруса, на котором

расположен один из непосредственных предшественников рабочего интервала p_i^s, для рабочих интервалов, не имеющих предшественников $lin=0$ (нулевой ярус всегда пуст). Операция $lout = \min_{OUT_i}(l)$ - получает минимальный номер яруса, на котором расположен один из непосредственных последователей рабочего интервала p_i^s, для рабочих интервалов, не имеющих последователей $lout=N$ (N – число ярусов в HP, для ярусной формы максимальной высоты число ярусов всегда равно числу рабочих интервалов). Тогда, рабочий интервал p_i^s может быть размещен в любой из списков SP_j на любой из ярусов из интервала $lin<s<lout$. Если выбранный ярус занят, то осуществляется коррекция ярусной формы путем соответствующего сдвига ярусов.

Пусть рабочий интервал p_i^s переносится в SP_j или изменяется его порядковый номер в этом списке. Разобьем множество SP_j на три подмножества: $PIN_j = \{p_k^l : l \geq lin,\ p_k^l \in SP_j\}$ - множество рабочих интервалов из списка SP_j, находящихся на ярусах, расположенных выше яруса $(lin-1)$; $PI_j = \{p_k^l : lin < l < lout,\ p_k^l \in SP_j\}$ - множество рабочих интервалов из списка SP_j, находящихся на ярусах $]lin, lout[$; $POUT_j = \{p_k^l : l \leq lout,\ p_k^l \in SP_j\}$ - множество рабочих интервалов из списка SP_j, находящихся на ярусах, расположенных ниже яруса $(lout+1)$. Параметр c при размещении рабочего интервала p_i^s в SP_j может принимать следующие значения, не нарушающие ограничения на HP (индекс j для PIN, PI, $POUT$ будем опускать):

	PIN	PI	POUT	C
1	$PIN=\varnothing$	$PI=\varnothing$	$POUT=\varnothing$	$c=1$
2	$PIN=\varnothing$	$PI=\varnothing$	$POUT\neq\varnothing$	$c=1$
3	$PIN=\varnothing$	$PI\neq\varnothing$	$POUT=\varnothing$	$1 \le c \le \max_{PI}(y_{jk})+1$
4	$PIN=\varnothing$	$PI\neq\varnothing$	$POUT\neq\varnothing$	$1 \le c \le \max_{PI}(y_{jk})+1$
5	$PIN\neq\varnothing$	$PI=\varnothing$	$POUT\neq\varnothing$	$c = \min_{PIN}(y_{jk})+1$
6	$PIN\neq\varnothing$	$PI=\varnothing$	$POUT=\varnothing$	$c = \min_{PIN}(y_{jk})+1$
7	$PIN\neq\varnothing$	$PI\neq\varnothing$	$POUT=\varnothing$	$\min_{PI}(y_{jk}) \le c \le \max_{PI}(y_{jk})+1$
8	$PIN\neq\varnothing$	$PI\neq\varnothing$	$POUT\neq\varnothing$	$\min_{PI}(y_{jk}) \le c \le \max_{PI}(y_{jk})+1$

Указанные выше интервалы значений параметра c не нарушающие ограничения на HP могут быть расширены, если ярусная форма максимальной высоты будет приведена к такому виду, что все рабочие интервалы SP_j, которые не находятся в отношении транзитивного порядка с непосредственным предшественником рабочего интервала p_i^s, находящимся на ярусе lin, и расположены на ярусах с меньшими номерами чем lin, будут перенесены на ярусы с номерами большими чем lin. Аналогичное преобразование ярусной формы может быть осуществлено и для непосредственного последователя рабочего интервала p_i^s, находящимся на ярусе $lout$.

Возможность снятия условий: каждый процесс имеет не более одного рабочего интервала или на расписание не накладывается ограничение 5

Поскольку, при доказательстве теоремы 1 и получении условий применимости операций *O1/O2* использована ярусная форма максимальной высоты (на каждом ярусе находится лишь один рабочий интервал), то полученные результаты легко могут быть обобщены на случаи когда: каждый процесс может иметь более одного рабочего интервала или на расписание накладывается ограничение 5. При перемещении рабочего интервала из одного списка в другой, перемещаются также и все рабочие интервалы процесса, которому принадлежит перемещаемый рабочий интервал, но при этом остаются на прежних ярусах. В результате получаем ярусную форму нового расписания, и, следовательно, полученное расписание удовлетворяет ограничениям 3-5.

Операции преобразования расписания в алгоритмах имитации отжига и случайного поиска

Получение расписания на очередной итерации алгоритма может быть осуществлено как случайный выбор:

- рабочих интервалов, к которым следует применить операцию из набора *{O1,O2}*, диапазон номеров интервалов: *[1,...,N]*;
- типа применяемой операции *O1/O2* (если операция *O2* выбрана для применения к рабочему интервалу некоторого процесса, то для остальных рабочих интервалов этого процесса могут применяться лишь операции *O1*);
- параметра операций *c* из интервала, определяемого условиями применимости операций, не нарушающими ограничений на *HP*;
- для операции *O2*: списка SP_i, в который переносится рабочий интервал.

Длина шага α_k равна числу применяемых операций.

4. Параметрическое представление расписания

Параметрическое представление расписаний с использованием приоритетов

Расписание задается вектором Y_{N+K}:

$$Y_{N+K} \equiv (\mathop{\mathrm{Y}}_{i=1}^{K} \langle YPR \rangle_i)$$

$$\langle YPR \rangle_i \equiv (\langle YE \rangle_i \, \mathrm{Y} \, \langle YP \rangle$$

$$\langle YP \rangle \equiv (\mathop{\mathrm{Y}}_{j=1}^{N_i} \langle YP \rangle_j)$$

K - число процессов в *H*, N_i – число рабочих интервалов в *i*-ом процессе, $N = \sum_{i=1}^{K} N_i$ - число рабочих интервалов в *H*.

Параметр $\langle YE \rangle_i \in [1,\mathrm{K},K] \subset Z$ содержит номер процессора, на котором выполняются рабочие интервалы *i*-го процесса, т.е. параметры $\langle YE \rangle$ однозначно определяют распределение рабочих интервалов по *SP* (привязку). Параметры $\langle YE \rangle$ могут принимать значения от 1 до *K*. Если значения всех параметров $\langle YE \rangle$ равны, то все рабочие интервалы выполняются на одном процессоре, если все значения различны, то рабочие интервалы каждого процессора выполняются на своем процессоре. В силу ограничения 5, максимальное число не пустых процессоров равно числу процессов *K* в *H*. Параметры

$\langle YP \rangle \in [1,\mathrm{K}, L] \subset Z$ используются алгоритмом восстановления расписания (определение отношения полного порядка в каждом SP_i) в качестве приоритетов рабочих интервалов.

При данном способе представления расписания число целочисленных переменных равно $N+K$.

Для описания алгоритма восстановления, множество рабочих интервалов разобьем на три подмножества: $P = P1 \cup P2 \cup P3$. *P1* - множество рабочих интервалов распределенных в *SP* (определен порядковый номер рабочего интервала) алгоритмом восстановления на предыдущих шагах; *P2* - множество рабочих интервалов, у которых все предшественники принадлежат *P1*; *P3* - множество рабочих интервалов, у которых хотя бы один из предшественников не принадлежит *P1*.

Алгоритм восстановления строго порядка рабочих интервалов в SP (A1).

1. Начальное разбиение множества *P*:

$P1 = \varnothing$, $P2 = \{p_j : \forall j, k \in [1,\mathrm{K}, N] \mid \pi_{jk} = \varnothing\}$ - множество рабочих интервалов в *H* без предшественников; $P3 = \{p_j : p_j \in P \setminus P2\}$ - - множество рабочих интервалов в *H*, у которых имеются предшественники.

2. Находим в *P2* рабочий интервал с наименьшим значением параметра $\langle YP \rangle$ (если таких интервалов более одного, то выбираем интервал с наименьшим номером):

- размещаем его в конец соответствующего списка *SP* (номер списка определяется значением параметра $\langle YE \rangle$);
- переносим его из *P2* в *P1*.

3. Проверяем *P3* с целью возможности переноса рабочих интервалов в *P2*:

если есть рабочие интервалы, у которых все предшественники принадлежат *P1*, то переносим их в *P2*.

4. Если $P2 \neq \varnothing$, то к п.2, иначе завершить работу.

Утверждение 1. Алгоритм *A1* восстановления расписания по его параметрическому представлению Y_{N+K} получает расписание $HP \in HP^*_{1-5}$ и расписание восстанавливается однозначно.

Теорема 2. Любое допустимое произвольное расписание $HP \in HP^*_{1-5}$ может быть задано параметрическим представлением с использованием приоритетов (Y_{N+K}) и однозначно восстановлено алгоритмом *A1*, если допустимая верхняя граница *L* значений параметров $\langle YP \rangle$ больше или равна числу рабочих интервалов ($L \geq N$).

Операции преобразования расписания

При параметрическом представлении расписания с использованием приоритетов операции преобразования расписания могут быть введены как операции изменения значений параметров $\langle YE \rangle$ и $\langle YP \rangle$:

1) параметры $\langle YE \rangle$ могут принимать целочисленные значения в диапазоне $[1,\ldots,K]$,
2) параметры $\langle YP \rangle$ могут принимать целочисленные значения в диапазоне $[1,\ldots,L]$,
3) количество изменяемых параметров может изменяться от 1 до $N+K$.

Операции преобразования параметрической формы представления расписаний, удовлетворяющие условиям 1-3 при использовании алгоритма восстановления $A1$, будут получать расписания $HP \in HP^{*}_{1-5}$, что следует непосредственно из утверждения 1. Из теоремы 2 следует, что данные операции преобразования расписания и алгоритм восстановления $A1$ позволяют получить любой допустимый вариант расписания.

Параметрическая форма представления расписаний может быть использована как в генетических и эволюционных алгоритмах, так и в алгоритмах имитации отжига и случайного поиска. Операции преобразования расписания в алгоритмах имитации отжига и случайного поиска могут быть введены как операции случайного выбора изменяемых параметров $\langle YE \rangle, \langle YP \rangle$ и их значений из допустимого диапазона. Длина шага α_k равна количеству изменяемых параметров.

Параметрическое представление расписаний с использованием характеристик рабочих интервалов

Расписание задается вектором Y_N:

$$Y_N \equiv (\overset{K}{\underset{i=1}{\mathrm{Y}}}\langle YE \rangle_i)$$

Параметр $\langle YE \rangle_i \in [1,\mathrm{K},K] \subset Z$ содержит номер процессора, на котором выполняются рабочие интервалы i-го процесса, т.е. параметры $\langle YE \rangle$ однозначно определяют распределение рабочих интервалов по SP (привязку). Параметры $\langle YE \rangle$ могут принимать значения от 1 до K. При данном способе представления расписания число целочисленных переменных равно K.

Алгоритм восстановления $A1$ вместо значений параметров $\langle YP \rangle$ в качестве приоритетов может использовать характеристики рабочих интервалов: вычислительную сложность рабочего интервала, число непосредственных последователей, число непосредственных предшественников, априорно заданные приоритеты. Для данного способа параметрического представления расписания справедливо утверждение 1, однако, теорема 2 не выполняется. При построении алгоритмов оптимизации, использующих данный способ представления расписания, следует обращать внимание на принципиальную возможность получения расписания с требуемым качеством.

Заключение

В работе предложены непосредственные и параметрические способы представления расписаний и соответствующие системы операций преобразования расписаний, обладающие свойствами: 1)функциональная полнота, 2)длина цепочки операций для перехода от одного произвольного допустимого расписания к другому не превосходит удвоенного числа работ, подлежащих планированию, 3)применение операций не приводит к получению недопустимых (неинтерпретируемых и нарушающих инвариант поведения программы) расписаний. Данные формы представления и системы операций используются для построения вычислительно эффективных итерационных алгоритмов построения расписаний: генетические и эволюционные алгоритмы, алгоритмы имитации отжига, алгоритмы случайного поиска (ненаправленного, направленного, направленного с самообучением), алгоритмы детерминированной коррекции расписания.

Литература

1. Теория расписаний и вычислительные машины/ Под ред Э.Г.Коффмана. М.: Наука, 1984. - 334с.
2. Гэри М., Джонсон Д. Вычислительные машины и трудно решаемые задачи. - М.: Мир, 1982. - 416с.
3. Смелянский Р.Л. Модель функционирования распределенных вычислительных систем// Вестн. Моск. Ун-та. сер 15, Вычисл. Матем. и Кибернетика. 1990, No. 3, стр. 3-21.
4. Смелянский Р.Л. Об инварианте поведения программ// Вестн. МГУ, сер. 15, Вычислительная математика и Кибернетика, 1990., No. 4, С. 54-60.
5. Воеводин В.В. Математические модели и методы в параллельных процессах. – М.: Наука, 1986.
6. Костенко В.А. Принципы построения генетических алгоритмов и их использование для решения задач оптимизации//

Труды IV Международной конференции "Дискретные модели в теории управляющих систем" (19-25 июня 2000 г.) – М.: МАКС Пресс, 2000., С.49-55.

7. Костенко В.А. Алгоритмы оптимизации, опирающиеся на метод проб и ошибок, в совместном проектировании аппаратных и программных средств ВС// Труды Всероссийской научной конференции "Высокопроизводительные вычисления и их приложения" (30 октября - 2 ноября 2000 г., г. Черноголовка). -М.: Изд-во МГУ, 2000., С.123-127.

8. Костенко В.А., Смелянский Р.Л., Трекин А.Г. Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов// Программирование, 2000., №5, С.63-72.

9. Л.А. Растригин. Статистические методы поиска.- М.: Наука, 1968.

10. Уоссермен Ф. Нейрокомпьютерная техника. Теория и практика.- М.: Мир, 1992.

А.В. Никитин

Эволюционный алгоритм оптимизации ассоциативной памяти.

1. Введение.

Одной из проблем, стоящих на пути создания вычислительных машин потока данных, является построение устройства, позволяющего выявлять готовые к обработке операнды (токены) и формировать из них пары [1]. В динамической модели потока данных операция подлежит выполнению при совпадении значений ключей токенов на ее входе. При обнаружении такого совпадения машина объединяет токены в пару для последующих вычислений на исполнительных устройствах. Так, например, в Манчестерской машине использовалось несколько блоков: память ожидания совпадений, объединяющее устройство и устройство переполнения [1, 2]. В связи с необходимостью получения максимальной производительности системы исключена возможность построения памяти объединения на базе обычной прямоадресуемой памяти с использованием программно – аппаратных средств. В работе [1] предлагается использовать память совпадения на аппаратной основе, а именно, на базе ассоциативной памяти (АП) выборки информации по ключам.

В настоящее время проводятся исследования по созданию оптической АП [3, 4]. Оптическая ассоциативная память обладает рядом преимуществ по сравнению с АП, реализованной на полупроводниковой основе. Важнейшими из преимуществ являются возможность параллельного одновременного сравнения на совпадение нескольких ключей, а также превосходящий в 10 раз темп считывания. Все это позволяет существенно ускорить работу потоковой машины в целом.

К недостаткам подобного подхода следует отнести высокую стоимость АП, а также тот факт, что предельная производительность ее работы обратно пропорциональна объему [3]. Технологическим решением, позволяющим повысить производительность работы АП при сохранении достаточно большого объема, является разбиение общей АП на независимые модули и использование хэширования при обращении к модулям. При этом пропускная способность всей памяти повышается как за счет увеличения количества модулей, так и за счет увеличения скорости работы самого модуля. Но, с другой стороны, возникает проблема переполнения модуля памяти, которая приводит к невозможности доступа к информации в модуле, а в итоге - к полной остановке машины. За распределение информации по модулям отвечает

функция хэширования. Необходимо организовать хэширование так, чтобы позволить оптимальным образом с минимальным временем вычисления алгоритма распределять данные между модулями.

В данной статье предложена математическая модель модульной АП, позволяющая определять оптимальную конфигурацию АП и исследовать работу хэширования. На основе данной модели поставлена оптимизационная задача для определения оптимального хэширования.

При выборе метода оптимизации необходимо учитывать большой размер реальных графов потока данных. Данный факт делает точные (персборные) методы малопригодными. В качестве метода оптимизации был выбран генетический алгоритм (ГА) [5, 6].

ГА успешно применяется для поиска решения целого ряда NP – сложных задач большой размерности, в том числе для поиска разрезания графов [7-9]. К такому классу задач относится и сформулированная в статье проблема поиска оптимального хэширования. ГА также обладает высокой степенью параллелизма, что позволяет эффективно его использовать на вычислительных системах с параллельной архитектурой.

2. Графовая модель вычислительного алгоритма.

Вычислительный алгоритм для машины потока данных будем представлять в виде графа вычислительного процесса без переиспользования цепей графа для различных данных [1]. Каждая вершина графа имеет уникальный номер - ключ. Кодируется граф набором троек, состоящих из кода операции и двух указателей (ключей) на вершины, в которые необходимо передать полученный при выполнении операции результат.

В настоящей работе будем использовать ориентированный ациклический граф потока данных алгоритма. Вершины графа определяют операции, которые необходимо выполнить над данными. Дуги графа соответствуют потоку данных между вершинами графа и указывают направление этого потока. Будем задавать граф алгоритма в виде $\Gamma_a = <n, C, S>$.

Здесь n – число вершин графа, $C = \{c_l, l = 1..n\}$ – вектор, элементы которого определяют операции, выполняемые в соответствующих вершинах графа, а S – матрица смежности графа. Аппаратная организация рассматриваемой машины потока данных требует, чтобы число предков у вершин графа не превышало двух. В случае наличия в алгоритме операций с большим числом аргументов необходимо преобразовать граф за счет введения «фиктивных» вершин, содержащих команды-фишки [1].

3. Поярусное разложение графа алгоритма.

Ациклические орграфы допускают поярусное разложение [10]. На первый ярус помещаются вершины, которые не имеют входных дуг. На каждый последующий ярус помещаются те вершины, которые не имеют предшественников, за исключением уже распределенных на предыдущие ярусы. Обозначим число таких ярусов за h. А само поярусное распределение будем представлять в виде матрицы H размерности hxn, элементы которой H_{kj} равны 1, если j – ая вершина принадлежит k – ому ярусу, и нулю – в противном случае.

Вершины, принадлежащие одному ярусу такого разложения, не связаны отношением предшествования. Поэтому, если отвлечься от ресурсов вычислительной системы и, следовательно, от временных факторов протекания вычислительного процесса, можно считать, что операции, соответствующие вершинам одного яруса, будут выполняться одновременно. А, как следствие, результаты этих операций одновременно поступят на вход АП.

При вычислении алгоритма исполняющие устройства выполняют операции, соответствующие вершинам графа алгоритма. Полученный результат объединяется с номером вершины - потомка и поступает на вход АП. Такая совокупность называется токеном.

Токены, соответствующие одному ярусу алгоритма, поступят на вход АП одновременно. Часть из них после обработки яруса останется в АП, другие найдут в АП для себя пару и будут отправлены на дальнейшее исполнение. Разделим токены на следующие 3 типа:

1. P-тип. Токены, для которых пары находятся на том же ярусе. Такие токены будут записаны и считаны из АП во время обработки текущего яруса. Число таких токенов на i - ом ярусе обозначим за P_i.
2. R-тип. Токены, для которых пары уже находились в АП к моменту начала обработки текущего яруса. При обработке таких токенов в АП произойдет выборка ранее записанных токенов, что освободит место в АП. Число таких токенов на i- ом ярусе обозначим за R_i.
3. W-тип. Токены, для которых пары поступят при обработке последующих ярусов. Такие токены будут записаны в АП и останутся в ней после окончания обработки данного яруса. Число таких токенов на i - ом ярусе обозначим за W_i.

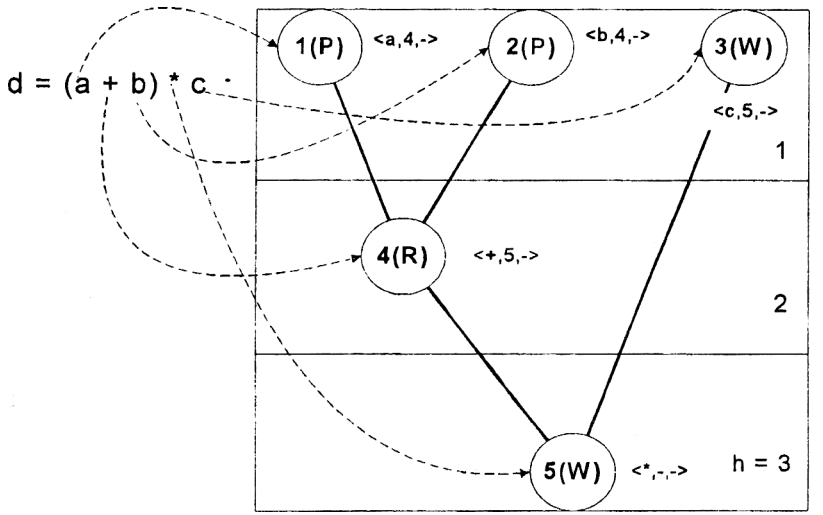


Рис. 1. Пример графа потока данных. В скобках указаны типы токенов.

На рис. 1. приведен пример графа потока данных получаемый при вычислении выражения $d = (a+b)*c$, где a, b, c – некоторые значения. Для каждой вершины графа указаны ключ и тип вершины (в круглых скобках). В угловых скобках заданы код команды и ключи вершин – приемников результата.

4. Ассоциативная память в вычислительных системах потока данных.

Использование памяти совпадения на аппаратной основе, а именно на базе АП выборки информации по ключам, позволяет решить целый ряд проблем при выявлении готовых к исполнению токенов и формировании из них пар.

Такая АП выполняет несколько команд в зависимости от кода операции. Основные из них – поиск парного токена по ключу и считывание его в случае нахождения, или запись вновь пришедшего токена на свободное место, если поиск закончился отрицательно. В дальнейшем будем рассматривать случай, когда считывание происходит со стиранием информации. Таким образом, при нахождении пары (по совпадению ключей) для токена происходит одновременное считывание и стирание обоих токенов из АП. Будем рассматривать ассоциативную

память выборки информации по ключам, построенную на оптических принципах [4].

Максимальная производительность оптической АП определяется следующим соотношением:

$$\Pi_{max} = P_{доп}\eta / 2nWM , \quad (1)$$

где W – минимальная энергия срабатывания фотоприемника, необходимая для различения нуля и единицы. Как показывает практический опыт, следует рассматривать $W = 10^{-14}$ Дж;

$P_{доп}$ – предельно допустимая мощность устройства. Учитывая возможности отвода тепла с поверхности устройства, зададим $P_{доп} = 10^4$ Вт;

$\eta = \eta_1\eta_2$, где η_1 и η_2 – коэффициенты, учитывающие потери в лазере и оптической системе, соответственно. В дальнейшем будем рассматривать $\eta = 10^{-2}$;

M – объем памяти ключей;

n – количество разрядов ключа.

Из соотношения (1) определим формулу длительности такта работы АП:

$$\tau = 2nWM / P_{доп}\eta \quad (2)$$

Для указанных выше величин при объеме памяти $M = 10^6$ ключей длиной $n = 50$ разрядов длительность такта составит $\tau = 10^{-8}$ с.

Из приведенного выше соотношения (2) видно, что при увеличении объема оптической АП скорость работы памяти уменьшается. С другой стороны, при маленькой АП может произойти переполнение, и вычислительный процесс остановится.

5. Математическая модель оптической АП.

Обозначим поток токенов, поступающих на вход АП, за $\{F_i\}$. Очередная порция данных F_{i+1} поступает на вход АП только после окончания обработки предыдущей F_i. Число таких порций обозначим за h. Входной поток F может состоять из токенов P, R и W- типов, а соответствующее число токенов каждого типа в i - ой порции данных обозначим P_i, R_i, W_i. Таким образом, определим входной поток F как набор векторов с компонентами $\{P_i, R_i, W_i\}$, а общее число токенов в i - ой порции данных определяется следующим выражением: $F_i = P_i + R + W_i$.

Время T, затрачиваемое АП на обработку всего входного потока, будет представлять собой сумму времен T_i, затрачиваемых на обработку каждой из порций данных:

$$T = \sum_{i=1}^{h} T_i \ . \qquad (3)$$

Как отмечено выше, оптическая АП позволяет проводить одновременное сравнение и запись нескольких токенов. Максимальное число токенов, которые могут быть обработаны параллельно за один такт работы АП, ограничено числом свободных ячеек АП.

Число токенов, записанных в АП к началу обработки i-ой порции данных, обозначим за B_i. Это токены W - типа из предыдущих порций данных, для которых еще не поступили парные R – токены:

$$B_i = \sum_{j=1}^{i-1} \left(W_j - R_j \right). \qquad (4)$$

Тогда во время прохождения i-ой порции данных оптическая АП может одновременно обработать M - B_i токенов, где M - размер ассоциативной памяти. При этом, если количество токенов во входном потоке не превышает объем свободной АП, т.е. $F(i) \le M - B_i$, - все поступившие токены будут обработаны за один такт работы АП. В противном случае - возникнет задержка, связанная с эффектом так называемого "натыка" [1]. АП будет разбирать входной поток токенов частями по $M - B_i$ штук.

Если $M < B_i$, т.е. число токенов, которые должны находится в АП к началу обработки очередного яруса, превышает размер АП, то произойдет остановка всего вычислительного процесса.

Оценим количество тактов t_i, затрачиваемых на обработку порции данных $F(i)$:

$$t_i = \begin{cases} \left\lceil \dfrac{F(i)}{M - B_i} \right\rceil, & B_i \le M \\ \infty, & B_i > M \end{cases}. \qquad (5)$$

Время, затрачиваемое на обработку одной порции данных:

$$T_i = t_i * \tau \ . \qquad (6)$$

Тогда время обработки всего потока F, при отсутствии переполнения АП:

$$T(F, M) = \sum_{i=1}^{h} \left\lceil \frac{F(i)}{M - B_i} \right\rceil \tau \ . \qquad (7)$$

Время $T(\Gamma_a, M)$ обработки потока, заданного графом потока данных вычислительного алгоритма Γ_a, определяется аналогично. Формулы для вычисления числа токенов каждого типа, приведены ниже:

$$P_i = 2\sum_{v=1}^{n}\sum_{d=v+1}^{n}\sum_{e=d+1}^{n} h_{i,v}h_{i,d}h_{i+1,e}s_{v,e}s_{d,e}\ , \qquad (8)$$

$$R_i = \sum_{l=1}^{i-2}\sum_{v=1}^{n}\sum_{d=v+1}^{n}\sum_{e=d+1}^{n} h_{i,v}h_{l,d}h_{i+1,e}s_{v,e}s_{d,e}\ , \qquad (9)$$

$$W_i = \sum_{l=i+2}^{h}\sum_{v=1}^{n}\sum_{d=v+1}^{n} h_{i,v}h_{l,d}s_{v,d}\ . \qquad (10)$$

6. Модульная АП.

Существенным недостатком оптической АП является тот факт, что ее производительность обратно пропорциональна объему. Технологическим решением, позволяющим повысить скорость работы, является разбиение общей АП на модули. Увеличение производительности системы вцелом достигается как за счет уменьшения нагрузки на каждый модуль памяти в отдельности, так и за счет повышения быстродействия самого модуля вследствие уменьшения его объема. За распределение информации по модулям отвечает функция хэширования.

При этом возникают проблемы построения хэширования и определения размера модулей АП. Неудачно выбранное хэширование, создавая неравномерную нагрузку на модули, может существенно снизить производительность системы или, в худшем случае, привести к полной остановке вычислительного процесса. Маленький размер модулей может привести к возникновению задержек при работе или переполнению, а очень большой – к снижению скорости работы АП.

Введем в предложенную модель АП разбиение на модули. Рассмотрим случай, когда модули равноправны и независимы. Под воздействием хэширования исходный поток данных разбивается на меньшие подпотоки, каждый из которых попадает в свой однозначно определенный модуль АП. Число подпотоков равно числу модулей АП. Обозначим число модулей за p.

Действие хэширования, с точки зрения графа алгоритма, эквивалентно разбиению графа алгоритма на подграфы. Каждому подграфу однозначно соответствует определенный модуль памяти. Таким образом, при хэшировании данные попадут в модуль АП, соответствующий подграфу, в котором находится вершина - потомок данной вершины графа алгоритма. Разбиение графа алгоритма будем представлять в виде матрицы разрезания $X_{p\times n}$, элементы χ_{il} которой

равны 1, если l - ая вершина принадлежит i - ому подграфу, нулю - в противном случае. X также будем называть матрицей хэширования.

Обозначим за M_j размер j – ого модуля АП. $P_{i,j}$, $R_{i,j}$, $W_{i,j}$- число вершин каждого типа с i - го яруса, направляемых в j – ый модуль АП. $F_{i,j}$ – число токенов, поступающих при обработки i – ой порции данных в j – ый модуль АП, $B_{i,j}$ – число занятых ячеек в j – ом модуле АП к началу обработки i – ой порции данных. Ниже приведены формулы для вычисления числа токенов для случая модульной АП:

$$P_{i,j} = 2\sum_{v=1}^{n}\sum_{d=v+1}^{n}\sum_{e=d+1}^{n} h_{i,v}h_{i,d}h_{i+1,e}\chi_{j,e}s_{v,e}s_{d,e}\,, \tag{11}$$

$$R_{i,j} = \sum_{l=1}^{i-2}\sum_{v=1}^{n}\sum_{d=v+1}^{n}\sum_{e=d+1}^{n} h_{l,v}h_{l,d}h_{l+1,e}\chi_{j,e}s_{v,e}s_{d,e}\,, \tag{12}$$

$$W_{i,j} = \sum_{l=i+2}^{h}\sum_{v=1}^{n}\sum_{d=v+1}^{n} h_{i,v}h_{l,d}\chi_{j,d}s_{v,d}\,, \tag{13}$$

$$B_{i,j} = \sum_{l=1}^{i-1}\left(W_{l,j} - R_{l,j}\right),\ F_{i,j} = P_{i,j} + R_{i,j} + W_{i,j}. \tag{14}$$

При оценке времени обработки яруса алгоритма необходимо отметить, что АП будет заблокирована на прием токенов с новых ярусов до тех пор, пока не ликвидируются все очереди на входах блоков. Следовательно, при возникновении переполнения хотя бы в одном модуле вычислительный процесс останавливается.

Введем следующую оценку времени, затрачиваемого на обработку i - го яруса:

$$t_i = \begin{cases} \max\limits_{j=1,p}\left\lceil \dfrac{F_{i,j}}{M_j - B_{i,j}} \right\rceil \tau_j, & B_{i,j} \le M_j \\ \infty, & B_{i,j} > M_j \end{cases}, \tag{15}$$

где τ_j - длительность такта работы j – ого модуля АП.

Тогда время обработки всего графа Γ_a при отсутствии переполнения модулей АП:

$$T(\Gamma_A, X, M) = \sum_{i=1}^{h} \max_{j-1,p}\left\lceil \frac{F_{i,j}}{M_j - B_{i,j}} \right\rceil \tau_j\,. \tag{16}$$

Можно рассматривать оценку времени обработки всего графа Γ_a (16) как функционал от X. Оптимизационную задачу определения

хэширования для графа потока данных сформулируем следующим образом:

Пусть заданы граф потока данных некоторого алгоритма $\Gamma_a = <n, C, S>$, число модулей оптической АП p и размеры модулей $M = \{M_j, j = 1..p\}$. Необходимо найти матрицу разрезания X графа Γ_a на p подграфов, дающую минимум функционалу (16):

$$T(\Gamma_A, X, M) \to \min \qquad (17)$$

7. Генетический алгоритм решения задачи оптимизации.

Решение задачи (17) состоит в нахождении функции распределения пар по модулям АП. Для этого необходимо исследовать возможные распределения пар и найти оптимальное. Предлагается производить поиск решения эволюционным методом, которым является ГА [5, 6].

ГА – итерационный стохастический алгоритм оптимизации, возникший из принципов биологической эволюции. В своей работе ГА исследует потенциальные решения задачи, записанные в форме векторов, называемых хромосомами, или особями. Элементы вектора называются генами. Матрицу хэширования $X_{рхп}$ можно представить в виде вектора $v=(v_1,\ldots,v_n)$, элементы которого соответствуют вершинам графа и принимают целые значения от 1 до p, указывающие на номер подграфа, в котором находится данная вершина. Таким образом, будем рассматривать вектор v в качестве хромосомы, v_i являются генами. Изменение значения гена приводит к новому разрезанию. Наша задача за счет эволюционного (итерационного) процесса ГА, путем модификации значений генов, построить оптимальное разрезание – решение (17). На каждом шаге ГА обрабатывает некоторое множество хромосом, называемое популяцией, число хромосом в популяции - размер популяции. Начальная популяция формируется из случайно составленных разрезаний графа.

С помощью оператора отбора из популяции выбираются пары лучших хромосом (родители). При этом чем лучше решение, закодированное в хромосоме, тем больше вероятность, что его отберут. Критерием оценки получаемого решения является функция качества, отображающая множество хромосом на множество реальных чисел. В качестве функции качества применялся функционал T (16).

Из родительской пары, посредством операторов скрещивания, получается пара потомков. Работа оператора скрещивания заключается

в обмене фрагментов родительских хромосом между собой. С точки зрения графа потока данных, это эквивалентно переносу группы пар токенов из одного подграфа в другой. Различные операторы скрещивания отличаются принципами группировки вершин. Затем к каждой из полученным хромосом - потомков применяется мутация, которая случайным образом перемещает отдельные пары между подграфами. Полученные таким образом хромосомы помещаются в новую популяцию, над которой повторяются описанные шаги. Цель работы ГА - улучшение значения функции качества. Более подробное описание основных операторов ГА и параметров можно найти в [5].

8. Результаты вычислений.

Было проведено исследование процесса поиска решения задачи (17) с помощью ГА. Исследовались графы потока данных для алгоритма обращения матриц по методу Гаусса с размером матрицы 20 x 20 элементов. Число вершин в графе составляет 9570, распределение производилось на 8 модулей АП, размер каждого модуля был равен 190. Для каждой комбинации входных параметров ГА было проведено 100 экспериментов. Полученные в ходе экспериментов данные были усреднены. Для таких параметров может быть найдено точное решение. Значение функционала T при этом равно $T_{\min}$.

Для оценки получаемых результатов использовалась относительная ошибка $E=(T(X)-T_{\min})/T(X)$, где $T_{\min}$ – точное решение задачи, полученное аналитически, X – разрезание.

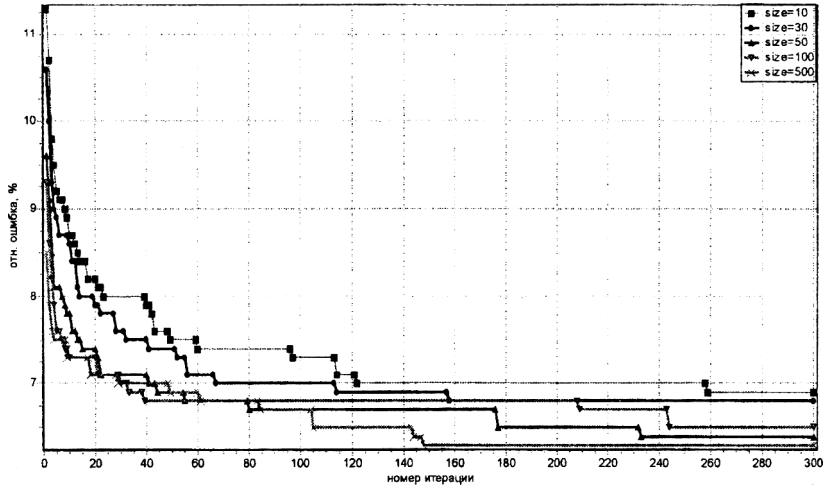


Рис. 2. Сходимость относительной ошибки ГА для различных размеров популяции.

