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

МЕТОДЫ И ИНСТРУМЕНТЫ КОНСТРУИРОВАНИЯ И ОПТИМИЗАЦИИ ПРОГРАММ

Под редакцией
проф. Виктора Николаевича Касьянова

Новосибирск 2005

УДК 519.68; 681.3.06
ББК З 22.183.49+ З 22.174.2

Методы и инструменты конструирования и оптимизации программ. — Новосибирск: Ин-т систем информатики им. А. П. Ершова СО РАН, 2005. — 274 с.

Является двенадцатым в серии сборников, издаваемых Институтом систем информатики им. А.П.Ершова СО РАН. Описывает методы и инструменты конструирования и оптимизации программ.

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

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

TOOLS AND TECHNIQUES OF PROGRAMM CONSTRUCTION AND OPTIMIZATION

Edited by
prof. V. N. Kasyanov

Novosibirsk 2005

This volume is the twelfth one in a series of books published in A.P. Ershov Institute of Informatics Systems. This volume is devoted to the tools and techniques of program construction and optimization.

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

ПРЕДИСЛОВИЕ РЕДАКТОРА

Двенадцатый выпуск серии «Конструирование и оптимизация программ» посвящен решению актуальных задач, связанных с разработкой методов и инструментов конструирования эффективных и надежных программ.

Продолжая уже сложившиеся традиции, данный выпуск, как и предыдущие, базируется на результатах исследований, выполненных в лаборатории по конструированию и оптимизации программ Института систем информатики СО РАН совместно с Новосибирским государственным университетом при финансовой поддержке Российского фонда фундаментальных исследований, Российского гуманитарного научного фонда, Министерства образования и науки Российской Федерации, а также компании «Майкрософт».

Открывает сборник совместная статья группы авторов (Т.В. Батура и др.), посвященная разрабатываемой авторами исследовательской системе для анализа текстов на естественном языке.

В статье Т. Ф. Валеева рассматривается проблема поиска регуляторных модулей в последовательностях ДНК и проводится сравнительный анализ трёх программных систем: TOUCAN, TELiS и Composite Module Analyst.

Статья А.А. Винокурова, И.В. Ильина, Ф.А. Мурзина и Д.Ф. Семича посвящена алгоритмам расчета нефтенасыщенности по данным ядерного каротажа и их реализации в виде программной системы OilTemper.

Статья Т.А. Волянской посвящена пользовательскому интерфейсу виртуального музея истории информатики в Сибири и содержит изложение его возможностей по управлению информационными ресурсами и управлению пользователями.

В статье Е.В. Касьяновой представлен вводный курс программирования на базе языка Zonnon. Это новый универсальный язык программирования в семействе языков Паскаль, Модула-2 и Оберон, работа над которым ведется в Цюриховском институте информатики.

В статье Т.Г. Коновалова и В.М Комашко рассматривается подход к решению задачи выделения генов, изменивших с определенной достоверностью уровень своей экспрессии на основе данных из нескольких микрочиповых экспериментов.

В статье Ю. В. Малининой рассматривается подход к извлечению и унификации информации, содержащейся в публикациях по преобразовани-

ям программ, в основе которого лежит автоматическое формирование смыслового портрета текста в виде ассоциативной (семантической) сети.

В статье Л.С. Мельникова и И.В. Петренко изучаются путевые ядра и разбиения в графах с малыми длинами циклов, доказывается теорема о том, что каждый граф имеет P_9-ядро.

В статье Г.П. Несговоровой делается обзор существующих в сети Интернет так называемых виртуальных музеев и делается попытка установить различия между сайтами — представительствами реальных музеев и собственно виртуальными музеями.

В статье Р. А. Осмонова изучаются преобразования гнезда циклов посредством унимодулярной матрицы, действующей на индексные переменные. Описываемые преобразования используются как для выявления параллелизма, так и для повышения его степени.

Статья К.А. Пыжова посвящена блоку редуцирующих преобразований, который может использоваться в системе функционального программирования SFP в качестве составной части оптимизатора, а также интерфейса, расширяющего возможности работы с внутренним IR-представлением.

В работе А. И. Синякова анализируются средства модульности в существующих языках программирования и приводится реализация модульности в функциональном языке Sisal 3.0.

В статье А. П. Стасенко представлена система COM-интерфейсов, задающих трансляцию Sisal-программы из её текстового представления во внутреннее представление IR1, основанное на графовой модели. Приводятся требования к желаемой функциональности системы интерфейсов и возможные направления её расширения.

В статье Ю. Хана проводится анализ современных средств отладки программ на функциональных языках программирования, в основе которых трассировка, пошаговое выполнение и декларативная отладка.

Статья Е. С. Черемушкина посвящена изучению структуры ДНК с целью нахождения тех различий, которые возможно не будут способствовать распознаванию неизвестных участков, но которые характеризуют качественные различия ДНК разных функций.

Завершает сборник статья Д.Н. Штокало и Е.С Черемушкина, в которой представлен программный комплекс Regulatory Sequences Analyzer, разработанный авторами для визуализации поиска потенциальных цис-элементов последовательности ДНК.

Проф. В.Н. Касьянов

Т.В. Батура, О.В. Корда, Ф.А. Мурзин, А.А. Позименко*

ИССЛЕДОВАТЕЛЬСКАЯ СИСТЕМА ДЛЯ АНАЛИЗА ТЕКСТОВ НА ЕСТЕСТВЕННОМ ЯЗЫКЕ

ВВЕДЕНИЕ

Исследования в области автоматической обработки текста (АОТ) и формализации естественных языков, планомерно продвигаясь от самых простых методов анализа к более сложным, постепенно приближаются к такому уровню обработки текста, на котором уже возможно представление текста не просто в виде последовательности слов, а единым целым, обладающим неким смыслом, что уже соответствует человеческому восприятию. Увеличение вычислительных мощностей сделало возможным применение трудоёмких лингвистических алгоритмов на больших объемах данных.

Основной целью данной работы является разработка исследовательской системы, реализующей новые подходы к анализу текстов на естественном языке. В результате построено приложение «испытательный стенд», использующее существующие системы для «первичного» анализа текста и реализующее собственные подходы к анализу и представлению естественного языка.

1. СИСТЕМЫ ОБРАБОТКИ ТЕКСТОВ

В настоящее время лингвистами сформулированы различные теории, позволяющие в какой-то степени формализовать естественный язык. В основном, суть этих теорий сводится к тому, что предложению в тексте сопоставляются различные конечные объекты — графы или, в общем случае, конечные модели, которые, как принято считать [1], отражают смысл предложений.

* murzin@academ.org, tbatura@ngs.ru

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 05-08-33448).

1.1. Общие принципы систем обработки текстов

Компоненты, составляющие структуру систем анализа текстов — лингвистические процессоры, которые последовательно обрабатывают входной текст. Вход одного процессора является выходом другого [2].

Выделяются следующие компоненты:

- графематический анализ — выделение слов, цифровых комплексов, формул и т.д.;
- морфологический анализ — построение морфологической интерпретации слов входного текста;
- синтаксический анализ — построение дерева зависимостей всего предложения;
- семантический анализ — построение семантического графа текста.

Для каждого уровня разрабатывается свой язык представления. Язык представления обычно состоит из констант и правила их комбинирования. На графематическом уровне используются константы, называемые графематическими дескрипторами (ЛЕ — лексема, ЦК — цифровой комплекс и т.д.). На морфологическом уровне — граммемы (**рд** — родительный падеж, **мн** — множественное число). На синтаксическом — названия отношений (**subj** — отношение между подлежащим и сказуемым, **circ** — обстоятельство). О семантическом анализе будет сказано ниже.

Основой для построения уровней служат результаты работы предыдущих этапов, но, что важно, последующие анализаторы также могут улучшить представление предыдущих. Например, для какого-то предложения синтаксический анализатор не смог построить полного дерева зависимостей, тогда, возможно, семантический анализатор сможет спроектировать построенный им семантический граф на синтаксис.

Такой многоуровневый подход позволяет предложить критерии оценки систем машинного перевода. Вполне можно утверждать, что разработчики ФРАП [6, 7] показали, что для достижения адекватности перевода (равенство по смыслу входному тексту) и грамматической правильности выходной фразы необходимо присутствие всех пяти этапов, причем адекватность перевода можно гарантировать только после работы «глубоких» — синтаксического и информационного — анализаторов.

1.2. Система Диалинг

Система Диалинг разрабатывалась как система русско-английского перевода с 1999 по 2002гг. на базе ООО «Диалинг» [2, 3]. В разное время в

работе над системой принимали участие 22 специалиста, большинство из которых — известные учёные-лингвисты [4, 5].

Как и все современные системы обработки текста, Диалинг включает в себя основные этапы анализа текста: графематический, морфологический и синтаксический, а также ещё один, не так давно появившийся, семантический этап. В отличие от морфологического и синтаксического, на семантическом этапе появляется формальное представление смысла текста.

За основу системы автоматического русско-английского перевода Диалинг были взяты система французско-русского автоматического перевода (ФРАП), разработанная в ВЦП совместно с МГПИИЯ им. М. Тореза в 1976–1986гг. [6, 7], и система анализа политических текстов на русском языке — ПОЛИТекст, разработанная в центре информационных исследований в 1991–1997гг. [8].

Система ФРАП содержала полную цепочку анализа текста, вплоть до семантического, который был реализован только частично. В системе ФРАП был разработан и опробован семантический аппарат, на основе которого в системе Диалинг был создан оригинальный метод семантического анализа — метод полных вариантов. В центре семантического аппарата ФРАП находятся два перечня (вернее, две грамматики): семантических характеристик (СХ) и смысловых отношений (СО). Используется минимальное количество семантических характеристик: ВЕЩВО (“вещество”), ИЗМ (“изменение”), ИНТЕЛ («интеллектуальность»), ИНФ (“информация”) и т. д.; слова характеризуются по признаку принадлежности к одному или нескольким классам. СХ обеспечивают проверку семантического согласования при интерпретации связей в тексте.

Система ПОЛИТекст была направлена на анализ официальных документов на русском языке и содержала полную цепочку анализаторов текста: графематический (первичный анализ), морфологический, синтаксический и частично семантический. Из системы Диалинг был частично заимствован графематический анализ, но адаптированный под новые стандарты программирования. Программа морфологического анализа была написана заново, поскольку скорость работы была низкой, но сам морфологический аппарат не изменился.

2. ПОСТАНОВКА ЗАДАЧИ

Целью данной работы являлось создание исследовательской системы для анализа текстов на естественном языке. Под этим подразумевается не некото-

рый продукт, готовый к применению в прикладных задачах, а система, в которой реализовываются различные подходы для анализа текстов, т.е. целью проекта является получение наработок в области анализа текстов на естественном языке. Эти наработки могут быть использованы в других проектах, а также для дальнейших исследований структуры естественных языков.

Конкретной целью работы является программная система, реализующая следующие подходы к семантическому анализу текстов на естественном языке:

- сопоставление тексту набора предикатов узкого исчисления (лексических функций, грамматических предикатов и др.) [9–12];
- представление предложений с помощью деревьев с пометками на основе определения слова (словарной статьи из словаря Ожегова).

Система должна обеспечивать:

- загрузку текста;
- морфологический и синтаксический анализы текста;
- реализацию перечисленных подходов к анализу текста;
- графический вывод результатов анализа.

Отметим, что морфологический и синтаксический анализы производятся посредством использования внешних модулей (системы Диалинг). Они необходимы в системе формирования грамматических предикатов и для других целей.

3. ПРЕДСТАВЛЕНИЕ ПРЕДЛОЖЕНИЙ С ПОМОЩЬЮ ДЕРЕВЬЕВ С ПОМЕТКАМИ

Считаем, что поисковый запрос представляет собой совокупность предложений на естественном языке. Эту совокупность предложений можно расширить, используя словарные статьи из толкового словаря (например, словаря Ожегова), т.е. фактически приписать определения отдельных слов. Следующий этап — представление данных предложений в виде помеченных деревьев. Вершины помечаются словами, а ребра — вопросами, задаваемыми от одного слова к другому.

Далее имеется текст достаточно большого объема, из которого необходимо выбрать предложения по тематике поискового запроса и таким образом сформировать аннотацию или решить, является ли текст релевантным к данному запросу. Для этого предложения данного текста также могут быть представлены в виде деревьев (вообще говоря, необязательно все предло-

жения, а выборочно, по некоторым критериям). После этого необходимо сопоставление на похожесть (соответствие) деревьев из запроса и деревьев, возникших из текста.