На рис. 2 приведена зависимость относительной ошибки E получаемого решения от номера итерации ГА для популяций различного размера. В нашем случае с различным числом параллельно анализируемых разрезаний. Построенные зависимости позволяют сделать вывод, что число итераций алгоритма, необходимых для получения решения с заранее фиксированной точностью, убывает с ростом размера популяции. Отличие в точности получаемых решений для популяций различных размеров после 150 итераций не превышает 0.5%. Для понимания получаемых результатов важно отметить, что наблюдаемая относительная ошибка 6,5% может возникнуть при перераспределении не более 15 пар вершин в оптимальном решении. Поскольку на вход АП поступают графы различной структуры, достигнутая точность вполне достаточна для решения поставленной задачи.

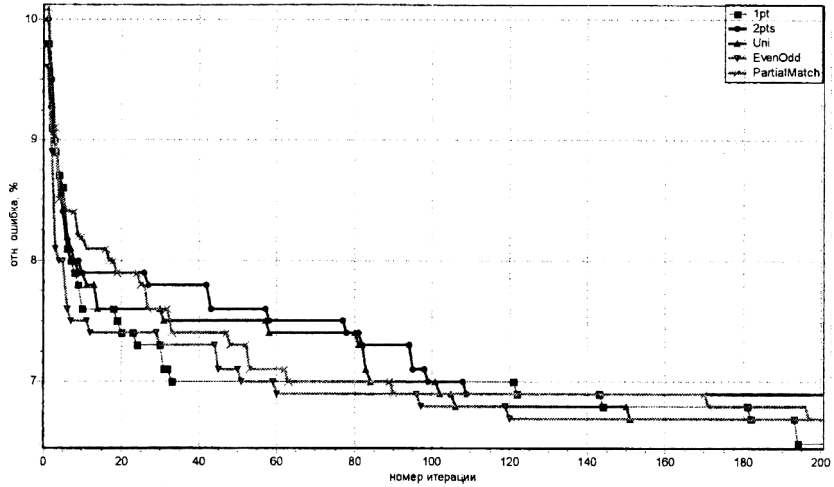


Рис. 3. Сходимость относительной ошибки ГА для различных операторов пересечения.

На рис. 3 показана сходимость относительной ошибки E для различных операторов скрещивания. Лучшие результаты демонстрирует одноточечный оператор скрещивания, который делит родительские хромосомы в случайной точке на две части и полученные фрагменты обменивает между родителями. Описание остальных операторов можно найти в [5]. Самая медленная сходимость наблюдается при применении двухточечного оператора скрещивания. При этом на первых 10 шагах генетического алгоритма все операторы дают практически одинаковые результаты.

На рис. 4 изображены результаты, характеризующие способность ГА выходить из локальных минимумов за счет мутации. Видно, что наилучшие результаты ГА показывает при вероятности мутации в 1%. Применяемый оператор мутации перемещал 2 вершины графа между подграфами.

Рис. 5 демонстрирует результаты работы ГА для различных размеров популяции при отсутствии элитизма. Элитизм означает сохранение части лучших хромосом родительской популяции в дочерней. На рис. 6 и 7 приведены, соответственно, средние значения (м.о.) и дисперсии относительных ошибок генерируемых ГА решений. При отсутствии возможности точно вычислить T_{min} на основании оценок дисперсии и среднего значения T можно принять решение об останове ГА.

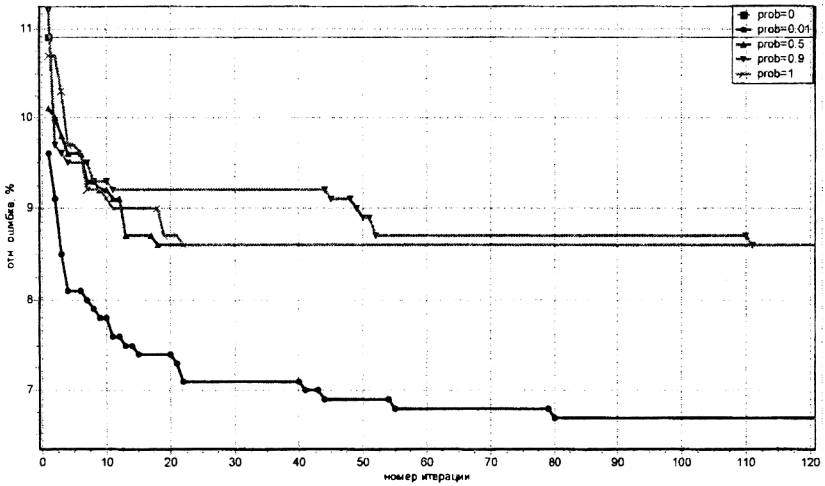


Рис. 4. Сходимость относительной ошибки ГА для различных вероятностей мутации.

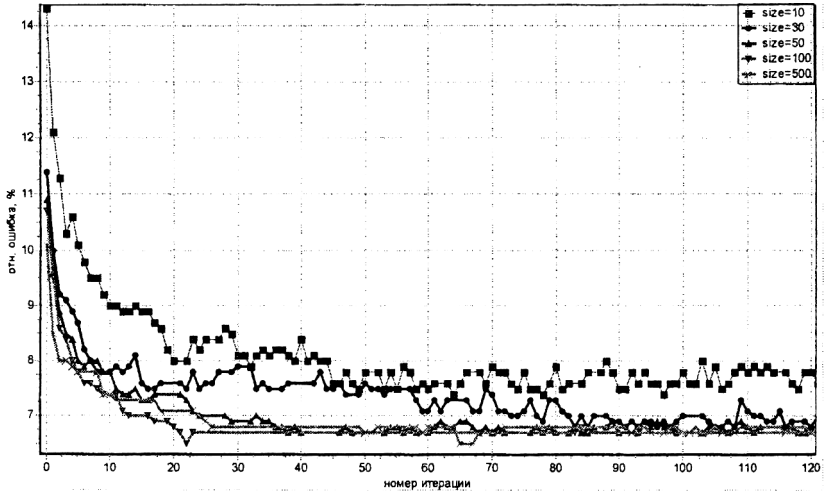


Рис. 5. Сходимость относительной ошибки ГА для различных размеров популяции при отсутствии элитизма.

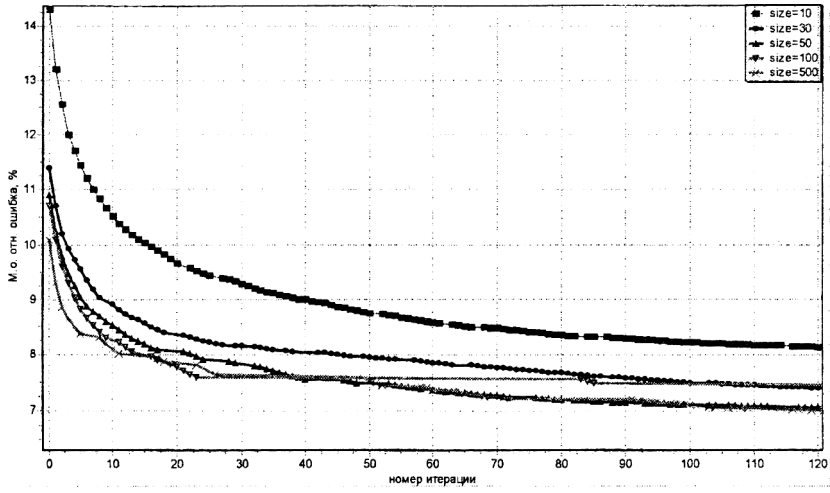


Рис. 6. Среднее значение относительной ошибки ГА для различных размеров популяции при отсутствии элитизма.

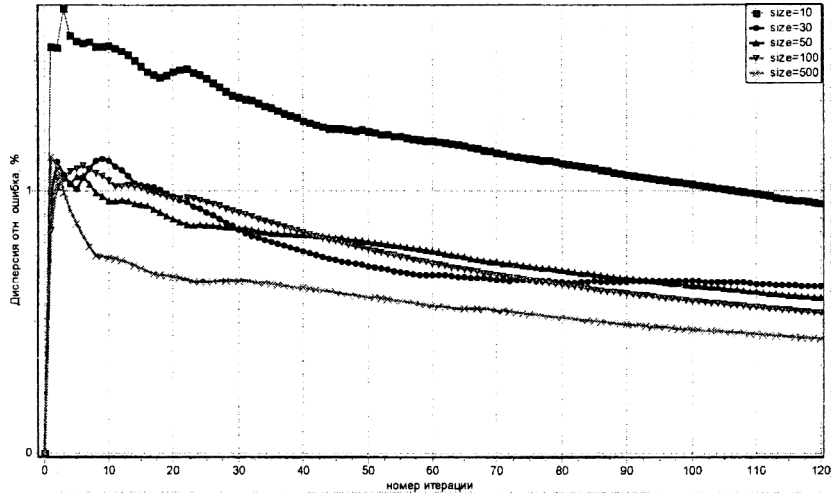


Рис. 7. Дисперсия относительной ошибки ГА для различных размеров популяции при отсутствии элитизма.

Полученное решение сформулированной выше задачи дает оптимальное разбиение конкретного графа алгоритма. В практических условиях необходимо создание единого хэширования для множества различных алгоритмов.

Разделим алгоритмы на классы, объединяя в рамках одного класса алгоритмы одинаковой размерности (число вершин) и однородностью вычислений [11]. Будем искать хэширование для алгоритмов одного класса и конкретной реализации АП.

В качестве примера рассмотрим графы алгоритма Гаусса (Γ_1), метода Жордана разложения по столбцу (Γ_2), LU – разложение (Γ_3), МК – метод Монте-Карло (Γ_4). Для АП были использованы указанные выше параметры.

Зафиксируем погрешность соответствующую относительной ошибке 10%, и, используя ГА, построим разрезания X_1, X_2, X_3 выбранных алгоритмов. Каждое из найденных разрезаний X_1, X_2, X_3 применим к двум другим алгоритмам и вычислим получаемые относительные ошибки. Результаты приведены в табл. 1.

В качестве общего хэширования выберем разрезание с наименьшей средней ошибкой. Для данного набора алгоритмов таким квазиоптимальным разрезанием будет являться X_1. Из приведенной таблицы видно, что большие однотипные графы алгоритмов Γ_1, Γ_2, Γ_3 дают разброс результатов в пределах точности разрезаний – не более 10%. Рассмотренный алгоритм метода Монте-Карло существенно отличался по своей структуре от остальных графов, что и отразилось на результатах. Так средняя ошибка при применении X_4 составляет 43% в сравнении с 23-28% для других разрезаний. Необходимо также отметить несимметричность таблицы относительно главной диагонали. Видно, что результаты применения X_4 будут существенно хуже, чем для остальных разрезаний. Для нахождения хэширования имеет смысл проводить исследования для конкретной архитектуры вычислительной системы методом ГА.

9. Обсуждение результатов и выводы.

Предложенный функционал T, описывающий работу АП в архитектуре потока данных, позволяет строить оптимизационные модели для исследования и оптимизации структуры АП. Поставлена оптимизационная задача для поиска оптимального хэширования при заданных структуре АП и графе алгоритма.

С помощью описанной модели несложно получить оценки работы АП, как, например, загрузку модулей АП и объем свободного пространства в модуле. Модифицируя модель можно рассматривать схемы организации работы АП, отличные от предложенной, например, с "выделенным буфером".

Порядок поступления вершин графа алгоритма на вход АП, а, следовательно, и порядок выполнения операций алгоритма, определяется с помощью поярусного строения графа алгоритма, т.е. при помощи матрицы *H*. Для отражения в модели факта ограниченности числа исполнительных устройств потоковой машины, достаточно внести изменения в матрицу *H*. С помощью перемещения вершин алгоритма между ярусами также возможно провести моделирование ситуаций перегрузки, когда резко увеличивается поток данных в АП.

Предложенный в статье метод оптимизации функционала *T* основан на использовании генетического алгоритма. ГА позволяет естественным эволюционным образом распределять токены в АП. Показана эффективность использования данного подхода к оптимизации. Достигаемая алгоритмом точность достаточна для рассматриваемой задачи. Из приведенных графиков видно, что для нахождения точности, соответствующей относительной ошибки в 6-8%, достаточно малого числа итераций 10-30, что с учетом параллельного исполнения позволяет находить решение за малое время.

Изменяя выражение под знаком суммы в функционале *T* (16), возможно рассматривать различные принципы организации сравнения ключей в АП: например, одновременное сравнение нескольких ключей, как в оптической АП, или последовательное, как в полупроводниковой АП. Дальнейшее развитие предложенной модели возможно провести также в направлении учета таких параметров графа алгоритма как вес дуг и тип вершин графа, что позволит ввести в рассмотрение векторные операции.

Проведенные исследования также наводят на мысль о разработке интеллектуальной системы подбора хэширования в процессе работы вычислительной системы (ВС). Включение эволюционного механизма, аналогичного описанному ГА, в ВС для формирования эффективного хэширования позволит создать АП с самонастраивающейся архитектурой. Такая ВС, обладающая возможностью эволюционировать, будет способна обучаться и быстро подстраиваться под меняющийся входной поток задач.

Процесс эволюционной адаптации АП в этом случае можно построить следующим образом:

1. В начале работы система случайным образом формирует разрезания входящего алгоритма. Каждое такое разрезание соответствует хромосоме. Множество хромосом образует начальную популяцию.

2. ВС проводит выполнение поступившего задания с использованием разрезаний, представленных в популяции. При этом реальное время работы ВС будем рассматривать как значение функции качества для используемого разрезания.
3. С помощью операторов ГА система формирует новую популяцию. Далее переходим к п.2.

В случае изменений входного потока заданий, система будет сама подстраиваться, за счет механизма эволюции. Из приведенных в предыдущем разделе результатов видно, что за первые пять итераций ГА проводит существенное улучшение значения T, при этом дальнейшие итерации не приводят к значительному уменьшению ошибки. Это позволяет сделать вывод, что такая ВС будет достаточно быстро за 1-5 итераций адаптироваться под изменяющийся поток входных заданий.

11. Литература.

1. Бурцев В.С. Выбор новой системы организации выполнения высокопараллельных вычислительных процессов, примеры возможных архитектурных решений построения суперЭВМ // Параллелизм вычислительных процессов и развитие архитектуры супер-ЭВМ. М.: Нефть и Газ. 1997. С. 41-78.

2. Popadopolous G., Traub K. Multithreading: A Revisionist View of Dataflow Architecture // Sigarch Computer Arch. News. 1991. Vol. 19, N.3.

3. Бурцев В.С. Использование оптических методов обработки информации в архитектуре суперЭВМ // Параллелизм вычислительных процессов и развитие архитектуры супер-ЭВМ. М.: Нефть и Газ, 1997. С. 79-104.

4. Бурцев В.С., Федоров В.Б. Ассоциативная память на принципах оптической обработки информации для супер-ЭВМ нового поколения // Вычислительные машины с нетрадиционной архитектурой. Супер-ВМ. Вып. 2, М.:ВЦКП РАН, 1994. С. 78-97.

5. Goldberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning. Reading. MA: Addison – Wesley, 1989.

6. De Jong K. Evolutionary Computation: Recent Development and Open Issues // Proceedings of EvCA'96. P. 7-17.

7. Nikitin A.V. Using GA for Parallelization of Programs // Proceedings of EvCA'96. P.273-277.

8. Thang Nguyen Bui, Byung Ro Moon. Genetic Algorithm and Graph Partitioning // IEEE Trans. On Computers. 1996. Vol. 45, N.7.

9. Никитин А.В., Попов А.М. Задача оптимизации модульной ассоциативной памяти // Численные методы и вычислительный эксперимент / Под ред. А.А. Самарского и В.И. Дмитриева. М.: Диалог-МГУ. 1998. С. 97-106.

10. Ершов Н.М. Алгоритм распараллеливания вычислений на основе метода Монте-Карло // Математические модели естествознания. М.: Изд-во МГУ. 1995. С. 150-156.

11. Оленин А.С. Представительные вычисления для исследования архитектур ЭВМ // Вычислительные машины с нетрадиционной архитектурой. Супер-ВМ. Вып. 6. М.:МИКОПРИНТ, 1997. С. 3-11.

Брусенцов Н.П., Деркач А.Ю.

Трехзначная логика, нечеткие множества и теория вероятностей

Трехзначная логика [1] восполняет существеннейший пробел в общепринятой двухзначной формальной логике – ее недиалектичность, неспособность адекватно отобразить непостоянный, непрерывно обновляющийся характер бытия. Принцип двухзначности (закон исключенного третьего) обуславливает бескомпромиссную дискретность отображения, не оставляя места неопределенности, нечеткости, модальным и вероятностным оценкам: «да-да, нет-нет, а что сверх того, то от лукавого». Но ведь «не-полный» не значит «пустой» и «не-пустой» не значит «полный», потому что существует третье – «не пустое и не полное».

Это промежуточное незавершенное третье Аристотель называет *привходящим* [2, «Метафизика», 1025a14], полагая его уделом диалектики – «искусства ставить наводящие вопросы» и «испытывания при помощи умозаключений» [2, «О софистических опровержениях», 172a17]. По-видимому, мнение Аристотеля, что «о привходящем нет доказывающего знания, так как заключение о нем нельзя доказать с необходимостью, поскольку привходящее может не быть присущим» [2, «Вторая аналитика», 75a18], последующими логиками было воспринято со свойственной им категоричностью: они устранили привходящее вместе с диалектикой из своей непогрешимой науки и приняли закон, исключающий какое бы то ни было третье. Правомерность такой перестройки обосновывают ссылками на аристотелевы же трактаты, напрочь игнорируя занимающие в них едва ли не главное место рассуждения о привходящем. Наука о методе, в которой Аристотель усматривал начало всех наук, выродилась в практически бесполезную, несовместимую со здравым смыслом и с учением самого Аристотеля схоластику.

Такая логика непригодна для решения реальных проблем, неприемлема в качестве основания наук и даже не может обеспечить развитие самой себя. Показательна безуспешная попытка «логистического» обоснования математики и предпочтения теоретико–множественной аксиоматики, в частности, в теории вероятностей [3, 4], которая, как и последующие теории нечетких множеств и логик [5], должна быть развитием учения о привходящем, непосредственным и неотъемлемым продолжением диалектической логики. В свое время на это указывал Дж.Буль [6], которого, как и Аристотеля, не захотели понять. Более того, на протяжении ста с лишним лет, как раз в эпоху гносеологического кризиса оснований остается «незамеченным»

построенное П.С. Порецким [7] исчерпывающее обоснование теории вероятностей средствами математической логики.

Концепцию Порецкого, воспользовавшись современной терминологией, можно охарактеризовать вкратце так [8].

Введем n независимых булевых переменных x, y, z, ... (по Аристотелю – «первичных терминов»), интерпретируемых как обозначения «простых» дискретных качеств, составляющих основу рассматриваемого «мира речи» (универсума). В n-терминном «мире речи» однозначно (четко) определимы 2^n элементарных конъюнкций, идентифицирующих атрибуты индивидных классов, а проще – индивидные классы, еще проще – индивидные вещи, одним словом – индивиды. Неиндивидные (нечеткие) атрибуты вещей – это дизъюнкции индивидных конъюнкций, т. е. СДНФ-выражения булевой алгебры. Общее число различных атрибутов и соответственно классов в n-терминном «мире речи» 2^{2^n}.

Булева алгебра классов позволяет, фиксируя значение атрибута класса, определить в виде уравнения $F(x, y, z,...) = 1$ «мир задачи», в котором термины x, y, z,... взаимосвязаны условием F. Решением уравнения относительно того или иного термина можно выявить зависимость этого термина от прочих терминов в случаях, когда она носит необходимый характер. Но в общем случае искомая зависимость неоднозначна, привходяща, и охарактеризовать ее можно лишь вероятностью того, что предполагаемая связь имеет место.

Переход от качественной характеристики взаимосвязи, выраженной булевым уравнением, к количественной, вероятностной характеристике статуса исследуемых классов, интерпретируемых теперь как «события», Порецкий вслед за Булем называет *пробабилизацией*. Совокупность членов СДНФ-выражения $e(x, y, z,...)$, фиксированного уравнением $e(x, y, z,...) = 1$, с теоретико-вероятностной точки зрения представляет собой полную группу событий (случаев, шансов) – они равновероятны, несоисключимы и попарно несовместимы. Общее число таких событий-индивидов в «мире речи» - 2^n.

Абсолютная вероятность $p(e)$ события e, представленного СДНФ-выражением, в котором содержится N_e индивидных членов равна

$$p(e) = N_e/2^n$$

Относительная (условная) вероятность $p(x|e)$ события x при наступившем e равна отношению числа N_{xe} тех индивидов в выражении e, которые содержат термин x неинвертированным, к общему числу индивидов в e:

$$p(x|e) = N_{xe}/N_e$$

Абсолютная вероятность $p(xe)$ совпадения (конъюнкции) событий x и e будет теперь:

$$p(xe) = N_{xe}/2^n = p(x|e)N_e/2^n = p(x|e)p(e).$$

Абсолютная вероятность $p(x \vee e)$ дизъюнкции событий x, e аналогично определяется как

$$p(x \vee e) = (N_x + N_e - N_{xe})/2^n = p(x) + p(e) - p(xe).$$

В свете такого логического истолкования понятий тории вероятностей вполне очевидным становится и смысл нечетких множеств Заде. Произвольное булево выражение $e(x, y, z,\ldots)$ преобразуется (пробабилизуется) в форму нечеткого множества Заде вычислением относительных вероятностей всех его терминов. Например, выражению материальной импликации $x' \vee y$, которое в СДНФ имеет вид $xy \vee x'y \vee x'y'$. соответствует $p(x|e) = 1/3$, $p(y|e) = 2/3$ и нечеткое множество $\{1/3x, 2/3y\}$.

В n-терминном «мире речи» четкие множества соответствуют 2^n индивидным (n-арным элементарным) конъюнкциям, в которых нет привходящих (умалчиваемых) терминов. События, представленные в такой конъюнкции неинвертированными терминами, достоверны, необходимо принадлежат описываемой конъюнкцией совокупности, вероятность их равна 1, события же представленные инверсиями терминов, невозможны в рассматриваемой совокупности, антипринадлежат ей, характеризуются нулевой вероятностью. За исключением индивидных конъюнкций, все прочие булевы выражения пробабилизуются в нечеткие совокупности терминов, содержащие, наряду с достоверными и невозможными, привходящие качества, присущность которых рассматриваемому объекту не необходима и не невозможна, а может быть только дробным значением вероятности: $0<p<1$

Литература

1. Брусенцов Н.П. Трехзначная диалектическая логика. – В этом сборнике.
2. Аристотель. Сочинения в четырех томах. – М.: «Мысль», т. 1 – 1975, т. 2 – 1978.
3. Бернштейн С.Н. Опыт аксиоматического обоснования теории вероятностей // Заметки Харьковского математического общества. – Харьков, 1917, с. 209-274.
4. Колмогоров А.Н. Основные понятия теории вероятностей. – М.: «Наука», 1974.
5. Zadeh L.A. Fuzzy sets // Fuzzy sets and systems, 1965. n. 8, pp. 338-353.
6. Boole G. An investigation of the laws of thought on which are founded the mathematical theories of logic and probabilities. – London, 1854.
7. Порецкий П.С. Решение общей задачи теории вероятностей при помощи математической логики. – Казань, 1887.

8. Брусенцов Н.П., Деркач А.Ю. Логическая модель теории вероятностей и нечстких множеств Заде. // Цифровая обработка информации и управление в чрезвычайных ситуациях. Вторая международная конференция, 28-30 ноября 2000г. - Минск. Доклады, т. 1, с. 41-44.

Дерий Д.М., Рамиль Альварес Х.

Троичный процессор на двоичном компьютере

Интерес к троичной системе счисления применительно к технике возник уже на начальном этапе развития цифровых машин в связи с замечательными арифметическими свойствами симметричного кода чисел [1].

Впервые в чистом виде симметричная троичная система счисления появилась в статье Л. Ланнана, изобретателя механических вычислительных устройств, в 1840 году. В течении следующих ста лет эта система счисления упоминалась лишь эпизодически, пока в Электротехническом институте Мура в 1945-46 годах не стали разрабатываться первые электронные вычислительные устройства. В этот период она серьезно рассматривалась наряду с двоичной системой счисления в качестве возможной замены десятичной.

Троичная цифровая техника базируется на трехзначных сигналах и трехстабильных элементах памяти (тритах). Объекты, принимающие более чем три значения, реализуются в ней как наборы тритов. Такая техника основывается уже на трехзначной логике, что необычно для распространенных в наше время двоичных компьютеров, в логике которых допустимо только два значения: "Да" и "Нет". Однако так ли необычна трехзначная логика? Ведь двузначные критерии вовсе не отрицают третьего значения: если мы не можем сказать о значении ничего определенного, то мы можем выйти из этого положения, введя особое значение "Неопределенность". В явном виде это значение ввел польский логик Я. Лукасевич (1878-1956), основав таким образом первую, как он считал, недвузначную логику, которую теперь называют трехзначной логикой Лукасевича. На самом деле уже за шесть веков до этого подобной логикой пользовался англичанин У. Оккам (1300-1349), один из крупнейших философов и логиков Средневековья. В своих работах Оккам рассматривал три значения истинности: истинно, ложно и неопределено.

Теория силлогистики, созданная Аристотелем в 4 в. до н.э., в сущности тоже построена на принципе трехзначности [2]. Кроме "истинно" и "ложно", у Аристотеля было еще понятие "привходяще", которое использовалось в тех случаях, когда не было оснований ни утверждать, ни отрицать истинность высказывания. Например, вопрос "Будет ли завтра в полдень морской бой?" предполагает три варианта ответа: "Да", "Нет" или "Может быть".

В реальной жизни трехзначные отношения очень распространены, например:

больше - равно - меньше,
увеличить - не изменять - уменьшить,
вперед - стой - назад,
победа - ничья - поражение и т.п.

В современном мире компьютеров главенствующее место занимает двузначная – булева - логика. Однако в традиционной булевой логике существует немало проблем связанных, например, с парадоксами. Предлагаемые решения сложны и трудны для понимания. Очевидно, имеет смысл попробовать идти не по пути сохранения двоичности ценой усложнения теории, а ввести третье значение и изучить ситуацию с точки зрения трехзначной логики. Для некоторых задач такой подход позволяет упростить решение.

Кроме трехзначной логики, троичный компьютер предлагает нам все преимущества троичного симметричного кода, значимость которых отмечалась признанными авторитетами в мире программирования, такими как Д. Кнут и К. Шеннон. В своей работе "Искусство программирования для ЭВМ" Д. Кнут отзывается о симметричной троичной системе счисления как о "быть может, самой изящной" и отмечает, что эта система "до сих пор все еще не нашла серьезного применения, но возможно, что ее симметричность и простая арифметика окажутся в один прекрасный день весьма существенными" [3].

Троичный симметричный (сбалансированный) код обладает следующими преимуществами:

1. Не приходится различать "числа без знака" и "числа со знаком", в отличие от двоичной системы счисления, так как естественное представление числа в симметричной системе счисления дает нам и отрицательные величины, и положительные, и нуль. Благодаря этому вдвое сокращается необходимое число условных команд и, как следствие, облегчается их использование.

2. Арифметические операции допускают свободное варьирование длины операндов и выполнение с операндами неравной длины. При этом не требуется никаких вспомогательных команд, таких как, например, "распространения знака" в двоичной технике.

3. Правильное округление чисел (до ближайшего представимого с равной вероятностью представления с избытком и недостатком в случае равного расстояния) достигается просто отсечением соответствующей хвостовой части.

4. Единственная операция сдвига заменяет и логический и арифметический сдвиги двоичной арифметики.

5. Знак числа задается наиболее значимым ненулевым тритом, причем для изменения знака числа достаточно произвести поразрядную инверсию.

Вопреки распространенному мнению, будто главное преимущество троичного кода перед двоичным состоит в его экономности, которая обусловлена тем, что 3 ближе чем 2 к основанию натурального логарифма e=2.718... , следует сказать, что эта экономия не превышает 5%, т.е. практически не имеет значения. Вместе с тем, может быть оправдана и существенная неэкономность кодирования трита парой битов ради реализации действительных преимуществ троичного кода. Эти преимущества обусловлены тем, что 3 - минимальное основание системы счисления с естественным представлением чисел со знаком [4].

В двоичной системе естественное представление возможно либо только неотрицательных чисел (с цифрами 0, 1), либо только неположительных (0, -1), либо неположительных и неотрицательных, но неоднозначное и без нуля. Обычно используются цифры 0 и 1, а для отрицательных чисел применяется дополнительный или обратный код или вводится особый бит знака числа. Троичный же код позволяет ввести знак (+, 0, -) на уровне элемента-трита и последующая обработка чисел со знаком упрощается.

Например, правильное округление, которое предоставляет нам симметричный код практически в готовом виде, имеет большое значение для численных методов. Было показано [5], что при любом закрепленном способе округления, определяемым лишь "отбрасываемыми" разрядами, ошибка при сложении случайных чисел в любой четной системе счисления в режиме плавающей запятой будет иметь систематическое смещение. Другими словами, ошибки не компенсируют друг друга, тогда как в любой системе счисления с нечетным основанием классический способ округления асимптотически приводит к несмещенным ошибкам для сложения в режиме плавающей запятой. Заметим, что данное утверждение не требует симметричности системы счисления, однако при ее отсутствии алгоритм правильного округления будет нетривиальным. Таким образом, троичные машины (в особенности с симметричным кодом) теоретически являются более подходящими для применения численных методов.

В симметричном троичном коде возможно не только непосредственное сложение чисел разной длины, но и можно любую часть слова интерпретировать как отдельное число со знаком и выполнять арифметические операции над словами, как над совокупностями таких чисел, что может оказаться полезным.

Перспективным научным направлением является использование симметричного троичного кода для кодирования функций [6].

Компьютер, построенный на основе такого кода, сможет непосредственно работать не только с числовыми константами, но и с функциями, что является качественным переходом на пути повышения "интеллекта" компьютера.

Заметим, что метод кодирования, предложенный в [6], позволяет представлять достаточно широкий класс функций (включающий в себя ряды Фурье) и производить над ними операции - изменение знака, алгебраическое сложение, умножение, деление, а также дифференцирование и интегрирование.

Серьезное препятствие на пути внедрения троичной техники - это то, что мир уже занят и насыщен двоичной техникой, для которой накоплено много программных продуктов. Поэтому, чтобы добиться успехов в этом деле, надо:

1) продемонстрировать реальные преимущества троичной техники;

2) обеспечить совместимость и эффективное взаимодействие с существующей двоичной техникой.

Поскольку создание троичной машины "в железе" сопряжено с различного рода трудностями, для исследования троичной техники целесообразно использовать ее программные модели. Кроме того, наличие таких моделей позволит воспользоваться преимуществами симметричного троичного кода широкому кругу пользователей. Создание этих моделей весьма актуально в связи с наметившимися практическими применениями троичного симметричного кода и фактическим отсутствием троичной цифровой техники.

Целью проведенной работы и было создание такой модели. Моделирование производилось в Диалоговой системе структурированного программирования (ДССП), которая предоставляет подходящий инструмент для конструирования новых сущностей. Также большим плюсом ДССП является ее свободная переносимость.

Методика моделирования в ДССП основывается на понятиях «конструкт» и «конструктив». Конструкт в ДССП - это процедура, располагающая собственной памятью и непосредственно ею манипулирующая. Конструкт характеризуется форматом принадлежащего ему участка памяти, то есть его размером (информационной емкостью), структурой и способами доступа, а также набором базисных процедур (операций), применимых к содержимому этой памяти. Формат и набор базисных операций в совокупности определяют класс, или тип, конструкта, а тем самым и тип представляемых им данных. Процедуры, порождающие конкретные экземпляры конструктов определенного типа, называются конструктивами [7]. Различные базисные процедуры вызываются при помощи указания у имени конструкта соответствующих префиксов,

одна из процедур выполняется при указании имени без префикса и называется процедурой по умолчанию.

Разработка модели производилась при помощи иерархического построения типов путем введения соответствующих конструктов. Таким образом, на основе имеющегося в двоичном компьютере бита был сконструирован трит, на основе трита был построен трайт, затем были реализованы типы "целое троичное число" и "вещественное троичное число", послужившие в свою очередь основой для модели процессора.

При конструировании трита первым делом необходимо было выбрать для него двоичное представление. Минимальное число бит для представления одного трита – это пара бит. Более сложные сущности в троичной машине представляются последовательностями тритов. Поскольку информационная емкость пары бит на четверть больше, чем требуется, можно было бы попытаться как-то упаковать представление последовательности тритов, однако это сделало бы выборку трита нетривиальной операцией и затруднило бы обработку последовательности тритов. Пожертвовать памятью представляется более предпочтительным, чем усложнять модель. Триты соответствуют цифрам симметричной троичной системы счисления и их представление таково: 00 для 0 , 01 для 1 и 10 для –1. Отметим, что первый и второй биты в таком представлении трита могут быть проинтерпретированы как, соответственно, отрицательная и положительная двоичные составляющие этого трита: такие x^+ и x^-, что $x^+ \& x^- = 0$ и $x^+ - x^- = x$, где x - значение трита.

На основе трита был сконструирован трайт (шестерка тритов), который, аналогично двоичному байту, является наименьшей адресуемой частью памяти. Сама же троичная память представляет собой последовательность трайтов.

Набор трайтов, который содержит в себе последовательность тритов, можно уже интерпретировать как целое число, число с плавающей запятой и, собственно, просто как вектор тритов. Введя конструкты с соответствующими базисными операциями, мы получаем целую и вещественную арифметики и трехзначную логику. Для реализации трехзначной логики был использован простой и универсальный метод, основанный на побитовых операциях двоичной логики, примененным к двоичным составляющим тритов. Очевидно, что трит результата любой логической операции однозначно определяется тритами-аргументами. Это значит, что каждая из двоичных составляющих трита-результата однозначно определяется составляющими тритов-аргументов, то есть является по существу значением функции обычной булевой логики. Для требуемых функций троичной логики были составлены таблицы истинности и были определены выражения вычисления двоичных составляющих результата.

Наименее сложной оказалась реализация двоичных функций в виде полиномов Жегалкина. Рассмотрим, например логическое сложение двух тритов (то есть сложение по модулю 3).

Таблица истинности для логического сложения следующая:

x	y	f
0	0	0
0	1	1
0	$\bar{1}$	$\bar{1}$
1	0	1
1	1	$\bar{1}$
1	$\bar{1}$	0
$\bar{1}$	0	$\bar{1}$
$\bar{1}$	1	0
$\bar{1}$	$\bar{1}$	1

Или, в двоичных кодах:

$x^- x^+$	$y^- y^+$	$f^- f^+$
00	00	00
00	01	01
00	10	10
01	00	01
01	01	10
01	10	00
10	00	10
10	01	00
10	10	01

Как видно, f^- и f^+ - обычные булевы функции от четырех переменных.

Нетрудно выписать СДНФ для функций f^- и f^+:

$$f^- = \neg x^- \& \neg x^+ \& y^- \& \neg y^+ \vee \neg x^- \& x^+ \& \neg y^- \& y^+ \vee x^- \& \neg x^+ \& \neg y^- \& \neg y^+,$$

$$f^+ = \neg x^- \& \neg x^+ \& \neg y^- \& y^+ \vee \neg x^- \& x^+ \& \neg y^- \& \neg y^+ \vee x^- \& \neg x^+ \& y^- \& \neg y^+.$$

(7 отрицаний , 12 конъюнкций, 3 дизъюнкции на каждую функцию)

СКНФ для функций f^- и f^+ содержат пять элементарных дизъюнкций и явно сложнее.

Полиномы Жегалкина для функций f^- и f^+ выглядят следующим образом:

$$f^- = x^- \oplus y^- \oplus x^+ \& y^+ \oplus x^+ \& y^- \oplus x^- \& y^+,$$

$$f^+ = x^+ \oplus y^+ \oplus x^- \& y^- \oplus x^+ \& y^- \oplus x^- \& y^+.$$

(3 конъюнкции и 4 сложения по модулю 2 на каждую функцию)

Рассмотренный выше метод реализации логических операций удобен тем, что позволяет обрабатывать все триты аргументов одновременно, что является более эффективным, чем последовательный перебор.

Поскольку построение модели процессора производилось при помощи введения в ДССП конструктивов «целое троичное число» и «троичное число с плавающей запятой», то архитектура процессора была выбрана аналогичной архитектуре ДССП, а именно стековой. Надо сказать, что ДССП в своей основе эмулирует архитектуру реальной троичной машины «Сетунь 70», опыт разработки и эксплуатации которой предоставил богатую базу для построения данной модели.

Выполнение команды реализовано следующим образом: из стека на регистры берутся операнды, производится соответствующая операция над ними соответствующим же функциональным устройством и результат засылается в стек. При этом ответственность за проверку корректности содержимого стека (например, имеется ли в стеке нужное количество операндов) целиком возложена на реализацию стека, а само функциональное устройство работает только с регистрами.

Поскольку процессор должен обрабатывать данные двух типов: целого и вещественного с плавающей запятой - то было введено два стека операндов, каждый для своего типа. В противном случае в элементы стека пришлось бы добавить тэги типов для контроля, что в любом случае привело бы к усложнению или формата команд или их реализации. Проще и безопаснее манипулировать разными типами в разных стеках, что не исключает возможности их «общения». В модели для целочисленной и соответствующей вещественной операции используется одна команда, а для определения со стеком какого типа производится операция вводится понятие «активного» стека и, соответственно две команды - для «активации» целого стека и для «активации» вещественного стека. Причем при операциях, могущих незаметно изменить состояние неактивного стека, возбуждается прерывание, реакцию на которое можно переопределять по усмотрению пользователя.

При попытке выполнения недопустимых арифметических операций (например, деление на 0), при переполнении регистров и при других исключительных ситуациях также возбуждаются соответствующие прерывания.

Кроме того, реализованы команды работы с памятью и, соответственно сама оперативная память. Адресация у этой памяти симметричная, то есть трайты нумеруются от заданного отрицательного значения до такого же по абсолютной величине положительного. На представление каждого трайта отводится два байта. Отметим, что для представления трайта достаточно 12 битов, то есть полтора байта, но усложняется обмен с памятью, поскольку, засылая в двоичную память

одну половинку байта, необходимо оставить неизменной вторую половину этого байта. Также усложняется отображение адресов троичной памяти модели в адреса двоичной памяти реальной машины. В данном случае предпочтение было отдано простоте модели и скорости выполнения обращений к троичной памяти в ущерб экономии памяти реальной машины. Таким образом, адресами у нас являются троичные числа и, соответственно адресная арифметика также троична.

Кроме низкоуровневых операций процессора, для удобства пользователя реализована надстройка, позволяющая писать программы для модели процессора на достаточно высоком уровне - с использованием переменных (как скалярных, так и массивов) и определением процедур.

Надо сказать, что, как оказалось, реализация чистой троичной памяти, состоящей из троично адресуемых трайтов не так проста, как хотелось бы, она усложняет модель и понижает эффективность ее работы, тогда как для реализации троичной арифметики она вовсе не необходима. Поэтому, создавая троичный инструмент для реальных практических применений, имеет смысл отказаться от троичных форматов и троичной адресной арифметики. Реализуя модель троичного компьютера на двоичной технике, более оптимально ввести двоично-троичные форматы, плотно укладывающиеся в существующие адресуемые форматы двоичной памяти традиционной техники. Эта работа и проводится в данное время в рамках создания варианта модели троичного процессора для практических применений.

Литература

1. Shannon C.E. A symmetrical notation for numbers.-"The American Mathematical Monthly", 1950, v.57, n.2, p.90-93.
2. Брусенцов Н.П. Диаграммы Льюиса Кэррола и аристотелева силлогистика. Сборник "Вычислительная техника и вопросы кибернетики", вып.13, изд-во МГУ, 1976, с.164-182.
3. Кнут Д. Искусство программирования для ЭВМ. Получисленные алгоритмы. Т.2. М.: Мир, 1977.
4. Брусенцов Н.П., Жоголев Е.А., Маслов С.П., Рамиль Альварес Х. Опыт создания троичных цифровых машин. //Компьютеры в Европе. Киев. Изд-во "Феникс",1998, с.67-71.
5. Воеводин В.В. Ошибки округления и устойчивость в прямых методах линейной алгебры.М., ротапринт ВЦ МГУ, 1969.
6. Хмельник С.И. Кодирование функций. // «Кибернетика», № 6, 1966, с. 88-92.
7. Концептуальная характеристика РИИИС-процессора // Брусенцов Н.П., Маслов С.П., Рамиль Альварес Х., Сидоров С.А. //Интегрированная система обучения, конструирования программ и

разработки дидактических материалов. - М.: Изд-во ф-та ВМиК МГУ,1996, с.25.

Раздел III
Прикладные программные системы

Веселов Н. А., Машечкин И. В.

О некоторых экспериментальных средствах анализа колебаний длин очередей сообщений в узлах сети в условиях возникновения перегрузок

Введение

Одной из проблем, возникающих при работе сети передачи данных являются перегрузки. Они могут быть связаны как с поступлением в сеть избыточного трафика, превышающего её пропускную способность, так и с неоптимальной, нерациональной работой алгоритмов, используемых в сети связи. При этом некоторые участки сети могут быть перегружены и в некоторых узлах могут возникать очереди пакетов, ждущих передачи по определённой линии связи, в то время как другие участки загружены не полностью.

Для того чтобы распределить входящий трафик, максимально используя пропускную способность сети, применяются алгоритмы маршрутизации. Количественно оценить загруженность сети позволяет потоковая модель. На неё опирается в своей работе алгоритм маршрутизации. Он, по сравнению с другими алгоритмами адаптивной маршрутизации (например алгоритмом кратчайшего пути) позволяет избежать многих нежелательных явлений. Этот алгоритм широко описан в литературе, в частности в работе Д. Бертсекас, Р. Галлагер. "Сети передачи данных"[1] и носит название "алгоритм оптимальной маршрутизации".

Однако алгоритму оптимальной маршрутизации присущи некоторые недостатки. Целью данной работы является его анализ и усовершенствование.

1. Постановка задачи, формулировка модели

Остановимся более подробно на модели сети передачи данных, используемой далее при описании алгоритмов. Выбор данной модели не является единственно возможным, но он не случаен, так как позволяет наглядно отразить особенности исследуемого подхода

---

[1] Бертсекас Д., Галлагер Р., Сети передачи данных. – М: Мир, 1989.

1.1 Описание модели функционирования сети передачи данных

Сеть передачи данных будем представлять в виде связного неориентированного графа $\Gamma = (U, V)$. В некоторых узлах графа $u \in U$ расположены абоненты сети. Ребрами графа сети $v = (i, j) \in V$ представлены каналы связи. При этом для каждого канала $v = (i, j)$ задана пропускная способность C_{ij}, измеряемая в единицах данных в секунду. По маршрутам графа Γ от одних абонентов сети к другим передаются сообщения в форме пакетов.

Обозначим некоторое множество пар $w = (i, j)$, где $i, j \in U$ как W и назовем его множеством пар отправитель-адресат.

Рассмотрим модель сети, в которой применяется метод пакетной коммутации с промежуточным накоплением, называемый маршрутизацией с виртуальными каналами. Пусть $L_w = \{L_w^j\}\, j = 1, s_w$ - множество виртуальных путей, соединяющих w -ю пару отправитель-адресат.

Рассмотрим некоторую пару абонентов отправитель-адресат $w \in W$. Пусть в некоторый момент времени к отправителю приходит сообщение извне, длина которого равна I. Разобьем сообщение на пакеты с длинами $I_1^w, I_2^w, \ldots, I_{s_w}^w$ и будем отправлять их параллельно по виртуальным путям таким образом, что пакет I_j^w посылается по пути L_j^w.

Тогда маршрутизация состоит в выделении для каждой пары абонентов $w \in W$ набора маршрутов $P_w = \{L_j^w, j = 1,2,\ldots, s_w\}$ в графе Γ и разбиения каждого приходящего сообщения на пакеты в соответствии с некоторым правилом.

1.2 Потоковые модели и задача оптимальной маршрутизации

В подобной постановке предлагается для измерение нагрузок в линиях связи использовать средний трафик, проходящий по линии. Точнее говоря, предполагается, что статистика потока, поступающего в любую линию (i, j), меняется только из-за обновлений маршрутов, и под нагрузкой в линии (i, j) понимается интенсивность поступающего трафика z_{ij}. Будем называть z_{ij} *потоком*, проходящим по линии

(i, j), и измерять его в единицах данных в секунду, где единицами данных могут быть биты, пакеты, сообщения и т.д.

Задача оптимальной маршрутизации формулируется тогда следующим образом:

для $\forall w = (i, j) \in W$ пары отправитель-адресат, входной процесс поступающих пакетов предполагается стационарным и имеет интенсивность λ_w. Таким образом, λ_w - интенсивность входного трафика (измеряемая в единицах данных в секунду) поступающего в сеть в узле i и адресованного узлу j. Цель маршрутизации состоит в том, чтобы трафик интенсивности λ_w разделить между несколькими путями от отправителя к адресату так, чтобы общий получающийся в результате поток по линиям в сети минимизировал некоторую стоимостную функцию, которая отражает работу сети в целом.

Примерами стоимостных функций могут служить:

$$\sum_{(i,j)} D_{ij}(z_{ij}), \text{ где } D_{ij}(z_{ij}) = \frac{z_{ij}}{C_{ij} - z_{ij}} + d_{ij} z_{ij},$$

C_{ij} - пропускная способность линии, а d_{ij} - задержка из-за обработки и распространения.

и $\max\limits_{(i,j)} \left\{ \dfrac{z_{i,j}}{C_{ij}} \right\}$,

т.е. максимум коэффициента использования линии.

Задача оптимальной маршрутизации поддается аналитическому исследованию и распределенному численному решению. Однако она имеет некоторые ограничения. Основное касается выбора стоимостной функции в качестве меры. Этот выбор основан на гипотезе, что хорошую маршрутизацию можно получить, оптимизируя средние уровни проходящего по линиям трафика. Таким образом стоимостная функция нечувствительна к возникновению и росту очередей в узлах конкретного виртуального пути. Этот недостаток, проявляется при условии возникновения и роста очередей в узлах виртуального пути.

Была построена специализированная программа моделирования, позволяющая оценить и сравнить этот алгоритм с предложенным усовершенствованием.

1.3 Функция состояния виртуального пути

Предлагается усовершенствование алгоритма оптимальной маршрутизации. Маршрутизация, основанная на измерении функции

состояния виртуального пути. Эта функция была предложена в результате проведённых экспериментов для различных конфигураций сетей, различных статистик входного трафика и различных нагрузок.

Рассмотрим некоторую пару "отправитель-адресат" $w \in W$. Пусть $P_w = \{L_j^w, j = 1,2,...,s_w\}$ - множество виртуальных путей, по которым может происходить передача приходящих пакетов. Множество P_w может представлять собой всевозможные маршруты, соединяющие данную пару или некоторое фиксированное его подмножество, это не имеет значения для дальнейшего рассмотрения. Пока будем считать, что в сети имеется единственная пара отправитель-адресат и опускать индекс w далее.

Введем *функцию состояния виртуального пути* таким образом: пусть отправителю необходимо отправить сообщение длины I в момент времени 0. Для параллельной транспортировки сообщение разбивается на пакеты длины $I_1, I_2, ..., I_s$ по числу виртуальных путей, соединяющих данную пару отправитель-адресат.

Рассмотрим, следующее соотношение $L_j \in P$, используемую далее в различных критериях, применяемых в практической реализации:

$$S_j(t) = \max_{(l,m)\in L^j} \left\{ 1 + \frac{dt_{lm}^{ожс}(t)}{dt} \right\},$$ если задержки в очередях растут и

$S_j(t) = 0$ в противном случае.

1.4 Алгоритм маршрутизации, использующий функцию состояния виртуального пути, его основные свойства

Применяя функцию состояния виртуального пути, рассмотрим алгоритм разбиения сообщения на пакеты для параллельной транспортировки.

1. Каждый отправитель пытается посылать пакеты по виртуальным путям согласно некоторому распределению потоков (первоначально сообщение может делиться на пакеты равной длины);

2. Каждый узел измеряет время задержки пакета в очереди для каждой уходящей из него линии

связи, а потом аппроксимирует производную функции времени ожидания пакета в очереди;

3. Каждый узел-адресат периодически посылает пакеты, собирающие информацию о функции состояния пути узлу-отправителю по каждому виртуального пути;

4. Узел отправитель периодически получает информацию о состоянии своих виртуальных путей, и функция состояния больше нуля, то пересчитывает коэффициенты. При этом пересчет производится таким образом, чтобы длина посылаемых пакетов по соответствующему виртуальному пути уменьшилась в $S_j(t)$ раз

При этом основными свойствами данного алгоритма являются:

- Перегрузки на определенном виртуальном пути определяются через рост очередей следующим образом:

$$\frac{dt_{ij}^{ож}}{dt}(t)=\frac{dq}{C_{ij}dt}(t)=\frac{f_{ij}(t)-C_{ij}(t)}{C_{ij}(t)}\Leftrightarrow 1+\frac{dt_{ij}^{ож}}{dt}(t)=\frac{f_{ij}(t)}{C_{ij}(t)},$$

где $q(t)$ - функция длины очереди (в ед. данных), $f_{ij}(t)$ - поток в ед. данных в ед. времени, приходящий в линию и предназначенный для передачи по ней. Далее, если каждая пара "отправитель-адресат", которая использует данную линию, будет следовать алгоритму, то соответствующий поток z_{ij} уменьшится ровно на столько, чтобы пропускной способности линии хватило на его обслуживание. Основными проблемами здесь являются лишь точность измерения производной времени ожидания (так как эта функция меняется от пакета к пакету дискретно) и выбор пути, на который будет перекладываться соответствующий поток. В практической реализации производная рассчитывалась по значениям времени ожидания в двух крайних точках, а для решения второй проблемы выбирался путь со средней задержкой сообщения на нем равной нулю.

- Функция состояния виртуального пути, как следует из предыдущего свойства, является

обобщением критерия $\max\limits_{(i,j)}\left\{\frac{z_{ij}}{C_{ij}}\right\}$ (максимум коэффициента использования линии).

2 Экспериментальная реализация, результаты проведённых экспериментов

2.1 Краткое описание программной реализации

Для проведения экспериментов и исследования функционирования предложенного алгоритма, была разработана программа имитационного моделирования поведения сети передачи данных. В ней используется дискретно-событийный принцип отсчета времени, а проверка поведения различных алгоритмов производится при различных трафиках входных потоков, для сетей с различной топологией. Система разработана с использованием COM – технологии фирмы Microsoft и библиотек ATL и MFC. В качестве хранилища для конфигурации сетей используется Microsoft DAO Jet DB.

В программу вводятся следующие характеристики сети:

1. число узлов N;
2. топологическое описание, в частности пропускные способности всех линий связи;
3. средняя длина сообщения;
4. интенсивности поступления сообщений в узлы отправители сети;
5. ограничивающий параметр (в качестве него выбирается время прогона модели);

Для каждой конкретной конфигурации сети определяется база данных, состоящая из следующих таблиц:

1. таблица узлов сети, которая используется для хранения информации об узлах сети передачи данных;
2. таблица дуг сети, каждая запись в которой представляет собой тройку (i, j, C_{ij}), где i и j - идентификаторы узлов, а C_{ij} - пропускная способность линии (i, j). Таким образом задается топологическая структура, которая будет в дальнейшем использоваться при работе алгоритма маршрутизации. Каждая дуга считается ориентированной, то есть для задания линии, способной пропускать трафик в обоих направлениях, необходимо определить 2 дуги (i, j, C_{ij}) и (j, i, C_{ji});

3. таблица пар узлов "отправитель-адресат". Эта таблица используется для генерации входного трафика в сети. Каждая запись представляет собой пару (i, j, r_{ij}), где i и j - идентификаторы узлов, i - отправитель и j -адресат, а r_{ij} -средняя интенсивность входного трафика.

Каждому узлу сети соответствует объект, который создается из выбранной конфигурации и представляет топологическую структуру сети. Он хранит следующую информацию:

1. список выходящих из данного узла линий вместе с их пропускными способностями;
2. список соседних с данным узлов;
3. статистическую информацию и информацию относящуюся к работе алгоритмов маршрутизации в данном узле;

Маршрутная таблица представляет информацию, необходимую для работы алгоритма и специфичную для него. В частности для узлов отправителей храниться список всех виртуальных путей, соединяющих их с соответствующими адресатами.

Для каждой линии выходящей из данного узла рассматривается отдельная очередь пакетов, предназначенных для отправки по данной линии. Поступающие пакеты попадают в эту очередь, если линия связи занята.

2.2 Результаты проведённых экспериментов

Проводились эксперименты на различных конфигурациях сетей и для различных условий. В статью включены 2 наглядных примера.

Конфигурация с одной парой отправитель-адресат

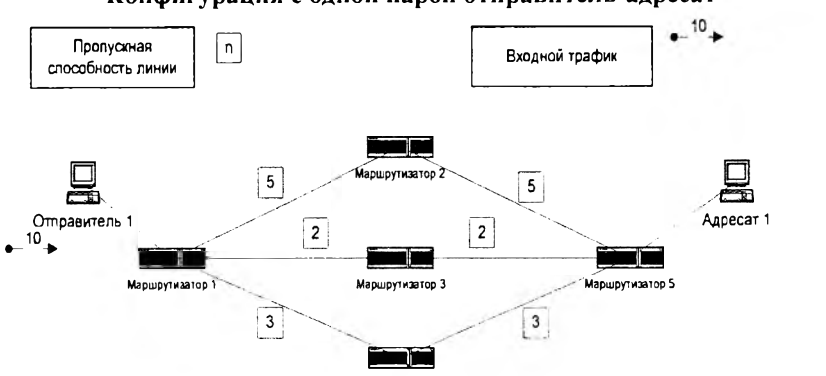


Рисунок 1. Поведение сети при предложенном алгоритме (сообщения поступают через равные интервалы времени с интенсивностью 10 сообщений в ед. времени, длина 0.97).

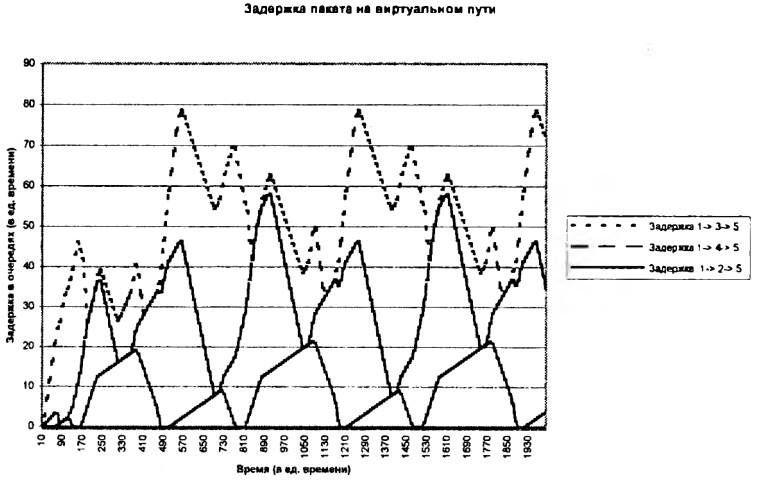


Рисунок 2. Поведение сети при алгоритме оптимальной маршрутизации (сообщения поступают через равные интервалы времени с интенсивностью 10 сообщений в ед. времени, длина сообщения 0.97).

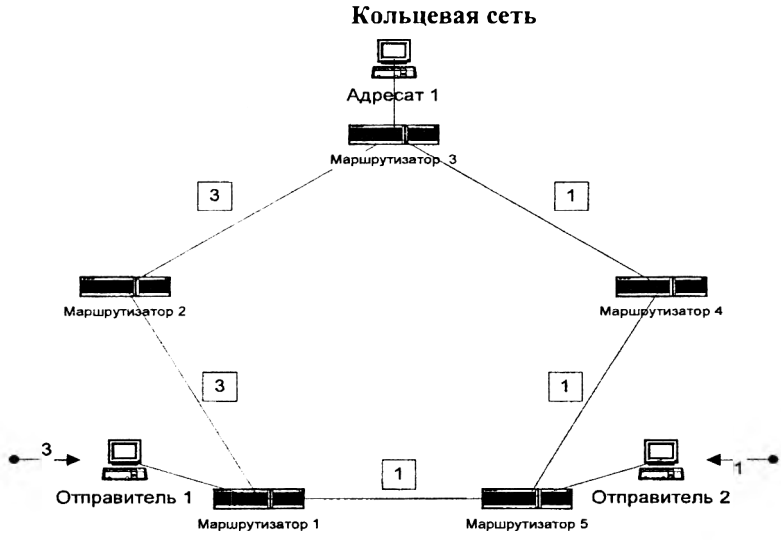


Рисунок 3. Поведение сети при предложенном алгоритме (сообщения поступают через равные интервалы времени с интенсивностями 3 и 1 сообщений в ед. времени, длина 0.97, рассматривается ОА – пара 1_3).

Рисунок 4. Поведение сети при алгоритме оптимальной маршрутизации (сообщения поступают через равные интервалы времени с интенсивностями 3 и 1 сообщений в ед. времени, длина сообщения 0.97, рассматривается ОА - пара 1_3).

Заключение

На основании полученных данных, можно говорить о том, что разработанный способ измерения загруженности виртуального пути в сети передачи данных может эффективно применяться на практике совместно с другими алгоритмами следующим образом: пересчет потоков, который передаются по сети производится по одному из известных алгоритмов, а в промежутках времени между итерациями применяется разработанный метод (используется величина изменения времени простоя сообщения в очередях). Эта схема чувствительна к изменениям в задержке на путях некоторой пары отправитель-адресат, вызванным возникновением перегрузок для каждого конкретном пути.

Литература

1. Бертсекас Д., Галлагер Р., Сети передачи данных. – М: Мир, 1989.
2. Васильев Н.С., Математическое моделирование в задачах маршрутизации сетей передачи данных (многокритериальный подход). – Диссертация на соискание ученой степени доктора физико-математических наук, Москва – 1999.
3. Клейнрок Л., Коммуникационные сети. Стохастические потоки и задержки сообщений. – М: Наука, 1970.
4. М. Шварц, Сети связи. Протоколы, моделирование и анализ.. – М: Наука, 1992.

5. Robert G. Gallager, A Minimum Delay Routing Algorithm Using Distributed Computation. – IEEE Trans. On Communications, COM-25, No.1, January – 1977.
6. D.G. Cantor, M. Gerla, "Optimal routing in a packet switched computer network," – IEEE Trans. Comput., vol. C-23, Oct. 1974.

Суслов А.А.

Извлечение данных из реляционных источников по метаданным

Введение

Применяемые на сегодняшний день поисковые машины опрашивают web-сайты и индексируют страницы по ключевым словам. При этом велика вероятность несоответствия результата поиска ожидаемому результату запроса, поскольку подстановка определенного количества невидимых для пользователя ключевых слов (чем иногда злоупотребляют кодировщики HTML) приводит к полному несоответствию запроса результатам поиска.

Идея поисковой машины по метаданным состоит в автоматизированном извлечении метаинформации о структуре и содержании данных, которая не зависит от желания создателей предоставить информацию в том или ином виде. Таким образом, пользователь видит формальное описание и часть реальной информации, которую ему предоставляет источник. Но, с одной стороны, каждый источник данных предоставляет собственные механизмы работы с хранящейся в нем информацией, с другой стороны, пользователю совершенно неважно, где и как хранятся нужные данные, а важно их найти и представить на экране. Так как же пользователю описывать то, чего он хочет найти?

Для каждой предметной области вводится метамодель, над которой собственно и будут строиться запросы. Метамодель с метаданными образуют репозитарий, представляющий собой набор XML-файлов. Репозитарий находится под управлением ядра системы, или, другими словами, метамашины.

Создание такой модели обеспечивает пользователю адекватное соответствие запрашиваемой и реально доступной информации. Кроме того, пользователь получает возможность формально определить и откорректировать свой запрос, используя языковые средства навигации по метаданным.

Безусловно, необходимо обеспечить пользователя удобными графическими интерфейсами и формальными языками для формулирования запросов. С другой стороны, придется разрабатывать средства преобразования метаданных из различных источников в общую метамодель, а также извлечения данных из источников по метаданным. Задача проектирования общей метамодели будет во многом похожа на создание Хранилищ Данных, поскольку придется как

проектировать общую модель данных, так и реализовать механизмы ETL (Extraction-Transformation-Loading).

В данной работе предложена архитектура поисковой системы по метаданным, источниками информации для которой служат реляционные базы данных. Обсуждаются наиболее важные аспекты проектирования системы и предлагаются механизмы для ее реализации.

Изложение основано на конкретной реализации прототипа системы, поэтому некоторые решения приводятся как фактически реализованные в прототипе и их обсуждения опущены.

1. Архитектура системы

Всю систему можно разделить на три относительно независимые части:

- Пользовательская интерфейсная часть
- Метамашина
- Источники данных (и механизмы доступа к ним)

За хранение и обработку метаданных отвечает ядро системы – метамашина, интерфейсная часть системы отображает метамодель на экране и контролирует взаимодействие с пользователем. В качестве языка запросов к метаданным предлагается использовать некоторое подмножество принятого в UML [1] языка выражений OCL (Object Constraint Language) [1]. Выделенное подмножество языка будем называть UQL (Unified Query Language –Унифицированный Язык Запросов). Формальное описание UQL выходит за рамки данной работы и не является здесь обязательным, отметим только, что поскольку OCL (и, соответственно, UQL) является языком выражений, он не позволяет каким либо образом «испортить» метаданные и не влечет побочных эффектов.

В качестве формата представления метаданных выберем MOF (Meta Object Facility) [2], принятый OMG, и XMI (eXtended Metadata Interchange) [3] как спецификацию отображения MOF на XML [4]. MOF является гораздо более проработанной и развитой спецификацией, чем, например, RDF (Resource Description Framework) [5], принятой W3C.

Идея организации метаинформации такова – метамодель и метаданные (представленные, например, как отдельный XMI-файл), должны содержать информацию об источнике данных и отображении из MOF-модели в схему базы данных (БД). Тем самым будет достигаться независимость метаинформации – по отдельному файлу можно восстановить подключение и преобразования, необходимые для доступа к данным.

Метаданные состоят из двух основных частей – пользовательской и служебной. Пользовательская содержит метаданные, которые будут доступны пользователю для

формулирования запросов. Служебная состоит из метаинформации, необходимой для соединения с источниками данных и перевода запросов из UQL во внутренние языки источников, а именно:

- Строки подключения (connect string)
- Информации для преобразования объектов MOF в объекты БД (имена структур данных источников, типы данных и ограничения)

1.1 Пользовательский интерфейс

В браузер пользователя загружается апплет, в котором в UML-подобном виде представлена метамодель по выбранной предметной области (метамодель строится на пользовательской части метаданных). Пользователь либо непосредственно формулирует запрос на UQL, либо использует графические возможности навигации по метамодели, в любом случае в результате формируется запрос на UQL. Метаданные содержат также значения основных атрибутов (реляционных таблиц), определяющих данные, поэтому пользователю предоставляется возможность выбрать заведомо существующие данные, пользуясь значениями этих атрибутов.

1.2 Метамашина

Метамашина хранит метаданные и выполняет преобразование запроса из UQL в специфичные для выбранных источников данных языки, для реляционных БД это SQL. Метамашина содержит библиотеку обработчиков (parsers), к которым обращается после определения источников, задействованных в запросе. В данной работе не рассматриваются конкретные системы хранения XML-данных, однако как вариант предлагается использовать XML сервер Tamino компании Software AG.

1.3 Доступ к источнику данных

Доступ к реляционному источнику осуществляется посредством ODBC/JDBC, адрес источника также является служебной метаинформацией, содержащейся в метаданных.

2. Особенности реализации

2.1 Проектирование общей модели метаданных

Целью проектирования является выработка спецификации метаданных для выбранной предметной области, которая предоставляется владельцам источников данных для организации автоматического взаимодействия с источником. Помимо метаданных предметной области, потребуются также и служебные метаданные, представленные как служебные классы метамодели.

Мы предполагаем, что в метамодели собраны данные из нескольких источников, поэтому каждый экземпляр каждого класса метамодели должен иметь ссылку на источник-происхождение данного экземпляра. В случае реляционных баз данных экземпляр соответствует строке таблицы. В объединенном классе его экземпляры могут принадлежать разным источникам. Для идентификации источника выделяется отдельный класс и создаются ассоциации композиции от каждого класса пользовательской части метаданных к данному. Композиция здесь оправдана, т.к. каждый экземпляр класса - строка в реляционных БД – принадлежит какому-либо источнику.

Подобно Хранилищам Данных, общая метамодель может иметь собственную систему именований классов и их атрибутов, отличающуюся от принятой в источнике данных (а при нескольких независимых источниках переименования и вовсе не избежать). Пользователь составляет запросы в пространстве имен общей модели, в то время как реально они будут выполняться для каждого источника в собственном пространстве имен. Поэтому необходимо ввести структуры для преобразований, которые физически могут быть представлены отдельным XMI-файлом и также образовывать репозитарий. Вполне вероятно, что для каждого источника потребуются свой набор служебных структур.

2.2 Представление реляционных данных в формате MOF

В итоге мы хотим привести реляционные метаданные к общей схеме. Поскольку источник данных сам по себе ничего не знает об общей метамодели, т.е. именах сущностей и типах данных, принятых в ней, необходимо учесть эти знания при генерации XMI-файла. Здесь возможно два принципиальных подхода:

Вариант А. Полученный непосредственно их реляционной схемы XMI-файл не является последней итерацией, в этом случае он будет простым отображением структур данных в XMI-формате. Необходимы дальнейшие преобразования файла с учетом общей метамодели.

Вариант В. При генерации из источника сразу применяются знания об общей метамодели и на выходе получается файл, «понятный» для метамашины. Создаются также и структуры для обратного преобразования.

Выбор подхода зависит от разделения задач между участниками проекта. Вариант А предпочтителен, когда задача извлечения метаданных их источника лежит на администраторе БД, а задача приведения к общей метамодели – на ее разработчике, второй подход оправдан, если оба по сути одно и тоже лицо – удобнее сразу же преобразовывать метаданные в необходимый формат.

На основе знаний об общей метамодели выполняются следующие действия:

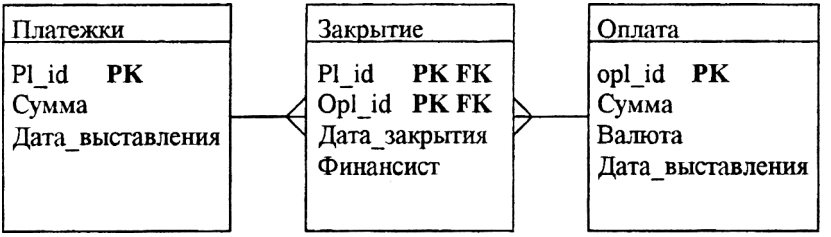
- Выборка таблиц, столбцов и их значений, которые будут выступать как метаданные
- Переименование, приведение типов данных и, возможно, значений (в разных системах возможно применение различных обозначений семантически одних и тех же сущностей – приведение их к единому формату).

На основе произведенных операций ETL создастся XMI-файл (возможные форматы файла обсуждается ниже), по которому возможно будет обратное преобразование из UQL в SQL.

Для решения этой задачи можно использовать графический инструмент проектирования прикладных систем (напр., Rational Rose UML Modeler), имеющий доступ к различным источникам данных (напр., через ODBC/JDBC); или же специфический инструмент для построения Хранилищ Данных. Разработчик метамодели «вытягивает» модель данных на экран и вручную осуществляет выбор требуемых таблиц и тех столбцов, значения которых будут расцениваться как метаданные. Также в качестве метаданных будут выступать схемы выбранных отношений в проекции на выбранные атрибуты. Приведение типов данных в простейшем случае может выполняться вручную на основе общей метамодели, либо автоматически сопоставлением таблиц источника и классов метамодели. Rational UML позволяет генерировать метамодель в MOF-формате.

В силу разницы реляционной и объектной модели, заложенной, соответственно, в БД и MOF, возможно возникновение сложностей, которые невозможно решить автоматически. Одна из таких сложностей заключается в том, что некоторые наборы таблиц отображаются в MOF в другом соотношении, нежели чем 1:1, например, ассоциация m:n в реляционной системе реализуется через три таблицы, в то время как в MOF (и соответственно UML) она реализуется как m:n ассоциация между двумя классами. Однако не всегда позволительно перевести три таблицы в два класса – в тех случаях, когда промежуточная таблица кроме foreign key на две другие также имеет смысловые атрибуты и тогда непонятно, что с ними делать – приходится оставлять как отдельный класс.

Рассмотрим пример: есть таблица выставленных платежных поручений-требований, и есть таблица оплат по этим поручениям. В примере для простоты опущено много деталей.



Здесь не получится перевести три таблицы в два класса Платежки и Оплата с множественной ассоциацией m:n, т.к. в таблице Закрытие существуют семантически значимые поля Дата_закрытия и Финансист, которые некорректно с точки зрения предметной области отнести к объекту Платежки или Оплата.

По опыту создания Хранилищ Данных, при объединении метаданных из различных источников могут возникать самые разные коллизии имен, типов данных и даже значений. Их разрешение может привести к следующим необходимым преобразованиям данных при интеграции:

1. Изменение типов элементов. Например, один источник данных использует числовой код для идентификации какой либо сущности, а другой – литерный. Очевидно, что в общей метамодели будет принят литерный код.
2. Атрибуты отношения могут переходить в атрибуты другого (не соответствующего ему) класса. Вариант, когда модель БД не проецируется непосредственно на MOF (класс в класс).
3. Отображение сущностей в соотношении другом, нежели чем 1:1. (см. рисунок выше)
4. Некоторые значения столбцов переходят в элементы или атрибуты или, более вероятно, наоборот – атрибуты или имена таблиц перейдут в значения элементов (как один из вариантов агрегации). Например, общая метамодель объединит монеты и бумажные денежные знаки и вместо таблиц coin и bond появится элемент или атрибут money_type, значения которого могут быть coin или bond.
5. Изменение значений атрибутов таблиц. Классический пример, обозначения пола в одной системе могут быть как «муж» и «жен», а в другой – «М» и «Ж». Общая метамодель возможно предлагает собственную систему обозначений. Т.е. запрос на UQL формулируется с условием where sex = 'M', а SQL нужно сформулировать с условием where sex = 'муж'. Таких элементов, как правило, бывает совсем немного.
6. Возникновение дополнительных элементов. Общая модель может предусматривать поддержку нескольких языковых интерфейсов и возможности задавать запросы на нескольких языках. Для этого

потребуется на уровни общей модели ввести дополнительные атрибуты, выраженные на различных языках. Источники данных таким многообразием могут не обладать.

Безусловно, здесь приведены далеко не все, а лишь наиболее очевидные преобразования. Для их корректного выполнения при переводе запроса из UQL над общей метамоделью в SQL над конкретным источником потребуются дополнительные структуры XML, определенные на разных уровнях иерархии MOF – отдельного элемента или всего экземпляра класса. Служебные атрибуты также потребуются и для выражения дополнительных сведений о метаданных. На уровне экземпляра класса определяется источник данных *data_source*. На уровне элемента определяется:

- Принадлежность к таблице *source_table*
- Имя элемента в источнике *source_name*
- Тип данных элемента в источнике *source_type*
- Значение элемента в источнике *source_value*
- Видимость элемента для запросов пользователя *is_visible*.

Выделим два основных подхода к XMI-представлению реляционных метаданных. Первый из них реализуем только при выборе варианта А и заключается в инкапсулировании частной схемы в общий формат описания реляционной модели. Примерная схема XMI-сущности может выглядеть так:

```
<Table XMI.id='t1'>
  <name>Emp</name>
  <columns>
   <column>
    <name>First name</name>
    <source_name>first_name</source_name>
    <source_type>varchar2</source_type>
    <value>John</value>
   </column>
   <column>
    <name>Last name</name>
    <source_name>last_name</source_name>
    <type>string</type>
    <source_type>varchar2</source_type>
    <value>John</value>
   </column>
  </columns>
  <targetTables>
   <XMI.reference target='t2' />
  </targetTables>
 </Table>
```

Достоинства и недостатки приведенного подхода:

+ Единая форма для всех реляционных источников данных – один DTD

- Громоздкость: для каждого атрибута в каждом кортеже указывается имя, тип, значение и т.д.