Для аннотации выбираются предложения, которые соотносятся по теме, имеют похожие структуры и т.д. На основании подобных идей можно судить о релевантности.

Данный вопрос еще требует серьезной доработки. Часть материала представлена ниже.

3.1. Структура словарной статьи в словаре С.И. Ожегова

Для описания структуры словарных статей необходимы некоторые обозначения.

Наклонным шрифтом в скобках будем обозначать пояснение, некоторую характеристику между двумя словами или между словом и группой слов, относящейся к этому слову. Полужирным шрифтом записываем вопросы. Вопросы можно было бы записывать везде, но в этом нет необходимости. Вопросы указаны только в тех случаях, когда они несут большую информативность, нежели пояснения, или для того, чтобы провести общую линию в схемах статей (наличие ограниченного числа вопросов, которые часто повторяются). Полужирным и наклонным одновременно записаны слова, которые подразумеваются, но не присутствуют в толковании. Это удобно при схематической записи статьи, в которой встречаются причастия (причастия легко представить через глагол, так как они производные от глаголов).

Если вопрос задан к одной самостоятельной части речи, то пишем *синоним* или *синоним, но из другой части речи*. Если вопрос задан к словосочетанию или целому предложению, то этот фрагмент разбирается самостоятельно.

К однородным членам предложения и от однородных членов предложения задается каждый вопрос в отдельности. Например:

«употребляется для указания на расстояние или время, отделяющие одно пространство или событие от другого».

для указания -> (**на что?**) на расстояние

для указания -> (**на что?**) на время

Если между однородными членами предложения стоит союз, и слово относится к каждому из них, то ставится знак $\vee$. Например:

на расстояние $\vee$ время -> (*признак предмета*) отделяющие одно пространство или событие от другого.

В скобки […] заключена конструкция (она является целиком одним членом предложения), состоящая из набора слов, целиком относящаяся к одному слову, но не сложное предложение.

В скобки {…} заключена часть сложного предложения (не первая), т.е., если предложение сложное, в нем обязательно встречаются такие скобки.

Предложение или конструкцию записываем полностью после слова, от которого задается вопрос. Затем в скобках идет детальный разбор.

Если возможно задать два вопроса или более, то они записаны через «;».

Структурная запись (запись, как в языках программирования) помогает разделить толкование на семантические слои. В каждом таком слое обязательно присутствует центральное (главное) слово, т.е. слово, от которого задаётся вопрос.

Типичная словарная статья, как правило, может быть представлена в виде $\langle t, m, s \rangle$, где

t — заглавное слово в словарной статье;

m — слово, выполняющее служебную функцию;

s — толкование значения слова (определение слова).

Пример.

ДО. *предлог с род. п.*

1. Употребляется для указания на расстояние или время, отделяющие одно пространство или событие от другого.

Здесь t = «до», m = «употребляется», s — остальная часть предложения.

Слово, выполняющее служебную функцию, иногда отсутствует, но подразумевается. Например, в статье «**ЗАГРАНИЧНЫЙ**». Тогда в толковании дается сразу синоним или синоним, но из другой части речи.

ЗАГРАНИЧНЫЙ. Относящийся к зарубежным странам, зарубежный.

Здесь t = «заграничный», m отсутствует, s — остальная часть предложения.

Случай, когда слово, выполняющее служебную функцию, присутствует, более всего характерен для толкования предлогов. Гораздо реже подобная ситуация встречается в объяснении наречий. Большинство статей для наречий начинается со слов:

в места, в месте, в место, из мест, из места, на место, от места

в направлении, по направлению

в сторону, на стороне, со стороны, со сторон

в части

из источника

на пространство
на расстоянии, с расстояния
по линии
по поверхности.

Большая часть статей для существительных начинается со слов:
полоса (*чего-то*)
предмет (*чего-то*)
род (*чего-то*)
слой (*чего-то*)
часть (*чего-то*).

В толкованиях прилагательных характерно наличие отношения *признак предмета* (так как объяснение часто дается не через прилагательные, а через причастия), а глаголов, как и всех вышеперечисленных частей речи, — наличие отношения *синоним*.

Паре $\langle m, s \rangle$ может быть сопоставлена схема, аналогичная схеме синтаксического разбора. Если m отсутствует, то такая схема представляет собой просто синтаксический разбор предложения s.

ДО. *предлог с род. п.*

1. Употребляется для указания на расстояние или время, отделяющие одно пространство или событие от другого.

употребляется -> (*цель*, **для чего?**) для указания
для указания -> (**на что?**) на расстояние
∨
для указания -> (**на что?**) время
на расстояние ∨ время -> (*признак предмета*) отделяющие
одно пространство или событие от другого
[
(***которые***) отделяют
отделяют -> (**что?**) пространство
∨
отделяют -> (**что?**) событие
пространство ∨ событие -> (*признак предмета*) одно
пространство ∨ событие -> (**от чего?**) от другого
]

Или другой пример, в котором m отсутствует.

ПЕРЕНОСИЦА. Верхняя часть носа, примыкающая ко лбу и образующая углубление между лбом и носом.

переносица -> (*синоним*) часть носа
часть -> (*признак предмета*) верхняя
часть -> (*признак предмета*) примыкающая ко лбу и образующая углубление между лбом и носом
[
(***которая***) примыкает
примыкает -> (**к чему?**) ко лбу
&
(***которая***) образует
образует -> (**что?**) углубление
углубление -> (**между чем?**) между лбом
углубление -> (**между чем?**) между носом
]

Таким образом, каждый узел схемы представляет собой либо пару $\langle w, q \rangle$, где *w* — слово (оно стоит перед стрелкой), *q* — вопрос (стоит в скобках после стрелки), либо $\langle s \rangle$ — а) предложение, или б) причастный, деепричастный оборот, или в) несогласованное определение, или г) сравнительная, превосходная степени сравнения прилагательных или наречий (их сложная форма образования), или д) некоторые неразрывные словосочетания.

Предположим, что дано предложение или часть предложения $w_1 p_1 w_2 p_2 ... w_n p_n$, где w_i — слова, p_i — разделители, т.е. пробелы или знаки препинания. Тогда можно рассмотреть упорядоченный кортеж $\langle w_1, ..., w_n \rangle$.

Переходы вида $\begin{matrix} \langle w_i, q_i \rangle \\ \uparrow \\ \langle w_j, q_j \rangle \end{matrix}$ и $\begin{matrix} \langle w_i, q_i \rangle \\ \uparrow \\ \langle s_j \rangle \end{matrix}$ могут рассматриваться как правила формальной грамматики Хомского. На каждом шаге вопрос (нетерминальный символ грамматики) заменяем словом или предложением (терминальным символом). Предложения $\langle s_j \rangle$, которые подставляются в схеме вместо вопроса q_i, могут быть разобраны аналогичным образом.

3.2. Пример построения дерева

Рассмотрим пример со словом «междуречье».

Местность между двумя или несколькими реками, включающая водоразделы и прилегающие склоны долин.

междуречье -> (*синоним*) местность

местность -> (*признак предмета;* **между чем?**) между реками
между реками -> (*количество,* **сколькими?**) двумя
∨
между реками -> (*количество,* **сколькими?**) несколькими
местность -> (*признак предмета*) включающая водоразделы и прилегающие склоны долин
[
(***которая***) включает
включает -> (**что?**) водоразделы
&
включает -> (**что?**) склоны
склоны -> (*признак предмета*) прилегающие
склоны -> (**чего?**) долин
]

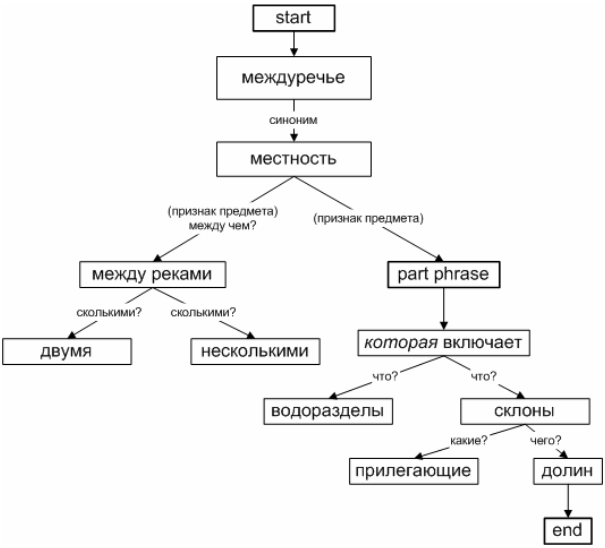


Рис. 1. Пример построения дерева, помеченного вопросами и ответами

4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Система состоит из набора функциональных модулей, представленных на рис. 2.

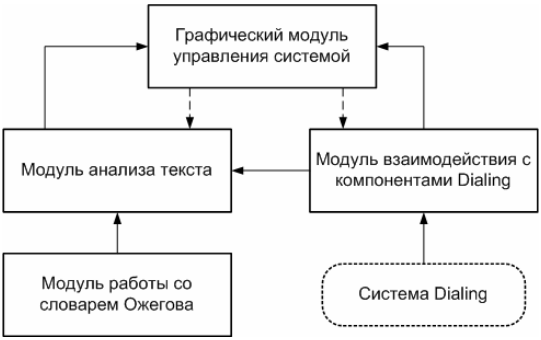


Рис. 2. Архитектура системы

Графический модуль управления системой выполняет функции общей координации работы системы. В этот модуль поступают все действия, произведённые пользователем, а также уведомления о внутренних событиях системы. В соответствии с полученной информацией этот модуль активизирует другие подсистемы для выполнения необходимой операции.

Модуль анализа текста является ядром системы и выполняет функции по генерации грамматических предикатов и построения деревьев с пометками по словарной статье из словаря Ожегова. Данный модуль взаимодействует с *модулем работы со словарем Ожегова* и с *модулем взаимодействия с компонентами Диалинг*.

Модуль взаимодействия с компонентами Диалинг. Задачей данного модуля является инициализация компонентов системы Диалинг и предоставление интерфейсов для основных функций, реализованных в этих модулях.

Модуль работы со словарем Ожегова обеспечивает доступ системы к словарю Ожегова, его основной задачей является получение словарной статьи из словаря.

Система реализована в среде Microsoft Visual Studio 6.0 на языке C++ с использованием MFC (Microsoft Foundation Classes). Такой выбор объясняется следующими преимуществами:

- язык C++ является одним из наиболее гибких языков в настоящее время;
- в связи с большой трудоёмкостью задачи, требуется наиболее производительное решение с большими возможностями к оптимизации.

4.1. Пользовательский интерфейс

Общий вид пользовательского интерфейса системы представлен на рис. 3.

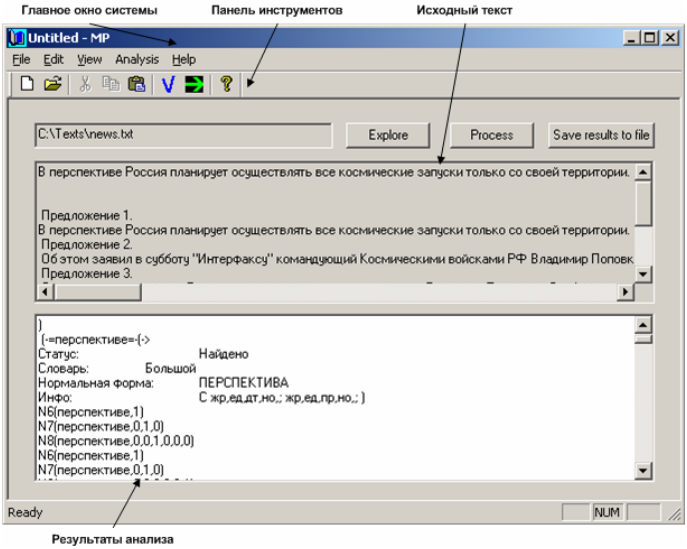


Рис. 3. Пользовательский интерфейс системы

На панели инструментов находятся кнопки, управляющие системой.

- Создание нового текста
- Загрузка текста из файла
- Вырезка выделенного участка текста с помещением в буфер обмена
- Копирование выделенного участка текста в буфер обмена
- Вставка текста из буфера обмена
- Установка параметров анализа
- Запуск анализа
- Информация о системе

Отметим, что все кнопки панели инструментов продублированы в меню. Кнопки входят в набор стандартных элементов управления большинства современных систем, работающих с текстом, поэтому

описание их работы приводить не будем; в системе они обладают стандартной функциональностью.

4.2. Анализ текстов

Перед запуском анализа текста пользователю необходимо выбрать файл, содержащий текст, который нужно проанализировать. Это можно сделать несколькими способами: нажав кнопку «Explore», через меню или через панель инструментов, нажав  . Текст загрузится в верхнее текстовое окно и для удобства будет разбит по предложениям (рис. 3).

Далее, для непосредственного запуска анализа текста пользователю необходимо нажать кнопку «Process» (то же самое действие производит  на панели инструментов, также анализ текста можно запустить из меню). Результаты анализа будут выведены в нижнем текстовом окне и будут содержать следующее.

- Информацию, полученную из системы Диалинг:
  - статус: найдено/не найдено слово в словаре системы;
  - словарь, в котором найдено слово;
  - лемма или нормальная форма слова;
  - набор граммем для данного слова.
- Грамматические предикаты, соответствующие данной части речи.
- Определение анализируемого слова из словаря Ожегова.
- Данные для построения дерева, помеченного вопросами и ответами.

Полученные результаты можно сохранить в текстовый файл посредством нажатия на кнопку «Save results to file» или через меню.

4.3. Установка параметров анализа

При нажатии на кнопку  появляется диалоговое окно, в котором устанавливаются параметры для анализа текстов (рис. 4). Пользователю предоставляется возможность выбрать, какие грамматические предикаты нужно генерировать по частям речи. Опция «Generate all» автоматически выбирает все предикаты. Также есть возможность включать/отключать вывод определения из словаря Ожегова и генерирование данных для построения дерева, помеченного вопросами и ответами.

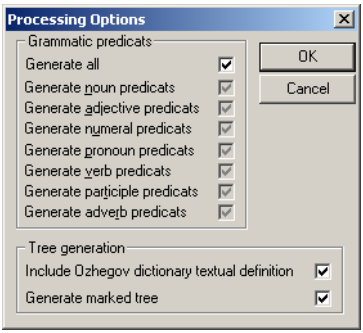


Рис. 4. Форма установки параметров анализа

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были изучены и проанализированы различные системы автоматической обработки текста и алгоритмы, на которых эти системы базируются, их возможности, достоинства и недостатки. Были подробно изучены функциональные возможности и интерфейсы, предоставляемые компонентами системы Диалинг, при этом пришлось преодолеть трудности, связанные с неполной и, к тому же, не всегда достоверной документацией интерфейсов. В связи с этим, приходилось опробовать разные варианты работы с внешними компонентами, а также исследовать их с привлечением некоторых дополнительных инструментов, что существенно усложнило процесс разработки.

Также, ввиду отсутствия толковых словарей в формате, удобном для программного доступа, словарь Ожегова был переведен из плохо структурированных текстовых документов в базу, к которой легко обращаться программно.

Основным результатом работы является исследовательская программная система, использующая два различных подхода для анализа текстов на естественном языке:

- построение лексических функций и грамматических предикатов;
- представление предложений с помощью деревьев с пометками, построенных по словарной статье словаря Ожегова.

Результаты работы предполагается использовать в самых разных областях: от фундаментальной лингвистики до прикладного уровня, например, в

интеллектуальных поисковых системах, в системах автоматического резюмирования и т.д.

На данный момент еще не разработан алгоритм для сравнения и сопоставления помеченных деревьев, построенных по словарной статье из словаря Ожегова. В перспективе планируется реализация такого алгоритма, что позволит проверить на практике эффективность подхода.

СПИСОК ЛИТЕРАТУРЫ

1. **Мельчук И.А.** Опыт теории лингвистических моделей типа «Смысл⇔Текст». — М.: Наука, 1974.
2. **Сокирко А.В.** Реализация первичного семантического анализа в системе Диалинг // Тр. Междунар. семинара «Диалог'2000» по компьютерной лингвистике и ее приложениям, 1–5 июня 2000 г., Протвино.
3. **Панкратов Д.В., Гершензон Л.М.** Описание синтаксического анализа в системе Диалинг. Техн. документация по сист. Диалинг. — М., 1999.
4. **Леонтьева Н.Н. и др.** Семантический компонент в системах автоматического понимания текстов // Обзорная информация. — М.,1982. — Вып. 6.
5. **Леонтьева Н.Н.** Этапы информационного анализа естественного текста // Международный форум по информации и документации. — М., 1987. — Т. 12, № 4. — С. 8–14.
6. **Леонтьева Н.Н.** Система французско-русского автоматического перевода (ФРАП): лингвистические решения, состав, реализация // МП и ПЛ. Проблемы создания системы автом. перевода: Сб. научн. трудов МГПИИЯ им. М. Тореза. — М.,1987. — Вып. 271. — С. 6–25.
7. **Кудряшова И.М.** О семантическом словаре в системе ФРАП // МГПИИЯ им. М. Тореза: Сб. научн. трудов. — М., 1986. — Вып. 271.
8. **Леонтьева Н.Н.** ПОЛИТекст: информационный анализ политических текстов: Сб. НТИ. — 1995. — Сер. 2, № 4.
9. **Батура Т.В.** Представление смысла текста на естественном языке и его лексический анализ // Технологии Microsoft в информатике и программировании. — Новосибирск, 2004. — С. 88–90.
10. **Батура Т.В., Еркаева О.Н., Мурзин Ф.А.** К вопросу об анализе текстов на естественном языке // Новые информационные технологии в науке и образовании. — Новосибирск, 2003. — С. 7–58.
11. **Батура Т.В., Мурзин Ф.А.** Логические методы представления смысла текста на естественном языке // Новые информационные технологии в науке и образовании. — Новосибирск, 2003. — С. 59–111.
12. **Батура Т.В., Корда О.В.** Программные средства для анализа текстов на естественном языке // МНСК-XLII, 15–19 апреля 2004 г., Новосибирск.

Т. Ф. Валеев*

СРАВНИТЕЛЬНЫЙ АНАЛИЗ МЕТОДОВ ПОИСКА РЕГУЛЯТОРНЫХ МОДУЛЕЙ В ПОСЛЕДОВАТЕЛЬНОСТЯХ ДНК, ИСПОЛЬЗУЮЩИХ ДАННЫЕ МИКРОЭРРЭЕВ

ВВЕДЕНИЕ

В связи с появлением и развитием технологии микроэррэев для получения значений экспрессии (величины, соответствующей количеству белка, производимого геном) многих генов одновременно, встала задача определения набора транскрипционных факторов, которые регулируют экспрессию данной группы генов [1]. Говорят, что фактор регулирует экспрессию данного гена, если от наличия данного фактора в достаточной концентрации в клетке зависит величина экспрессии гена. Другими словами, задача состоит в том, чтобы найти набор транскрипционных факторов, которые связываются с промоторами (регуляторными участками) генов, показывающих высокую экспрессию.

Также используют данные из нескольких экспериментов по экспрессии для более точного предсказания. Тогда экспрессия гена представляется в виде вектора значений экспрессии в каждом эксперименте.

Задача поиска набора транскрипционных факторов, которые участвуют в регуляции экспрессии генов, решается в два этапа. Во-первых, ищутся потенциальные сайты связывания транскрипционных факторов с промоторами генов. Здесь на каждом промоторе по отдельности выполняется поиск сайтов связывания каждого фактора, вычислительная сложность пропорциональна числу промоторов и числу факторов. Эта часть, как правило, реализуется сходным образом в различных разработках, её обычно можно выполнить независимо, сохранив результаты для второго этапа.

Второй этап заключается в поиске композитного модуля, набора факторов, регулирующих данный набор генов. Здесь возможны значительные вариации в постановке задачи и методах её решения.

* lan@biorainbow.com

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 05-08-33448).

В настоящее время разработано несколько программных продуктов, которые позволяют решать задачу поиска регуляторных факторов. В данной статье рассмотрены три такие разработки. Это пакет TOUCAN [2], разработанный группой Bioi в Католическом Университете города Леувен, Бельгия; система TELiS [3], созданная в Калифорнийском университете, а также наша собственная разработка, Composite Module Analyst [4]. Ниже приведены описание и обзор возможностей каждой из этих систем, а затем проанализированы их достоинства и недостатки.

1. ПАКЕТ TOUCAN

Пакет TOUCAN представляет собой набор программ для решения различных задач в области анализа регуляторных последовательностей промоторов. Он включает в себя клиентское приложение на языке Java, которое загружает данные, решает часть задач на стороне пользователя, а также позволяет посредством технологии SOAP (Simple Object Access Protocol) связываться с веб-службами на Tomcat-сервере разработчиков и пользоваться процессорными ресурсами кластера на 10 процессоров для запуска особо трудоёмких вычислений. Управление распределёнными вычислениями осуществляется при помощи Java RMI (Remote Method Invocation). Пакет доступен на веб-сервере разработчиков:

`http://www.esat.kuleuven.ac.be/~saerts/software/toucan.php`

Среди доступных веб-служб были проанализированы MotifScanner и ModuleSearcher. Первое приложение используется для поиска сайтов связывания с транскрипционными факторами, а второе — для поиска транскрипционного модуля, регулирующего заданный набор генов.

На вход программы ModuleSearcher подаётся набор промоторов, показавших высокую экспрессию в эксперименте. Входные параметры включают количество факторов в модуле, максимально допустимое расстояние (количество нуклеотидов) между сайтами связывания отдельных факторов модуля на каждом промоторе. В основе поиска набора факторов лежит направленный перебор возможных наборов одним из двух способов, алгоритмом A* или генетическим алгоритмом. При этом максимизируется весовая функция, характеризующая, насколько хорошо данный набор факторов соответствует набору промоторов. Весовая функция учитывает, сколько факторов из пробного набора имеют сайты связывания на каждом из промоторов и насколько близко они расположены.

Алгоритм A* [5] представляет собой метод ветвей и границ, адаптированный под данную задачу. Метод ветвей и границ впервые был предложен Лендом и Дойгом [6] для решения общей задачи линейного программирования. Он позволяет найти оптимальное решение NP-полных задач, существенно снижая трудоёмкость вычислений в большинстве ситуаций. Тем не менее, в данной задаче вычисления могут занять много времени (до нескольких суток на современных процессорах) и изрядный объём памяти. Учитывая, что общее число факторов может превышать 100, число всевозможных модулей из пяти факторов превышает 10^{10}, практически невозможно искать оптимальный модуль, состоящий более чем из четырёх факторов.

В большинстве случаев на практике имеет смысл использовать генетический алгоритм [7]. Этот алгоритм не находит гарантированно оптимального решения, но результаты тестирования показали, что модуль, найденный генетическим алгоритмом, в подавляющем большинстве случаев совпадает с оптимальным модулем, найденным алгоритмом A*. Генетический алгоритм выполняется на порядок быстрее и требует намного меньше памяти. Суть его состоит в следующем. Вначале генерируется популяция из случайных модулей (~200−1000), они сортируются в порядке убывания весов, после чего некоторая доля лучших (30−50%) остаётся (выживает) и дает «потомство», а остальные вымирают. Потомство создается уже не случайно, а посредством скрещивания существующих модулей (случайно выбираются два модуля и генерируется новый, содержащий часть факторов из первого, а часть — из второго) и мутации (в полученном после скрещивания модуле каждый фактор с некоторой вероятностью может быть заменён на случайный). Эта процедура повторяется заданное число итераций (поколений), порядка 100−1000 в зависимости от других входных параметров. Затем модуль в популяции, обладающий наибольшим весом, выдаётся как результат.

2. СИСТЕМА TELIS

Система TELiS также реализована на Java и разделена на два модуля, PromoterScan и PromoterStats, первый из которых выполняет поиск сайтов связывания, а второй находит факторы, которые наиболее (или наименее) представлены на заданной пользователем выборке промоторов. Чтобы не выполнять многократно поиск сайтов, авторы выполнили эту операцию для промоторов всех генов (~40000) человека и для всех известных регули-

рующих факторов позвоночных (~200). Результаты этого анализа сохранены в базе данных MySQL и доступны на сайте разработчиков:

`http://www.telis.ucla.edu/`

Также предоставляется возможность запустить через веб-интерфейс второй модуль PromoterStats, который выполняется на стороне сервера (Java-servlet) и выдаёт результаты пользователю.

Здесь используется упрощённый подход: PromoterStats не ищет модуль целиком, а выполняет поиск отдельных факторов, которые статистически встречаются чаще (или реже) на заданной пользователем выборке промоторов (по сравнению со всем набором промоторов, использованных в данном эксперименте). Для статистической оценки используется z-тест [8]. Оценивается среднее количество сайтов связывания каждого фактора на промоторе, а также количество промоторов, где присутствует хотя бы один сайт связывания. Задача решается за линейное время от числа промоторов и числа факторов, результат получается сразу же после отправки формы с исходными данными.