- Имя столбца, поскольку является значением, а не именем тега, не поддается проверке на целостность

+ Возможно хранение типов данных столбцов, а также любых других важных данных путем простого добавления соответствующих тегов – расширяемость схемы без нарушения концепции ее построения

- Данный XMI не может выступать как конечный файл, т.к. общая метамодель не будет представлена в терминах таблиц. Требуется дальнейшее преобразование подобного файла

Второй подход может быть реализован как при варианте А, так и при варианте В и представляет по сути XMI запись реляционных структур:

```
<coin xmi.id = 'c1' type='SQL'>
 <id type='INTEGER'>
     source_name='ID'
  000001
 </id>
 <nominal type = 'REAL(4,2)'
       source_name = 'NOMINAL'>
   2,5
 </nominal>
 <name type = 'REAL(4,2)'
        source_name = 'NOMINAL'>
   FRANCS
 </name>
 <REF_metal>
   <Coins.Metal xmi.idref = 'm1'/>
 </REF_metal>

 <metal xmi.id='m1' type='SQL'>
 <id type='INTEGER'>
     source_name='ID'
  000001
 </id>
 <name type = 'VARCHAR2(30)'
        source_name = 'METAL'>
  Alluminium
 </name>
 </metal>
```

Преимущества и недостатки:

\+ Структура метаданных отражает реальную структуру в базе, поэтому более наглядна и близка к структуре MOF.

\+ Более краткая запись, элементы для преобразования спрятаны в атрибуты.

\- Для каждой предметной области нужно составлять собственные DTD или XML Schema.

Теперь необходимо обеспечить механизм, который позволит по метамодели и метаданным получать данные из соответствующих источников.

2.3 Извлечение данных по метаданным

Вся задача построения запроса к источникам разбивается на три подзадачи:

1. Формулировка UQL-запроса.
2. Преобразование UQL в SQL
3. Перевод SQL в ODBCSQL

Условия запроса формируется по имеющимся данным, и, соответственно, определяют конкретные экземпляры классов. Становится известной область значений xmi.id (см. примеры выше) для каждого класса, удовлетворяющая условиям запроса. Каждый экземпляр класса имеет указатель на свой источник данных; таким образом определяются источники, обладающие запрошенными данными. Ядро метамашины вызывает соответствующие обработчики, использующие структуры метаданных, необходимые для преобразования имен и типов данных.

Как должны выглядеть эти структуры данных? Здесь, опять же, возможно два принципиальных подхода. Либо мы объединяем служебные и пользовательские метаданные, либо отдельно оставляем представление объектов в общей метамодели, а отдельно выносим их реальную структуру. В первом варианте объект получается самодостаточным - содержит как собственное описание, так и информацию обо всех необходимых преобразованиях для доступа к его данным, во втором объект не перегружен излишней информацией о своих свойствах в источнике. Оба варианта альтернативны, если в качестве источников используются только, например, реляционные БД, однако сравнивая оба подхода в применении к разнородным источникам данных, становится очевидным преимущество второго, либо напрашивается компромисс между ними. Поясним недостатки первого варианта.

Тип источника определяет только экземпляр класса, значит на уровне самого класса все объекты имеют одинаковую структуру. Это означает, что все объекты одного класса должны содержать все структуры, необходимые для преобразовании запроса для всех

источников данных, кроме того, при добавлении нового типа источников будет меняться вся метамодель.

Поэтому в качестве варианта решения предлагается в общей метамодели оставлять только указатель на источник данных, точнее, на некоторый фильтр, отображающий структуру общей метамодели на структуру отдельно взятого источника. Фильтр может быть построен по принципу XSL – каждому элементу метамодели в зависимости от его расположения в дереве XMI-документа ставится в соответствие структурный элемент источника данных. Анализатор UQL запроса использует этот фильтр подобно тому, как компилятор использует таблицы имен, замещая их при преобразовании запроса. Поскольку фильтр также является XMI-файлом, все метаданные могут храниться и обрабатываться единым XML-сервером.

В результате на выходе мы получаем SQL запрос (на стандартном SQL или же сразу в виде вызовов ODBC), ориентированный на структуру таблиц в источниках данных.

Заключение

В данной работе предложена архитектура поисковой системы по реляционным данным с предварительным выделением метаданных и построением запроса на формальном языке. Система, прототип которой находится в стадии реализации, предоставит возможность использования реляционных баз данных как источников информации для Интернет-ориентированных поисковых систем. Существенным является использование открытых технологий и форматов хранения и передачи данных между компонентами системы, таких как XML, MOF и XMI, что является гарантом гибкости системы и открытости для интеграции с другими приложениями.

Литература

1. OMG Unified Modeling Language Specification
   http://www.omg.org/cgi-bin/doc?formal/00-03-01.pdf
2. Meta Object Facility (MOF) Specification. Version 1.3 RTF.
   http://www.omg.org/cgi-bin/doc?formal/00-04-03.pdf
3. XML Metadata Interchange (XMI) Proposal to the OMG OA&DTF RFP 3: Stream-based Model Interchange Format (SMIF)
   http://cgi.omg.org/cgi-bin/doc?ad/98-07-01
4. Extensible Markup Language (XML) 1.0 W3C Recommendation.
   http://www.w3.org/TR/2000/REC-xml-20001006
5. Introduction to RDF Metadata. W3C Note.
   http://www.w3.org/TR/REC-rdf-syntax,
   http://www.w3.org/TR/rdf-schema

Калугина Н.В., Шляховая Е.М.

Автоматизация построения распределенных информационных систем.[2]

Введение.

В настоящее время является актуальной задача построения распределенных информационных систем (ИС). Примерами областей применения распределенных ИС могут быть задача автоматизации документооборота [1], задача построения распределенных систем управления исходным кодом и контроля версий [2], [5], а также задачи автоматизации сбора и анализа информации от множественных источников. Следует отметить, что в данной работе авторы понимают под термином «информационная система» любую систему, обеспечивающую в той или иной форме хранение, преобразование и доступ к пользовательской информации, хотя такое толкование и является более широким, нежели традиционная трактовка этого термина. При таком понимании к классу информационных систем могут быть отнесены, в частности, системы контроля версий, которые, как правило, являются интегрированными в системы программирования.

Ведущие корпорации-производители программного обеспечения выпустили на рынок ряд распределенных ИС, покрывающих потребности указанных предметных областей. Однако эти продукты преимущественно решают частные задачи распределенного хранения и обработки информации. В то же время более глубокий анализ показывает, что большинство распределенных ИС опирается на одинаковую модель хранения, организации и управления данными и для их реализации применяются концептуально схожие методы и технологии [3], [4], что позволяет говорить об актуальности задачи построения ИС, способных адаптироваться к специфике конкретных предметных областей.

В результате детального анализа можно выделить некоторое необходимое функциональное ядро информационной системы. Причем данная общность не ограничивается только лишь требованиями наличия определенных функциональных возможностей, но и предполагает использование определенных общих идеологических подходов к организации как систем в целом, так и отдельных их компонент. Такой взгляд приводит к выводу о разумности и целесообразности создания некоторого ***адаптируемого базового инструментального средства***, при помощи которого путем настроек на индивидуальные особенности

[2] Данная работа поддерживается грантами РФФИ №№ 00-01-00385, 00-01-06162, 00-01-06163.

можно получать информационные системы для решения частных прикладных задач. Наличие такого конструктора позволит охватить достаточно широкий класс задач построения распределенных информационных систем, удовлетворяющих нуждам различных областей коллективной деятельности. Ниже мы рассмотрим основные подходы к построению предлагаемого инструментария, а также обсудим структурную схему и реализацию этого средства.

1. Основные понятия и модели.

Как уже отмечалось, целью построения системы-конструктора является реализация базовой функциональности, а именно:

- функции локального хранения и передачи данных между узлами хранения
- механизм авторизованного доступа
- механизм отслеживания событий, происходящих в системе, и средства определения обработчиков этих событий.

При этом важно построить модель организации и управления данными, которая впоследствии может быть применена к широкому классу частных задач. Это означает, что в рамках этой модели нам следует описать структуры данных и логику управления ими в терминах некоторых базовых понятий, позволяющих абстрагироваться от специфики конкретной прикладной области, но дающих нам возможность реализовать необходимое функциональное ядро.

Для описания данных прикладной области вводится абстракция **документа** - элементарной единицы пользовательских данных произвольного формата и смыслового содержания, обладающей определенной законченностью и самостоятельностью. Весь объем хранимых и обрабатываемых системой пользовательских данных может быть представлен как некоторая совокупность отдельных документов, как независимых, так и, возможно, определенным образом связанных между собой.

В рамках предлагаемой системы не делается никаких предположений о содержании и смысловой нагрузке конкретных документов. Вместо этого каждый документ наделяется некоторым набором атрибутов различных типов, значения которых в совокупности полностью определяют статус и поведение документа в системе. Значения атрибутов документа, в общем случае, - это единственная информация, доступная обобщенной системе для анализа и, при определенных обстоятельствах, для изменения.

Атрибуты могут быть как общими для всех документов (например, автор, дата создания и т.п.), так и специфичными для определенного класса документов. Таким образом, мы приходим к

понятию типа документа: описание типа документов определяет совокупность атрибутов, ассоциированных с документами этого типа[3].

Функциональные возможности системы в части управления документами определяются, таким образом, ее способностью задавать, изменять и интерпретировать значения атрибутов документов. При этом важно отметить, что в большинстве частных случаев действия информационной системы по управлению данными (в терминах нашей модели – действия по изменению значений атрибутов документов) будут зависеть от их текущего состояния (т.е. от текущих значений этих атрибутов). Таким образом, было бы весьма желательно определить в нашей модели некоторый механизм, позволяющий запрограммировать выполнение определенных действий над документами при условии достижения ими определенного состояния (т.е. определенной совокупности значений атрибутов).[4] Для решения этой задачи вводится в рассмотрение механизм триггеров - процедур, выполнение которых привязывается к специфической совокупности значений атрибутов для документов определенного типа. Любой такой триггер имеет шаблон, представляющий собой условия на значения атрибутов; выполнение триггера активизируется всякий раз при появлении в системе документа, обладающего набором значений атрибутов, совпадающим с шаблоном данного триггера. Триггеры задаются при настройке базовой системы, и для каждой частной системы, получаемой из базовой, они будут своими.

Заметим, что несмотря на то, что базовая система не предполагает какой-либо интерпретации пользовательской части документов, при построении частной системы может возникнуть необходимость при определенных условиях производить действия с содержимым документов, причем это могут быть действия, как связанные с изменением этого содержимого, так и не изменяющие данные, содержащиеся в документе, но включающие в себя некоторую их интерпретацию (примером такого действия может служить поиск статей по определенной тематике в информационной системе библиотечного типа или статистическая обработка поступающих данных в системах сбора и анализа информации). Такой доступ к пользовательской части документов может быть осуществлен посредством триггеров.

---

[3] Напомним, что с точки зрения абстракции документа пользовательские данные, содержащиеся в нем, не интерпретируются системой хранения, и поэтому мы не можем говорить о каких-либо смысловых критериях, задающих тип документа по его содержанию.

[4] Понятно, что логика управления данными в каждой частной системе, построенной с помощью базового инструментария, будет своей.

Таким образом, триггеры представляют собой механизм, позволяющий в рамках абстракции документа осуществление доступа к пользовательской части документа, и являются тем самым некоторым аналогом методов, привязанных к типам, в объектно-ориентированном подходе к организации данных. Базовая система сама по себе не включает в себя каких-либо средств обработки пользовательских данных, но предоставляет гибкий инструментарий для определения таких средств в рамках конструирования частных систем. Кроме того, удается инкапсулировать все процедуры обработки данных внутри определяемых при настройке событийно-ориентированных процедур, что позволяет говорить об объектно-ориентированном взгляде на организацию пользовательских данных в базовой системе.

Сама идея распределенной информационной системы предполагает поддержку доступа к содержащейся в ней информации множества пользователей. Во многих, если не во всех случаях реального применения информационных систем в прикладных областях возникает необходимость помимо обеспечения работы в многопользовательском режиме контролировать и/или разграничивать доступ к определенным частям хранимой информации. Таким образом, необходимо определить некоторую модель авторизации и разграничения доступа к документам, хранящимся в системе.

В нашей модели мы будем рассматривать следующие основные типы доступа к документам: доступ к пользовательской части документа по чтению (просмотр), по изменению, создание и удаление документов. Кроме того, отдельно следует рассматривать права на изменение значений атрибутов.

В основу модели авторизации и разграничения доступа положено следующее предположение. Будем считать, что как пользователи, так и документы в системе можно некоторым образом разбить на группы, так что все пользователи внутри одной группы обладают одинаковыми правами доступа к документам, причем права доступа группы пользователей определяются не в терминах отдельных документов, а в терминах групп документов. Каждый пользователь может входить в одну или более групп пользователей, называемых далее рабочими группами. Аналогично каждый документ может входить в одну или более групп документов. В общем случае если пользователь не входит ни в одну рабочую группу из числа тех, для которых определены какие-либо права доступа к некоторой конкретной группе документов, то система никаким образом не ставит пользователя в известность о существовании этих документов. Тем самым, задание системы прав доступа не только позволяет контролировать создание, удаление и модификацию документов, но и позволяет определять области видимости документов в разрезе пользователей.

Развивая концепцию рабочих групп, можно выдвинуть идею наследования групповых прав, т.е. новая рабочая группа может быть получена на основе уже существующей путем выделения некоторого подмножества пользователей из числа входящих в первую группу и некоторого подмножества объектов доступа из числа определенных для базовой группы. Заметим, что механизм наследования прав не предполагает расширения множества прав при наследовании, напротив, такое множество может быть только сужено. Механизм наследования удобен при построении иерархических моделей доступа, а также при определении дополнительных ограничений внутри однородных рабочих групп с учетом специфики конкретной задачи.

Такая модель определения и разграничения прав доступа при своей логической простоте обладает необходимой гибкостью, позволяя задавать как очень простые, так и весьма разветвленные системы прав доступа, а также при необходимости усложнять и масштабировать уже существующие схемы доступа без потери имеющихся ограничений.

Рассмотрим теперь абстрактную модель, описывающую принципы организации распределенного хранения данных с учетом уже введенных нами концепций документов и их поведения, пользователей и модели авторизации и разграничения доступа.

Систему распределенного хранения информации можно рассматривать как логически единое хранилище документов, представляющее собой некоторую совокупность локальных узлов физического хранения информации, взаимодействующих друг с другом для обеспечения общей логики хранения и обработки данных. Логическая целостность данных независимо от способа и места их хранения, а также разделяемый доступ к общим данным обеспечиваются системой автоматически.

Будем рассматривать узел хранения как некоторую локальную среду физического хранения, в которую при необходимости может быть помещена или извлечена порция информации. Априори предполагаем, что любая информация, помещенная в систему, в любой момент времени физически находится в одном и только одном узле хранения; дублирование информации одного узла на другом узле не допускается. Таким образом, для каждого документа в системе можно однозначно указать узел, на котором он хранится, хотя такой подход не запрещает ситуаций, когда на протяжении жизненного цикла документа он может перемещаться с одного узла системы на другой.

Рассмотрим теперь принцип, по которому документы распределяются между отдельными узлами, а также необходимые механизмы взаимодействия узлов друг с другом. В качестве базовой предпосылки будем исходить из того факта, что, как правило, идея распределенного хранения данных в ИС предполагает целесообразность группировки этих данных в зависимости от того, какой частью

пользователей системы данная информация с наибольшей вероятностью будет востребована. Кроме того, в случае сильно распределенных многопользовательских систем кажется разумным предположение об определенной локализации пользователей. С точки зрения экономии накладных и транспортных расходов было бы желательно организовать хранение всей или большей части информации, необходимой данному пользователю, на том локальном хранилище, которое расположено, с учетом топологии и транспортных сред, наиболее близко к данному сегменту (либо входит в него).

Объединяя эту идею с вышеизложенной концепцией рабочих групп, приходим к следующему базовому принципу распределения информации по узлам. С каждым узлом системы ассоциируется одна или несколько рабочих групп пользователей, причем каждая рабочая группа закрепляется ровно за одним узлом (пользователей, входящих в рабочую группу, закрепленную за узлом, будем называть зарегистрированными на данном узле). После того, как данное закрепление осуществлено, распределение документов по узлам производится таким образом, что документы из некоторой группы документов хранятся на узле, ассоциированном с рабочей группой, которая имеет доступ к документам из этой группы. Если же существует другой узел, за которым закреплена рабочая группа, также имеющая доступ к этим документам, то такой узел обеспечивает логическое хранение документа за счет взаимодействия узлов, т.е. он в случае необходимости предоставляет необходимые документы своим пользователям, запрашивая их у узла, на котором в реальности производится хранение. С точки зрения пользователя такая ситуация ничем не отличается от той, в которой документ хранится на том узле, где зарегистрирован пользователь.

Заметим, что при таком определении места хранения возможен некий произвол, связанный с тем, что к одной и той же группе документов могут иметь доступ различные рабочие группы, причем они, в общем случае, могут быть закреплены за разными узлами, и тем самым, документы из указанной группы могут быть помещены на любой из этих узлов. В качестве дополнительных факторов, принимаемых во внимание при выборе места хранения конкретного документа, могут выступать как характеристики самих узлов-кандидатов, задаваемые при конфигурировании узла (например, емкость конкретного локального хранилища), так и аспекты, связанные с документом, а именно: авторство (предпочтение может отдаваться узлу, на котором зарегистрирован пользователь-автор документа, исходя из предположения, что он востребует этот документ с наибольшей вероятностью), приоритет типа права (предпочтение отдается узлу, рабочая группа которого имеет доступ к документам из этой группы по изменению, нежели по чтению) и т.п.

2. Концепция работы системы.

Общая схема работы системы представляется следующей. Система принимает от пользователей на хранение данные в виде документов - объектов, содержащих порцию пользовательских данных и обладающих некоторым набором атрибутов, состав которого задается типом документа. Логически документ помещается в распределенное хранилище и каждому пользователю, в зависимости от его принадлежности к определенным группам, присваивается тот или иной диапазон прав на доступ к данному документу (вплоть до отсутствия каких бы то ни было прав).

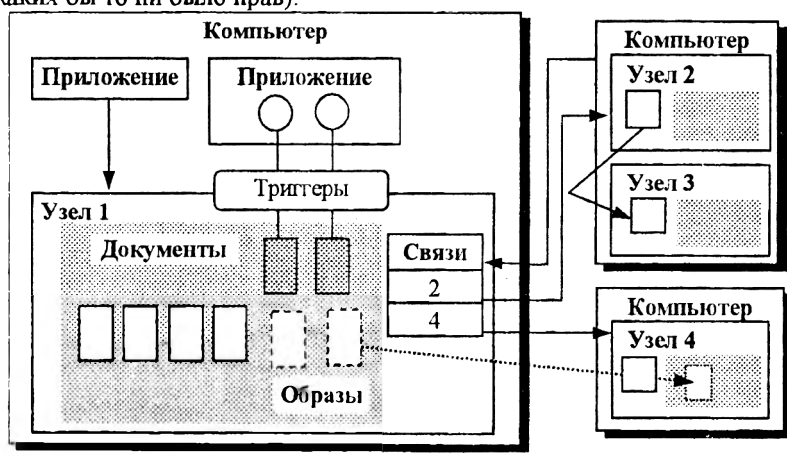


Рис. 1 Взаимодействие узлов системы.

Физическое хранение документа осуществляется локальным хранилищем одного из узлов системы, все остальные узлы системы при необходимости получают доступ к этому документу, используя механизм внутреннего взаимодействия. Этот механизм основывается на следующем принципиальном положении. Любой документ хранится в системе в единственном экземпляре на одном и только одном узле системы. На всех остальных узлах, которым необходимо работать с этим документом (т.е., к ним прикреплены пользователи, у которых есть права на доступ к нему), хранится "образ" данного документа - специальный объект, содержащий ссылку на реальное месторасположение документа. При необходимости доступа к документу узел, на котором находится образ изменяемого документа, обращается к узлу, на котором реально хранится документ, с соответствующим запросом (Рис. 1). Такой подход позволяет избежать копирования информации и эффективно организовать взаимное исключение при работе с разделяемыми данными.

Образ хранится в системе как обычный документ и обладает таким же набором атрибутов, как и документ, на который он ссылается, но вместо пользовательских данных содержит информацию о документе, расположенном на другом узле. При этом для образа устанавливается некоторое специфическое значение определенного служебного атрибута, позволяющее системе отличать образы от обычных документов. Что касается остальных атрибутов образа, то здесь предлагается настраиваемый механизм тиражирования, позволяющий выбрать один из трех режимов обновления значений:

- Локальные поля. Значения атрибутов образа никак не связаны со свойствами оригинала и являются локальными для данного узла, т.е. значения атрибутов образа берутся по месту хранения образа, а пользовательские данные - по ссылке образа.
- Пассивное обновление. Значения атрибутов образа обновляются после обращения к оригиналу по значениям полей оригинала, т.е. значения атрибутов образа отражают значение и состав атрибутов оригинала на момент последнего обращения к пользовательским данным образа (т.е. перехода по ссылке, хранящейся в образе).
- Активное обновление. Узел-хранитель оригинала сохраняет также информацию обо всех ссылках на оригинал и автоматически обновляет значение атрибутов всех образов в момент обновления значений атрибутов оригинала, т.е. реализуется обычное тиражирование.

На каждом узле системы систематически производится проверка текущего состояния значений атрибутов для каждого документа и вызываются триггеры, к шаблонам которых подходит набор значений атрибутов данного документа (если такие триггеры имеются).[5] Таким образом, триггеры можно представлять себе как своего рода обработчики событий, при этом в качестве событий выступает появление на узле документа определенного типа с заданным набором атрибутов.

Действие триггеров распространяется только на документы, хранящиеся на данном узле, при этом на каждом узле системы могут быть зарегистрированы свои триггеры, что позволяет на этапе написания триггеров реализовать на каждом узле системы свои специфические профили обработки документов. Таким образом, механизм триггеров, предоставляемый системой-конструктором, является одним из наиболее значительных функциональных средств,

---

5 Заметим, что результатом срабатывания триггера может являться изменение значений атрибутов документа, что, в свою очередь, может повлечь за собой вызов другого триггера и т.д. С другой стороны, изменение атрибутов документа происходит главным образом в результате работы триггеров.

позволяющих создавать частные информационные системы, отвечающие специфике конкретной области их применения.

3. Компоненты и их взаимодействие.

Для того, чтобы обеспечить разрабатываемой системе-конструктору необходимую гибкость, настраиваемость и переносимость, разумно проектировать систему как совокупность самостоятельных компонент, реализующих отдельные блоки функциональных возможностей. При этом необходимо предусмотреть возможность функционирования и взаимодействия компонент при различных конфигурациях составленной из них системы. Это означает, что компоненты системы должны поддерживать как локальные, так и сетевые механизмы взаимодействия друг с другом.

Выше были рассмотрены основные функциональные возможности системы. Исходя из этого, было проведено разбиение системы на компоненты, реализующие отдельные функциональные блоки (Рис. 2). Ниже будут рассмотрены указанные на схеме компоненты и их функции.

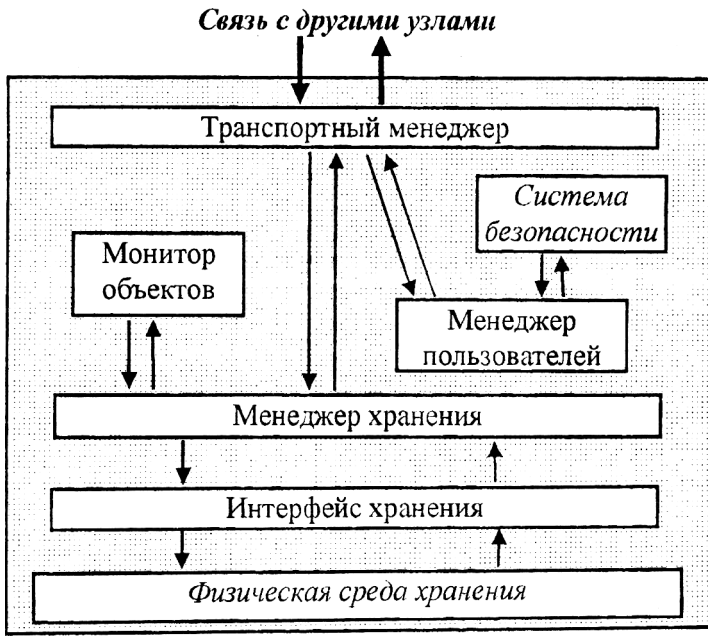


Рис. 2 Структурная организация узла системы.

При проектировании нашей системы нами были взяты в расчет два практических соображения: во-первых, предполагается, что узлы

системы могут функционировать под управлением разных операционных систем, предоставляющих различные возможности и идеологию постоянного хранения данных, и во-вторых, как следствие, не существует универсального способа постоянного хранения данных, эффективного для любых операционной системы, характера и объемов данных. В связи с этим было принято решение об инкапсуляции способа хранения данных. В системе выделяется специальная компонента, называемая далее *интерфейс хранения*, монопольным образом отвечающая за сохранение локальных данных. Использование интерфейса хранения позволит:

- Абстрагироваться от конкретной системы постоянного хранения данных, специфичной для каждого узла.
- Унифицировать способ размещения данных с точки зрения остальных компонент системы, базируясь на терминах документной модели хранения.
- Увеличить гибкость системы, т.к. при переходе от одного способа хранения данных к другому (или при замене структуры хранения на более эффективную) потребуется заменить лишь интерфейс хранения.

Интерфейс хранения может иметь несколько реализаций для разных способов физического локального хранения данных; выбор конкретного способа определяется при настройке готовой системы средствами администрирования.

С интерфейсом хранения взаимодействует только одна компонента системы, которая представляет абстракцию механизма хранения для остальных компонент, а также обеспечивает одновременный доступ к интерфейсу хранения со с стороны нескольких компонент более высокого уровня - *менеджер хранения*. Эта компонента решает следующие задачи:

- Поддержка одновременного подсоединения нескольких приложений и прием их запросов.
- Корректное разрешение коллизий при попытке одновременного доступа к одним и тем же документам.

Отдельная компонента - *монитор объектов* реализует функционирование механизма триггеров. В ее задачи входит периодическая проверка текущего состояния атрибутов объектов и инициация триггеров. Для монитора объектов важным параметром являются условия его активизации. Они могут быть двух типов: связанные с событиями, происходящими в системе, и не связанные с ними. Предполагается, что при отсутствии указаний администратора (задаваемых при настройке и конфигурировании конкретного узла) монитор активизируется всякий раз, когда в системе появляется новый объект либо в результате действий пользователя изменяются значения атрибутов существующих объектов (т.е. происходит обращение к

атрибутам объекта "по записи"). Однако, если документы в системе обрабатываются интенсивно, а ресурсов недостаточно, и при этом специфика задачи позволяет производить действия, предусмотренные триггерами, в "отсроченном" режиме, то при администрировании можно принудительно задать временной интервал, через который монитор будет активизироваться. (Такая необходимость может возникнуть, например, при работе на удаленном компьютере с медленной связью, если узел системы находится на некотором сервере, а на хосте присутствует только пользовательский интерфейс и система мониторинга)

Три предыдущие компоненты в совокупности реализуют базовые возможности узла системы с точки зрения хранения и обработки документов. Отдельная компонента - *менеджер пользователей* реализует разграничение и авторизацию доступа. Она отвечает за администрирование пользователей, проверку прав доступа, а также (при необходимости) позволяет подключить отдельную систему шифрования для обеспечения безопасной передачи конфиденциальной информации по сети. Таким образом, менеджер пользователей связывается через *интерфейс безопасности* с некоторой самостоятельной системой безопасности, выполняющей следующие функции:

- По имени пользователя, имени узла и запросу определить, разрешено ли выполнение запроса (контроль прав доступа).
- По имени пользователя, имени узла и запросу изменить данные запроса (шифрование/дешифрование)

Инкапсуляция системы безопасности предоставляет практически те же преимущества, что и инкапсуляция системы хранения - увеличение гибкости и абстрагирование от конкретных способов шифрования и контроля прав доступа. В случае отсутствия внешней системы безопасности любому пользователю считается доступна любая операция с любыми документами, а сами данные не шифруются.

Система разрабатывается как некоторый логический сервер, предоставляющий услуги по распределенному хранению документов. Соответственно, должен быть разработан механизм, обеспечивающий взаимодействие системы с приложениями, запрашивающими эти услуги. Специальная компонента, реализующая этот механизм - *интерфейс использования* должна обеспечивать:

- Передачу запросов и ответов на запросы между приложениями и системой.
- Нотификацию о срабатывании триггеров у приложений, запросивших такие услуги.

Взаимодействие узлов, как расположенных на одной машине (локальных), так и удаленных, друг с другом, а также с интерфейсами

использования, которые желают работать с этими узлами, обеспечивается *транспортным менеджером*. Он занимается собственно передачей между компонентами запросов и ответов на них и инкапсулирует всю реализацию межпроцессного и сетевого взаимодействия, обеспечивая взаимодействие компонент системы друг с другом.

Заключение.

В данной работе предлагается перспективная концепция построения распределенных ИС с использованием адаптируемого инструментального средства, реализующего базовое ядро функциональности и предоставляющего возможности параметризации типов и жизненного цикла хранимых данных, логики управления информационным потоком, системы авторизации и разграничения доступа, использования различных физических сред хранения и передачи данных, аппаратных и программных платформ. Предложенное инструментальное средство может быть использовано при построении широкого класса распределенных ИС, таких как системы электронного документооборота, распределенные системы контроля версий, системы сбора и анализа информации.

Литература.

1. Lotus Notes: An Overview — Lotus Development Corporation, 1996.
2. Brian Berliner «CVS II: Parallelizing Software Development» — Prisma, Inc.
3. Весли П. Меллинг. Корпоративные информационные архитектуры: и все-таки они меняются. — СУБД, N2 1995
4. Каменнова М.С. Корпоративные информационные системы: технологии и решения. — СУБД, N3 1995
5. The Visual SourceSafe — Microsoft, апрель 1997г.

Смирнов Д.В.

Проект трехуровневой инструментальной системы обработки динамических процессов

Введение

В данной публикации рассматривается структура системы, предназначенной для анализа динамических процессов. Система представляет собой набор модулей, необходимых для связывания штатных интерпретаторов языков Python [2] и Prolog [1,3], низкоуровневых библиотек на С и модулей передачи по каналам связи, а также модулей взаимодействия с СУБД и управления файлами. Система должна позволить в автоматическом режиме производить обработку сигналов на распределенных вычислительных системах.

Пользовательский интерфейс большинства существующих систем обработки экспериментальных данных разработан для начинающих пользователей. Такого рода система базируется на небольшом наборе инструментов, применяя которые можно совершать преобразования данных. Но простота интерфейса приводит к тому, что в тех случаях, когда число преобразований велико, исправить ошибку или пересчитать данные эксперимента в случае изменения исходных данных довольно сложно, т.к. для этого обычно необходимо все преобразования повторить. Широко распространенные в программистской среде Makefile-ы позволяют одной командой пересчитывать огромное количество промежуточных данных, но для работы с Makefile-ом необходимо оформить каждое преобразование в виде отдельной программы, что в большинстве пакетов обработки экспериментальных данных сделать невозможно.

Другими важными особенностями предлагаемой инструментальной системы являются ее открытость и гибкость, достигаемые применением распространенных языков для написания модулей, стандартизацией межмодульных связей и структуризацией модулей в целом.

Стоит отметить также возможность удаленной работы с системой по Internet и на распределенных вычислителях, достигаемую за счет применения модулей управления каналами.

Система предназначена для широкого круга пользователей от начинающих, знакомых с несколькими Prolog-предикатами в своей узкой предметной области до опытных специалистов, разрабатывающих дополнительные модули на всех трех уровнях системы.

1.Арихитектура системы

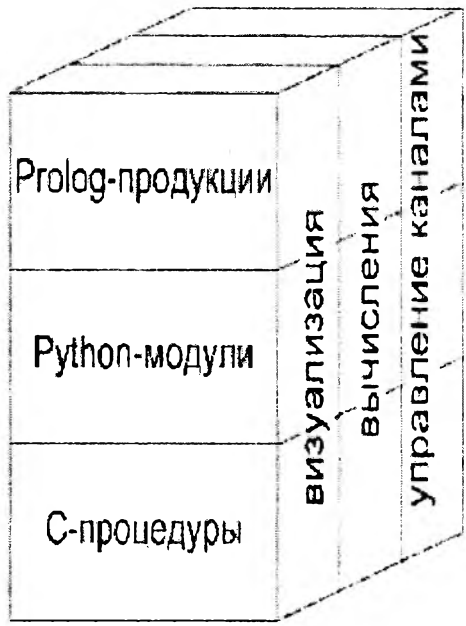


Рис. 5. Архитектура системы

На рис 1 изображена архитектура системы, в которой можно выделить 3 уровня иерархии: уровень низкоуровневых С-процедур, уровень языка Python, и уровень продукций языка Prolog. Низкоуровневые С-процедуры может хорошо оптимизировать по быстродействию, при их написании и использовании необходимо постоянно отводить, освобождать и по определенным правилам заполнять блоки памяти, подбирать наиболее подходящие функции и формировать большое количество параметров. Язык Python включает достаточно много конструкций, упрощающих формирование

параметров, написание стандартных циклов и контроль ссылок на блоки памяти. Продукционно-ориентированная база знаний на основе языка Prolog предназначена для автоматического планирования вычислений.

В другом измерении систему можно разбить на 3 части: управления визуализацией, непосредственно вычислениями и каналами межпроцессного обмена.

2.Низкоуровневые модули на языке С

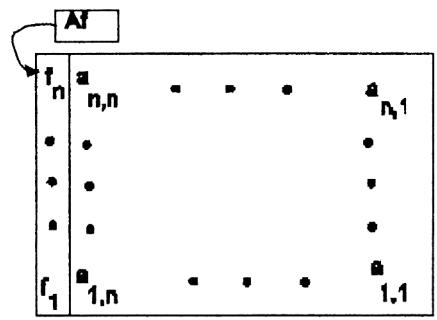


Рис. 6. Размещение расширенной матрицы системы в блоке памяти

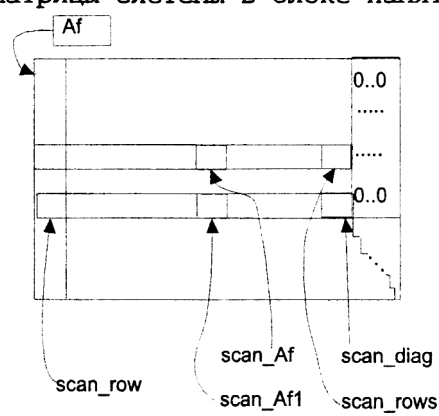


Рис. 7. Указатели, используемые во время прямого хода метода Гаусса.

На самом низком языковом уровне система позволяет подключать модули, написанные на языке С и вызывать С-функции с

более высоких языковых уровней. Рассмотрим пример функции на С, решающей систему линейных уравнений методом Гаусса.

```
void gauss(double* Af,double* x,unsigned int n)
{
double* scan_row;
double* scan_diag;

scan_diag=(scan_row=Af+(n*(n+1)))+(n+1);
while((scan_row-=n+1)>=Af)
   {
   double* scan_rows;
   scan_row-=n+1;
   scan_rows=Af+(scan_diag-scan_row);
   while((scan_rows+=n+1)<=scan_row)
      {
      double* scan_Af;
      double* scan_Af1;
      double k;
      k=scan_rows/*(scan_diag);
      scan_Af=scan_rows;
      scan_Af1=scan_diag;
      while (scan_Af1-->=scan_row)
         {
         *(--scan_Af)-=k**scan_Af1;
         };
      }
   }
.....
}
```

Для краткости обратный ход метода Гаусса опущен. Используемые переменные приведены на рис. 3 , где

Af - указатель на расширенную матрицу системы, записанную в обратном порядке (на рис. 2 числа в блоке памяти размещаются сверху вниз и слева направо),

x - указатель на вектор, в который помещается решение

n - количество уравнений и неизвестных

scan_row - указатель на строку в матрице системы, содержащую ведущий элемент.

scan_rows - указатель на обрабатываемую строку

scan_diag - указатель на ведущий элемент

scan_Af - указатель на обрабатываемое значение

scan_Af1 - указатель на значение в строке, содержащей ведущий элемент.

Основное внимание при оптимизации уделяется самому внутреннему циклу: из него вынесена константа k, два указателя легко пересчитываются от итерации к итерации, а условие выхода - простое сравнение двух указателей. Кроме того, цикл векторизован, т.е. приведен к виду

```
F(*scan_Af,*scan_Af1);scan_Af--;scan_Af1--;
```

F - процедура обработки

Но написание и понимание процедуры затруднено из-за необходимости распределения матриц и векторов по областям памяти, а также из-за применения сложных процедур оптимального вычисления адресов. Следует заметить, что оптимальность указанного кода во многом определяет производительность системы в.целом.

Приведённый пример показывает какие возможности оптимизации программ заложены в язык С, используемый на самом низком уровне языковой иерархии.

3.Python-модули

Как показывают Д. Кнута [6], около половины времени тратится на выполнение менее чем 4% программного кода. Основное влияние на производительность оказывают рассмотренные выше циклы самого высокого уровня вложенности на языке С. На более высоком уровне важное значение приобретает гибкость параметризации модулей, экономия памяти и легкость изменения. Для достижения вышеуказанных целей часто используются классы объектно-ориентированных языков, таких как C++, Object Pascal и даже Visual Basic. К недостаткам такого подхода следует отнести необходимость описания большого количества переменных, сложность работы с функциями и низкоуровневые циклы. Все вышеуказанное серьезно затрудняет модификацию кода и согласование изменений. Другим подходом к решению этой проблемы являются языки спецификации задач. К их недостаткам следует отнести узкоспециализированность и уникальный синтаксис.

Язык Python следует отнести к универсальным языкам, близким по синтаксису к языкам спецификаций. Например, в библиотеке оконных компонент

```
map(lambda fname,boxlist:(GtkFrame(fname),lambda
cont,comp:cont.add(comp),
```

```
    map(lambda box:(GtkVbox(GtkVbox())))
[("Frame1",

      [comp1_1, comp1_2, comp1_3],
      [comp2_1, comp2_2, comp2_3]
  ],
  [
      [comp3_1,comp3_2,comp3_3]
  ]
),
("Frame2")
]
```

Результатом вычисления выражения будет список упаковки объектов в окно. По заданному списку вызовом одной функции могут быть созданы все объекты- экземпляры и заполнены контейнеры, в результате чего объект-окно и его подобъекты будут полностью инициализированы и после вызова функции gtkmainloop появятся на экране.

В C и C++ для инициализации подобных объектов необходимо сделать много объявлений, что серьёзно замедляет написание программы. Средства, подобные C++ Builder лишь частично решают эту проблему, т.к. написание процедур обработки событий производится на языке C.

4.Prolog-предикаты

Ныне наиболее распространенным способом управления вычислениями является ручное управление, при котором пользователь, выбирая пункты меню или инструменты из панели инструментов, выполняя команды и т.п. производит те или иные преобразования данных. Описанный подход хорош тогда, когда преобразования выполняются один раз. Если же обработка многократно повторяется с небольшими изменениями параметров, то после каждого изменения пользователю приходится повторять вышеперечисленные действия заново.

prolog-продукции предназначены для того, чтобы по одному запросу пользователя пересчитать промежуточные и выходные данные при изменении параметров. при этом работа пользователя идет в двух направлениях: изменение системы продукций и подбор параметров. Любая продукция - это пара

ситуация -> реакция

Простейшим примером продукции являются операторы makefile-а. Ситуация в продукции Makefile-а - это факт того, что результат устарел, т.е. исходные данные модифицировались после получения результата. Реакция в Makefile-е - это последовательность команд вычисления результата.

Рассмотрим процесс решения задачи методом построения продукций на примере расчетов оценок в норме максимума модуля. Для вычисления оценки необходимы аппроксимирующие функции, и аппроксимируемая функция, т.е. функция к которой строится приближение. В результате получается набор коэффициентов разложения оценивающей функции по аппроксимирующим и отклонение Не вдаваясь в суть указанных функции, норм и оценок, рассмотрим предикат

max_estminate(F , f ,M , A)
,где
F={f1...fn} - множество аппроксимирующих функций
f - аппроксимируемая функция
M - оценка в норме максимума модуля
A={a1...am} - множество коэффициентов

Предикат истинен, если M и a1..am являются решениями рассмотренной задачи оценки с исходными данными f1..fm и f.

Первое решение, которое принимает экспериментатор - это определение системы функций. Пусть система функций тригонометрическая, т.е.

1 sin(t) cos(t) sin(2t) cos(2t) .....

Указанную систему функций можно задать продукциями

trig_func(1).
trig_func(F U sin(nt) U cos(nt),n):trig_func(F,n-1).

или python-выражением:

```
|lambda t:1,]+map(lambda n:lambda t:sin(n*t),range(0,n))
+map(lambda n:lambda t:sin(n*t),range(1,n))
```

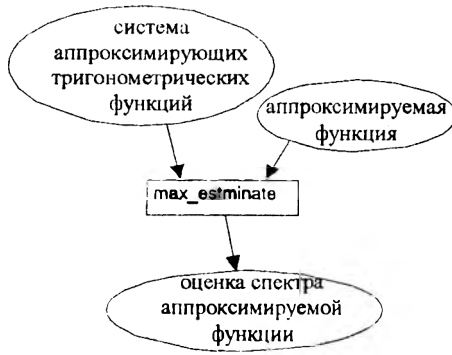


Рис. 8. Диаграмма потока данных вычислительного эксперимента оценивания спектра

Пусть аппроксимируемая заданная аналитически функция, оценка спектра которой будет получена в результате, задана python-выражением. Диаграмма потока данных эксперимента приведена на рис.4. Следует заметить, что не всякое множество продукций описывается диаграммой потока данных. Например, рассмотренные выше продукции задания тригонометрической системы функций рекурсивны и потоки данных динамичны и связаны с потоком управления.

Следует заметить, что продукции - это декларативное задание эксперимента, по которому автоматически строится алгоритм решения задачи или пересчета. По системе продукций можно построить множество алгоритмов, перенося переменные из результатов в исходные данные и наоборот. Такая возможность открывает огромные возможности для подбора параметров эксперимента и сопоставления результатов эксперимента и модели.

5.Реализованные части

К моменту публикации реализована матричная библиотека, построенная на основе алгоритмов из [5], предоставляющая пользователям и программистам определять и преобразовывать матрицы на высоком уровне. На рис. 5 приведён результат расчета фермы моста. Стержни фермы свободно закреплены в узлах без трения. По заданным силам в вершинах фермы рассчитываются усилия, сжимающие или растягивающие стержни фермы. Ферма задается в текстовом файле, который легко может подготовить даже начинающий пользователь. Эта задача сводится к решению системы линейных уравнений. Расчет вызывается командой

pl -f ferm.pl,

Затем следующим Prolog-запросами формируются необходимые данные.

makeferm_preview. - произвести расчет фермы и показать результат расчета в окне

makeferm. - произвести расчет фермы и результат записать в Postscript-файл.

makefermforces. - произвести только расчет усилий и записать значения усилий в текстовый файл.

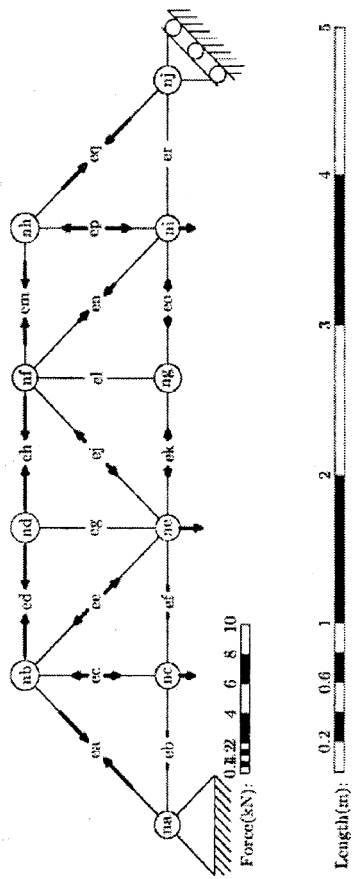


Рис. 9 Расчет фермы

При работе в системах типа [4,7,8,9], где операции визуализации, обращения матриц, формирования матриц из графа фермы, чтения графа фермы из текстового файла оформлены в виде отдельных процедур, пересчет усилий затруднителен. Это происходит из-за того, что вместо одного из вышеперечисленных Prolog- запросов нужно вызвать все процедуры от чтения графа до обращения матрицы и визуализации. Некоторые системы позволяют одной командой вызывать специальные файлы-скрипты, синтаксис которых подобен языку Python. Такой подход часто приводит к избыточным вычислениям. Например, при изменении параметров визуализации нет необходимости пересчитывать значения усилий, а скрипт производит все действия от начала до конца.

Заключение

Из рассмотрения примеров можно сделать вывод о возможности построения трехзвенной инструментальной системы для решения широкого круга задач.

Оптимизированные по быстродействию низкоуровневые процедуры на языке С несут основную вычислительную нагрузку.

Более высокоуроневые модули на языке Python - это наиболее компактное, модифицируемое и экономное средство связывания низкоуровневых модулей.

Заполняемая пользователем в процессе построения модели эксперимента продукционная база знаний позволяет путем дедуктивного синтеза строить алгоритмы решения задач и пересчитывать промежуточные результаты при изменении параметров эксперимента.

Результирующий программный код разбивается на 3 раздела:

управления каналами межпроцессного обмена;

непосредственно вычисления;

модули визуализации.

Автор выражает благодарность чл.-корр. РАН Л.Н. Королёву за ценные замечания и идеи без которых публикация не увидела бы свет.

Литература

1. Братко Программирование на языке Prolog для искусственного интеллекта М. Мир 1990.

2. www.python .org

3. www.psy.uva.nl/projects/~SWIprolog/

4. www.mathworks.com

5. Дж.Голуб Ч. Ван Лоун. Матричные вычисления. М. Мир 1999.

6.Knuth D.E. An Empirical study of FORTRAN programs.Softwate-Pract/ and Exper.1971.V.1. N 2. P 105-133.

7.Ю.Н. Тюрин. А.А. Макаров. Анализ данных на компьютере. М. Инфра-М Финансы и статистика 1995.

8.А.П. Кулаичев. Методы и средства анализа данных в среде Windows. STADIA 6.0.НПО "Информатика и компьютеры". Москва 1998 г.

9.www.statsoft.ru

Маслов С.П., Рамиль Альварес Х., Сидоров С.А.

Многотерминальная МСО "Наставник" на IBM PC

Описывается перенос многотерминального варианта микрокомпьютерной системы обучения (МСО) "Наставник" на IBM PC. Рассматриваются нестандартная аппаратура, поддерживающие ее программы (драйверы) и программа, управляющая деятельностью обучаемых, для двух вариантов системы.

Введение

Прототип Микрокомпьютерной Системы Обучения (МСО) "Наставник" [1] был создан в начале 70-х годов. Впоследствии система реализовывалась в многочисленных вариантах [2] и остается востребованной до сих пор. Это говорит о правильности использованной концепции, но порождает трудности при переносе системы на современную техническую основу с целью "продлить ее жизнь".

Свойство переносимости постоянно имелось в виду разработчиками. В отношении программного обеспечения она достигалось использованием ДССП [4], в отношении аппаратуры - стандартных машинно-независимых портов ввода-вывода [5]. Однако материализовать какой-либо из имеющихся переносимых вариантов трудно - нужное оборудование не выпускается. Более реально модернизовать существующие системы максимально сохранив при этом их аппаратуру.

Уникальные минитерминалы являются наиболее дорогой частью системы и сохранить их особенно важно. Кроме минитерминалов в системе имеется еще одна нестандартная компонента – контроллер, сопрягающей минитерминалы с компьютером. При замене компьютера должен быть спроектирован и изготовлен новый контроллер. Переход на современные компьютеры типа IBM PC позволяет радикально упростить контроллер за счет выполнения большинства его функций программным путем. Такой контроллер уместнее называть адаптером.

Наиболее популярный вариант МСО "Наставника" [6] базируется на микрокомпьтерах типа Электроника-60 или ДВК, выпуск которых давно прекращен. Такая система уже 14 лет функционирует на факультете ВМК МГУ. Существовала необходимость перевести систему на PC и сделать это оперативно, чтобы не пострадал учебный процесс.

Перенос происходил в два этапа. На первом этапе в качестве временного (пока не будет сделан адаптер) варианта была реализована система, содержащая связанные ДВК и РС. При этом ДВК работает с существующим контроллером и не имеет другой периферии Его функцией является обслуживание терминалов и связь с РС. Программа, управляющей деятельностью учащихся, при этом выполняется в РС. На втором этапе ДВК был заменен адаптером, через который к РС подключаются минитерминалы.

Промежуточный вариант МСО

ДВК и РС связаны через COM-порты. У ДВК использован вход для подключения консоли. Программа в РС, имитируя работу за консолью в пультовом режиме, загружает в ДВК и запускает драйвер.

Драйвер промежуточного варианта МСО

Драйвер в ДВК выполняет функции по управлению контроллером [6] и поддерживает связь с выполняемой в РС программой, управляющей деятельностью учащихся. Драйвер реализован в виде двух процедур: обработки прерывания после завершения полного цикла обслуживания контроллером всех терминалов и обработки прерываний от их клавиатур. Драйвер написан на языке ассемблера PDP-11, для чего был создан кросс-компилятор для РС, поскольку использование штатного компилятора затруднено из-за отсутствия у ДВК жесткого диска.

Программа поддержки связи с РС работает в фоновом режиме. Она принимает команды от РС и вызывает реализующие их процедуры:

`[ ] TZ [ ]`	загружает в ДВК программу драйвера минитерминалов
`[ ] INIT [ ]`	инициализирует драйвер класса в ДВК
`[ ] CLON [ ]`	включает минитерминалы
`[ ] CLOFF [ ]`	выключает минитерминалы
`[cc,tp] SEND [ ]`	посылает код <cc> для индикации на терминале <t> в позиции <p> cc(7:0) - код; tp(7:3) - номер терминала; tp(2:0) - номер позиции
`[t] MESEND [ ]`	конец сообщения терминалу t (t: 0..31)
`[ ] GET [ttkk]`	чтение информации о нажатых клавишах: ttkk (15:8)=tt, номер терминала + 1; ttkk(7:0)=kk, код клавиши

Основной вариант МСО
Адаптер

Адаптер подключен к PC через стандартный параллельный порт, обычно используемый для подключения принтера. Со стороны PC адаптер получает и выдает типовой набор сигналов управления принтером и сигналов о его состоянии, однако интерпретирует их по своему. Терминалам адаптер выдает тот же набор сигналов, который раньше они получали от контроллера.

В составе адаптера имеются два байтовых регистра с параллельным приемом и последовательной выдачей: регистр номера позиции **РНП** и регистр кода символа **РКС**; генератор синхроимпульсов; 4-разрядный счетчик; триггер фиксации свертки и схема управления.

Описание устройства и работы минитерминалов и контроллера содержится в [6]. Здесь опишем функции, выполняемые адаптером.

1. Циклически (каждые 100 мкс) выдает текущему активизированному терминалу 16-битный код, несущий информацию о том, какой символ и в какой позиции должен быть на нем высвечен. Этот код предварительно загружается в **РНП** и **РКС**. Его выдача сопровождается выдачей по отдельным линиям синхросерий. Каждый раз после активизации 32 терминалов дополнительно к перечисленным сигналам выдается импульс, активизирующий первый терминал и инициирующий по цепочке активизацию остальных.

2. Принимает от активизированного терминала синхронно с выдачей ему кода символа 16-битный унитарный код, содержащий информацию о нажатой на терминале клавише и преобразует его в 4-битный программно доступный код свертки.

Драйвер основного варианта МСО

Драйвер поддерживает индикацию на минитерминалах, распознает и анализирует нажатия клавиш, обрабатывает запросы управляющей программе и упорядочивает их поступление к ней.

Адаптер отличается очень простым устройством, что было достигнуто за счет программной реализации механизма поддержки индикации. 8-позиционный индикатор терминала может единовременно высвечивать одну позицию. Индикация происходит в мультиплексном режиме, т.е. каждая позиция отображается 1/8 часть времени. Чтобы мигания не были заметны требуется обновлять позиции индикатора с частотой не менее 40 Гц. Для этого драйвер должен загружать регистры адаптера с интервалом не более 100 мкс.

Чтобы снизить требования к быстродействию используемого компьютеру программа драйвера организована не в форме циклов, а в виде линейной последовательности команд. Ускорение достигается не только за счет отсутствия накладных расходов на организацию циклов, но и благодаря сокращению числа обращений к памяти, поскольку

практически все переменные задаются напрямую, операндами в командах.

Функционирование драйвера основывается на отслеживании и анализе текущего состояния каждого минитерминала. Последовательность этих состояний такова:

0	Индикация включена, нажатия клавиши нет
1	Индикация включена, первый раз зафиксировано нажатие клавиши
2	Индикация выключена ("темный" индикатор), факт нажатия клавиши подтвердился
3	Индикация выключена, запрос (код клавиши) передан управляющей программе
4	Индикация включена, получен ответ управляющей программы, клавиша еще нажата

Управляющая программа

Управляющая программа реализована в ДССП [4]. Ее перенос на PC не составил труда. К системе были добавлены программы статистической обработки архивов занятий, а подсистемы "Подготовка" и "Обработка" включены в число основных.

Литература

1. Брусенцов Н.П., Маслов С.П., Рамиль Альварес Х. Автоматизированная система обучения "Наставник" - М.: В сб. Вычислительная техника и вопросы кибернетики. Вып.13 - М.: Изд-во Моск. ун-та, 1977. - С.3-13.

2. Брусенцов Н.П., Маслов С.П., Рамиль Альварес Х. Микрокомпьютерная система обучения "Наставник" - М.: Наука, 1990. - 223с.

3. Брусенцов Н.П., Маслов С.П., Рамиль Альварес Х. Методическое пособие по разработке учебных материалов в микрокомпьютерной системе обучения "Наставник" - М.: Изд-во Моск. ун-та, 1992. - 95с.

4. Н.П.Брусенцов, В.Б.Захаров, И.А.Руднев, С.А.Сидоров. Диалоговая система структурированного программирования ДССП-80. В сб. Диалоговые микрокомпьютерные системы. - М.: Изд-во Моск. ун-та, 1986. - С.3-21.

5. Маслов С.П., Сидоров С.А. Локальная сеть минитерминалов, управляемая персональным компьютером В сб. Программное оснащение персональных компьютеров. - М.: Изд-во Моск. ун-та, 1990. - С. 99-114.

6. Маслов С.П. Аппаратура многотерминальной микрокомпьютерной системы. В сб. Диалоговые микрокомпьютерные системы. - М.: Изд-во Моск. ун-та, 1986. - С. 121-132.

Демаков А.В., Зеленова С.А., Зеленов С.В.

Тестирование парсеров текстов на формальных языках[6]

Введение

В данной работе рассматриваются вопросы, связанные с систематическим тестированием на основе формальных спецификаций.

Спецификация – это абстрактное, независимое от реализации, описание поведения программы. Тестирование на основе спецификаций есть проверка соответствия поведения реальной системы ее специфицированному поведению.

Тестирование на основе спецификаций обладает рядом достоинств:

- спецификации позволяют отвлечься от деталей реализации, а значит предоставляют возможность использовать один и тот же набор тестов, построенный по этим спецификациям, для проверки нескольких различных реализаций тестируемой системы;
- тестирование на основе спецификаций, т.е. эталона поведения, дает возможность выбрать критерий полноты тестового набора, позволяющий выявить неполную функциональность программного продукта;
- спецификация обычно менее объемна и легче для восприятия, чем реализация, поскольку представляет собой некоторую факторизацию реальной системы.

В настоящее время в качестве спецификации компилятора рассматривают неформальное или частично-формальное описание входного языка этого компилятора. Лишь парсер обычно полностью формализован.

Для тестирования компиляторов в основном применяются комплекты тестовых программ, написанных вручную на основе такого неформального описания целевого языка.

Такой подход, безусловно, имеет право на существование. При этом для обеспечения систематичности тестирования главное – это задать критерий покрытия и создать множество тестов, которое ему

[6] Данная работа частично поддержана грантом РФФИ 99-01-00207.

соответствует. Однако, использование неформальных спецификаций обладает существенными недостатками:

- при построении тестов для фиксированного критерия покрытия большую роль играет субъективный фактор, а следовательно, нельзя гарантировать качества тестового покрытия;
- ручная разработка требует слишком много ресурсов.

В то же время использование формальных спецификаций при тестировании имеет много положительных сторон:

- легко формулировать критерии полноты покрытия;
- возможна автоматизация построения тестов;
- для автоматически построенного тестового набора легко проверить, достигнуто ли требуемое покрытие.

1. Постановка задачи

В настоящей работе мы ограничимся рассмотрением парсеров, то есть синтаксических анализаторов. Более того, мы предполагаем, что целевой язык парсера является контекстно-свободным.

Пусть задан парсер, отвечающий некоторому формальному контекстно-свободному языку. Требуется проверить, что он работает правильно, т.е. корректно выносит вердикт о принадлежности входного предложения языку.

При этом задача разбивается на две:

- построение набора «правильных тестов», то есть предложений, принадлежащих языку;
- построение набора «неправильных тестов», то есть предложений, не принадлежащих языку.

Решение каждой из этих подзадач в свою очередь состоит из двух этапов: описание метрики тестового покрытия и разработка генератора входных тестовых предложений.

Здесь мы не будем останавливаться на алгоритме генерации, а подробно рассмотрим метрику тестового покрытия, которая позволяет построить семейство критериев, удобных для определения полноты тестирования парсеров на основе «правильных тестов».

2. Метрика покрытия для правильных тестов

Основные определения.

Назовем *нетерминальным подвыражением* такую часть правила BNF, которая является либо ссылкой на некоторое правило, либо выражением в скобках (см. Приложение А).

С каждым нетерминальным подвыражением свяжем *фронтальное множество*, состоящее из всех токенов (т.е. терминальных символов), каждый из которых служит началом в какому-нибудь раскрытии исходного подвыражения.

Будем говорить, что нетерминальное подвыражение является *точкой однозначного локального продолжения*, если его фронтальное множество состоит из одного элемента.

Нетерминальное подвыражение, не являющееся точкой однозначного локального продолжения, назовем *точкой ветвления*.

Назовем *множеством локальных ситуаций* множество всевозможных пар (*f*, **t**), где *f* – точка ветвления, а **t** – токен из фронтального множества точки *f*.

Критерий покрытия.

Предложенный выше набор понятий позволяет формулировать разнообразные критерии покрытия различной мощности. Остановимся подробно на критерии, который мы использовали в своей работе.

Рассмотрим какое-нибудь предложение языка и все выводы этого предложения в данной грамматике. Будем говорить, что предложение *покрывает* локальную ситуацию (*f*, **t**) если в некотором выводе встречается эта локальная ситуация.

Наш критерий требует построить такой набор предложений языка, которые в совокупности покрывают все элементы множества локальных ситуаций. Будем называть этот критерий «все локальные ситуации».

Заметим, что в предложенных терминах можно определить много других критериев.

Пример критерия меньшей мощности: для каждой точки ветвления *f* покрыть какую-нибудь локальную ситуацию (*f*, **t**).

Пример критерия большей мощности: для любой пары локальных ситуаций построить предложение языка, покрывающее их обе, если это возможно, или каждую по отдельности в противном случае.

Построенный по критерию «все локальные ситуации» набор тестов для некоторой BNF контекстно-свободного языка проверяет поведение парсера для этого языка в точках ветвления.

Существует широкий класс парсеров, именно – парсеров, осуществляющих LL-разбор (т.е. двигающихся сверху вниз), для

которых повсдение в этих точках особенно важно. Это утверждение основывается на том факте, что любой подобный парсер, дойдя в разбираемом предложении до места, соответствуюшего некоторой точке ветвления, должен будет сделать выбор на основании нескольких следующих символов.

Эту особенность LL-разбора не учитывают методики генерации тестов, которые пытаются покрыть лишь все правила или все альтернативы.

Пример.

Проиллюстрируем обсуждавшиеся выше понятия на примере.

Пусть *A*, *B*, *C* – правила грамматики; **a**, **b**, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{c}_1$, $\mathbf{c}_2$ – терминальные символы. Рассмотрим следующую BNF:

$$A ::= \mathbf{a}\, B\, C;$$
$$B ::= \mathbf{b}\,(\mathbf{b}_1 \mid \mathbf{b}_2);$$
$$C ::= (\mathbf{c}_1 \mid \mathbf{c}_2);$$

В ней можно указать три точки ветвления :

$$A ::= \mathbf{a}\, B \underset{\uparrow f_1}{} C;$$
$$B ::= \mathbf{b} \underset{\uparrow f_2}{} (\mathbf{b}_1 \mid \mathbf{b}_2);$$
$$C ::= \underset{\uparrow f_3}{} (\mathbf{c}_1 \mid \mathbf{c}_2);$$

Множество локальных ситуаций состоит из шести элементов:

$$\{ (f_1, \mathbf{c}_1), (f_1, \mathbf{c}_2), (f_2, \mathbf{b}_1), (f_2, \mathbf{b}_2), (f_3, \mathbf{c}_1), (f_3, \mathbf{c}_2) \}$$

Распишем вывод для предложения $\mathbf{a}\ \mathbf{b}\ \mathbf{b}_1\ \mathbf{c}_1$:

$$A \to \mathbf{a}\, B\, C \to \mathbf{a}\, \mathbf{b}\, (\mathbf{b}_1 \mid \mathbf{b}_2)(\mathbf{c}_1 \mid \mathbf{c}_2) \to \mathbf{a}\, \mathbf{b}\, \mathbf{b}_1\, \mathbf{c}_1.$$

Легко заметить, что в нем встречаются три точки ветвления нашей BNF:

$$A \to \mathbf{a}\,B \quad C \to \mathbf{a}\,\mathbf{b} \quad (\mathbf{b}_1 \mid \mathbf{b}_2) \quad (\mathbf{c}_1 \mid \mathbf{c}_2) \to \mathbf{a}\,\mathbf{b}\,\mathbf{b}_1\,\mathbf{c}_1.$$
$$\uparrow f_1 \qquad \uparrow f_2 \qquad \uparrow f_3$$

Поэтому можно указать три локальных ситуации, покрытых предложением :

$$A \to \mathbf{a}\,B \qquad C \to \mathbf{a}\,\mathbf{b} \qquad (\mathbf{b}_1 \mid \mathbf{b}_2) \qquad (\mathbf{c}_1 \mid \mathbf{c}_2) \to \mathbf{a}\,\mathbf{b}\,\mathbf{b}$$
$$\uparrow (f_1, \mathbf{c}_1) \qquad \uparrow (f_2, \mathbf{b}_1) \qquad \uparrow (f_3, \mathbf{c}_1)$$

Тестовый набор, удовлетворяющий нашему критерию, выглядит так:

$\mathbf{a}\,\mathbf{b}\,\mathbf{b}_1\,\mathbf{c}_1$;
$\mathbf{a}\,\mathbf{b}\,\mathbf{b}_2\,\mathbf{c}_1$;
$\mathbf{a}\,\mathbf{b}\,\mathbf{b}_1\,\mathbf{c}_2$.

3. Практические результаты

Нами разработан и реализован генератор тестового набора, отвечающего описанной выше метрике.

Характеристики генератора: память не является критическим фактором; скорость генерации на P-III-450MHz составила 60 тестов в секунду. Для удобства использования генератора введены некоторые параметры, позволяющие строить тестовые последовательности, отвечающие более слабым критериям.

Генератор используется для построения тестов в промышленной разработке компиляторов.

Следует отметить, что в современной практике автоматизированной разработки парсеров, в основоном, стараются свести грамматику к виду LL(1). В тех местах, где грамматика не укладывается в такой вид, дописывают ручные компоненты анализатора, например, процедуры "LookAHead" в инструменте JavaCC [1], обеспечивающие «заглядывание вперед».

Для грамматик языков Java и его спецификационного расширения J@va [2, 3] был сгенерирован набор тестов (около 170 тысяч). При прогоне построенного набора было выявлено восемь ошибок. Все они были в ручных компонентах.

Для грамматики языка mpC [4, 5] при различных значениях параметров генератора было получено два набора тестов – «маленький» (около 30 тысяч предложений) и «большой» (почти 1,4 миллиона предложений). В результате прогона полученных тестов было

обнаружено двенадцать ошибок в парсере и семантическом анализаторе компилятора с языка mpC, которые приводили к аварийному завершению или зацикливанию. Интересно отметить, что все ошибки, которые были найдены с помощью «большого» набора тестов, также обнаруживались и с помощью «маленького» набора.

4. Направления дальнейшей работы

Для второй задачи, именно, задачи построения «неправильного» тестового набора, традиционен следующий подход: неправильные тесты строят на основе правильных путем некоторых мутаций. Основной проблемой при этом является обеспечение гарантии того, что полученные тесты действительно не принадлежат целевому языку.

Кроме тестирования парсеров, большой интерес представляет также разработка методов тестирования других компонентов инструментов для работы с текстами на формальном языке, таких как:

- парсер с сообщениями об ошибках;
- tree builder, т.е. парсер, возвращающий дерево разбора входного предложения;
- анализатор статической семантики.

В настоящее время нами ведутся исследования в этих направлениях.

Литература.

1. <http://www.metamata.com/JavaCC>
2. <http://www.ispras.ru/~RedVerst/RedVerst/White Papers/Java Specification Extension for Automated Test Development/Main.html>
3. I. B. Bourdonov et al. "Java Specification Extension for Automated Test Development". In proceedings of the Fourth A. P. Ershov International Conference, PSI'2001
4. http://www.ispras.ru/~mpc, "The mpC Programming Language Specification", The Institute for System Programming of Russian Academy of Science.
5. "The mpC Programming Language Specification", The Institute for System Programming of Russian Academy of Science.

Приложение А. Синтаксис BNF

BNF, используемая в данной работе, имеет следующий вид:

grammar ::= (*rule*)+ ;
rule ::= *id* **::=** *expression* **;** ;
expression ::= *term* (**|** *term*)***** ;
term ::= (*factor*)+ ;
factor ::= *id*
| <LEXEME_TEXT>
| <LEXEME_NAME>
| **(** *expression* **)** (***** | **+** | **?**)?
;
id ::= <ID> ;

Ющенко Н.В.

Оценка времени выполнения программ статико-динамическим методом

Введение

Задача оценки времени выполнения программ является одной из основных подзадач в области оценки эффективности вычислительных систем. Как показано в [1], корректное сравнение производительности различных вычислительных систем возможно только на основе сравнения времен выполнения программ на этих системах. Кроме того, существуют задачи, в которых время выполнения программ является показателем не только эффективности, но и корректности решения: если решение получено слишком поздно, то оно уже не представляет интереса.

Распространены две постановки задачи оценки времени - оценка наихудшего случая и оценка конкретного случая. В первой постановке требуется дать верхнюю оценку времени выполнения программы, которая не будет превышена ни при каких входных данных этой программы. Во второй постановке входные данные программы считаются известными и требуется с максимальной точностью оценить время выполнения программы именно с этими входными данными.

В настоящей статье рассматривается вторая постановка - оценка конкретного случая. Эта постановка возникает, например, при имитационном моделировании распределенных вычислительных систем [4]. В процессе моделирования требуется определять, сколько времени займет автономная работа последовательного процесса, входящего в состав распределенной системы, между операциями взаимодействия с другими процессами. От этого времени может существенно зависеть дальнейшее развитие системы, поэтому время требуется оценивать с высокой точностью.

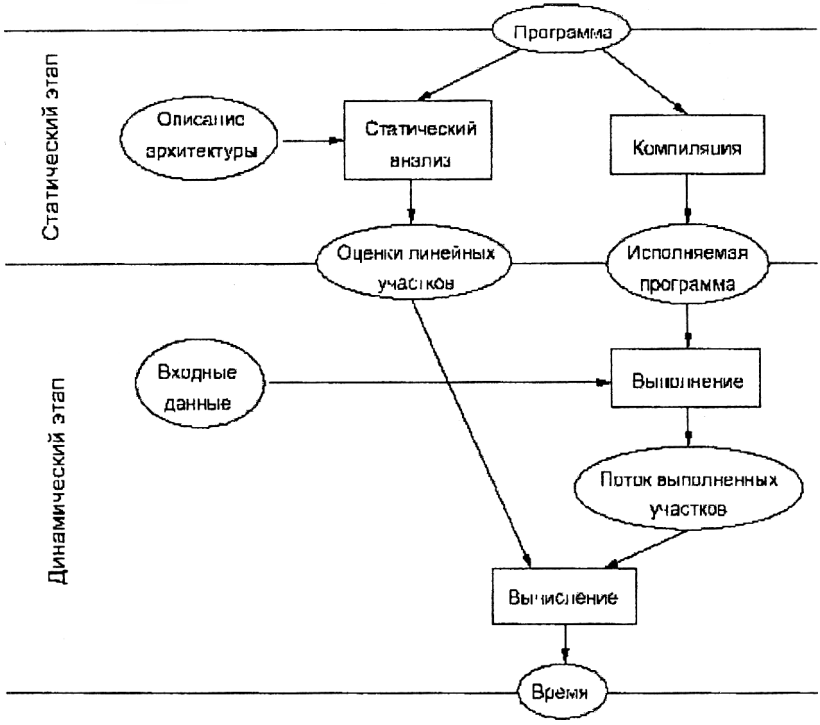
Оценка времени конкретного выполнения последовательной программы на сколь угодно сложном процессоре может быть осуществлена с любой требуемой точностью при помощи эмуляции работы этого процессора на достаточно детальном уровне. Однако этот метод часто оказывается неприменим из-за того, что эмуляция - очень медленный процесс. Типична ситуация, когда время выполнения программы под эмулятором на несколько порядков меньше времени ее выполнения на реальном процессоре, и эмуляция нескольких секунд работы программы требует многих часов или даже суток.

В данной статье рассматривается другой способ получения оценок времени выполнения программ - статико-динамический подход.

Описывается общая схема получения оценок. Анализируются достоинства и ограничения подхода. Наконец, предлагается путь применения статико-динамического подхода для оценки времени выполнения программ, обработанных оптимизирующим компилятором.

Статико-динамический подход

Исходный вариант статико-динамического подхода был предложен в [6]. Схема этого варианта представлена на рисунке 1.



Оценка времени выполнения программы состоит из следующих основных этапов.

- Статический этап. Программа, представленная на языке высокого уровня Си, разбивается на линейные участки и для каждого участка вычисляется статическая оценка времени его выполнения. Для вычисления статической оценки строится *модельный код* для этого участка, с использованием алгоритмов оптимальной кодогенерации - алгоритма Ахо-Джонсона [2] для регистровых машин и алгоритма Бруно-Лессаджа [3] для стековых машин. Предполагается, что модельный код близок к реальному, и за оценку времени

выполнения линейного участка принимается время выполнения модельного кода. Время выполнения модельного кода получается как сумма времен выполнения отдельных операций. Времена выполнения операций указываются в описании архитектуры.

- Динамический этап. Программа компилируется и запускается на выполнение на доступной инструментальной машине, и определяется последовательность выполненных участков. Для получения результирующей оценки времени складываются полученные статические оценки выполненных участков.

Для оценки времени выполнения той же программы с другим набором входных данных достаточно повторить только динамический этап.

Экспериментальные исследования статико-динамического метода показали его высокую эффективность - время получения оценки отличается от времени работы анализируемой программы всего в несколько раз. Высокая эффективность метода связана с тем, что на статическом этапе каждый линейный участок программы анализируется однократно, а на динамическом этапе количество дополнительных действий, которые требуется выполнить, невелико - фактически только извлечение оценок из таблицы и добавление их к счетчику.