3. COMPOSITE MODULE ANALYST

Composite Module Analyst (CMA), разрабатываемый нашей группой, написан на C++ и на данный момент представляет приложение с интерфейсом командной строки, компилируемое под Win32 и Unix. Планируется также разработать веб-интерфейс и Win32 GUI. Он включает в себя два этапа: поиск сайтов связывания и поиск модуля, причём этапы могут быть выполнены по отдельности (с сохранением промежуточных результатов) или вместе.

Здесь на вход подаётся либо два набора промоторов, либо набор промоторов и соответствующие им численные значения экспрессии. В первом случае один из наборов включает промоторы с повышенной экспрессией, а второй — остальные промоторы, использованные в эксперименте, и выполняется поиск такого модуля, который даёт большой вес на первом наборе и малый на втором. Во втором случае учитывается также величина экспрессии и ищется такой модуль, веса которого для каждого промотора лучше всего соответствуют величинам экспрессии. Строится зависимость величины веса промотора от его экспрессии, аппроксимируется прямой и вычисляется квадрат смешанной корреляции для аппроксимации, который и максимизируется.

Поиск лучшего модуля осуществляется генетическим алгоритмом, его реализация похожа на реализацию в TOUCAN. Кроме поиска обычных модулей, состоящих из заданного числа факторов, реализован также поиск так называемого булева модуля. Булев модуль представляет собой булеву формулу следующего вида:

$$\overline{\left(\bigcup_i p_{0i}\right)} \cap \left(\left(\bigcap_j \bigcup_i p_{ij}\right)\right).$$

Здесь p_{ij} — логическая величина, определяющая, есть ли на данном промоторе сайты связывания определённого фактора. Формула представляет собой конъюнкцию дизъюнкций, причём первый конъюнкт с отрицанием. С точки зрения биологии это означает, что факторы, вошедшие в первый конъюнкт, препятствуют повышению экспрессии, а факторы, оказавшиеся внутри одного конъюнкта, взаимозаменяемы. Число конъюнктов, дизъюнкций внутри конъюнкта и общее число матриц могут варьироваться в заданных пределах. Такие модули также могут мутировать и скрещиваться, как и обычные.

Помимо нахождения лучшего модуля, CMA позволяет решать некоторые сопутствующие задачи: кластеризацию входного набора промоторов (разделение набора на подмножества, к которым найденный модуль подходит лучше всего), прогон алгоритма несколько раз и сравнение результатов для тестирования устойчивости, подсчёт веса конкретного модуля, заданного пользователем, и другие.

4. СРАВНЕНИЕ ПРОГРАММНЫХ ПАКЕТОВ

Три рассмотренных разработки обладают своими достоинствами и недостатками. Система TELiS больше отличается от остальных: в ней не используется модульный подход, а рассматривается каждый фактор по отдельности. Основной недостаток такого подхода в том, что не учитывается расстояние между сайтами связывания на промоторе. Отчасти эта проблема решена: можно выполнить расчёт либо для 300 первых нуклеотидов промотора, либо для 600 или 1200. Однако этого параметра недостаточно, сайты связывания конкретного комплекса могут располагаться, скажем, с 200-го по 400-й нуклеотид. С другой стороны, такое упрощение сводит задачу к линейной сложности и позволяет быстро найти факторы, которые потенциально могут присутствовать в оптимальном модуле. Кроме того, TELiS не ограничивает число найденных факторов, тогда как TOUCAN и CMA ищут

модули с конкретным числом факторов, хотя это число может быть заранее неизвестно, и придётся прогонять алгоритм, меняя эту величину.

Система TOUCAN реализует алгоритм, который находит заведомо лучший модуль. Несмотря на то, что он выполняется долго и требует много памяти, иногда может быть полезно получить гарантированно оптимальный результат. Хотя, как показывает практика, генетический алгоритм, реализованный и в TOUCAN, и в CMA, также даёт хорошие результаты за гораздо меньшее время.

Система TOUCAN имеет некоторые дополнительные возможности, отсутствующие в CMA: возможность искать модули с повторяющимися факторами или запрет поиска модулей, где сайты факторов перекрываются. Кроме того, в TOUCAN для пар факторов введена степень сходства, и можно запретить наличие в модуле факторов со степенью сходства выше некоторой заданной величины.

TELiS и TOUCAN реализованы на Java, что упрощает переносимость этих систем, но значительно снижает быстродействие. В частности, это стало причиной того, что поиск сайтов связывания разработчики TELiS выполнили заранее для всевозможных генов и матриц, упомянув, что эти вычисления могут занять несколько дней. Та же процедура в CMA для аналогичных объёмов входных данных занимает около часа. Реально же в каждом эксперименте не требуется информация про абсолютно все гены, поэтому поиск сайтов для конкретного эксперимента может занимать меньше минуты. Следует также заметить, что полученная база данных TELiS не содержит информации о положении сайтов связывания, а содержит лишь количество сайтов определённого фактора для промотора каждого гена. Это делает невозможным использование её в других разработках, где необходимо учитывать расстояние между сайтами.

Трудно сравнить быстродействие TOUCAN и CMA, так как реальные вычисления, выполняемые TOUCAN, производятся на кластере разработчиков. Тем не менее, CMA демонстрирует вполне удовлетворительное быстродействие на одном компьютере класса Pentium III, необходимости переносить вычисления на кластер нет. В настоящий момент программа CMA используется в рамках работы по гранту INTAS «Построение модели регуляторной сети в нормальном и патологическом состоянии для предсказания потенциальных противораковых фармакологических агентов для ключевых молекул» для выявления комплексов факторов, регулирующих работу генов клеточного цикла в зависимости от его стадии.

ЗАКЛЮЧЕНИЕ

В статье проведён обзор различных систем, предназначенных для поиска наборов транскрипционных факторов. Каждая из рассмотренных систем обладает своими достоинствами и недостатками и может быть полезна в определённых ситуациях. Система TELiS полезна, если необходимо узнать, какие факторы повышают экспрессию в данном эксперименте, и не так важно, рядом расположены их сайты или нет. Пакет TOUCAN позволяет найти оптимальный модуль для небольших наборов данных, а также выполнить генетический алгоритм. Систему CMA можно использовать для достаточно больших наборов входных данных, выполнять поиск булева модуля и решать смежные задачи типа кластеризации набора промоторов.

Дальнейшее исследование этой области неизбежно. Планируется более точное математическое моделирование биологических процессов, происходящих при регуляции генов, и программная реализация этих моделей. Также будет исследована возможность усовершенствования генетического алгоритма с целью ускорения поиска модулей с большим числом факторов, когда количество возможных модулей резко возрастает. Кроме того, планируется ввести новые статистические тесты для оценки надёжности полученного результата.

СПИСОК ЛИТЕРАТУРЫ

1. **Velculescu V. E., Zhang L., et al.** Serial analysis of gene expression // Science. — 1995. — N 270 (5235). — P. 484–487.
2. **Aerts S., Thijs G., Coessens B., et al.** TOUCAN: Deciphering the Cis-Regulatory Logic of Coregulated Genes // Nuclear Acids Research. — 2003. — Vol. 31, N 6 — P. 1753–1764.
3. **Cole S., Yan W., Galic Z., et al.** Expression-based monitoring of transcription factor activity: The TELiS database // Bioinformatics. — 2005. — N 21 (6). — P. 803–810.
4. **Konovalova T., Cheremushkin E., Beschastnov E., Kel. A.** Applicating of the metropolis algorithm to reveal composite modules in promoters of eukaryotic genes // Proc. European Conf. on Computational Biology, Paris, France, September 2003. — Paris, 2003. — P. 447–448.
5. **Aerts S., Van Loo P., Thijs G., et al.** Computational detection of cis-regulatory modules // Bioinformatics. — 2003. — Vol. 19, Suppl. 2. — P. ii5–ii14.
6. **Land A.H., Doig A.G.** An automatic method of solving discrete programming problems // Econometrica. — 1960. — Vol. 28 — P. 497–520.

7. **Aerts S, Van Loo P, Moreau Y, De Moor B.** A genetic algorithm for the detection of new cis-regulatory modules in sets of coregulated genes // Bioinformatics. — 2004. — Vol. 20, N 12. — P. 1974–1976.
8. **Kanji G. K.** 100 Statistical Tests. — London, Sage, 1999. — 224 p.

А.А. Винокуров, И.В. Ильин, Ф.А. Мурзин, Д.Ф. Семич*

РАСЧЕТ КОЭФФИЦИЕНТА НЕФТЕНАСЫЩЕННОСТИ ПО РЕЗУЛЬТАТАМ ЯДЕРНОГО КАРОТАЖА

ВВЕДЕНИЕ

В статье описаны алгоритмы для расчета нефтенасыщенности по данным ядерного каротажа. Рассмотрены два метода расчета нефтенасыщенности.

Первый метод основан на применении кросс-плот зависимости аналитических параметров C/O, Ca/Si и коэффициента пористости, которая получена на базе исследований моделей различной литологии, пористости и насыщенности (метрологический центр Западно-Сибирской Корпорации ТюменьПромГеофизика, г. Мегион, Ханты-Мансийский Автономный Округ, Россия). Вариант данного метода рассматривается также в работе [1].

Второй метод представляет собой модифицированный вариант классического метода “Дельта C/O”, созданного и описанного фирмой Halliburton [2].

Каждый из них может базироваться на использовании спектров ГИРЗ (Гамма Излучения Радиоактивного Захвата) или ГИНР (Гамма Излучения Наведенной Радиоактивности). Поэтому можно считать, что исследуются четыре метода.

Алгоритмы, рассматриваемые в данной статье, реализованы в программе OilTemper, которая передана заказчику.

1. РАСЧЕТ МЕТОДОМ “КРОСС-ПЛОТ”

В данном разделе описывается алгоритм обработки данных методом кросс-плот. Данный метод расчета нефтенасыщенности основан на использовании кросс-плота, построенного по результатам модельных работ прибором ИНГК-С-95 (C/O-каротаж) в метрологическом центре на моделях

* murzin@academ.org, D.Semich@ftc.ru, ilyin@megasignal.com

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 05-08-33448).

пластов различной пористости, литологии и насыщенности. Кросс-плот описывает взаимную зависимость аналитических параметров C/O, Ca/Si (отдельно по ГИНР и по ГИРЗ), пористости и нефтенасыщенности.

Итоговая нефтенасыщенность определяется путем сопоставления аналитических параметров Ca/Si, C/O и пористости по скважине с соответствующими параметрами для моделей пластов. Дополнительно, для компенсации влияния неучтенных в кросс-плот факторов (наличие обсадной колонны и т.п.), производится так называемая калибровка метода — привязка к опорным пластам с известной нефтенасыщенностью.

1.1. Построение кросс-плота по результатам модельных работ

Расчет методом кросс-плот требует в качестве входных данных три аналитических параметра: **C/O** (по спектру ГИНР), **Ca/Si** (по спектру ГИРЗ или по спектру ГИНР) и **пористость** (в процентах). В программной реализации предполагается, что параметры сведены по глубине и находятся в одном или нескольких LAS-файлах.

С математической точки зрения получается, что мы работаем в трехмерном пространстве. Будем откладывать значения **Ca/Si** на горизонтальной оси $0x$, значения **C/O** — на вертикальной оси $0y$. Ось $0z$, перпендикулярная плоскости $0xy$, будет соответствовать пористости.

По результатам модельных работ для высоких пористостей $z_{A1}, z_{A2}, z_{A3}, z_{A4}$ порядка 32–35% получены следующие 4 точки:

(x_{A1}, y_{A1}, z_{A1}) — точка, соответствующая водонасыщенному песчанику,

(x_{A2}, y_{A2}, z_{A2}) — точка, соответствующая водонасыщенному известняку,

(x_{A3}, y_{A3}, z_{A3}) — точка, соответствующая нефтенасыщенному песчанику,

(x_{A4}, y_{A4}, z_{A4}) — точка, соответствующая нефтенасыщенному известняку.

Для низких пористостей $z_{C1}, z_{C2}, z_{C3}, z_{C4}$ порядка 15–18% получены аналогичные 4 точки:

(x_{C1}, y_{C1}, z_{C1}) — точка, соответствующая водонасыщенному песчанику,

(x_{C2}, y_{C2}, z_{C2}) — точка, соответствующая водонасыщенному известняку,

(x_{C3}, y_{C3}, z_{C3}) — точка, соответствующая нефтенасыщенному песчанику,

(x_{C4}, y_{C4}, z_{C4}) — точка, соответствующая нефтенасыщенному известняку.

Отметим, что точки, соответствующие известняку, лежат правее точек, соответствующих песчанику, т.е. для них отношение **Ca/Si** больше. Точки,

соответствующие нефтенасыщенным образцам, лежат выше точек, соответствующих водонасыщенным образцам, т.е. для них отношение **C/O** больше.

Соединяя соответствующие точки отрезками прямых линий, получим фигуру, изображенную ниже.

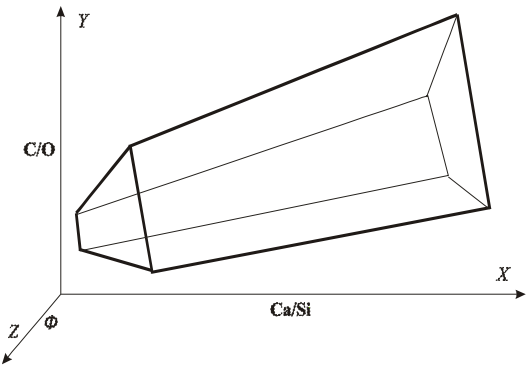


Рис. 1. Номограмма для определения нефтенасыщенноси по отношениям **C/O** и **Ca/Si** в терригенных и карбонатных коллекторах

Можно также отметить, что при фиксированной пористости четырехугольник, получаемый в сечении, “слегка сужается” при уменьшении значений **Ca/Si.** Кроме того, фигура “резко сужается” при уменьшении значений пористости.

Рассматривались также 4 модельные точки $(x_{Bi}, y_{Bi}, z_{Bi}), i = 1,\ldots,4$, соответствующие средним пористостям, примерно 23–25%. В этом случае в итоге получаем 12 точек и, соответственно, фигуру, “склеенную” из двух частей. Но подобное усложнение мало повлияло на результаты, о которых пойдет речь ниже, поэтому ограничились использованием восьми модельных точек.

1.2. Вычисление индекса нефтенасыщенности по кросс-плоту

Предположим, что зафиксирована пористость $z = p_0$. Мы также считаем, что $p_0 \geq 12\ \%$.

Рассмотрим четыре прямые линии L_i, проходящие через пары точек (x_{Ai}, y_{Ai}, z_{Ai}) и (x_{Ci}, y_{Ci}, z_{Ci}), соответственно. Обозначим (x_i, y_i, p_0) координаты пересечения данных прямых с плоскостью $z = p_0$. Соединяя их соответствующими отрезками прямых, получаем кросс-плот, изображенный на рис. 2, который расположен в плоскости.

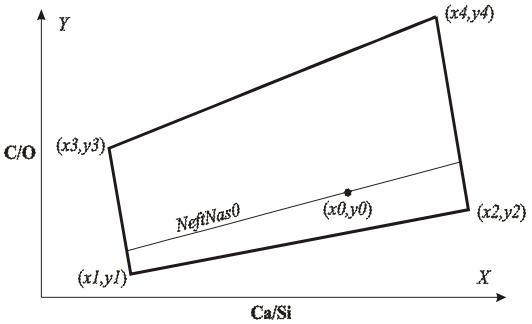


Рис. 2. Кросс-плот, полученный в сечении $z = p_0$

Предположим, что $0 \le \lambda \le 1$, и пусть

$$x_L = (1-\lambda)x_1 + \lambda x_3;$$
$$y_L = (1-\lambda)y_1 + \lambda y_3;$$
$$x_R = (1-\lambda)x_2 + \lambda x_4;$$
$$y_R = (1-\lambda)y_2 + \lambda y_4.$$

Тогда точка (x_L, y_L) делит отрезок, соединяющий точки (x_1, y_1) и (x_3, y_3), в пропорции $\lambda/(1-\lambda)$, и то же самое справедливо для (x_R, y_R) и отрезка, соединяющего точки (x_2, y_2) и (x_4, y_4).

Метод вычисления индекса нефтенасыщенности по кросс-плоту состоит в следующем. Допустим, что в результате измерений получена точка с координатами (x_0, y_0, p_0). Рассматриваем плоский кросс-плот

в сечении $z = p_0$. Далее находим λ такое, что точка (x_0, y_0) лежит на отрезке, соединяющем точки (x_L, y_L) и (x_R, y_R). Полученное λ называется индексом нефтенасыщенности (рис. 2).

Здесь предполагается, что точка (x_0, y_0) лежит внутри кросс-плота. В первоначальном варианте программы полагалось $\lambda = 0$, если точка лежит ниже кросс-плота, и $\lambda = 1$, если точка лежит выше кросс-плота. Но потом, чтобы лучше видеть динамику индекса нефтенасыщенности по глубине, стали считать, что λ меняется от –1 до +2. Соответствующая геометрическая интерпретация очевидна. Грубо говоря, снизу и сверху достраиваются аналогичные кросс-плоты, и мы анализируем местоположение точки уже в более широком кросс-плоте.

Задача нахождения λ решается численно. Проходим все его значения от нуля до единицы с шагом $\Delta\lambda = 0.001$. На каждом шаге вычисляем соответствующие точки (x_L, y_L) и (x_R, y_R). Определяем расстояние от точки (x_0, y_0) до прямой, соответствующей данному λ, и берем то значение λ, для которого это расстояние минимально (рис. 3).

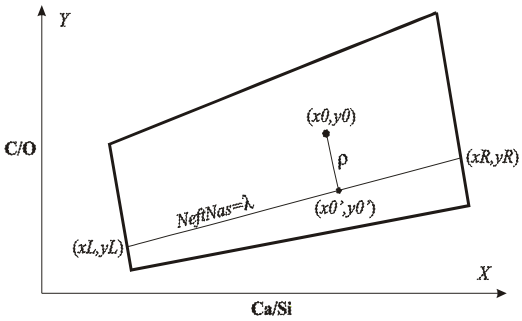


Рис. 3. Итерационный шаг определения λ, определяется расстояние ρ

Точка (x, y) лежит на прямой, соединяющей точки (x_L, y_L) и (x_R, y_R), и соответственно имеем пропорцию, так называемое “уравнение в отрезках”

$$\frac{x - x_L}{x_R - x_L} = \frac{y - y_L}{y_R - y_L}.$$

Далее имеем, что $(x - x_0, y - y_0)$ ортогонален вектору $(x_R - x_L, y_R - y_L)$, и соответствующее скалярное произведение

$$(x - x_0)(x_R - x_L) + (y - y_0)(y_R - y_L) = 0.$$

Опуская выкладки, напишем, что решение имеет вид:

$$x = \frac{x_B}{x_A}; \; x_B = x_{B1} + x_{B2};$$
$$x_A = (x_R - x_L)^2 + (y_R - y_L)^2;$$
$$x_{B1} = x_0 (x_R - x_L)^2;$$
$$x_{B2} = [x_L (y_R - y_L) - (y_L - y_0)(x_R - x_L)] * (y_R - y_L).$$

$$y = \frac{(x - x_L)(y_R - y_L)}{x_R - x_L} + y_L$$
$$\rho^2 = (x - x_0)^2 + (y - y_0)^2.$$

Напомним еще раз, что выбирается $\lambda_{\min}$, которому соответствует минимальное значение ρ.

1.3. Коэффициент и индекс нефтенасыщенности

Величину λ, которую мы считаем, называем индексом нефтенасыщенности. Рассмотрение скважинных данных, полученных альтернативными методами, а также литературных источников, убедило нас, что эта величина и есть реальная нефтенасыщенность.

Однако для данных на моделях, в которых использован спирт вместо нефти, получается, что реальная нефтенасыщенность нелинейно зависит от индекса нефтенасыщенности.

Например, для спектра ГИРЗ при $\lambda = 0.4$ нефтенасыщенность оказывается равной 0.1, или в процентном выражении — 10%, а при $\lambda = 0.6$ нефтенасыщенность оказывается равной 0.25, или соответственно — 25%,

Имея набор моделей с разной насыщенностью флюидом (в данном случае — спиртом), иначе говоря, набор экспериментальных точек, можно пытаться приблизить эту зависимость некоторой кривой.

Нами эта зависимость была представлена в виде трех кусков парабол. Оба их варианта, для спектров ГИРЗ и ГИНР, представлены ниже на графиках.

Для спектра ГИРЗ зависимость нефтенасыщенности от индекса имеет следующий вид:

$$NeftNas(x) = \begin{cases} 0.8333333333 * x^2 - 0.08333333333 * x, & \text{if } x \le 0.4, \\ 1.35416666665 * x^2 - 0.604166666665 * x + 0.125, & \text{if } 0.4 < x < 0.6, \\ 1.875 * x^2 - 1.125 * x + 0.25, & \text{if } x \ge 0.6. \end{cases}$$

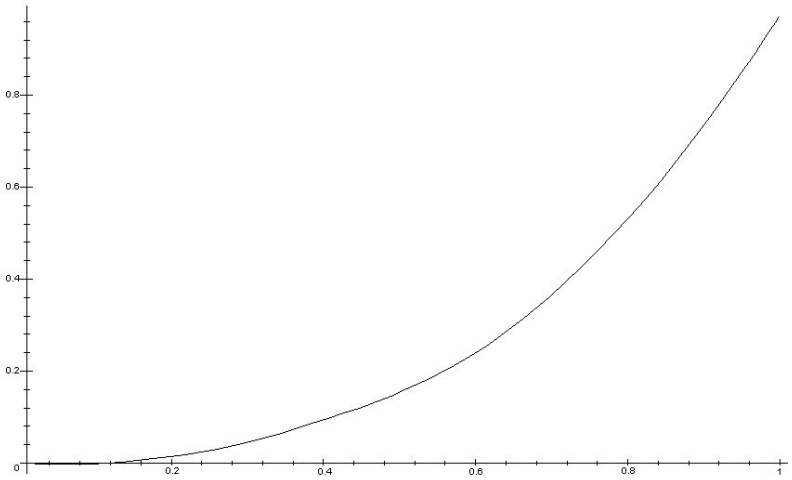


Рис. 4. График зависимости нефтенасыщенности от индекса для спектра ГИРЗ

Для спектра ГИНР аналогичная зависимость нефтенасыщенности от индекса имеет несколько иной вид:

$$NeftNas(x) = \begin{cases} 0.2777777778 * x^2 + 0.25 * x, & \text{if } x \le 0.3, \\ 1.121031746 * x^2 - 0.5089285710 * x + 0.1517857142, & \text{if } 0.3 < x < 0.6, \\ 1.964285714 * x^2 - 1.267857142 * x + 0.3035714283, & \text{if } x \ge 0.6. \end{cases}$$

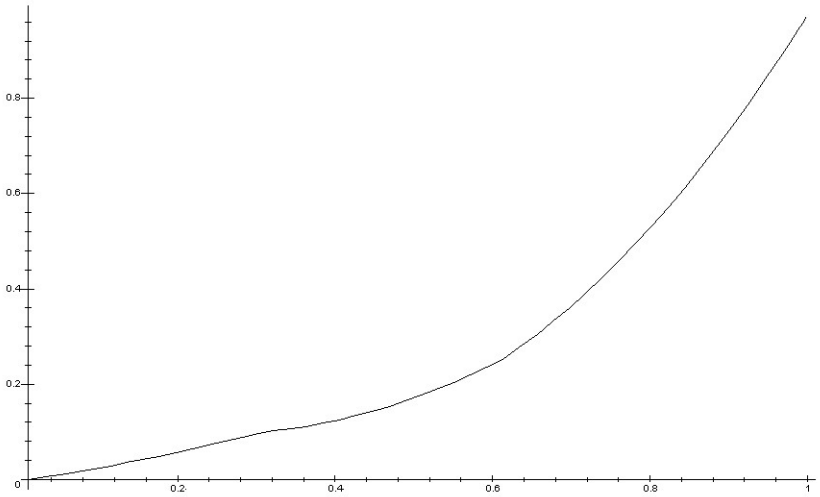


Рис. 5. График зависимости нефтенасыщенности от индекса для спектра ГИНР

1.4. Калибровка метода кросс-плот

Калибровка метода кросс-плот необходима для приведения скважинных данных к данным на моделях пластов, учета влияния скважинных условий и режимов работы прибора при проведении регистрации.

Калибровка необходима для осуществления количественного расчета нефтенасыщенности. Она заключается в указании приблизительных значений нефтенасыщенности на так называемых опорных пластах.

Мы считаем, что кросс-плот может перемещаться вдоль вертикальной оси **C/O** и сжиматься (растягиваться).