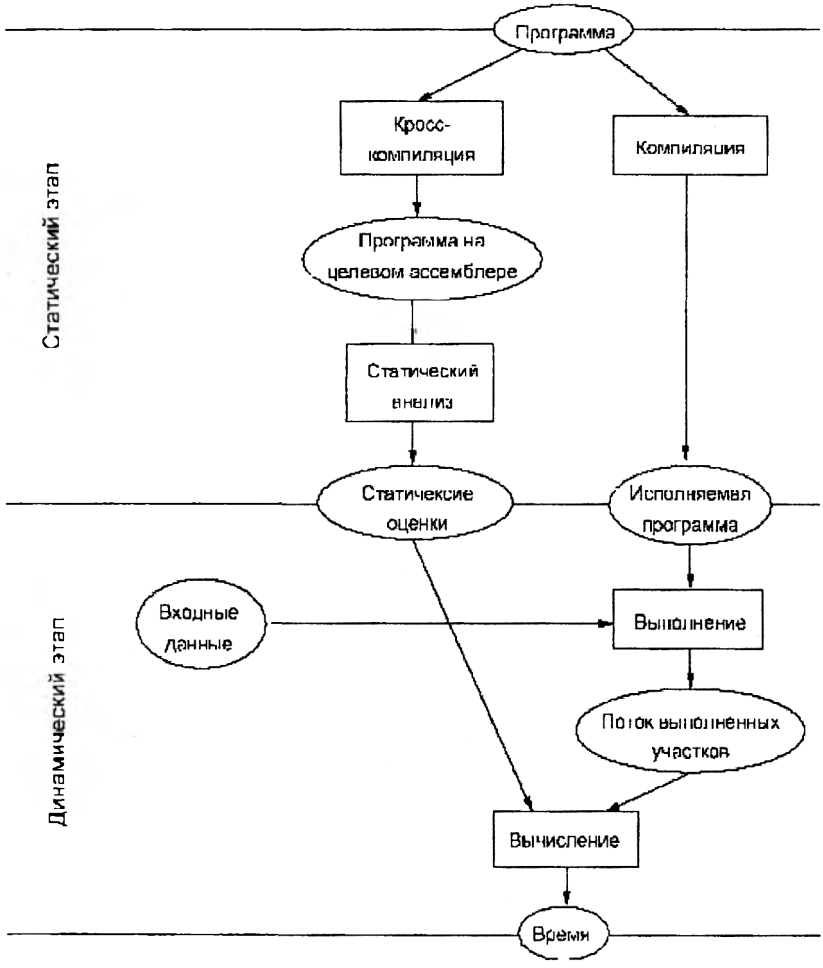
Для сравнительно простых архитектур, таких как Intel 8088, точность метода также была высокой. Погрешность оценок не превосходила 15 процентов.

Статико-динамический подход с кросс-компиляцией

С развитием архитектуры процессоров и методов компиляции погрешность статико-динамического метода стала увеличиваться. Это связано со следующими причинами:

- применение в компиляторах методов глобальной оптимизации кода сделало неадекватным использованный метод построения модельного кода (модельный код использовал алгоритмы, оптимальные для отдельных выражений, но не оптимальные для более крупных фрагментов программы);
- в связи с использованием в процессорах параллельно работающих функциональных устройств стала неадекватной оценка времени выполнения последовательности операций в виде суммы времен выполнения отдельных операций.

После анализа этих причин статико-динамическая схема была модисфицирована [7]. Новая схема представлена на рисунке 2.



В новой схеме набор машинных операций, которые будут выполнены в процессе выполнения линейного участка, определяется при помощи кросс-компилятора. Это позволяет решить обе отмеченные выше проблемы:

- анализируется именно тот машинный код, который будет выполняться при выполнении программы на целевой машине;
- код на ассемблере содержит достаточно информации для оценки времени его работы с учетом параллельно работающих устройств процессора.

Статико-динамическая модель процессора, построенная по этой схеме, состоит из кросс-компилятора, статического анализатора,

библиотеки поддержки динамического анализа и вспомогательных программ.

Кросс-компилятор используется для преобразования программы, представленной на языке Си, в программу на языке ассемблера целевого процессора.

Статический анализатор принимает на вход программу на языке ассемблера, разбивает ее на линейные участки и вычисляет статические оценки времен выполнения отдельных участков. Кроме того, статический анализатор может выдавать дополнительную информацию, которая может быть использована на динамическом этапе для коррекции статических оценок.

Библиотека поддержки динамического анализа используется вычисления оценки времени в процессе выполнения анализируемой программы. Перед компиляцией на инструментальную машину в анализируемую программу вставляются вызовы библиотеки поддержки динамического анализа так, чтобы по последовательности этих вызовов можно было получить информации о последовательности выполненных линейных участков.

Вспомогательные программы используются для установления соответствия между линейными участками программы на языке Си и линейными участками программы на языке ассемблера.

На основе модифицированного статико-динамического подхода были построены модели RISC-процессоров семейства Sparc [8] и процессора цифровой обработки сигналов Motorola DSP96002 [9]. В качестве кросс-компилятора были использованы соответствующие версии компилятора GNU C. В моделях процессоров семейства Sparc были реализованы алгоритмы анализа работы конвейера команд и кэш команд. Погрешность моделей процессоров Sparc не превышала 15 процентов. Модель процессора DSP96002 на типичных задачах цифровой выдавала оценки с потактовой точностью.

Статико-динамический подход для оптимизированных программ

Статико-динамический подход опирается на предположение о том, что поток выполненных линейных участков на целевой и инструментальной машине одинаков.

Однако, это предположение верно не всегда. Формально оно перестает быть верным уже при введении кросс-компилятора, т.к. на уровне ассемблера возникают новые линейные участки, связанные с особенностями архитектуры целевого процессора. Например, если в целевой архитектуре условные переходы возможны только на небольшие расстояния (из-за ограниченного размера поля смещения в

двоичном коде команды), то возникают дополнительные линейные участки для замены дальнего условного перехода на ближний условный и дальний безусловный.

В статико-динамических моделях процессоров Sparc и DSP96002 эта проблема решалась при помощи анализа путей в графе управления программы и установления соответствия между непересекающимися последовательностями линейных участков (а не отдельными линейными участками). Этот способ, однако, пригоден только при условии гомеоморфности графов управления программы на уровне Си и на уровне ассемблера.

Оптимизирующие компиляторы могут существенно менять граф управления программы и делать его не гомеоморфным исходному. Примером оптимизаций, способных привести к не гомеоморфному графу, являются стандартные машинно-зависимые оптимизации для VLIW процессоров - if-conversion [14] и конвейеризация циклов [13].

Для оценки времени программ, обработанных оптимизирующим компилятором, статико-динамическая схема была модифицирована. Получившаяся схема представлена на рисунке 3.

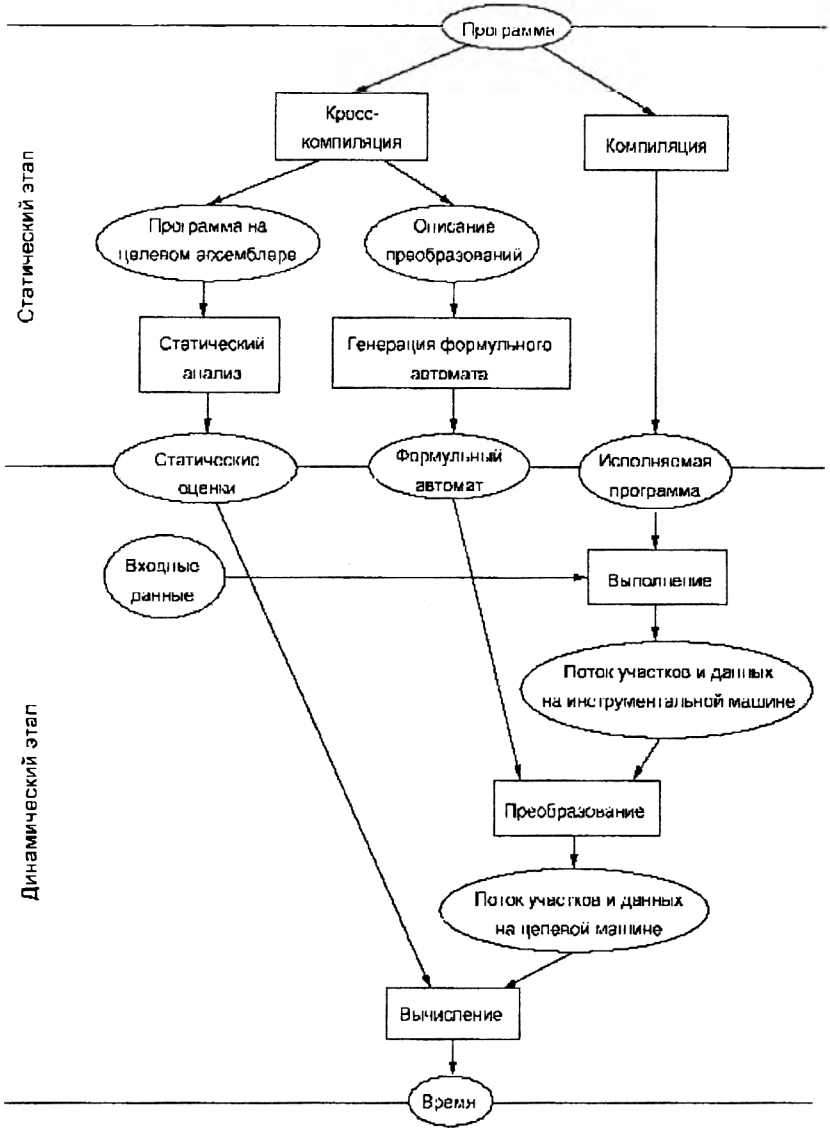
На динамическом этапе модифицированной схемы явно присутствует преобразование потока выполненных участков на инструментальной машине в поток участков для целевой машины.

Разработан формализм для строгого описания этого преобразования. Формализм позволяет описать не только преобразование последовательностей линейных участков, но и преобразование значений, вычисляемых в этих участках. А именно, описывается связь значений, вычисляемых при выполнении линейных участков оптимизированной программы на целевой машине, со значениями, вычисляемыми при выполнении линейных участков неоптимизированной программы на инструментальной машине.

Основные понятия разработанного формализма приводятся ниже.

- Представление программы - множество линейных участков, каждый из которых имеет свой набор виртуальных регистров. Каждый виртуальный регистр соответствует единственному значению, вычисляемому или хранимому в процессе выполнения линейного участка.

- Выполнение представления программы - последовательность шагов. Каждый шаг - выполнение некоторого линейного участка. Этому выполнению соответствуют некоторые конкретные значения всех виртуальных регистров данного линейного участка.



- Формульный автомат - автомат, принимающий последовательность шагов одного представления программы, и выдающий соответствующую последовательность шагов другого представления программы.

С введением этого формализма статико-динамическая схема превращается из метода оценки в метод строгого вычисления времени

выполнения программы с потактовой точностью. Возможность точного вычисления времени вытекает из следующих утверждений.

• Время выполнения представления программы можно вычислить, зная полную последовательность шагов этого выполнения. Это следует из того, что функционирование последовательной вычислительной системы полностью определяется программой (и ее входными данными).

• Последовательность шагов выполнения одного представления программы можно вычислить, зная последовательность шагов выполнения другого представления программы. Такое преобразование осуществляется формульным автоматом.

• Последовательность шагов выполнения представления программы можно вычислить путем наблюдения этого выполнения на некотором исполнителе.

При вычислении времени модифицированным статико-динамическим методом, на динамическом этапе определяется последовательность шагов выполнения представления программы для инструментального вычислителя. Затем при помощи формульного автомата она преобразуется в последовательность шагов выполнения представления программы для целевого вычислителя. Наконец, проводится вычисление времени выполнения получившейся последовательность шагов на целевом вычислителе. Все перечисленные действия могут быть выполнены точно, без внесения погрешности. Соответственно, получившееся значение времени будет точным.

Для того, чтобы эффективно и точно вычислять время выполнения данной последовательности шагов, требуются специальные алгоритмы, разбивающие вычисление на статический и динамический этапы и сводящие количество действий на динамическом этапе к минимуму. Такие алгоритмы для моделирования различных устройств процессоров были разработаны [5].

Для выполнения преобразования последовательностей шагов выполнения программы на инструментальный машине в последовательность шагов выполнения программы на целевой машине был разработан следующий метод.

• Обработка программы оптимизирующим компилятором рассматривается как последовательность преобразований. Каждое преобразование имеет на входе одно представление программы и получает в результате другое представление программы. Такая модель достаточно точно описывает действия реальных компиляторов [10].

• Каждое преобразование описывается на специально разработанном языке описания преобразований. Язык основан на регулярных выражениях, символами которых являются линейные участки исходного представления программы. К символам регулярных

выражений приписываются участки преобразованного представления программы и формулы для преобразования значений. Выразительной силы языка описания преобразований оказалось достаточно для описания широкого класса оптимизаций, используемых при генерации кода для VLIW процессоров.

• По описанию каждого преобразования строится формульный автомат, реализующий это преобразование.

• Вычисляется последовательная композиция построенных формульных автоматов. Получается формульный автомат, преобразующий шаги исходного представления программы (на языке Си) в шаги оптимизированного представления программы.

Для получения описания преобразований программы, осуществляемых оптимизирующим компилятором, существуют два пути.

• Можно внести изменения в компилятор, чтобы он выдавал описания преобразований. Этот метод позволяет сохранить абсолютную точность.

• Можно разработать специальный инструмент, который на основе выдаваемой компилятором отладочной информации и, возможно, некоторых дополнительных знаний о компиляторе, будет строить описание преобразований. В этом случае может возникнуть погрешность, если информации для точного описания преобразования окажется недостаточно.

Следуст заметить, что в работе [12] также ставится вопрос об описании преобразований программы оптимизирующим компилятором. Предложенный там метод используется для оценки времени наихудшего случая. Однако использованный там формализм непригоден для высокоточной оценки конкретного случая, рассматриваемой в данной статье.

Формульный автомат реализуется на языке Си в виде набора функций. Каждая функция соответствует приему автоматом на вход одного шага. Размер кода для таких функции, получаемый компилятором GNU C версии 2.95, составляет в среднем от 10 до 30 команд процессора x86. Это значит, что работа формульного автомата не замедлит существенно динамический этап статико-динамической схемы, т.е. основное достоинство схемы - ее высокая эффективность - сохранится. Что же касается точности, то, как было показано выше, она может быть потактовой.

В настоящее время ведется работа по реализации рассмотренного выше модифицированного статико-динамического метода на основе кросс-компилятора Trimaran [11]. Ожидаемым результатом является построение системы оценки времени, которая

будет выдавать в точности те значения, что и входящий в состав системы Trimaran эмулятор, однако значительно быстрее.

Заключение

В настоящей статье был рассмотрен статико-динамический метод получения оценок времени выполнения программ. Был рассмотрен исходный вариант этого метода, разработанный для процессоров со сравнительно простой архитектурой. Был описан механизм введения в статико-динамический метод кросс-компилятора, позволивший работать с более сложными процессорами. Наконец, было предложено расширение статико-динамического метода для поддержки оптимизирующих компиляторов.

Был разработан формализм, превращающий статико-динамический метод в строго определенное средство точной оценки времени. Точность оценок, получаемых в рамках этого формализма, не должна уступать точности оценок, получаемых на эмуляторе. При этом скорость получения оценок значительно выше, чем в случае использования эмулятора.

Литература

1 Hennessy J.L., Patterson D.A. *Computer Architecture: A Quantitative Approach.* // Morgan Kaufmann Publishers, CA 1990

2 Aho A.V., Johnson S.C. *Optimal Code Generation for Expression Tree.* // J.ACM, 1976, v.23, No 3, pp. 488-501

3 Bruno J.L., Lassagne T. *The Code Generation of Optimal Code for Stack Machines.* // J.ACM, 1975, v.22, No 3, pp. 382-396

4 Bahmurov A.G., Smeliansky R.L. *DYANA: an Environment for Distributed System Design and Analysis* // Proc. of the III International Workshop on Parallel Processing by Celluar Automata and Analysis. 1996, pp. 85-92

5 Калиниченко Д.В., Капитонова А.П., Ющенко Н.В. *Методы и средства прогнозирования времени выполнения последовательных программ.* // Методы математического моделирования, МГУ, 1997, N2

6 Капитонова А.П. *Методы и средства прогнозирования времени выполнения последовательных фрагментов программ на вычислителях с различной архитектурой.* // Диссертация на соискание степени кандидата физико-математических наук. 1997

7 Балашов В.В., Капитонова А.П., Костенко В.А., Смелянский Р.Л., Ющенко Н.В. *Построение моделей процессоров цифровой обработки сигналов на основе статико-динамического подхода.* // Труды конференции DSPA98 (цифровая обработка сигналов и ее применение). Москва, 1998

8 *The SPARC Architecture Manual: Version 8*, Prentice Hall, ISBN 0-13-825001-4

9 *Процессор цифровой обработки сигналов Motorola DSP96002. Руководство пользователя.* // Пакет документации фирмы Motorola

10 *Using and Porting the GNU Compiler Collection (GCC).* Пакет документации проекта GNU

11 Интернет-сайт проекта Trimaran: http://www.trimaran.org/

12 Jakob Engblom. *Worst-case execution time analysis for optimized code*. Master's thesis, Department of Computer Systems, Uppsala University, September 1997. DoCS MSc Thesis 97/94

13 Vicki H. Allan, Reese B. Jones, Randal M. Lee, Stephen J. Allan. *Software Pipelining* // ACM SIGPLAN, 1992

14 B.Ramakrishna Rau, Joseph A. Fisher. *Instruction-Level Parallelel Processing: History, Overview and Perspective* // Hewlett Packard Labs Technical Report, 1992

Леонов М.В.

Комплекс программ для автоматизации исследований в таксономической ботанике.*

«Таксономия – раздел систематики, учение о системе таксономических категорий, обозначающих соподчиненные группы объектов»

Большой Энциклопедический Словарь

Введение

В статье кратко представлены программные результаты, полученные в лаборатории вычислительного практикума и информационных систем факультета ВМиК МГУ за последние годы в процессе межфакультетского сотрудничества с коллективом Отдела систематики и географии растений Ботанического сада МГУ.

1. Актуальность и предпосылки .

Сначала несколько слов об актуальности разработанных программных средств и предпосылках нашего межфакультетского сотрудничества. Проблема сохранения биоразнообразия, по-видимому, не нуждается в обосновании актуальности, а одна из основных задач в этой сфере – инвентаризация объектов исследования и результатов, полученных при их изучении. Разработанные нами настольные информационные системы с базами данных по различным аспектам изучения растений семейства Зонтичных как раз и предназначены для автоматизации таких задач – типичных для ботаника-систематика. В частности, одна из основных целей комплекса – генерировать монографии по объектам исследования, а также давать всевозможные справки, некоторые из которых полезны для подготовки новых экспедиций и т.д.

* В настоящее время работа поддержана грантом **РФФИ 00-07-0066.**

Предпосылки. В Ботаническом саду МГУ сложилась группа исследователей под руководством проф. М.Г.Пименова, занимающая лидирующую позицию в мировой ботанике по изучению семейства Зонтичных, очень важного как в практическом, так и в теоретическом отношении. А с другой стороны, наше желание помочь ботаникам в автоматизации их работы нашло в свое время поддержку А.Н.Тихонова, а затем Д.П.Костомарова.

2. Специфика предметной области.

Основные объекты – таксоны растительного мира (точнее, таксоны семейства Зонтичных) – род, вид и другие; статьи, книги, благодаря которым описания этих таксонов приобретают законную силу; так называемые типовые образцы, хранящиеся в гербариях мира; результаты кариологических исследований. (Работа с изображениями растений и морфологическими описаниями пока не входит в функции комплекса). Специфика семейства Зонтичных состоит в частности и в том, что границы между таксонами не являются незыблемыми, а время от времени подвергаются ревизиям. В результате некоторые названия переходят в разряд так называемых синонимов, а иногда и наоборот. Но эти синонимы продолжают сохранять ценность, так как в соответствии с Ботаническим Кодексом, при номенклатурных рассмотрениях должны учитываться все названия, опубликованные после 1753 года. Старые книги, гербарные этикетки могут, естественно, содержать названия, перешедшие в разряд синонимов.

Число объектов довольно велико – названий принятых родов – сотни, видов и внутривидовых таксонов – тысячи. Учитывая номенклатурную динамику, очевидно, что без СУБД здесь обойтись сложно.

3. Специфика требований к аппаратному и программному обеспечению.

Ботаника не относится к хорошо финансируемым отраслям знаний, поэтому приходилось и приходится это учитывать и ориентироваться на простые доступные ресурсы. К примеру, наша БД CARUM [3] по кариологии была сначала создана на ЕС-1045 с использованием сетевой СУБД АСО ЭКСТЕРН (производства факультета ВМиК, автор – Ю.П.Лященко). БД по номенклатуре родов сем. Зонтичных GNOM [3]– появилась сначала на Ямахе под управлением dBase II. Опуская промежуточные этапы, сообщим, что основной пакет существует на PC-486 под управлением FoxPro 2.6 (MS-DOS), а доступ к одной из БД по Интернет осуществляется через сервер факультета ВМиК. Интерфейс основных программ выполнен в терминах предметной области, то есть созданы дружелюбные к

ботаникам пользовательские оболочки, что было особенно важно для не слишком компьютерно образованных ботаников. К особенностям нашего комплекса можно отнести и то, что он является эволюционирующим – не только из-за появления новых аппаратных и программных возможностей, но и в связи с тем, что по мере освоения старых задач у ботаников появляются новые, которые получают в дальнейшем компьютерную поддержку [4]. Можно сказать, что при разработке программ нами активно используется метод макетирования (или быстрого прототипа).

4. Основные компоненты комплекса

Основу комплекса составляют ИПС GNOM, CARUM и ASIUM, подробно описанные в работах [1], [3] и [5] соответственно.

Они обеспечивают в интерактивном режиме ввод и исправление данных, получение разнообразных справок, а также позволяют генерировать в нужном формате монографии по соответствующим темам: GNOM – по номенклатуре родов в объеме мировой флоры, CARUM – по кариологии (по хромосомным числам) растений семейства Зонтичных и ASIUM – ботанико-географическую и таксономическую сводку по видовому составу азиатского континента (центра видового разнообразия Зонтичных). С помощью первой из этих систем была сгенерирована монография [2], опубликованная в Великобритании. Это была первая в российской (и советской) ботанике монография, сгенерированная с использованием СУБД.

5. Интернет-компоненты комплекса

С появлением Интернета возникли и новые задачи. Вот три из них: 1) возникла потребность сообщить о себе и своих базах данных мировому ботаническому сообществу; 2) в Интернете появились БД и другие источники данных, содержащие полезную для ботаников информацию, но пользоваться которыми не очень удобно; 3) необходимость автоматизировать поиск нужной для ботанического коллектива информации.

Кратко остановимся на наших решениях этих задач.

1)Так как благодаря публикации [2] в Великобритании наша БД GNOM приобрела довольно широкую известность, то в первую очередь было решено обеспечить доступ через Интернет именно к этой базе. На сервере факультета ВМиК был размещен сайт, отражающий основные результаты сотрудничества нашей лаборатории с Отделом систематики Ботанического сада. В силу уже упомянутых ограничений был выбран с нашей точки зрения простейший, но довольно

оригинальный путь доступа к БД GNOM: создана программа генерации из БД множества связанных между собой HTML-файлов, которые представляли на нашем сайте как бы квази-базу данных по номенклатуре родов, удобную для просмотра.

Затем были реализованы (с помощью CGI-механизма) и запросы по географическому распространению таксонов (весьма важные для ботаников), включая определение списка эндемиков регионов. Соответствующие программы были написаны на языке Perl.

2)Как уже было сказано, ботаника относится к слабо финансируемой отрасли и на Западе, и наверное по этой причине многие Интернет-базы далеки от совершенства, а точнее, ими иногда неудобно пользоваться. Например, в некоторых БД отсутствует возможность получить данные сразу по всему семейству (либо по роду) – и приходится в этих случаях повторять десятки, а то и сотни нажатий комбинаций тех же самых клавиш. Для автоматизации таких процессов были созданы программные агенты на языке Perl для пакетного опроса по заданному списку, что естественно, существенно экономит время и нервы.

3) Поиск новых источников по Интернет – третья задача, которая была автоматизирована [7]. При использовании стандартных поисковых машин получалось много «шума», и поэтому у нас была разработана программа (сначала на языке Perl, а потом с помощью системы Delphi), с помощью которой реализуется мониторинг для обнаружения новых и обновленных страниц; ведется протокол просмотренных сайтов – фиксируются изменения на сайтах, включенных в список, и благодаря этому также сокращается время ручной работы.

Заключение

Как уже было сказано выше, комплекс продолжает развиваться. В ближайшей перспективе в комплекс войдет новая ИПС по персоналиям исследователей семейства зонтичных, содержащая различные транслитерации их фамилий и различные варианты сокращений, использовавшиеся при публикации названий таксонов. Второе важное направление - интеграция полезной информации по семейству Зонтичных в электронной форме из различных источников, что особенно важно для преодоления трудностей, связанных с плохим качеством линий связи с Интернет, другими словами, преобразование нашего сайта в «виртуальный читальный зал» для ботаников-систематиков.

Литература.

1. Пименов М.Г., Леонов М.В. Компьютерная база данных по номенклатуре родов Umbelliferae мира. 1992, Бот. Журнал, 77, 12, с.69-77.

2. M.G.Pimenov, M.V.Leonov. The genera of the Umbelliferae. A nomenclator. 1993. Royal Botanic Gardens, 156pp. Kew.

3. М.Г.Пименов, М.В.Леонов, М.Г.Васильева, Ю.В.Даушкевич. Компьютерная база данных по кариологии Umbelliferae мировой флоры. Бот. журн., 1993, 78, 10: 65-71.

4. Леонов М.В. Базы данных как средство повышения компьютерной грамотности ботаников. II совещание "Компьютерные базы данных в ботанических исследованиях". Тезисы докладов. С-Петербург, 1995, с. 28-30.

5. Пименов М.Г., Леонов М.В. ASIUM - таксономическая и ботанико-географическая база данных по зонтичным Азии. Там же, с 43-44.

6. М.В.Леонов. Доступ к базе данных GNOM в Internet: использование несложной технологии (http://www.botanik.cs.msu.su). В кн.: Информационно-Поисковые системы в Зоологии и Ботанике. - Труды Зоологического Института РАН, Санкт-Петербург, 1999, Том 278, стр. 98-99.

7. М.В.Леонов, Д.Ю.Малинин. Автоматизированный мониторинг и накопление электронных данных по растениям семейства Umbelliferae из источников Интернет. В кн.: Информационные и телекомуникационные ресурсы в Зоологии и Ботанике. –Санкт-Петербург, 2001 г., стр. 99-100.

Маслов С.П.

Карманная АСО "Наставник" (стартовая версия)

Обсуждаются достоинства автономной, компактной (карманной) и доступной АСО "Наставник" с точки зрения повышения удобства работы в системе и упрощения ее установки и эксплуатации. Предлагается программная реализация карманной АСО на существующей тиражируемой технике. Описывается система на устройстве Organiser II XP фирмы PSION. Рассмотрены "облик" системы, организация и кодирование управляющей информации, структура программы. Приведены результаты тестирования. Описанная в статье реализация была выполнена автором в инициативном порядке летом 2000г.

Введение

Микрокомпьютерная Система Обучения (МСО) "Наставник" [1] существует с начала 70 годов, реализована в многочисленных вариантах [2] и используется поныне. Это говорит о том, что система уверенно занимает свою нишу в существующем многообразии компьютерных обучающих систем. Вместе с тем, распространенность системы могла бы быть значительно большей, если бы желающим иметь ее в своем распоряжении не приходилось преодолевать значительные трудности. Несмотря на то, что разработчики смогли в свое время наладить мелкосерийное производство многих компонентов системы, будущий ее владелец сталкивался с затратами "ручного труда" значительного объема - работой по размещению минитерминалов, соединению их друг с другом и с управляющим компьютером, приобретению и установке источников питания и т.д. Лишь немногие энтузиасты оказались в состоянии довести дело до конца. В результате, в частности, не удалось в нужной степени привлекать педагогов для написания курсов, что, в свою очередь, не способствовало расширению применений системы.

Решением проблемы мог бы стать автономный вариант системы - компактное (карманное) и недорогое устройство, внешне подобное простому калькулятору, но содержащее в себе все необходимое для выполнения функций "Наставника". Карманную систему не нужно устанавливать - она работает от батарейки и всегда готова к употреблению, ее можно носить с собой и использовать в любом месте. При автономной работе учебные курсы могут либо изначально находиться в устройстве, либо загружаться в него из

сменных модулей. При коллективном использовании (в школах, ВУЗ-ах и т.д.) установка системы - связь карманных "Наставников" с центральным компьютером - осуществляется на основе стандартного связного оборудования. Требуемый курс тогда загружают в устройство из "большого" компьютера через связной порт; через него же выгружают архив - информацию о том, как обучаемый проходил этот курс. Сохраняя имеющиеся достоинства система приобретает новые качества - мобильность и автономность.

Карманный "Наставник" органично вписывается в Интернет - средствами последнего можно распространять учебные пособия, дистанционно производить загрузку курсов и выгрузку архива и обеспечивать участие преподавателя в процессе обучения.

Относительно того, какой быть карманной системе, имеются различные мнения. Согласно одному из них она должна представлять собой специализированное (предназначенное исключительно для работы в качестве "Наставника") устройство отечественного производства. Реализовать единичный экземпляр-макет такого устройства вполне возможно, однако представляется неоправданной столь узкая его специализация. Дело в том, что устройство должно быть дешевым, а организовать массовое производство специализированного "карманного Наставника" и обеспечить спрос такого уровня, чтобы он был "по карману" широкому кругу пользователей, нереально.

Другое мнение основывается на том, что для этого следует использовать существующую тиражируемую технику - выпускаемые в настоящее время многими фирмами карманные калькуляторы, электронные записные книжки, органайзеры, переводчики и т.д. Все они имеют в своем составе микропроцессор, память, клавиатуру и индикатор. Многие оснащены также связным интерфейсом. Штатные приложения таких устройств реализуются программно. При наличии запаса программной памяти к имеющимся приложениям можно добавить новое - работу в системе "Наставник". Для этого программное оснащение пополняется новой компонентой. Такая модернизация практически не сказывается на стоимости устройства.

В статье содержится описание "карманного Наставника", реализованного на базе последнего подхода. В основу взята модель Organiser II XP, выпускаемая фирмой PSION [3]. Выбор модели обусловлен прозаическими причинами - на момент начала работы (1998г.) это было единственное доступное по цене устройство с возможностью программирования задач невычислительного характера.

Стартовый вариант карманной системы является однотерминальным по определению и несколько урезан по сравнению со штатным. Реализована только подсистема "Обучение". Секции должны иметь сплошную нумерацию. Не ведется архив. Нет упражнений типа ПП, обязательных упражнений и ссылок на абзацы.

Organiser II-XP

Размеры: 142 x 78 x 29,3 мм. Вес: 250 гр.

Индикатор: две строки по 16 символов. Клавиатура: 36 клавиш

Микропроцессор: HD6303X с тактовой частотой 3,6864 МГц

Память: ОЗУ 32КВ (из них свободно до 23,5КВ), ПЗУ 32КВ

Сменные модули: Rampak (ОЗУ) 32КВ; Datapak (ППЗУ) 8/16/32/64/128КВ;

CommLink (связной порт типа RS-232C)

Приложения: зап. книжка, часы, калькулятор, язык OPL. и др.

Внешний "облик" карманного "Наставника"

Терминал "Наставника" эмулирован на Organiser-е. Задействованы клавиша включения "ON/CLEAR", цифровые клавиши от "1" до "8" для ввода ответов и клавиша "EXE", соответствующая клавише "," в "Наставнике". Остальные клавиши заблокированы и нажатия на них не воспринимаются. Устройство рассчитано на англоязычного пользователя поэтому все надписи на индикаторе и выдачи делаются на английском языке. Например: "ПР" - "**OK**". OPL позволяет определить до 8 новых символов и систему можно русифицировать.

Запуск осуществляется выбором процедуры **NAST:** в Главном Меню. На индикаторе в верхней строке высвечивается "шапка" полей выдач, а в нижней - "**press EXE**". При последовательных нажатиях клавиши "EXE" в соответствующих полях нижней строки появляются номера: курса **k**, секции **sec** и упражнения **exr**. Если в упражнении менее 8 ответов, ненужные цифровые клавиши блокируются. Эхо ответа видно в поле **rp**. Реакция на ответ выдается в поле справки **inf**. Изображение на индикаторе имеет вид:

k	sec	exr	rp	inf
X	XX	XX	X	XX

При успешном завершении работы в верхней строке вместо "шапки" выдается сообщение: ****GOOD!THE END****, а при неудаче: **WRONG! TRY AGAIN**. При этом в поле **inf** появляется поясняющая информация как при успешном окончании работы: **EoK** - конец курса; так и при досрочном ее прекращении: **S=0** - возврат в нулевую секцию или **R>4** - допустимое число возвратов (4) исчерпано. Это уместно, поскольку преподавателя поблизости может не оказаться.

Если клавиши долго не нажимались, устройство "засыпает" и индикатор погасает. Его "пробуждают" нажатием клавиши ON/CLEAR , а затем продолжают работу с того места, на котором "заснули".

Реализация карманного "Наставника"

Штатный вариант системы написан на ДССП [4,1]. Поэтому, на первый взгляд, следовало пойти "каноническим" путем - реализацией ДССП на Organiser-е и последующим переносом на него "Наставника". Однако сделать это в данных условиях было затруднительно - мала пользовательская часть ОЗУ. Даже если не учитывать других приложений имелось лишь 23,5 КВ. Одна из причин этого - наличие у Organiser II-XP собственных средств программирования - процедурного языка OPL. Заменить его на ДССП невозможно ни технически (OPL частично "зашит" в ПЗУ), ни "потребительски" (на OPL реализованы штатные приложения). По самым скромным оценкам ядро ДССП заняло бы не менее 15КВ и места для "Наставника" и всего остального просто не осталось бы. Таким образом не было иного способа, как реализовывать "Наставник" заново на OPL.

Краткие сведения о языке OPL

OPL сходен с BASIC-ом. Программы на OPL, представляют собой набор процедур, которые пишутся и отлаживаются по отдельности. Выполнение программы начинается с главной процедуры. В данном случае это процедура **NAST:**. В ней задаются параметры текущего курса - его номер, пределы отношения числа правильных ответов к числу попыток и максимальное число возвратов. В **NAST:** можно реализовать меню для выбора желаемого курса из имеющихся.

Процедуры вложены друг в друга и вся программа работает таким образом, что из главной процедуры вызывается процедура второго уровня, которая, в свою очередь вызывает процедуру третьего уровня и т.д. Каждая процедура после выполнения своих функций возвращает управление процедуре, откуда он была вызвана. Возможно также иметь процедуры, которые вызываются с любого уровня. Написанные на OPL процедуры преобразуются в специальную (транслированную) форму и выполняются в режиме интерпретации.

Данные хранятся в ОЗУ в виде файлов. Файлы состоят из однотипных последовательно пронумерованных записей, количество которых не ограничено. Записи могут иметь до 16 определяемых полей. В разных полях можно хранить данные разных типов, но все они будут иметь символьное представление. Так, действительные и целые числа записываются в десятичном представлении в виде последовательности символов (цифр и пр.) - т.е. очень не экономно. Целое число, например, будет занимать столько байтов, сколько в нем значащих цифр.

При создании файла определяют тип данных, размещаемых в данном поле. Размеры полей не указываются и не ограничены - требуется только, чтобы длина записи не превышала 254 символов

(байтов). Файлы имеют последовательную структуру: после того, как файл создан и заполнен, нельзя менять ничего внутри него, а только удалять отдельные записи либо прибавлять их к концу файла.

Представление управляющей информации

Управляющая информация (УИ) - это совокупность данных, описывающих реакцию системы на ответы обучаемых. В "Наставнике" каждому из ответов в УИ сопоставлен элемент, содержащий от одного до трех целых значений: номер справки; номер справки + номер вспомогательного упражнения в текущей секции; номер справки + номер упражнения на повторение в предшествующей секции + номер этой секции. (Описание реакции на правильный ответ, по существу, тоже является справкой). В состав УИ, кроме этого, входит описание секции: сколько в ней упражнений, какого они типа и т.д.