Чтобы определить величину параллельного переноса вдоль вертикальной оси **C/O** и коэффициента сжатия, предложено использовать калибровку по двум пластам. Оператор выделяет два слоя — с малой нефтенасыщенностью (в идеале, водонасыщенный) и с большой нефтенасыщенностью — и указывает их предполагаемую нефтенасыщенность (например, 0% и 50%). Из этих данных программа вычисляет величину параллельного переноса вдоль вертикальной оси **C/O** и коэффициент сжатия кросс-плота. Дальше мы их используем для обработки всех остальных данных по скважине.

Опишем более подробно алгоритм калибровки.

1.4.1. Первый этап калибровки — параллельный перенос данных

Во-первых, заметим, что на опорных пластах в качестве эталонных значений параметров C/O Ca/Si и пористости будет взята точка — среднее арифметическое точек соответствующих кривых по указанному интервалу глубин опорного пласта.

Пусть теперь (x_0, y_0, p_0) — данные, усредненные по пласту с малой нефтенасыщенностью.

Вычисляем $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ — координаты углов в плоскости $z = p_0$.

Полагаем, $\lambda = NeftNas_0$ — заданная оператором нефтенасыщенность, и вычисляем соответствующие точки на левой и правой сторонах кросс-плота:

$$\begin{aligned} x^* &= (1-\lambda)x_1 + \lambda x_3; \\ y^* &= (1-\lambda)y_1 + \lambda y_3; \\ x^{**} &= (1-\lambda)x_2 + \lambda x_4; \\ y^{**} &= (1-\lambda)y_2 + \lambda y_4. \end{aligned}$$

Находим координаты x_0', y_0', полагая

$$\begin{cases} x_0' = x_0 \\ \dfrac{x_0' - x^*}{x^{**} - x^*} = \dfrac{y_0' - y^*}{y^{**} - y^*} \end{cases} \rightarrow y_0' = y^* + \frac{(x_0' - x^*)(y^{**} - y^*)}{x^{**} - x^*} \ .$$

Таким образом, поправка на координату y, т.е. на C/O равна

$$\Delta y = y_0' - y_0.$$

Иначе говоря, точка (x_0, y_0) размещалась не там, где надо, осуществляя параллельный перенос, мы ее перемещаем на линию нефтенасыщенности, указанную оператором.

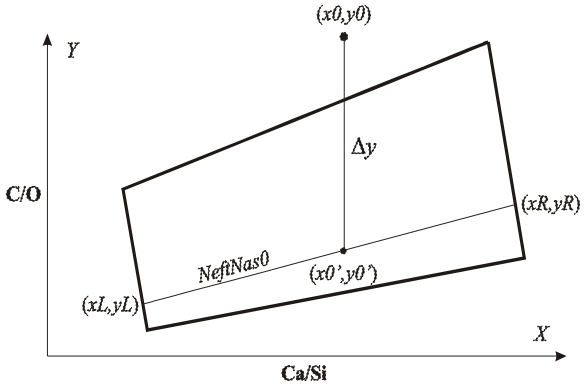


Рис. 6. Калибровка по первому пласту, вычисление параметра Δy

1.4.2. Второй этап калибровки — сжатие/растяжение кросс-плота

В нашем распоряжении имеются следующие данные:

kn_0 — коэффициент нефтенасыщенности на пласте с малой нефтенасыщенностью, заданный оператором;

kn_1 — коэффициент нефтенасыщенности на пласте с большой нефтенасыщенностью, заданный оператором;

kn_{Calc} — вычисленный коэффициент нефтенасыщенности на пласте с большой нефтенасыщенностью;

Обозначим

$$\delta = kn_1 - kn_0;$$
$$\Delta = kn_{Calc} - kn_0.$$

Очевидно, что

$$kn_1 = kn_0 + \Delta\left(\frac{\delta}{\Delta}\right)$$

или, что то же самое,

$$kn_1 = kn_0 + (kn_{Calc} - kn_0) * \kappa \text{, где } \kappa = \frac{\delta}{\Delta}.$$

Отсюда ясной становится идея метода. Используя опорные пласты, вычисляем величину κ, это и есть искомый коэффициент сжатия. Далее для каждой конкретной глубины мы пересчитываем значение коэффициента нефтенасыщенности, используя предпоследнюю формулу, полагая

$$kn = kn_0 + (kn_{Calc} - kn_0) * \kappa .$$

Заметим, что опорный пласт “станет туда, куда положено”. Возникающая ситуация изображена на рис. 7.

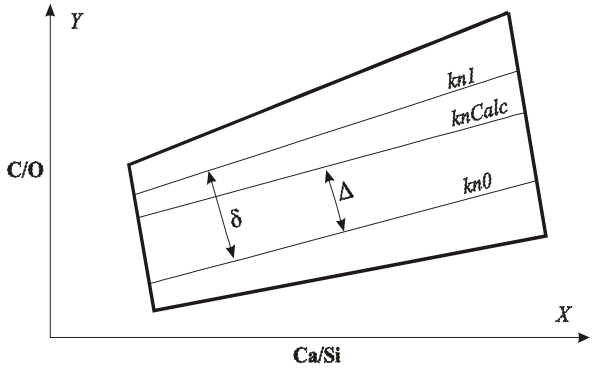


Рис. 7. Калибровка по второму пласту, вычисление параметра $\kappa = \frac{\delta}{\Delta}$

1.4.3. Коррекция угла поворота кросс-плота

Рассмотрение данных со скважин с помощью специальной программы, позволяющей увидеть точки в трехмерном пространстве, показало, что кросс-плот может поворачиваться в пространстве.

С одновременным ростом карбонатности и нефтенасыщенности правая сторона кросс-плота резко поднимается вверх.

Фактически необходимо корректно выставить “линию воды”, или, если работать в трехмерном пространстве (учесть еще пористость), то “плоскость воды”. Один из вариантов решения этой задачи предложен ниже, но отметим, что предложенные алгоритмы требуют дальнейшего совершенствования.

Допустим сначала, что первый калибровочный пласт чисто водяной, т.е. нефтенасыщенность равна нулю, (x_0, y_0, p_0) — данные, усредненные по данному пласту. Как и раньше, переходим к сечению $z = p_0$. Далее, используя первый этап калибровки (параллельный перенос данных), можно в данном сечении точку (x_0, y_0) переместить на линию воды кросс-плота. Поэтому для простоты сразу предполагаем, что (x_0, y_0) расположена на линии воды.

Предположим также, что в данном сечении имеется достаточно много других точек $(x_i, y_i), i = 1, \ldots, n$, расположенных вблизи линии воды.

Необходимо восстановить эту линию так, чтобы точка (x_0, y_0) лежала на ней и достаточно хорошо приближала набор точек $(x_i, y_i), i = 1, \ldots, n$. Возникающая ситуация изображена на рис. 8.

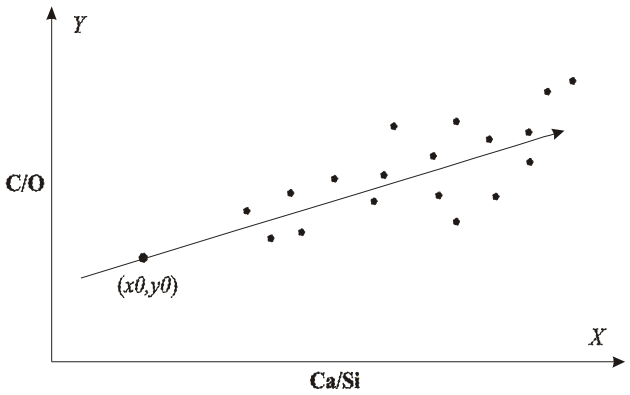


Рис. 8. Нахождение линии воды

Данная задача может быть решена с помощью метода наименьших квадратов.

Ищем прямую в виде $f(x) = a + bx$ с неизвестными коэффициентами a, b. Ввиду того, что (x_0, y_0) лежит на данной прямой, имеем $y_0 = a + bx_0$.

Отсюда получаем выражение для первого коэффициента

$$a = y_0 - bx_0 .$$

Далее минимизируем функционал

$$\Phi(b) = \sum_{i=1}^{n} (f(x_i) - y_i)^2 .$$

Запишем более подробно

$$\Phi(b) = \sum_{i=1}^{n} (b(x_i - x_0) + y_0 - y_i)^2 .$$

Приравнивая производную нулю, получаем

$$\frac{\partial \Phi}{\partial b} = 0 ,$$

или соответственно

$$2\sum_{i=1}^{n} (b(x_i - x_0) + y_0 - y_i)(x_i - x_0) = 0 .$$

Делая очевидные преобразования, получаем

$$\sum_{i=1}^{n} b(x_i - x_0)^2 = \sum_{i=1}^{n} (y_0 - y_i)(x_i - x_0) = 0 ,$$

и в итоге находится второй коэффициент

$$b = \frac{\sum_{i=1}^{n} (y_i - y_0)(x_i - x_0)}{\sum_{i=1}^{n} (x_i - x_0)^2} .$$

Далее мы имеем две линии воды: новая линия воды, которую мы нашли с помощью описанной выше процедуры, и старая линия воды кросс-плота. Соответственно, их уравнения будут

$$f(x) = a + bx,$$
$$g(x) = \bar{a} + \bar{b}x,$$
$$\bar{a} = y_1 - x_1 * tg\theta,$$
$$\bar{b} = tg\theta,$$
$$tg\theta = \frac{y_2 - y_1}{x_2 - x_1},$$

где (x_1, y_1), (x_2, y_2) — нижние угловые точки кросс-плота.

Угол между данными прямыми равен

$$\varphi = \arccos\left|\frac{1 + b\bar{b}}{\sqrt{1 + b^2}\sqrt{1 + \bar{b}^2}}\right|.$$

Далее, вместо поворота кросс-плота можно “повернуть все данные” в обратном направлении вокруг точки (x_0, y_0) с целью перемещения их в старый кросс-плот (рис. 9). Соответствующая формула приведена ниже:

$$\begin{cases} x' = (x - x_0)\cos\varphi - (y - y_0)\sin\varphi, \\ y' = (x - x_0)\sin\varphi + (y - y_0)\cos\varphi. \end{cases}$$

Калибровка по второму пласту (растяжение/сжатие) производится после поворота.

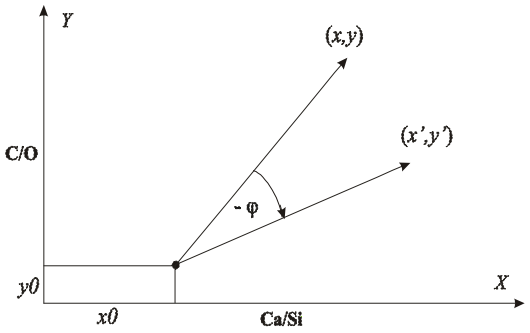


Рис. 9. Преобразование “поворота данных” вокруг точки (x_0, y_0) с целью перемещения их в старый кросс-плот

Здесь возникает ряд вопросов.

Если множество точек (x_i, y_i) действительно образует некоторое вытянутое множество, то прямая линия находится достаточно корректно, а если оно имеет вид шарового скопления, то прямая не может быть выставлена корректно. "Вытянутость" данного множества, как правило, означает присутствие точек с различной карбонатностью. Сейчас в программе наряду с автоматическим определением угла поворота кросс-плота предусмотрена также возможность его ручной корректировки.

Далее, в данном сечении может оказаться слишком мало точек. Тогда можно некоторое количество точек спроектировать в данное сечение из близких сечений. Проектировать их можно обычным способом, а можно учесть сужение кросс-плота.

Наконец, как можно действовать, если первый калибровочный пласт не с нулевой нефтенасыщенностью. В этом случае задача модифицируется следующим образом.

С помощью параллельного переноса перемещаем точку на линию в кросс-плоте, соответствующую $NeftNas_0$.

Далее для простоты считаем, что точка (x_0, y_0) уже на данной линии. Вычисляем расстояние от нее до линии воды кросс-плота. Обозначим это расстояние ρ_0.

Аналогично ищем прямую, приближающую набор точек, и такую, что расстояние от (x_0, y_0) до данной прямой равнялось бы ρ_0, т.е. если вычисление индекса нефтенасыщенности провести для нового повернутого кросс-плота, то для калибровочной точки (x_0, y_0) это значение не изменится.

В таком виде задача решаема, но довольно громоздка. В действительности могут быть предложены несколько более простых вариантов, но необходимы дополнительные исследования, чтобы выяснить, насколько они целесообразны.

2. РАСЧЕТ ПО МЕТОДУ ДЕЛЬТА С/О

2.1. Входные данные

Входными данными являются аналитические параметры, рассчитанные по моделям. Используем обозначения, принятые в компании Halliburton.

Для каждой из моделей указывается:

Параметр	Комментарий
Номер модели	В соответствии с нумерацией в метрологическом центре ТПГ
Отношение Ca/Si по ГИНР (Rc/o)	
Отношение Ca/Si по ГИРЗ (Rca/si)	
Пористость в долях единицы (Ф)	
Нефтенасыщенности в долях единицы (So)	
Объемная доля известняка (Vls)	=0 для песчаников, =1 для известняков

Кроме того, указываются атомные плотности некоторых веществ:

Вещество	Плотность
Углерод в нефти Nch	42.9×10^{23} /см^3
Углерод в известняке Ncca	16.2×10^{23} /см^3
Кислород в воде Now	33.3×10^{23} /см^3
Кислород в известняке Noca	48.6×10^{23} /см^3
Кислород в песчанике Nosi	53.0×10^{23} /см^3

2.2. Математическая модель параметра «COIR»

Параметр COIR — отношение счета в окне углерода к счету в окне кислорода по спектру ГИНР (COIR=Rc/o).

Общий вид выражений для модели:

$$COIR = \alpha 1 * \frac{Yc}{Yo} + \beta 1 * (1 - \Phi) + \gamma 1, \qquad [1]$$

здесь $\alpha 1$, $\beta 1$, $\gamma 1$ *– искомые коэффициенты,*

$$\frac{Yc}{Yo} \approx \frac{\Phi * So * Nch + (1 - \Phi) * Ncca}{\Phi * (1 - So) * Now + (1 - \Phi) * Nosi}. \qquad [1.0]$$

Уравнение 1.

Модель Вода (Ф = 1, So = 0)

$$COIRw = \alpha 1 * 0 + \beta 1 * 0 + \gamma 1 . \qquad [1.1]$$

Уравнение 2.

Модель X — водонасыщенный песчаник (Ф = Фx, So = 0)

$$COIRx = \alpha 1 * \frac{(1 - \Phi x) * Ncca}{\Phi x * (1 - So) * Now + (1 - \Phi x) * Nosi} + \beta 1 * (1 - \Phi x) + \gamma 1 . \qquad [1.2]$$

Уравнение 3.
Модель Y — нефтенасыщенный песчаник (Ф = Фу, So = 1)

$$COIRy = \alpha 1 * \frac{\Phi y * So * Nch + (1 - \Phi y) * Ncca}{(1 - \Phi y) * Nosi} + \beta 1 * (1 - \Phi y) + \gamma 1 . \quad [1.3]$$

Система $\begin{cases} [1.1] \\ [1.2] \\ [1.3] \end{cases}$ решается относительно $\alpha 1, \beta 1, \gamma 1$.

2.3. Математическая модель параметра «LIRI»

Параметр LIRI — отношение счета в окне кальция к счету в окне кремния по спектру ГИНР (LIRI=Rca/si).
Общий вид выражений для модели:

$$LIRI = (\alpha 2 * Vls + \beta 2) * (1 - \Phi) + \gamma 2,$$
здесь $\alpha 2, \beta 2, \gamma 2$ *– искомые коэффициенты.*

Уравнение 1.
Модель Вода (Ф = 1, So = 0, Vls = 0)

$$LIRIw = (\alpha 2 * 0 + \beta 2) * 0 + \gamma 2. \quad [2.1]$$

Уравнение 2.
Модель X — водонасыщенный/нефтенасыщенный песчаник
(Ф = Фx, So = 0/1, Vls = 0)

$$LIRIx = (\alpha 2 * 0 + \beta 2) * (1 - \Phi x) + \gamma 2. \quad [2.2]$$

Уравнение 3.
Модель Y — водонасыщенный/нефтенасыщенный известняк
(Ф = Фу, So = 0/1, Vls = 1)

$$LIRIy = (\alpha 2 + \beta 2) * (1 - \Phi y) + \gamma 2. \quad [2.3]$$

Система $\begin{cases} [2.1] \\ [2.2] \\ [2.3] \end{cases}$ решается относительно $\alpha 2, \beta 2, \gamma 2$.

2.4. Получение параметра «ΔC/O»

Уравнение 1.
Модель Вода (Ф = 1, So = 0, Vls = 0)

$$A * LIRI(\Phi = 1, So = 0, Vls = 0) + B + C = COIR(\Phi = 1, So = 0). \quad [3.1]$$

Уравнение 2.
Водонасыщенный песчаник с нулевой пористостью
(Ф = 0, So = 0, Vls = 0)

$$A * LIRI(\Phi = 0, So = 0, Vls = 0) + C = COIR(\Phi = 0, So = 0). \quad [3.2]$$

Уравнение 3.
Водонасыщенный известняк с нулевой пористостью
(Ф = 0, So = 0, Vls = 1)

$$A * LIRI(\Phi = 0, So = 0, Vls = 1) + C = COIR(\Phi = 0, So = 0). \quad [3.3]$$

Система $\begin{cases}[3.1]\\[3.2]\\[3.3]\end{cases}$ решается относительно A, B, C .

В итоге получаем:

$\Delta C / O = COIR - A * LIRI - B * \Phi - C + k$,
где k *выбирается по известному водосодержащему пласту*,
чтобы минимизировать $\Delta C / O$ *на нем*.

Согласно публикациям компании Halliburton, в настоящее время параметр k оператор подбирает экспериментально, опираясь на интуицию.

Дальше мы покажем, как распространить идею калибровки по двум пластам на метод Дельта С/О так, что данный параметр может быть вычислен автоматически.

Фактически мы усовершенствуем метод Дельта С/О.

2.5. Получение параметра «нефтенасыщенность»

Основное соотношение, которое используется, следующее:

$$\Delta C / O = \alpha 1 * \frac{Yc}{Yo}. \quad [4]$$

Подставим в [4] выражение [1.0] и выразим искомый параметр S_0, т.е. в итоге имеем два уравнения

$$\Delta C / O = \alpha 1 * \frac{Yc}{Yo};$$
$$\frac{Yc}{Yo} \approx \frac{\Phi * So * Nch + (1-\Phi) * Ncca}{\Phi * (1-So) * Now + (1-\Phi) * ((1-Vls) Nosi * Vls * Noca)}.$$

Введем обозначения:

$$A1 = \Phi * Nch; \quad B1 = (1-\Phi) * Ncca;$$
$$C1 = \Phi * Now; \quad D1 = (1-\Phi) * ((1-Vls) Nosi * Vls * Noca);$$
$$\Delta C / O = \alpha 1 * \frac{A1 * So + B1}{C1 * (1-So) + D1}.$$

Тогда очевидно, что

$$\Delta C / O * (C1 * (1-So) + D1) = \alpha 1 * (A1 * So + B1).$$

Далее имеем

$$\Delta C / O * (C1 + D1) - \alpha 1 * B1 = So * (\alpha 1 * A1 + \Delta C / O * C1),$$

и отсюда получаем

$$So = \frac{\Delta C / O * (C1 + D1) - \alpha 1 * B1}{\alpha 1 * A1 + \Delta C / O * C1}.$$

Нетрудно видеть, что в выражение для S_0 входит неизвестная величина Vls.

Величину Vls можно найти из выражения для

$$LIRI = (\alpha 2 * Vls + \beta 2) * (1-\Phi) + \gamma 2.$$

В итоге получаем равенство

$$Vls = \left(\frac{LIRI - \gamma 2}{1-\Phi} \right) / \alpha 2.$$

В случае терригенных отложений $(Vls = 0)$ используют упрощенную формулу

$$\Delta C / O \approx \alpha 1 * \frac{\Phi * So * Nch}{\Phi * (1 - So) * Now + (1 - \Phi) * Nosi} .$$

Данная формула может быть переписана в виде

$$\Delta C / O = \alpha 1 * \frac{A1 * So}{C1 * (1 - So) + D1}.$$

Отсюда получаем

$$\Delta C / O * (C1 * (1 - So) + D1) = \alpha 1 * A1 * So.$$

Далее, делая очевидные преобразования, имеем

$$\Delta C / O * (C1 + D1) = So * (\alpha 1 * A1 + \Delta C / O * C1),$$

и отсюда

$$So = \frac{\Delta C / O * (C1 + D1)}{\alpha 1 * A1 + \Delta C / O * C1}.$$

2.6. Калибровка по пластам

Сначала рассмотрим более простой случай, а именно, калибровку для терригенных отложений.

Во-первых, имеем вышеприведенную формулу для нефтенасыщенности So.

Обозначим $\overline{COIR}, \overline{LIRI}, \overline{\Phi}$ — усредненные значения $COIR, LIRI, \Phi$ по пласту с малой нефтенасыщенностью.

Соответственно получаем усредненное значение

$$\Delta C / O = \overline{COIR} - A * \overline{LIRI} - B * \overline{\Phi} - C + k.$$

Введем обозначение

$$T = \overline{COIR} - A * \overline{LIRI} - B * \overline{\Phi} - C,$$

тогда имеем равенство

$$\Delta C / O = T + k.$$

Формулу для вычисления So можно переписать в виде

$$So = \frac{(T + k) * (C1 + D1)}{\alpha 1 * A1 + (T + k) * C1}.$$

Далее, делая очевидные преобразования, имеем

$$So * (\alpha 1 * A1 + T * C1) + So * C1 * k = T * (C1 + D1) + k * (C1 + D1).$$

Отсюда, очевидно, следует

$$So * (\alpha 1 * A1 + T * C1) - T * (C1 + D1) = k * (C1 + D1 - So * C1).$$

В итоге получаем значение калибровочного малого параметра

$$k = \frac{So * (\alpha 1 * A1 + T * C1) - T * (C1 + D1)}{C1 + D1 - So * C1}.$$

Этот параметр аналогичен параметру Δy, который рассматривался в методе кросс-плот.

Заметим, что, если для калибровки использовать чисто водяной пласт $(S_0 = 0)$, то преобразование

$$\Delta C / O_{New} = \Delta C / O_{Old} + k$$

“посадит $\Delta C / O$ на водяном пласте на ноль” ввиду того, что

$$So = 0 \rightarrow k = -T.$$

Теперь рассмотрим калибровку для произвольных отложений, т.е. с учетом литологии.

Имеем аналогичные формулы

$$So = \frac{\Delta C / O * (C1 + D1) - \alpha 1 * B1}{\alpha 1 * A1 + \Delta C / O * C1},$$

$$\Delta C / O = T + k,$$
$$So = \frac{(T+k)*(C1+D1)-\alpha 1*B1}{\alpha 1*A1+(T+k)*C1}.$$

Отличие состоит в том, что в числителе появилось дополнительное слагаемое $-\alpha 1*B1$.

Далее получаем

$$So*(\alpha 1*A1+T*C1)+So*C1*k = T*(C1+D1)-\alpha 1*B1+k*(C1+D1),$$
$$So*(\alpha 1*A1+T*C1)-T*(C1+D1)+\alpha 1*B1 = k*(C1+D1-So*C1),$$

$$k = \frac{So*(\alpha 1*A1+T*C1)-T*(C1+D1)+\alpha 1*B1}{C1+D1-So*C1}.$$

Заметим, что в данном случае преобразование

$$\Delta C / O_{New} = \Delta C / O_{Old} + k$$

“не сажает $\Delta C / O$ на водяном пласте на ноль” ввиду того, что

$$So = 0 \rightarrow k = \frac{-T*(C1+D1)+\alpha 1*B1}{C1+D1} = -T + \frac{\alpha 1*B1}{C1+D1}.$$

Понятно, что “$\Delta C / O$ может быть посажено на водяном пласте на ноль”, если из него вычесть константу $\varepsilon = \frac{\alpha 1*B1}{C1+D1}$.

Калибровка по второму пласту с высокой нефтенасыщенностью выполняется таким же образом, как в методе кросс-плот.

Используя опорные пласты, вычисляем величину коэффициента сжатия κ. Аналогично, для каждой конкретной глубины мы пересчитываем значение коэффициента нефтенасыщенности, полагая

$$kn = kn_0 + (kn_{Calc} - kn_0)*\kappa.$$

В отличие от метода кросс-плот, в данном случае величина kn_{Calc} вычислена другим способом.

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

На основе проведенных исследований и разработанных алгоритмов создана программная система OilTemper для автоматизации процесса вычисления коэффициента нефтенасыщенности.

Она предназначена для проведения расчета нефтенасыщенности по данным углеродно-кислородного каротажа, проводимого приборами ИНГК-С-95 (производитель аппаратуры — НОКБ ГП).