Имеются три формы представления УИ: пользовательское, файловое и рабочее. Пользовательское представление (ПП) служит для подготовки управляющей информации составителем курса с целью ввода ее в систему и последующего преобразования в файловое представление (ФП). В ФП УИ хранится в Organiser-е. Рабочее представление (РП) получается из ФП и существует только для текущих упражнения и секции.

Пользовательское представление

Имеется "штатное" ПП, используемое в других реализациях "Наставника" [2]. Из него формируется внутреннее представление УИ, которое может быть разным у различных вариантов системы. В карманном варианте предусмотрены как загрузка в устройство УИ во внутреннем (файловом) представлении из персонального компьютера или из модуля дополнительной памяти, так и автономная подготовка УИ (здесь не описывается). В первом случае ПП не отличается от штатного. Во втором случае ПП похоже на штатное, но для упрощения процесса перекодирования и сокращения объема программы подготовки все числовые значения, в том числе и однозначные, записываются двухсимвольными фрагментами вида: <xx> (x - цифры от 0 до 9).

Например:

1 секция (упражнение, справка)	записывается: 01
21 секция (упражнение, справка)	записывается: 21

Если в ПП указывают на выдачу только справки, записывают единственный фрагмент: <номер справки>. Если указывают на выдачу вспомогательного упражнения, записывают подряд два фрагмента:

<номер справки><номер упражнения>. Если указывают на выдачу упражнения на повторение, пишут три фрагмента: <номер справки><номер секции><номер упражнения>. Если указывают на правильный (или частично правильный) ответы, записывают односимвольные фрагменты: <+> (или <*>. Последовательности, состоящие из 1, 2, 4 или 6 символьных фрагментов, называются элементами ПП. Элементы разделяются пробелами. Одна строка соответствует одному упражнению. Последний символ строки - пробел.

Например:

Строка: "`21 0214 133006 +` " означает: на 1-й ответ выдать справку 21, на 2-й - справку 2 и вспомогательное упражнение 14 в той же секции, на 3-й - справку 13 и упражнение 6 в 30 секции, на 4-й - справку "правильный ответ".

Кодировка информация о самой секции в ПП и ФП одинакова и будет описана ниже.

Файловое представление.

Это представление является внутренним. Оно называется файловым (ФП) поскольку используется при формировании файла с УИ, из которого программа, управляющая деятельностью обучаемых, получает сведения о реакции на их ответы. Эффективное кодирование УИ в ФП особенно важно для карманной системы с ограниченными ресурсами памяти.

Элемент УИ содержит от одного до трех целых значений, диапазон которых невелик. Так, номер справки лежит в интервале от 1 до 30; номер упражнения - от 1 до 15; номер секции - от 1 до 99. Для кодирования информации такого объема в ФП достаточно одного байта. Записи в файле с УИ содержат поля для данных символьного типа. В соответствующей символу ячейке ОЗУ помещается 8-битный код ASCII этого символа. Таким образом, фрагмент в ФП кодируется символом.

Секции курса соответствует одна запись в файле. Каждая запись содержат 16 полей. Нулевое поле несет сведения о самой секции. Поля с 1 по 15 хранят УИ упражнений в этой секции.

Поле упражнения представляет собой сплошную последовательность символов. Имеющиеся в ПП пробелы между элементами в ФП игнорируются. Игнорировать пробелы и достигнуть тем самым экономии размера файла стало возможным в результате того, что элемент всегда начинается со справки. Благодаря этому можно отыскать в строке ФП последовательность символов, соответствующую элементу ПП, по следующему признаку: она начинается символом-справкой и заканчивается символом, предшествующим символу-справке (либо концу поля). Чтобы такой поиск был возможен нужно уметь отличать символ-справку от других символов. С этой целью для кодирования справок используют символы, коды ASCII которых лежат

в диапазоне от 42 до 78. Символы "+" и "*"соответствуют реакциям на правильные ответы, а для кодирования номеров "настоящих" справок служат символы с кодами ASCII из диапазона от 49 до 78. Действительный номер справки определяют, вычитая из кода ASCII символа число 48. При этом символу "+" соответствует число -5, символу"*" - число -4, символу "1" - номер справки 1 ,.. и символу "N" - номер справки 30.

Символы с кодами ASCII из диапазона от 80 до 180 служат для кодирования номеров секций, вспомогательных упражнений и упражнений на повторение. Действительный номер в этом случае определяют, вычитая из кода ASCII символа число 79, а что он означает следует из того, на какой позиции символ находится. Если элемент состоит из 2-х символов, то 2-й символ - номер вспомогательного упражнения; если из 3-х, то 2-й символ - номер секции, а 3-й - номер упражнения в ней.

Нулевое поле в записи (поле секции) несет сведения о том, сколько в секции упражнений и какого они типа.

Например:

Строка вида "BBBBBBSSSSHHNNN" в поле секции информирует о том, что в секции имеется 6 основных упражнений B-типа (с 1 по 6), 4 итоговых S-типа (с 7 по 10) и 2 вспомогательных H-типа (11 и 12). Всего в секции 12 упражнений из допустимых 15 (в последних трех позициях строки стоит символ N).

В ФП секции учебного курса соответствует запись в файле, размеры которой в OPL ограничены 254 байтами. Для оценки влияния этого ограничения рассматрим гипотетическую секцию с максимально допустимым числом упражнений (15), каждое из которых имеет максимально допустимое число ответов (8). В 3-х ответах предусмотрен вызов вспомогательного упражнения, а в 2-х - упражнения на повторение. Запись, соответствующая такой секции, содержит 240 байтов. Таким образом, даже в этой практически невероятной ситуации место остается. Реально размеры записей-секций существенно меньше.

Рабочее представление

РП УИ существует для текущих упражнения и секции. РП упражнения хранится в массиве GE$(8,3) (8-элементный символьный массив, элементы которого содержат от 1 до 3 символов), формируемом с помощью процедуры CE:. Она копирует поле текущего упражнения номер GE% в переменную GU$ и "разбирает" его на элементы.

Поясним структуру ФП и РП на основе приведенного ранее примера ПП. В левой колонке таблицы записано ПП, в средней - ФП либо его копия в переменной GU$, а в правой выписаны 4 элемента массива GE$.

ПП	ФП	РП (эл-ты			
	(GU$)	1	2	3	4
21 0214 133006 +	E2^=nV+	E	2^	=nV	+

РП секции размещается в переменной GS$. При переходе на новую секцию туда копируется строка из нулевого поля соответствующей записи. Впоследствии там будут фиксироваться выдавемые упражнения и, таким образом, находить отражение текущее состояние секции.

Для выбора упражнения служит процедура RN:(P$). Процедура анализирует переменную GS$ и выясняет, имеются ли в секции не выданные ранее упражнения типа P$. Далее выдается выбранный случайным образом номер GE% такого упражнения, а соответствующий символ в строке GS% заменяет символом N.

Например:

GS$ = "BBNBNBSSNSHHNNN" - указывает, что упражнения 3,5 и 9 уже выдавались. Если в секции нет упражнения заданного типа - GE% =0.

Программа карманного "Наставника"

Программа состоит из 16 процедур разного объема и сложности. Большинство процедур являются вложенными. Глубина вложенности достигает 7.

Блок-схема программы показана на рисунке. Процедуры, которые могут быть вызваны с любого уровня, стоят особняком. Название, краткое содержание и размер процедур приведены в таблице.

Блок-схема программы

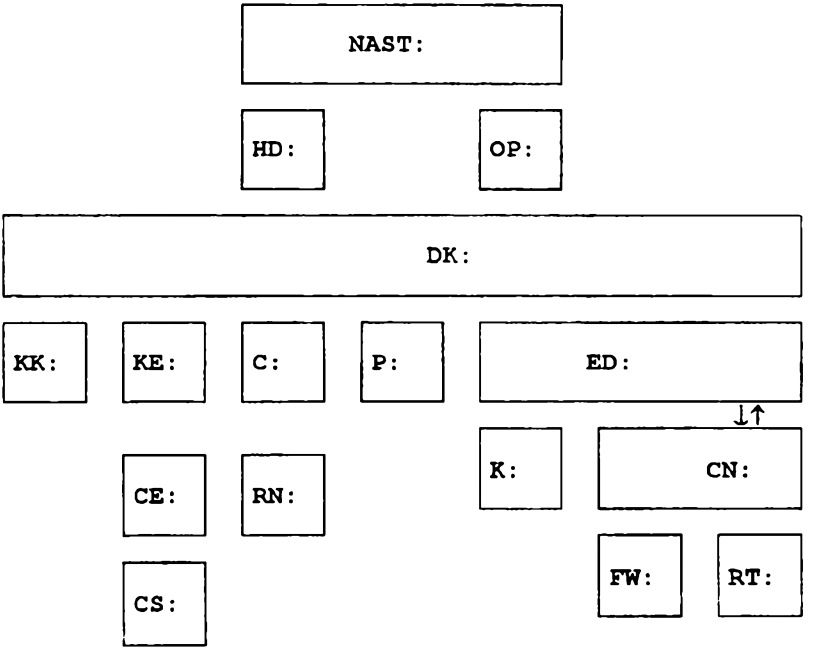


Таблица процедур

Процедура	Описание	Размер
NAST:	Задает параметры текущего курса	162 B
OP:	Открывает файл с УИ текущего курса	206 B
HD:	Задает номера секции, упражнения, справки и пр.; выводит "шапку" и финишные сообщения	807 B
DK:	Обслуживает клавиатуру и индикатор; поддерживает протокол работы в системе	1041 B
KK:	Срабатывает только при нажатии клавиши "EXE"	120 B
KE:	Срабатывает при ответе или при нажатии на "EXE"	228 B
C:	Очищает заданную часть индикатора	61 B
P:	Выводит номера сек.,упр.,спр. В нужной форме	61 B
ED:	Анализирует ответ и управляет выдачей справки, вспом. упражнения или упражнения на повторение	1016 B
CN:	Анализирует правильные ответы; подсчитывает отношение их числа к числу попыток; управляет продвижением по курсу	663 B
K:	Выбирает фрагмент УИ текущего упражнения	89 B
FW:	Разрешает продвижение вперед по курсу	246 B
RT:	Разрешает продвижение назад по курсу	333 B
CE:	Преобразует УИ упражнения из ФП в РП	436 B
CS:	Копирует текущее значение ФП УИ из поля в GU$	758 B
RN:	Выбирает случайным образом текущее упр-ие	306 B

Заключение

Программа испытывалась на специально составленном тест-курсе. На нем, в частности, оценивалось время реакции системы. Для описываемой реализации изначально представлялось не очевидным будет ли система интерактивной, поскольку выполнение программы на OPL происходит в режиме интерпретации, а микропроцессор HD6303X имеет низкую тактовую частоту (3,6864 Мгц). Удалось достигнуть того, чтобы время ожидания ответа не превышало 1 секунды. Наиболее узким местом оказалась "разборка" поля упражнения. Чтобы повысить скорость, пришлось отказаться от выделения повторяющихся частей CE: в отдельные процедуры. Это сократило время реакции втрое.

Объем подсистемы "Обучение" стартовой версии карманного "Наставника" составляет: тексты процедур на OPL - 3507 байт;

транслированные процедуры - 6533 байт. Программа "Наставника", состоящая только из транслированных процедур, занимает около четверти объема пользовательского ОЗУ. Управляющая информация кодируется также экономно. Так, чтобы разместить УИ в ФП для наибольшего из имеющихся на настоящий момент учебного курса "Базисный ФОРТРАН", в котором 36 секций [5], требуется 2,5КВ, что соответствует в среднем 70 байтам на секцию (допустимое значение 254 байта). Достаточное место для размещения штатных приложений Organiser-а (например, записной книжки) остается.

Проделанная работа продемонстрировала возможность создания автономного и доступного “Наставника” на основе недорогой тиражируемой техники и дала ответ на ряд вопросов: сколько для этого потребуется памяти, как следует ее организовать, при каком быстродействии микропроцессора система будет интерактивной и т.д.

Не менее важно и другое: большинство устройств, потенциально пригодных для модернизации под "Наставник", не имеют штатных средств программирования и оно возможно лишь при содействии фирмы-изготовителя. Убедить фирму оказать такое содействие трудно. Демонстрируя работающее устройство можно рассчитывать на благоприятное отношение.

Литература

1. Брусенцов Н.П., Маслов С.П., Рамиль Альварес Х. Микрокомпьютерная система обучения "Наставник" - М.: Наука, 1990. - 223с.

2. Брусенцов Н.П., Маслов С.П., Рамиль Альварес Х. Методическое пособие по разработке учебных материалов в микрокомпьютерной системе обучения "Наставник" - М.: Изд-во Моск.ун-та, 1992. - 95с.

3. PSION Organiser II-XP. Operating manual. 1994.

4. Н.П.Брусенцов, В.Б.Захаров, И.А.Руднев, С.А.Сидоров. Диалоговая система структурированного программирования ДССП-80. В сб. Диалоговые микрокомпьютерные системы. - М.: Изд-во Моск.ун-та, 1986. - С.3-21.

5. А.Л.Александров, Н.П.Брусенцов, Ю.Ю.Галимов, В.Ш.Кауфман, Н.Б.Лебедева, С.П.Маслов, Х.Рамиль Альварес. Базисный ФОРТРАН (Учебное пособие).- М.: Изд-во Моск. ун-та, 1982. - 200с.

А.В. Столяров

Расширенный функциональный аналог языка Рефал для мультипарадигмального программирования

Введение

Язык Рефал был предложен В.Ф.Турчиным в 1966 году[8]. В.Ш.Кауфман отмечает сходство изобразительных средств Рефала с нормальными алгоритмами Маркова[1]. Парадигму языка Рефал Кауфман называет *ситуационным программированием*, выделяя две ключевые абстракции - *анализ* исходной структуры данных и *синтез* результирующей структуры.

Рефал чрезвычайно удобен при решении задач, связанных с преобразованиями символьных данных. В.Ф.Турчин изначально позиционировал Рефал как *метаалгоритмический* язык[9] и позднее использовал его, в числе прочего, для изучения эквивалентных преобразований программ[10]. В.И.Сердобольский отметил удобство языка для работы с алгебраическими формулами[3]. С середины восьмидесятых годов и до настоящего времени Рефал успешно применяется в исследованиях, связанных с суперкомпиляцией и частичными вычислениями[11].

К настоящему времени существует несколько диалектов языка Рефал, среди которых наиболее динамично развиваются Рефал-5[12] и Рефал Плюс. Разработаны методы эффективной реализации языка; созданы системы программирования для различных программно-аппаратных платформ.

Несмотря на это, Рефал крайне редко применяется в индустриальном программировании (под индустриальным программированием мы понимаем прежде всего изготовление программных продуктов для конечного пользователя, находящегося за пределами компьютерной индустрии). При этом Рефал нельзя назвать неизвестным языком; многие профессиональные программисты с этим языком знакомы. По-видимому, основной причиной слабой востребованности языка является трудность его интеграции с другими языками (в частности, с языками, представляющими наиболее популярный в современной индустрии гибрид императивного и объектно-ориентированного программирования).

В статье [5] можно найти более подробное обсуждение проблемы межъязыковой (мультипарадигмальной) интеграции и предварительное сообщение о методе, основанном на моделировании альтернативных парадигм как предметных областей средствами

базового объектно-ориентированного языка (на примере языков Си++[7] и Лисп). Детальное описание экспериментальной реализации метода (библиотеки классов Си++ InteLib, моделирующей алгебру S-выражений языка Лисп) дается в работе [6]. Там же приводятся соображения по поводу моделирования средствами языка Си++ парадигмы языка Рефал.

Целью настоящей статьи является описание результатов экспериментальной реализации объектно-ориентированной модели парадигмы языка Рефал.

Краткий обзор метода. Библиотека InteLib

Библиотека InteLib, первоначально ориентированная на импорт в Си++-проекты парадигм языка Лисп, основана на моделировании алгебры S-выражений, представленной в языке Лисп, с помощью иерархии полиморфных классов, для которых переопределяются некоторые стандартные операции Си++ таким образом, чтобы обеспечить удобное для программиста отображение привычных Лисп-конструкций на синтаксис, допустимый в Си++-коде.

Все S-выражения представляются в InteLib объектами некоторых классов, являющихся потомками общего базового класса LTerm (это имя возникло на ранней стадии проектирования библиотеки и сохранено в силу исторических причин). Так, S-выражение типа "<строковая константа"> представляется объектом класса LTermString, лисповский символ (в терминологии, традиционной для стандарта Common Lisp[4]) или идентификатор (в терминах, принятых в языке Scheme[2]) моделируется объектом класса LTermSymbol, точечная пара - объектом класса LDotPair и т.п. Для доступа к объектам иерархии LTerm, существующим всегда в динамической памяти, используются объекты класса LReference. Класс LReference по своим свойствам аналогичен простому указателю, в дополнение к чему исполняет функции сборщика мусора.

Для построения лисповских списков, состоящих из точечных пар, используются две базовые операции - создание списка из одного (первого) элемента и добавление нового элемента в конец. Для введения первой операции используется класс LListConstructor с перекрытой операцией |; в качестве символа для обозначения второй операции из соображений наглядности выбрана двухместная операция "<запятая">, переопределенная в классе LReference. Класс LReference имеет конструкторы преобразования для всех стандартных встроенных типов языка Си++, что, в сочетании с механизмом неявных преобразований, дает возможность использования, например, следующих конструкций:

```
(L| 25, 36, 49)              // (25 36 49)
(L| "I am the walrus", 1965)  // ("I am the walrus" 1965)
```

```
(L| 1, 2, (L| 3, 4), 5, 6)      // (1 2 (3 4) 5 6)
(L| (L| 1, 2), 3, 4)            // ((1 2) 3 4)
```

Для удобства построения точечных пар и точечных списков применяется операция ||, заменяющая метку конца списка (символ NIL) в левом операнде на правый операнд. В роли лисповского сокращения для формы (QUOTE A) (апостроф) используется операция ~ (тильда). Приведем еще несколько примеров:

```
(L| 1 || 2)                  // (1 . 2)
((L| 1, 2, 3)|| 4)           // (1 2 3 . 4)
(L| MEMBER, 1, ~(L| 1, 3, 5))   // (member 1 '(1 3 5))
```

Необходимо пояснить, что L - это имя объекта класса LListConstructor, MEMBER - это имя объекта LReference, ссылающегося на объект класса LTermSymbol, и т.д.

Наконец, для прямого построения точечной пары из левого и правого значения используется операция ^.

Класс LTerm имеет виртуальную функцию Evaluate(), производящую вычисление соответствующего выражения в смысле Лиспа. Для констант (числовых, строковых и константных символов) метод Evaluate возвращает ссылку на исходное выражение. Для классов, наследуемых от LTerm и моделирующих другие (неконстантные) типы S-выражений, функция Evaluate() соответствующим образом перекрывается.

Рассмотрим для примера функцию на языке Лисп, определяющую, имеют ли два дерева изоморфную структуру (два дерева считаются изоморфными тогда и только тогда, когда они отличаются только значениями в листьях):

```
(defun isomorphic (tree1 tree2)
 (cond ((atom tree1) (atom tree2))
     ((atom tree2) NIL)
     (t (and (isomorphic (car tree1) (car tree2))
          (isomorphic (cdr tree1) (cdr tree2))))))
```

Если считать, что символы, соответствующие стандартным функциям Лиспа, уже описаны соответствующими объектами класса LSymbol, то можно записать на Си++ код, эквивалентный вышеприведенному:

```
#include "intelib.h"
LSymbol ISOMORPHIC("ISOMORPHIC");
void LispInit_isomorphic() {
 static LSymbol TREE1("TREE1");
 static LSymbol TREE2("TREE2");
```

```
(L|DEFUN, ISOMORPHIC, (L|TREE1, TREE2),
 (L|COND, (L|(L|ATOM, TREE1), (L|ATOM, TREE2)),
       (L|(L|ATOM, TREE2), NIL),
       (L|T, (L|AND,
       (L|ISOMORPHIC, (L|CAR, TREE1), (L|CAR, TREE2)),
       (L|ISOMORPHIC, (L|CDR, TREE1), (L|CDR, TREE2))))
 )).Evaluate();
}
```

Следует особо отметить, что данный фрагмент кода является кодом на языке C++, не требующим какого-либо дополнительного препроцессирования. Таким образом, не выходя за пределы языка C++, мы получаем возможность применения парадигм языка Lisp в C++-проекте.

Моделирование рефал-выражений с помощью лисповских списков

Древовидные данные, состоящие из строк терминальных символов, символов-меток, символов-чисел и структурных скобок (то есть выражения, допустимые на входе в рефал-функцию), очевидным образом изоморфны лисповским деревьям, состоящим из строковых констант, символов и чисел. Например, рефальское выражение

```
'abc' symb ('def' 25)
```

соответствует в смысле рассматриваемого изоморфизма лисповскому

```
("abc" symb ("def" 25))
```

Необходимо отметить, что изоморфизм является достаточно условным. На самом деле, рефальские объектные выражения изоморфны подмножеству лисповских S-выражений, а именно - множеству списков, в которых нигде не встречается двух символьных строк подряд. В реализованной версии библиотеки атомарные S-выражения и точечные списки считаются не соответствующими никаким выражениям языка Рефал и при попытке использования в качестве таковых вызывают ошибку. Что касается списков с двумя и более строковыми константами подряд, то такие списки считаются эквивалентными спискам, в которых следующие подряд константы заменены на их конкатенацию.

Рассмотренный изоморфизм позволяет использовать для представления рефальских выражений уже существующие средства, разработанные для Лиспа. Приведенное выше выражение можно представить на Си++ как

(L|"abc", SYMB, (L|"def", 25))

Для представления рефальских угловых скобок, обозначающих вызовы функций (так называемые "<активные списки">), введем новый класс RfCall, наследуемый от LTerm и инкапсулирующий обычный список. Для построения таких активных списков введем класс, аналогичный описанному ранее LListConstructor. Назовем его, например, RfListConstructor и опишем его единственный экземпляр с именем R. Также введем унаследованный от LReference класс RfCallReference, предназначенный для работы с активными списками. Основное отличие RfCallReference от LReference состоит в измененной семантике операции "<запятая">, которая в данном случае предполагает, что левым операндом является не списочное S-выражение, а выражение типа RfCall. Соответственно, добавление нового элемента производится к инкапсулированному в него списку.

Теперь рефальское `<fun1 'a' sym1 'b'>` может быть представлено в виде `(R|FUN1,"a",SYM1,"b")`.

Для представления рефальских переменных опишем специальный класс RfVariable, также наследуемый от LTerm. Строго говоря, никакими свойствами терма рефал-переменная не обладает, и такое наследование преследует единственную цель - позволить использовать для построения правил преобразования уже существующие средства.

Класс RfVariable имеет методы, используемые в процессе сопоставления выражений с образцами. Сам класс RfVariable является абстрактным, а переменные конкретных типов описываются с помощью наследуемых от него подклассов.

Путем введения дополнительных классов - наследников LReference и перекрытия для этих классов операций () (штатно обозначает вызов функции) и [] (штатно используется для адресации в массиве) можно дать достаточно наглядные средства для описания рефал-функций. Например, функция "<палиндром">, имеющая на Рефале-5 вид

```
Pal {          = True;
    s.1        = True;
    s.1 e.2 s.1 = <Pal e.2>;
    e.1        = False; }
```

может быть представлена на Си++ как

```
LSymbol PAL("PAL");
LSymbol True("True");
LSymbol False("False");
RfVariable_E e_1("e.1"), e_2("e.2");
RfVariable_S s_1("s.1");
```

```
// ...
RFUNC(PAL) [(L)          ^ (L| True)]
      [(L| s_1)      ^ (L| True)]
      [(L| s_1. e_2, s_1) ^ (L| (R| PAL, e_2))]
      [(L| e_1)      ^ (L| False)] ;
```

Стратегия вычисления рефал-выражений существенно отличается от стратегии вычисления S-выражений в смысле Лиспа. Например, результатом вычисления простого списка в смысле Лиспа будет применение функции, заданной первым элементом списка, к остатку списка как к списку фактических параметров (обычно подлежащих, в свою очередь, вычислению перед передачей в функцию), тогда как результатом вычисления того же списка в смысле Рефала будет простая конкатенация результатов вычисления всех элементов списка (опять таки, в смысле Рефала). Поэтому для вычисления S-выражений в смысле Рефала применяется не метод Evaluate(), введенный в классе LTerm, а внешняя по отношению к иерархии LTerm функция, названная RefalMachineCall.

Кроме того, пользователю доступны функции RefalMatch (простое рефальское сопоставление с образцом), RefalReplaceVars (подстановка значений вместо переменных в выражении) и RefalSimplifyView (нормализация рефал-выражения). Под нормализацией понимается замена строковых констант, идущих в списке подряд, на их конкатенацию, так что в результирующем выражении нигде не встречается подряд двух строковых констант. В ситуации мультипарадигмального программирования, т.е. в условиях, в которых рефал-конструкции являются одним из возможных, но не единственным способом обработки данных, применение таких функций, реализующих частные возможности рефал-машины, зачастую бывает очень удобно.

Рефал-переменные. Возможность расширения их функциональности

Под переменной в Рефале понимается специфическая атомарная составляющая выражения-образца, которой может ставиться в соответствие любое рефал-выражение из некоторого множества.

В Рефале-5 существуют переменные трех типов (S-переменные, T-переменные и E-переменные). S-переменной может быть поставлен в соответствие любой символ (в том числе символ-метка или символ-число). Если воспользоваться введенным выше изоморфизмом, можно говорить, что S-переменная ставится в соответствие символу внутри строковой константы, лисп-символу или числовой константе. T-переменной может быть поставлено в соответствие, кроме вышеперечисленного, любое выражение, заключенное в скобки.

Нзконец, E-переменная может соответствовать любому выражению, в котором соблюден баланс скобок, в том числе пустому. Несложно видеть, что при выполнении операции сопоставления выражения с образцом (например, слева направо, как это делается в Рефале-5), для встреченных на очередном шаге S- и T-переменных можно сразу определить, возможно ли сопоставление вообще и если да, то какое именно выражение должно быть сопоставлено данной переменной. Если же в выражении встречается переменная типа E, в общем случае необходимо последовательно рассмотреть разные варианты сопоставления, для чего может понадобиться алгоритм поиска решения с возвратами (backtracking).

В более ранних версиях Рефала можно было указать ограничения на выражения, которые могут быть сопоставлены данной переменной (в данном образце). Так, образцу `'ab' s('cd')1 'ef'` могли ставиться в соответствие только две строки - `'abcef'` и `'abdef'`. В современные версии Рефала эти возможности не вошли, видимо, из соображений элегантности языка. Дополнительные условия на значения переменных в Рефале-5 могут быть заданы с помощью так называемых "< *где*-предложений"> (*where* clauses), что по своим возможностям превосходит механизм спецификаторов ранних версий Рефала, т.к. позволяет накладывать на переменные совершенно произвольные ограничения. Каково бы ни было ограничение на переменные `v.1, v.2, ..., v.n`, всегда возможно описать на Рефале характеристическую функцию-предикат вида `<my_fpred (v.1)(v.2)...(v.n)>`, которая возвращала бы, например, значение `true` для допустимой комбинации значений переменных и `false` в противном случае. Тогда конструкция типа where вида `,<my_fpred (v.1)(v.2)...(v.n)> : true` введет в рефал-предложение требуемое ограничение. Универсальность механизма следует из алгоритмической полноты языка Рефал.

Для программ, составляемых целиком на Рефале, такой механизм, безусловно, предпочтительнее. Однако в ситуации, когда основным языком проекта является Си++ или другой объектно-ориентированный язык, а от Рефала используются только изобразительные парадигмы, возможность введения дополнительных типов рефал-переменных может повысить наглядность, а в некоторых случаях - и эффективность составляемых программ.

В связи с этим библиотека InteLib делает минимально возможное количество предположений относительно выражений, которые могут быть сопоставлены рефал-переменной. Переменные делятся на две категории - однозначные и многозначные. В базовой библиотеке к первой категории относятся S- и T-переменные, ко второй - E-переменные. Обработка переменных второй категории отличается

тем, что в ситуации, когда такая переменная встречается в открытой позиции (то есть за ней не следует непосредственно конец выражения или закрывающая скобка), процедура сопоставления организует перебор вариантов. Количество возможных вариантов, как и сами эти варианты, полностью определяется виртуальными функциями класса `RfVariable_E` (потомка класса RfVariable, представляющего E-переменные).

Таким образом, при необходимости можно описать дополнительные классы рефал-переменных. Например, можно ввести переменные, в соответствие которым ставится любая цепочка символов, допустимая в качестве имени идентификатора (в виде многозначной рефал-переменной). Более того, в случае, если в анализируемом языке имеется требование, что в качестве имени идентификатора всегда берется самая длинная из допустимых цепочек (в большинстве современных языков программирования это так и есть), то грамматическое понятие "<идентификатор"> может быть описано в виде однозначной переменной (то есть переменной, не допускающей вариантного сопоставления), что, безусловно, повысит эффективность программы и упростит реализацию лексического анализа.

Конструкции расширенного Рефала-5 и их моделирование

В описании Рефала-5 [12] под "<средствами расширения"> языка понимаются уже упоминавшаяся where-конструкция, или *условие*, а также with-конструкция, или *блок*.

При интерпретации where-конструкции производится вычисление некоторого рефал-выражения и попытка сопоставления результата с заданным образцом, которая может окончится успехом или неудачей. Возможно, что некоторые не связанные до выполнения конструкции переменные получат значения в процессе ее выполнения. В случае успеха производися обычная подстановка переменных в правой части предложения. В случае неудачи производится откат до ближайшей точки ветвления, если такая есть, или фиксируется неудача сопоставления для всего предложения.

With-конструкция фактически заменяет собой правую часть. При успешном сопоставлении аргумента функции с образцом предложения, содержащего with-конструкцию, вместо обычной подстановки производится вычисление аргумента with-конструкции; результат вычисления передается в безымянную рефал-функцию, заданную телом with-конструкции в виде обычного блока, состоящего из одного и более рефал-предложений.

В рассматриваемой объектно-ориентированной модели Рефала обе эти конструкции реализованы как частные случаи обобщенных

возможностей, реализованных, соответственно, абстрактными классами RfLeftSubclause и RfRightSubclause.

Один или несколько объектов класса RfLeftSubclause могут быть помещены в конец образца, т.е. левой части рефал-предложения, что является оправданием названия класса. Когда в процесе сопоставления обнаруживается объект этого класса, считается, что образец закончен (то есть остаток сопоставляемого выражения должен быть к этому моменту пуст). Если сопоставление было успешным, производится вызов метода RfLeftSubclause::Process, которому передается текущий контекст сопоставления (совокупность связей между рефал-переменными и их значениями). Метод возвращает true для индикации успеха, false - для индикации неудачи.

Объект класса RfRightSubclause может быть помещен в рефал-предложении на место правой части. В этом случае вместо обычной процедуры подстановки значений переменных происходит вызов метода RfRightSubclause::Process. Метод возвращает рефал-выражение, которое и используется в качестве результата работы всей функции.

Собственно конструкции where и with реализованы в виде классов RfWhereClause и RfWithClause, наследуемых от RfLeftSubclause и RfRightSubclause соответственно.

Следует заметить, что обычную процедуру подстановки значений переменных базисного Рефала тоже можно реализовать в виде наследника класса RfRightSubclause. Этим возможности обобщенного механизма, разумеется, не исчерпываются.

Введенные обобщения позволяют существенно расширить изобразительные возможности рассматриваемой модели по сравнению с исходным языком Рефал-5.

Лисп-функции и их вызов из рефал-функций

Реализация функционального аналога языка Рефал в виде надстройки над функциональным аналогом языка Лисп логично приводит к мысли об использовании уже наработанных для лисповской части InteLib библиотечных функций, соответствующих встроенным функциям языка Лисп.

При этом следует помнить о различии между множеством всех S-выражений и множеством S-выражений, допустимых как представление рефал-выражения. Кроме того, необходимо учитывать различие семантик двух языков. Проиллюстрируем возникающую проблему на примере. Рассмотрим рефал-выражение

<CAR (1 2 3)><CAR (10 20 30)><CAR (100 200 300)>

где CAR обозначает стандартную Лисп-функцию. Логично ожидать, что результатом вычисления такой конструкции должно стать рефал-выражение `1 10 100`, или, иначе говоря, S-выражение `(1 10`

100). Однако не все так просто. Первый активный вызов, если не принять специальных мер, вернет S-выражение 1, второй - 10, третий - 100. Все три, следует отметить, не являются допустимыми в качестве рефал-выражения, что уже означает возникновение ошибочной ситуации. Но даже если бы все три результата случайно оказались принадлежащими допустимому множеству, - например, если бы исходный вызов выглядел как

<CAR ((1) 2 3)><CAR ((10) 20 30)><CAR ((100) 200 300)>

то даже в этом случае полученный результат несколько отличался бы от того, чего мы ожидали ((1 10 100) вместо ((1)(10)(100)).

Причина этого в семантическом различии интерпретаций двух языков. В Рефале, в отличие от Лиспа, результат вычисления конкатенации нескольких выражений есть *конкатенация* результатов, то есть, в терминах Лиспа, результат применения функции APPEND к полученным результатам, тогда как в Лиспе в аналогичной ситуации традиции подсказывают ожидать чего-то похожего на результат функции MAP (список, построенный из результатов).

Преодолеть проблему можно, введя возможность помечать особым образом те активные (построенные с помощью класса RfListConstructor) списки, которые вызывают лисповскую функцию. При вычислении такого списка полученный результат, прежде чем быть возвращенным вызывающей функции, инкапсулируется в список из одного элемента. Для этого в классе RfListConstructor предусмотрен дополнительный оператор ||, который делает то же, что и введенный ранее оператор |, отличаясь от него только выставлением соответствующего признака у порождаемого активного списка.

Непосредственное сочетание языков Лисп и Рефал

Описанная выше объектно-ориентированная модель рефал-машины в сочетании с реализованной ранее моделью алгебры лисповских S-выражений приводит к идее создания гибрида языков Лисп и Рефал.

В работах [5,6] описан транслятор усеченного диалекта языка Лисп, названного InteLib Lisp. Транслятор генерирует на выходе модуль, написанный на языке Си++, предназначенный для использования с библиотекой InteLib. Отмечается, что устройство этого транслятора чрезвычайно примитивно в силу подобия получаемых конструкций Си++ и исходного кода на Лиспе.

Аналогичный транслятор может быть создан для языка Рефал (например, для диалекта Рефал-5, все возможности которого имеют адекватную модель в библиотеке). Очевидно, при построении такого

транслятора необходимо предусмотреть возможность использования обобщенных рефал-переменных, а также обобщений конструкций with и where (естественно предположить, что сами обобщения реализуются на Си++, и в конструкциях Рефала должен быть предусмотрен только некоторый интерфейс к таким обобщениям). Следует ожидать, что сложность построения такого транслятора также будет невысока, что позволяет рассматривать его как побочный продукт разработки библиотеки InteLib.

Наконец, для построения транслятора для гибрида Лиспа и Рефала достаточно разработать синтаксические соглашения. Можно ожидать, что построение транслятора с такого гибридного языка также не составит проблемы и не окажется ресурсоемким.

Заключение

Предложенная в статье объектно-ориентированная модель расширенной рефал-машины позволяет, не выходя за рамки языка Си++ и используя, соответственно, существующие системы программирования без специальных модификаций, применять в проектах изобразительные парадигмы, наработанные в языке Рефал, а при необходимости и расширять возможности этих парадигм.

В сочетании с разработанной ранее моделью алгебры S-выражений языка Лисп, предложенная модель порождает среду для мультипарадигмального программирования в рамках проектов, основным языком которых является Си++.

Поскольку применение предложенных методов не требует ни модификации существующих систем программирования, ни внедрения новых, барьер внедрения для предложенной методологии оказывается чрезвычайно низок и сводится к освоению библиотеки классов InteLib, имеющей достаточно простую структуру. Это позволяет ожидать, что предложенный метод может найти применение в индустриальном программировании.

Литература

1 Кауфман В.Ш. Языки программирования. Концепции и принципы. М.: Радио и Связь, 1993.

2 Kelsey R. at al. Revised report on Algorithmic Language Scheme.

3 Сердобольский В.И. Язык РЕФАЛ и его использование для преобразования алгебраических выражений. // Кибернетика. 1969. N 3. Стр. 45-51.

4 Steele G. L. Common Lisp the Language, 2nd edition. Digital Press, 1990.

5 Stolyarov A., Bolshakova E., Building Functional Techniques into an Object-oriented System. // Knowledge-Based Software Engineering

(proceedings of the 4th JCKBSE, Brno, Czech Republic, 2000), IOS Press, 2000. p.101-106

6 Столяров А. Интеграция изобразительных средств альтернативных языков программирования в проекты на C++. Москва, 2001. *Рукопись деп. в ВИНИТИ РАН*

7 Stroustrup B. The C++ Programming Language, 3rd edition. Addison-Wesley. 1997.

8 Турчин В.Ф. Метаязык для формального описания алгоритмических языков. // Цифровая вычислительная техника и программирование. М.: Сов. радио, 1966. Стр. 116-124

9 Турчин В.Ф. Метаалгоритмический язык. // Кибернетика. 1968. N 4. Стр. 45-54.

10 Турчин В.Ф. Эквивалентные преобразования программ на Рефале // Автоматизированная система управления строительством. М.: ЦНИПИАСС, 1974. Стр. 36-68.

11 Turchin V.F. The concept of a supercompiler // ACM Transactions on Programming Languages and Systems. 1986. Vol. 8, N 3. Pg. 292-325.

12 Turchin V. REFAL-5, Programming Guide and Reference Manual. New England Publishing Co., Holyoke, 1989.

Раздел IV
Сообщения

Новиков М.Д.

Среда dBASE как средство создания автоматизированных систем обработки анкет

Введение

В статье описывается структура и возможности двух автоматизированных информационных систем: «Международные научно-технические связи» (далее кратко – «МНТС») и «Иностранные студенты». Эти системы в настоящее время находятся в эксплуатации в иностранном отделе факультета ВМК МГУ. Система «МНТС» предназначена для учёта сотрудников факультета, командированных за рубеж и иностранных специалистов, находящихся на факультете ВМК. Система «Иностранные студенты» предназначена для обработки анкетных данных и сведений об успеваемости иностранных студентов, обучающихся на факультете ВМК.

Обе системы созданы для IBM-совместимых ПЭВМ в среде dBASE-подобных СУБД [1]. Родоначальником этих СУБД явилась система dBASE-II, созданная фирмой Ashton-Tate в 80-х г.г. Затем были созданы системы dBASE-III, dBASE-III-Plus, FoxBASE+, Clipper и, наконец, FoxPro. Система Clipper является компилятором, а остальные системы – интерпретаторами. Система FoxPro выделяется высокими скоростными характеристиками и обладает большим набором команд и функций, с помощью которых можно создавать программы, отвечающие всем современным требованиям, предъявляемым к дружественному интерфейсу «ЭВМ – пользователь» и к обработке данных. Система «МНТС» создана в среде FoxBASE+, «Иностранные студенты» – в среде FoxPro 2.6. Заметим, что программы, работающие в среде FoxBASE+, могут работать и в среде FoxPro.

Сотрудничество автора с иностранным отделом факультета ВМК началось в 1990 г. В 1992 г. было завершено создание системы «МНТС». В этом же году система «МНТС» начала эксплуатироваться и затем неоднократно совершенствовалась и дополнялась новыми модулями. В 1993 г. была создана первая версия системы «Иностранные студенты». Однако практическая работа с этой системой показала необходимость её существенной переработки, и в середине 2000 г. была создана новая версия системы, которая эксплуатируется с сентября 2000г.

1. Система «МНТС»

Система «МНТС» состоит из трёх подсистем: 1)*Командирование*, 2)*Приём*, 3)*Генерация отчётов*. Эти подсистемы предназначены, соответственно, для обработки анкет сотрудников факультета ВМК, командируемых за рубеж, для обработки анкет иностранных специалистов, находящихся на факультете ВМК и для формирования и вывода таблиц, содержащих те или иные сведения об учитываемых лицах. К моменту начала эксплуатации системы «МНТС» были созданы только подсистемы *Командирование* и *Приём*; подсистема *Генерация отчётов* была создана позже, когда выявилась необходимость вывода специальных форм отчётов.

Любая анкета учитываемого лица состоит из совокупности полей (см. рис.1 и 2). Значением поля анкеты может быть число, дата, произвольный текст или слово из списка допустимых значений. Например, анкета сотрудника, командированного за рубеж, содержит, в частности, следующие поля: 1)Фамилия, имя и отчество (значение – произвольный текст), 2)Место работы (значение – слово из списка), 3)Процент сохранения зарплаты (значение – число) и 4)Дата выезда (значение – дата).

Фамилия имя и отчество	Иванов Алексей Иванович
Место работы (кафедра)	МФ
Должность	Доцент
Страна командирования	Германия
Длительность командировки	14 дней
Тип визита	Международная конференция
Основание командировки	Приглашение
Линия командирования	МГУ
Принимающая организация	Технический университет
Оплата пребывания за границей	За счет принимающей стороны
Оплата транспортных расходов	За счет принимающей стороны
Дата представления документов в УВС МГУ	07.06.1998
Номер приказа о командировании	9999 млк от 30.08.1998
Процент сохранения зарплаты	100
Дата высзда	20.09.1998
Дата возвращения	03.10.1998
Дата представления отчета о командировке	11.10.1998

Рис.1. Форма анкеты «Командирование»

Фамилия и имя	Марко Кирстен
Дата рождения	20.07.1958
Место работы	Технический университет
Должность	н.с.
Страна откуда прибыл	Нидерданды
Длительность пребывания	14 дней
Тип визита	Научная работа
Принимающая кафедра	СП
Ответственный за прием	Иванов Алексей Иванович
Адрес проживания в Москве	ДС МГУ Б-1111
Паспорт (номер дата выдачи)	99999999999
Дата представления документов в УВС	06.06.1999
Номер приказа о приеме	1000е от 19.09.1999
Дата приезда	26.09.1999
Дата отъезда	10.10.1999
Дата представления отчета о приеме	17.10.1999

Рис.2. Форма анкеты «Приём»

Подсистемы *Командирование* и *Приём* состоят из пяти основных модулей: 1)Ввод новых анкет, 2)Изменение анкет, 3)Вывод анкет, 4)Удаление анкет, 5)Системный модуль. При вводе и изменении анкет производится контроль правильности вводимой информации (проверяется принадлежность дат и чисел заданному диапазону, а также наличие значения поля анкеты в списке). В модуле «Вывод анкет» реализован общий отбор анкет по значениям полей и вывод полей анкет на экран дисплея, на принтер или в файл на диске для последующей обработки. Например, можно вывести фамилии всех лиц, командированных от какой-либо кафедры или лаборатории, анкеты лиц, командированных за последние 5 лет и т.д. Системный модуль позволяет менять списки допустимых значений для полей анкеты, устанавливать и отменять пароли и т.п.

Подсистема *Генерация отчётов* предназначена для создания и вывода отчётов. Реализованы следующие формы отчётов: 1)Вывод сведений о международных связях кафедр; 2)Вывод структуры международных связей факультета ВМК; 3)Вывод анкет сотрудников факультета, находящихся за рубежом; 4)Вывод анкет сотрудников факультета, превысивших срок командирования; 5)Вывод анкет сотрудников факультета, не представивших отчет; 6)Вывод сведений о командировании за определенный период времени; 7)Вывод анкет

иностранных специалистов, находящихся на факультете; 8)Вывод анкет иностранных специалистов, превысивших срок пребывания в МГУ; 9)Вывод анкет иностранных специалистов, для которых нет отчета о приёме; 10)Вывод сведений о приеме за определенный период времени. Эти же отчёты можно получить и при помощи модуля «Вывод анкет» подсистем *Командирование* и *Приём* с последующим редактированием файлов (вставка заголовка и проч.), однако данная подсистема обеспечивает более быструю реализацию наиболее часто встречающихся запросов.

Подсистемы *Командирование* и *Приём* реализованы одними и теми же программными модулями, различается лишь форма анкеты. Особенностью системы «МНТС» является то, что форму анкеты можно менять, корректируя лишь записи в файлах базы данных, не меняя программы (в базе данных есть специальные файлы, содержащие количество форм анкет и форму каждой анкеты). Таким образом, систему «МНТС» можно легко приспособить для обработки других анкет; важно лишь, чтобы форма анкеты укладывалась в реализованный формат.

Программные файлы системы «МНТС» занимают объём около 110 Кб. К марту 2001 г. в базу данных были введены 832 анкеты лиц, командированных за рубеж и 147 анкет лиц, принятых из-за рубежа; объём файлов базы данных составил около 380 Кб. В архивированном виде система «МНТС» вместе с базой данных и средой FoxBASE+ занимает объём около 500 Кб, т.е. умещается на одной дискете.

Время обработки анкет следующее. На ПЭВМ Pentium с частотой 300MHz общий вывод 832 анкет в файл занимает около 1 минуты, формирование любого специального отчёта занимает не более 10 сек. Зависимость времени обработки от количества анкет – линейная.

Система «МНТС» используется иностранным отделом факультета ВМК следующим образом. 1)Представляются ежегодные отчёты о командировании и приёме в ректорат МГУ, деканат факультета ВМК МГУ, бухгалтерию и отдел кадров. 2)Осуществляется контроль за командированием и приёмом со стороны самого иностранного отдела.

2. Система «Иностранные студенты»

Система «Иностранные студенты» состоит из двух подсистем: *Анкеты студентов* и *Оценки студентов*. Первая подсистема основана на тех же принципах, что и подсистемы *Командирование* и *Приём* системы МНТС: можно вводить, изменять, удалять анкеты, выводить их на экран или в файл, устанавливать и отменять пароли, корректировать списки допустимых значений полей анкеты. Вторая подсистема позволяет вводить учебные планы и оценки студентов. Она состоит из следующих модулей: 1)Шаблоны; 2)Учебные планы; 3)Оценки;

4)Отчёты. Для иностранных студентов характерно то, что учебный план каждого студента в каждом семестре индивидуален: есть предметы, общие для большинства студентов и предметы, которые присутствуют в учебном плане лишь отдельных студентов. У разных студентов могут быть также разные формы отчётности по тому или иному предмету и не совпадать сроки сдачи зачётов и экзаменов.

Модуль «Шаблоны» позволяет вводить шаблоны учебных планов – предметы, которые присутствуют в учебном плане большинства студентов некоторой группы или курса. Всего может быть задано 140 различных шаблонов. Модуль «Учебные планы» позволяет вводить индивидуальный учебный план студента в каждом семестре; обычно базовый набор предметов копируется из шаблона, а затем учебный план корректируется для конкретного студента. Модуль «Оценки» позволяет вводить оценки студентов, а модуль «Отчёты» – формировать и выводить различные сведения о студентах (список неуспевающих, сроки сдачи сессии и проч.)

Принципиально новым моментом в системе «Иностранные студенты» по сравнению с системой «МНТС» является то, что вывод любой информации в файл производится с расчётом на последующую работу с файлом в среде текстового редактора Word [2]. По запросу из базы данных вначале формируется файл в формате txt и производится запуск текстового редактора Word. Затем с помощью макрокоманды текст преобразуется к нужному формату: изменяются шрифты, форматируются абзацы, формируются таблицы и проч. Можно сформировать девять документов: 1)Личный листок; 2)Сведения об окончании срока действия паспорта; 3)Сведения об окончании срока регистрации паспорта; 4)Сведения об окончании срока оплаты обучения; 5)Сведения об окончании срока медицинской страховки; 6)Формируемая таблица; 7)Учебные планы; 8)Успеваемость; 9)Расчёт рейтинга. Первые 6 документов относятся к подсистеме *Анкеты студентов*, а последние 3 – к подсистеме *Оценки студентов*; для их формирования создано шесть макрокоманд на языке Visual Basic [3] (каждая макрокоманда формирует свой документ, а все документы об окончании сроков формируются одной и той же макрокомандой).

Опишем подробнее некоторые документы. Так, документы 2, 3, 4 и 5 выдают список студентов, у которых срок действия паспорта, регистрации паспорта, оплаты и медицинской страховки, соответственно, истекает ранее заданной даты. Формируемая таблица получается аналогично документу из блока «Вывод анкет» системы «МНТС»: задаётся критерий отбора анкет, поля анкеты, необходимые для вывода, как упорядочивать анкеты и форма вывода – в столбец или в строку. Документ «Успеваемость» выдаёт список всех студентов с указанием академической задолженности каждого студента; учитываются индивидуальные сроки сдачи сессии. Что касается

рейтинга, то можно вычислить: общий рейтинг, рейтинг по курсу, рейтинг по семестру и рейтинг выпускника. Общий рейтинг вычисляется по всем предметам, рейтинг по курсу – по предметам курса, рейтинг по семестру – по предметам семестра и рейтинг выпускника – по предметам, идущим в диплом. Формула следующая:

$$(T_1*O_1+ T_2*O_2+\ldots+ T_n*O_n)/(T_1+T_2+\ldots+T_n),$$

где T_i - количество часов, за которые изучается i-й предмет, а O_i – оценка по этому предмету.

Примеры форм, получаемых с помощью системы «Иностранные студенты», приведены на рис.3 и 4.

Программные файлы системы «Иностранные студенты» занимают объём около 100 Кб. В базе данных планируется держать анкеты и оценки нескольких сотен студентов; объём базы данных составит ориентировочно несколько сотен Кб (к моменту написания статьи заполнение базы данных ещё не завершено). В архивированном виде система «Иностранные студенты» вместе с базой данных и средой FoxPro занимает объём около 1150 Кб, т.е. умещается на одной дискете.

В настоящее время идёт опытная эксплуатация системы «Иностранные студенты». Предполагается, что с её помощью будут формироваться документы для личных дел студентов (например, личный листок, учебный план) и осуществляться контроль за студентами (успеваемостью, оплатой обучения и проч.) со стороны иностранного отдела.

Заключение

Почти десятилетняя эксплуатация системы «МНТС» показала её высокую надёжность и эффективность. В ближайшее время предполагается дополнить систему модулями контроля и вывода суммарного времени пребывания каждого сотрудника факультета ВМК за границей в течение года и за 5 лет. Что касается системы «Иностранные студенты», то в процессе её опытной эксплуатации предполагается выявить и исправить возможные недоработки. В дальнейшем планируется создать подсистему «Выпускники», в которой будут храниться сведения об иностранных студентах, закончивших факультет ВМК.

Литература

1. А.А. Попов. FoxPro 2.5/2.6. – М.: Изд-во «ДЕССКОМ», 2000.
2. А. Хомоненко. MS Word 97. - Изд-во «BHW-Санкт-Петербург», 1998.
3. Руководство программиста по Visual Basic для Microsoft Office 97. – М.: Издательский отдел «Русская редакция», 1997.

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени М.В.ЛОМОНОСОВА
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ
ОТДЕЛЕНИЕ ПО РАБОТЕ С ИНОСТРАННЫМИ УЧАЩИМИСЯ

ЛИЧНЫЙ ЛИСТОК УЧАЩЕГОСЯ № 512134

Фамилия и имя	Ван Вэй	
Пол	Мужской	
Страна	КНР	
Дата рождения	21.06.1981	
Место рождения	КНР, Шанхай	
Номер национального паспорта	111111111	до 10.08.2004
Постоянный адрес		
Семейное положение	Холост	
Образование и год его получения	Среднее,1999 ЦМО МГУ,2000	
Год поступления В МГУ, номер приказа	2000, Приказ N 1111 от 24.09.2000	
Форма обучения	Дневная	
Курс	1	
Учебная группа	К133	
Начало обучения	01.09.2000	
Конец обучения	15.07.2004	
Программа обучения	Бакалавр	
Кафедра		
Линия прибытия	Индивидуальный контракт	
Номер контракта	N 111111/0	
Стоимость обучения	2500 у.е. по семестрам	
Стипендия	без выплаты стипендии	
Оплата обучения		до 01.10.2001
Регистрация паспорта	Р33/3333/33	до 18.03.2001
Адрес проживания в Москве	ДС МГУ,Б-111	
Приказы о переводе на следующий курс		
Медицинское страхование		
ВИЧ-тест	27.08.2000	
Приказ об окончании		
Номер диплома		

Рис.3. Личный листок студента

Утверждён
на заседании Учёного Совета
факультета ВМК МГУ
от 24 сентября 2000 г.

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени М.В.ЛОМОНОСОВА
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ
ОТДЕЛЕНИЕ ПО РАБОТЕ С ИНОСТРАННЫМИ УЧАЩИМИСЯ

Учебный план студента Ван Вэй (КНР)
на 2001/2002/2003 учебный год

Академическая программа специалиста

Общая подготовка

Семестр	**1**		**2**		**3**		**4**	
Дисциплины	**Час**		**Час**		**Час**		**Час**	
Алгебра	7	зач/экз						
Математический анализ	6	зач/экз	7	зач/экз	6	зач/экз	6	зач/экз
Практикум на ЭВМ	6	з/оц	4	з/оц	4	з/оц	4	з/оц
Основы программирования	3	Экз						
Экономика	3	Зач						
Русский язык	6	Зач						
Алгебра и геометрия			6	зач/экз	6	зач/экз	4	зач/экз
Алгоритмы и алгоритмические языки			3	Экз	3	экз	3	экз
История России			2	Экз	2	экз	2	экз
Английский язык			3	Зач	3	зач	3	зач
Библиография			1	Зач	2	зач		
Архитектура ЭВМ и язык Ассемблера			4	Экз				
Физика					4	зач	4	зач/экз
Численные методы							4	зач
Всего часов	**31**		**30**		**30**		**30**	
Всего зачётов		**5**		**5**		**6**		**6**
Всего экзаменов		**3**		**5**		**4**		**5**

Зам. декана факультета ВМиК МГУ А.В.Лукшин

Рис.4. Учебный план студента

Брусенцов Н.П., Владимирова Ю.С.

Троичный минимизатор булевых выражений

Минимизация булевых выражений относится к труднорешаемым, так называемым «NP-полным», задачам [1]. Действительно, настойчивые попытки, предпринимаемые на протяжении десятков лет, найти эффективный универсальный алгоритм минимизации все еще не достигли цели, хотя существуют уже алгоритмы минимизации в классе «почти всех булевых выражений» [2] и имеется информация о положительном решении NP-проблемы [Miron Telpiz's P=NP Page – http://www.tarusa.ru/~mit/RUS/rus.html].

Полагают, что трудности минимизации и проблемы NP-полноты в целом обусловлены алгебраичностью математической логики, использованием алгебраического представления булевых функций в качестве основного, что не способствует раскрытию их сущности в той степени, в которой это достижимо на картах Карно, не ущемляющих геометрическую интуицию и диалектическую логику как основу развития интуитивного знания [3]. Однако диалектическая логика возможна и в алгебраической форме [4], равно как и формалистическая, имеющаяся в виде и неалгебраической (традиционной), и алгебраической (математической). Дело не в форме, а в адекватности отображения бытия. Беда формалистов в том, что не приемлют третьего [5].

Предлагаемый алгоритм минимизации булевых выражений построен в трехзначной логике, которая более адекватно отображает сущность булевой алгебры, чем общепринятая сегодня двузначная логика, потому что сама булева алгебра в сущности трехзначна. Верно, что ее независимым переменным (первичным терминам) x, y, z,... допустимо присваивать только два значения: 0 и 1, но будучи связанными логическими условиями (отношениями) эти термины вынуждены принимать также третье логическое значение – не-0 и не-1, которое Аристотель называл *привходящим* ($\sigma\upsilon\mu\beta\varepsilon\beta\eta\kappa\acute{o}\varsigma$), а Буль считал неопределенностью. Обозначим его буквой σ.

В процедуре минимизации выражений это третье играет главную роль. Минимизация осуществляется путем попарного «склеивания»
n-арных конъюнкций, различающихся только по одному из терминов. Например:

$$xy'zu \vee xy'z'u \equiv xy'z^{\sigma}u.$$

В «двузначной» булевой алгебре термин, пребывающий в привходящем статусе опускают, т. е. вместо $xy'z^{\sigma}u$ пишут $xy'u$,

означающее, что отсутствующий термин z не утверждается и не отрицается с необходимостью. В дизъюнктивной нормальной форме (ДНФ) n-арного выражения конъюнкция, содержащая все n терминов, означает индивид, а конъюнкции с привходящими (опущенными) терминами именуют неиндивидные классы. Минимизация состоит в том, чтобы произведя все возможные склейки, свести к минимуму количество конъюнкций в ДНФ-выражении, а вместе с тем и количества терминов в отдельных конъюнкциях.

Для простоты допустим, что исходное выражение представлено в совершенной форме (в СДНФ), т. е. включает только n-арные конъюнкции.

Процедура минимизации начинается с выявления для каждого члена СДНФ всех «парных» для него, т. е. склеивающихся с ним по тому или иному термину членов. Число этих возможностей склеивания члена назовем его «парностью». Парность принимает целые значения от 0 до n: члены, не допускающие ни одной склейки, характеризуются нулевой парностью, склеивающиеся только по одному из терминов – парностью 1, по двум – 2, по всем терминам – парность n.

Далее производится упорядочение СДНФ по возрастанию парности ее членов: первые места занимают члены нулевой парности, если они имеются, за ними следуют, если имеются, члены с парностью 1, затем 2, 3, ..., n. Такая упорядоченность позволяет естественно отделить включаемые в минимальную ДНФ неизменными члены нулевой парности и утратившие парность результаты склеек от тех членов, которые уже учтены в этих результатах («покрыты» ими) и должны быть опущены.

В силу упорядоченности в склеивающихся парах членов один член, назовем его *ведущим*, предшествует другому, *ведомому*. Ведомые члены можно игнорировать, если результат склейки образуется из ведущего члена элиминацией в нем тех терминов, по которым произведена склейка. Например:

$$xyzu \vee xy'z'u \vee xy'zu \equiv xy^{\sigma}zu \vee xy'z^{\sigma}u \equiv xzu \vee xy'u.$$

Один и тот же член может оказаться ведущим относительно одного термина и ведомым относительно другого. Такой член относится к ведомым, т. е. игнорируется. Например:

$$xyzu \vee xy'z'u' \vee xy'zu \vee xy'z'u \equiv xy^{\sigma}zu \vee xy'z'u^{\sigma} \equiv xzu \vee xy'z'.$$

Таким образом, процедура минимизации состоит в выявлении всех возможных склеек, упорядочении по нарастанию парности и удалении всех ведомых членов, а в ведущих членах – удалении всех терминов, относительно которых допустимо склеивание.

Компьютерная версия этой процедуры реализована методом конструктов [6] в диалоговой системе структурированного программирования ДССП [7]. Минимизируемое выражение представлено конструктом вида динамическая цепь n-тритных (троичных) слов, над которыми определены необходимые операции трехзначной логики. Триты моделируются парами битов (бибитами), так что n-тритным словом служит $2n$-битное слово.

Литература

1. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи (введение в теорию NP-полноты). – М.: «Мир», 1982.

2. Стрыгин В.З. Матричная логика и комбинаторные машины. – М: ЦАГИ, Препринт № 120, 1998.

3. Стрыгин В.З. Критика формальнологического метода исследования в дискретной математике. – М.: ЦАГИ, Препринт № 121, 1998.

4. Брусенцов Н.П. Трехзначная диалектическая логика. – В этом сборнике.

5. Брусенцов Н.П. Блуждание в трех соснах (Приключения диалектики в информатике)//Программные системы и инструменты. Тематический сборник № 1. – М.: МАКС Пресс, 2000, с. 13-23.

6. Владимирова Ю.С. Конструктная реализация булевой алгебры // Интегрированная система обучения, конструирования программ и разработки дидактических материалов (учебно-методическое пособие) – М.: ВМиК МГУ, 1996, с. 44-69.

7. Развиваемый адаптивный язык РАЯ диалоговой системы программирования ДССП. / Брусенцов Н.П., Захаров В.Б., Руднев И.А., Сидоров С.А., Чанышев Н.А. - М.: Изд-во Моск. Ун-та, 1987.

Аннотация

Королев Л. Н. Архитектура и программирование (размышления). //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В статье рассматриваются некоторые проблемы программирования, связанные с развитием архитектур вычислительных систем и взаимным влиянием аппаратных решений и требований человеко-машинного интерфейса.

Кратко рассказывается также о некоторых исследованиях и разработках в области системного программирования, проводимых «в ближайшем окружении»: на факультете Вычислительной Математики и Кибернетики (ВМиК) МГУ, и, прежде всего, на кафедре Автоматизации Систем Вычислительных Комплексов (АСВК)

Автору в течение почти 50 лет (с1953 г.) посчастливилось быть свидетелем и непосредственным участником разработки аппаратуры и программного обеспечения некоторых отечественных вычислительных систем и их программного обеспечения. Суждения автора, хотя и носят субъективный характер, могут быть интересны современным компьютерщикам.

Предполагается остановиться на следующих вопросах и проблемах:

- CISC, RISC, VLIW, EPIC, DFC архитектуры и их связь с программированием.
- Объектно-ориентированная парадигма программирования – существо или мода.
- Что такое программирование сегодня и что будет завтра?
- Параллельные вычислительные системы и параллельные (распределенные) программы.

Ил.: нет. Библиогр.: 4 назв.

Машечкин И.В, Петровский М.И, Попов И.С., Шляховая Е.М. Web-система моделирования средств планирования операционных систем mainframe компьютеров. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Данная статья посвящена исследованию проблемы эффективности настройки средств планирования вычислительных комплексов на основе мэйнфреймов фирмы IBM. Предлагаемое решение представляет собой разработанную экспериментальную систему дискретно-событийного имитационного моделирования ThruPut Simulator, позже получившее развитие в качестве WWW-системы моделирования.

Ил.: 5. Библиогр.: 9 назв.

Брусенцов Н.П. Трехзначная диалектическая логика. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Указазан достоверный принцип диалектической логики - сосуществование противоположностей как необходимое третье, восполняющее неполноценность логики двухзначной упразднением ее закона исключенного третьего. Отмечено, что неотобразимость аристотелевой силлогистики в двухзначных исчислениях обусловлена ее трехзначной диалектической природой.

Библиогр. 15.

Доложено на Ломоносовских чтениях 2001 г. на факультете ВМиК МГУ.

Ил.: нет. Библиогр.: 15 назв.

Петровский М.И. Применение методов теории нечетких множеств в системах поддержки принятия решений. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В работе предлагается подход к решению задачи, относящейся к классу плохо структурированных задач, решаемых с помощью систем поддержки принятия решений. Данный подход базируется на применении функции предпочтения построенной с использованием метода нечеткого вывода, а также мер включения, совпадения и нечеткой арифметики. Предложенный метод был реализован в прикладной системе поддержки принятия решений и подтвердил свою адекватность на реальных задачах.

Ил.: 1. Библиогр.: 5 назв.

Костенко В.А. Способы представления и преобразования расписаний в итерационных алгоритмах. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В работе рассмотрены непосредственные и параметрические способы представления расписаний и соответствующие системы операции коррекции расписаний для построения итерационных алгоритмов. Как непосредственный, так и параметрический способ представления расписаний и соответствующие операции коррекции, допускают построение итерационных алгоритмов, обеспечивающих переход от произвольного допустимого расписания к оптимальному за

линейное число итераций и исключают возможность получения недопустимых расписаний на всех итерациях алгоритма.

Табл.: 1. Библиогр.: 10 назв.

Никитин А.В. Эволюционный алгоритм оптимизации ассоциативной памяти. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В статье предложена модель модульной оптической ассоциативной памяти для машин потока данных. Сформулирована задача определения оптимального распределения потока данных в вычислительной системе с модульной ассоциативной памятью. В основе данной модели лежит графовое представление потока данных. Предложен метод поиска оптимального распределение потока по модулям с использованием генетического алгоритма. Исследована сходимость генетического алгоритма. Получены оценки оптимального распределения по модулям ассоциативной памяти для различных вычислительных задач.

Ил.: 7. Библиогр.: 11 назв.

Афанасьев М.К. Моделирование рынка с помощью генетических алгоритмов. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Ил.: нет. Библиогр.: 2 назв.

Брусенцов Н.П., Деркач А.Ю. Трехзначная логика, нечеткие множества и теория вероятностей. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В развитие аристотелевой с третьим–привходящим логики и предложенной П.С.Порецким пробализации логических выражений достигнуто единое диалектическое истолкование сущности логики, теории вероятностей и нечетких множеств Заде.

Ил.: нет. Библиогр.: 8 назв.

Доложено на Ломоносовских чтениях 2001 г. на факультете ВМиК МГУ.

Дерий Д.М., Рамиль Альварес Х. Троичный процессор на двоичном компьютере. //Программные системы и инструменты.

Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В данной статье рассматриваются вопросы моделирования троичного процессора на традиционных двоичных машинах. Статья начинается с описания истории симметричного троичного кода. Далее описывается процесс создания конкретного примера программной модели троичного процессора, обсуждаются возникшие при этом альтернативы и, соответственно, обосновывается их выбор.

Ил.: 1. Библиогр.: 7 назв.

Веселов Н. А., Машечкин И. В. О некоторых экспериментальных средствах анализа колебаний длин очередей сообщений в узлах сети в условиях возникновения перегрузок. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Данная статья посвящена исследованиям алгоритмов маршрутизации и экспериментам, проводимым авторами в этой области. Авторами предложен подход, позволяющий избежать нежелательных явлений возникновения колебаний в сетях передачи данных при использовании широко известных алгоритмов оптимальной маршрутизации. Проведена серия экспериментов, для сравнения работы предложенного подхода с работой алгоритма оптимальной маршрутизации. Полученные результаты позволяют говорить о возможности использования рассмотренного в статье способа управления входным трафиком в условиях возникновения перегрузок в сети.

Ил.: 6. Библиогр.: 6 назв.

Суслов А.А. Извлечение данных из реляционных источников по метаданным. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Идея поисковой машины по метаданным состоит в автоматизированном извлечении метаинформации о структуре и содержании данных, которая не зависит от желания создателей предоставить информацию в том или ином виде. Таким образом, пользователь видит формальное описание и часть реальной информации, которую ему предоставляет источник.

Для каждой предметной области вводится метамодель, на которой собственно и будут задаваться запросы. Созданием такой модели обеспечивается переход от неформального к формальному – ставятся в соответствие желания пользователя и реально существующая информация. Кроме того, пользователь получает возможность более точно и корректно формулировать свой запрос.

Безусловно, для упрощения работы пользователя необходимо создать максимально удобные средства работы с метамоделью и формулировки запросов. С другой стороны, придется разрабатывать средства преобразования метаданных из различных источников в общую метамодель, а также извлечения данных из источников по метаданным.

В данной работе предложена архитектура системы для решения данной задачи для одного типа источников – реляционных баз данных. Также обсуждаются особенности ее проектирования и реализации.

Ил.: 3. Библиогр.: 5 назв.

Калугина Н.В., Шляховая Е.М. Автоматизация построения распределенных информационных систем. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Работа посвящена проблеме параметризации задач построения распределенных информационных систем. Рассматривается экспериментальное инструментальное средство построения информационных систем, которые могут принадлежать различным предметным областям. Обсуждаются основные концепции, функциональность, структурная схема и возможности параметризации, предлагаемые данным инструментальным средством.

Ил.: 2. Библиогр.: 5 назв.

Смирнов Д.В. Проект трехуровневой инструментальной системы обработки динамических процессов. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В данной публикации предлагается проект инструментальной системы и проводится несколько примеров расчетов с её использованием. Трехуровневая организация позволяет без существенной потери эффективности по времени выполнять расчёты на высоком уровне абстракции.

Ил.: 7. Библиогр.: 9 назв.

Маслов С.П., Рамиль Альварес Х., Сидоров С.А. Многотерминальная МСО "Наставник" на IBM PC. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Описывается перенос многотерминального варианта микрокомпьютерной системы обучения (МСО) "Наставник" на IBM PC.

Маслов С.П. Карманная АСО "Наставник" (стартовая версия). //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Обсуждаются достоинства автономной, компактной и доступной АСО "Наставник" с точки зрения повышения удобства работы в системе и упрощения ее установки и эксплуатации. Предлагается программная реализация карманной АСО на существующей тиражируемой технике. Описывается система на устройстве Organiser II XP фирмы PSION. Рассмотрены "облик" системы, организация и кодирование управляющей информации, структура программы. Приведены результаты тестирования. Реализация была выполнена автором в инициативном порядке летом 2000г.

Ил.:1. Библиогр. 5 назв.

Доложено на Ломоносовских чтениях 23 апреля 2001 г.

Столяров А.В. Расширенный функциональный аналог языка Рефал для мультипарадигмального программирования. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В статье рассматривается объектно-ориентированная модель рефал-машины с расширяемыми возможностями, предназначенная для практического применения мультипарадигмального программирования в проектах на языке Си++. Дается описание возможностей библиотеки классов Си++ InteLib, реализующей эту модель.

Ил.: нет. Библиогр.: 12 назв.

Новиков М.Д. Среда dBASE как средство создания автоматизированных систем обработки анкет. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Описывается структура и возможности систем «Международные научно-технические связи» и «Иностранные студенты», находящихся в эксплуатации в иностранном отделе факультета ВМК МГУ.

Ил.:4. Библиогр.: 3 назв.

Брусенцов Н.П., Владимирова Ю.С. Троичный минимизатор булевых выражений. //Программные системы и инструменты.

Рассматриваются нестандартная аппаратура, поддерживающие ее программы (драйверы) и программа, управляющая деятельностью обучаемых, для двух вариантов системы.

Табл.: 2. Библиогр.: 6 назв.

Доложено на Ломоносовских чтения 2001 г. на факультете ВМиК МГУ.

Демаков А.В., Зеленова С.А., Зеленов С.В. Тестирование парсеров текстов на формальных языках. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Рассматривается вопрос автоматизации тестирования парсеров текстов на формальном языке. Грамматика такого языка формализована, например, в форме BNF.

Подобная проблема возникает при разработке компиляторов, телекоммуникационных протоколов и других видов программного обеспечения. Предлагаемый метод используется в промышленной разработке компиляторов языков высокого уровня.

Ил.: 2. Библиогр.: 5 назв.

Ющенко Н.В. Оценка времени выполнения программ статико-динамическим методом. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В данной статье рассматривается метод оценки времени выполнения последовательных программ на современных процессорах с высокой (вплоть до потактовой) точностью, не прибегая к эмуляции процессора. Метод позволяет оценивать время выполнения программ, представленных на языке высокого уровня и преобразуемых в машинный код при помощи оптимизирующего компилятора.

Ил.: 3. Библиогр.: 14 назв.

Леонов М.В. Комплекс программ для автоматизации исследований в таксономической ботанике. //Программные системы и инструменты. Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

В статье представлены программные средства для автоматизации научных исследований в области таксономической ботаники, разработанные в лаборатории вычислительного практикума и информационных систем Факультета ВМиК МГУ для специалистов по растениям семейства Зонтичных Ботанического сада МГУ.

Ил.: нет. Библиогр.: 7 назв.

Тематический сборник №2, М.: Издательский отдел факультета ВМиК МГУ, 2001.

Описана универсальная процедура минимизации булевых выражений, базирующаяся на трехзначной логике и использующая троичный код, благодаря чему преодолена «труднорешаемость» задачи.

Ил.: нет. Библиогр. 7назв.

Доложено на Ломоносовских чтениях 2001 г. на факультете ВМиК МГУ.

ТРУДЫ ФАКУЛЬТЕТА
ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

Выпуск 2

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник факультета ВМиК МГУ

Под ред. чл.- корр. РАН Л.Н. Королева

Издательский отдел факультета ВМиК МГУ
(лицензия ИД №05899 от 24.09.01г.)

Напечатано с готового оригинал-макета
в издательстве ООО "МАКС Пресс"
Лицензия ИД N 00510 от 01.12.99 г.
Подписано к печати 08.11.2001 г.
Формат 60x90 1/16. Усл.печ.л. 13,5. Тираж 200 экз. Заказ 792.
Тел. 939-3890, 939-3891, 928-1042. Тел./Факс 939-3891.
119899, Москва, Воробьевы горы, МГУ.