В программе реализованы все алгоритмы, описанные в предыдущих разделах: метод, основанный на применении кросс-плот зависимости аналитических параметров C/O, Ca/Si и коэффициента пористости; модифицированный вариант классического метода “Дельта C/O”, созданного и описанного фирмой Halliburton; различные типы калибровок и т.д.

Программа OilTemper на входе принимает увязанные по глубине данные в файлах формата LAS, производит расчет нефтенасыщенности, визуализирует планшет с исходными данными и результатами расчета, формирует итоговую таблицу результатов и производит экспорт выбранных оператором кривых в файл формата LAS версии 2.0.

Программа обладает развитыми средствами визуализации данных, предоставляет возможности для настройки алгоритмов расчета, ручного выделения опорных пластов в процессе калибровки. В программу встроен калькулятор кривых, позволяющий производить простейшие арифметические действия над выбранной кривой: сложение/вычитание и умножение/деление.

Кроме того, предусмотрен механизм сохранения текущего состояния программы с возможностью последующей загрузки для просмотра и анализа. Это позволяет контролировать действия пользователя программы при проведении расчета и при необходимости вносить коррективы.

Программа OilTemper реализована в среде Visual C++ 6.0 и является полнофункциональным 32-х разрядным приложением, работающим под операционной системой из семейства Microsoft Windows.

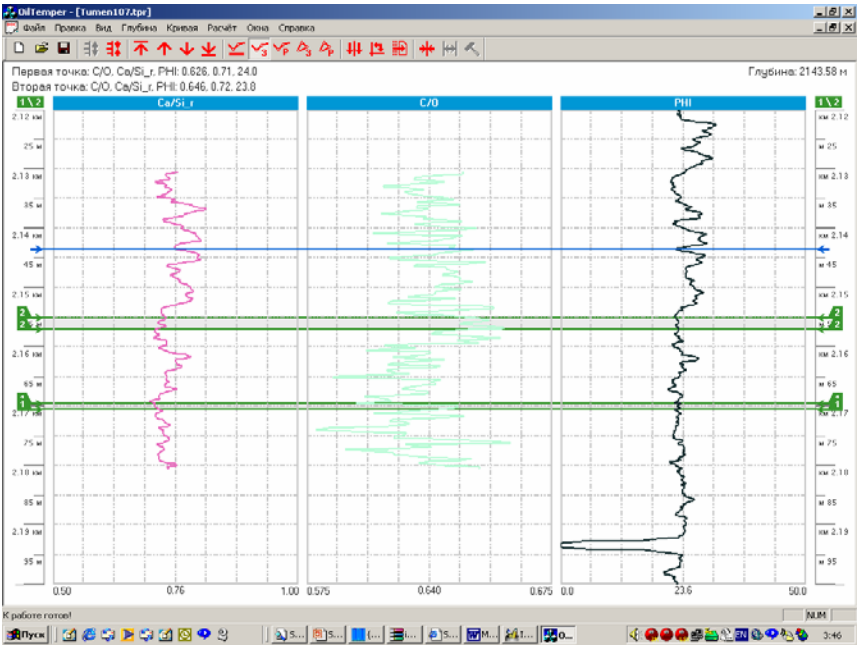


Рис. 10. Главное окно программы OilTemper

4. РЕЗУЛЬТАТЫ ПРОЕКТА ПО РАСЧЕТУ КОЭФФИЦИЕНТА НЕФТЕНАСЫЩЕННОСТИ

1. Предложены и испытаны 4 метода расчета коэффициента нефтенасыщенности:
   - по спектру ГИРЗ с помощью кросс-плота;
   - по спектру ГИНР с помощью кросс-плота;
   - методом ΔC/O (Halliburton) по спектру ГИРЗ;
   - методом ΔC/O (Halliburton) по спектру ГИНР.
2. На моделях все они работают достаточно хорошо.
3. Расчеты на скважинах показали, что наиболее хорошим методом является расчет по спектру ГИРЗ с помощью кросс-плота.
4. Заслуживают также внимания расчеты методом ΔC/O (Halliburton) по спектру ГИНР.

5. Отличие данных на моделях и в скважинах состоит в следующем. Вследствие литологии, обсадной колонны и других причин соответствующие измерениям точки смещаются в пространстве C/O, Ca/Si, или можно считать, что смещается или даже деформируется кросс-плот.
6. Сейчас мы считаем, что кросс-плот может перемещаться вдоль вертикальной оси C/O и сжиматься (растягиваться).
7. Чтобы определить величину параллельного переноса вдоль вертикальной оси C/O и коэффициента сжатия, предложено использовать калибровку по двум пластам. Оператор выделяет два слоя: с малой нефтенасыщенностью (в идеале, водонасыщенный) и с большой нефтенасыщенности и указывает их предполагаемую нефтенасыщенность (например 0% и 50%). Из этих данных программа вычисляет величину параллельного переноса вдоль вертикальной оси C/O и коэффициент сжатия кросс-плота. Дальше мы их используем для обработки всех остальных данных по скважине.
8. Калибровка по двум пластам была распространена и на методы ΔC/O (Halliburton), т.е. они были уточнены.
9. Рассмотрение данных со скважин с помощью специальной программы, позволяющей увидеть точки в трехмерном пространстве, показало, что кросс-плот может поворачиваться в пространстве. С одновременным ростом карбонатности и нефтенасыщенности правая сторона кросс-плота резко поднимается вверх. Необходимо уточнение этого момента.
10. Фактически необходимо корректно выставить “линию воды”, или, если работать в трехмерном пространстве (учесть еще пористость), то — “плоскость воды”. После чего можно предположить, что все 4 методики сблизятся, они будут давать близкие результаты, с точностью до расхождений, которые здесь не описываются, например, вызванные присутствием газа.
11. По поводу метода, используемого в настоящее время интерпретаторами, обрабатывающими скважинные данные, можно сказать следующее. Они домножают графики C/O, Ca/Si на некоторые два коэффициента, полагаясь на интуицию и опыт, совмещают и дальше смотрят расхождение.
12. В действительности, мы поняли, что дело обстоит следующим образом. Можно считать, что коэффициент один, а второй равен единице, просто поделим их на один из них. Мы поняли, что этот коэффициент в идеале должен быть равен тангенсу угла наклона кросс-плота (или

величине обратной, смотря что на что поделили, на первый или на второй коэффициент).

13. Таким образом, интерпретаторы, обрабатывающие скважинные данные, интуитивно подбирают тангенс угла наклона кросс-плота. Поэтому, если предложить алгоритм вычисления этого угла, то фактически произойдет формализация их эмпирического метода, всё сможет сделать программа и более точно, т.к. используется кросс-плот и т.д.
14. Вывод можно сделать следующий. Разработку алгоритмов по расчету нефтенасыщенности следует продолжить. В итоге может быть достигнут прогресс в уточнении алгоритмов в той мере, в которой это устроило бы практиков, непосредственно работающих на нефтепромыслах.

СПИСОК ЛИТЕРАТУРЫ

1. **Xu Jinwu, Zhang Zongjian.** Improved Carbon/Oxigen Log Interpretation Techniques under Variable Formation Water Salinity. — Shengli Well Logging Co., December 1999. — 12 p.
2. **Джекобсон Л.А., Этридж Р., Симпсон Дж.** Новый прибор малого диаметра с высокими характеристиками для мониторинга продуктивных пластов. — Hulliburton Energy Services, 1994. — 14 с.

Т.А. Волянская*

ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЕЙ ВИРТУАЛЬНОГО МУЗЕЯ ИСТОРИИ ИНФОРМАТИКИ В СИБИРИ

ВВЕДЕНИЕ

В статье описывается пользовательский интерфейс виртуального музея истории информатики в Сибири (СВМ), работа над которым ведется коллективом сотрудников ИСИ СО РАН, ИМ СО РАН и НГУ. Разрабатываемый виртуальный музей предназначен для накопления, систематизации и использования информации, относящейся к становлению и развитию информатики в Сибири. Музей создается в виде информационно-поисковой, справочной адаптивной гипермедиа-системы, доступной в Интернет [1, 3, 4].

Большинство виртуальных музеев, представленных в Интернет, реализованы при использовании традиционных технологий, одним из ограничений которых является то, что всем пользователям предоставляются одно и то же информационное содержание и один и тот же механизм навигации. Разрабатываемый виртуальный музей предназначен для использования различными категориями пользователей, и его посетители, имеющие различные цели, интересы, знания и предпочтения, могут нуждаться в различных частях содержащейся информации и использовать различные пути для навигации. Поэтому при создании музея особое внимание уделяется вопросам адаптации его интерфейса [1, 2, 3].

Структура статьи следующая. В первом разделе кратко описаны структура и содержимое музея, также приведена классификация пользователей на категории по уровню доступа к информационным ресурсам. Второй раздел статьи посвящен описанию пользовательского интерфейса музея. Рассматривается интерфейс управления информационными ресурсами: механизм навигации и просмотр информации, поиск, ввод и редактирование информации. Описывается интерфейс управления пользователями: регистрация, аутентификация, авторизация и администрирование пользователей.

* tanya@iis.nsk.su

Работа выполнена при финансовой поддержке Российского гуманитарного научного фонда (грант № 02-05-12010).

1. ВИРТУАЛЬНЫЙ МУЗЕЙ ИСТОРИИ ИНФОРМАТИКИ В СИБИРИ

1.1. Структура музея

Виртуальный музей включает в основном те же составляющие структурные единицы, что и реальные музеи: экспонаты, экскурсии, экспозиции и залы.

Минимальной структурной единицей музея является *экспонат*, в качестве экспонатов выступают следующие объекты: ученые-информатики, коллективы, документы архива, публикации, проекты, события, конференции и вычислительная техника.

Следующими структурными единицами являются *экскурсия и экспозиция*: это множества экспонатов, объединенных по тематическому, хронологическому или типологическому критерию. Экскурсия — это протекающий во времени рассказ о музее, в ходе которого происходит демонстрация экспонатов в определенной последовательности. В отличие от экскурсии, при просмотре экспозиции, составляющие ее экспонаты посетитель просматривает сам, причем только в режиме on-line. Обычно экспозиция предоставляет пользователю несколько способов навигации, в том числе возможность свободно перемещаться по экспонатам.

Следующей структурной единицей музея является *зал*. В общем случае, зал представляет собой совокупность экспонатов одного типа, при этом каждому типу экспонатов соответствует одноименный зал. В музее имеются о*ткрытые залы,* доступные для просмотра всем посетителям, и *запасники* — залы, доступные только для зарегистрированных пользователей.

Открытые залы содержат зал экспозиций и зал экскурсий, а запасники включают следующие залы: библиотеку, архив, хронику событий, зал ученых-информатиков, зал коллективов, зал проектов, зал вычислительной техники, зал конференций, зал новых поступлений и зал подготовки экспозиций и экскурсий.

В библиотеке собраны книги, монографии, сборники статей, учебные и методические пособия, статьи из научных журналов, тезисы конференций и т.д. Архив представляет собой совокупность текстовых, графических, звуковых и видео материалов. Хроника событий включает описания наиболее выдающихся событий из истории развития информатики в Сибири. Зал информатиков содержит информацию о наиболее выдающихся ученых-информатиках, включая биографии, основные печатные труды и достиже-

ния, фото и пр. В зале коллективов содержатся данные о коллективах: группах, лабораториях и институтах. В зале проектов размещены данные о проектах, создаваемых в рамках работ по информатике (темы, системы). В зале вычислительной техники расположены экспонаты, имеющие отношение к вычислительной технике, которая использовалась и разрабатывалась с начала создания Сибирского отделения Академии наук. Зал конференций содержит информацию о различных научных мероприятиях. Новые экспонаты, добавляемые пользователями музея, помещаются в зал новых поступлений. В зале подготовки экспозиций и экскурсий размещаются экспозиции и экскурсии, создаваемые пользователями музея [4].

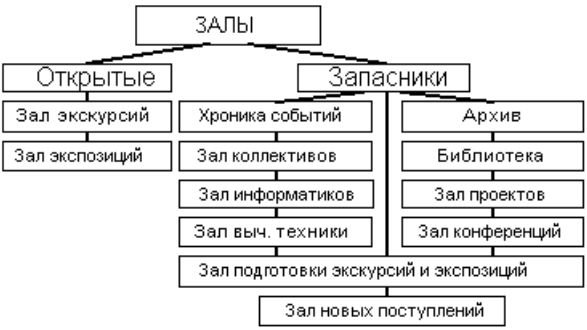


Рис. 1. Структура виртуального музея

1.2. Категории пользователей

Интерфейс музея разрабатывается с учетом его использования различными категориями пользователей. Все пользователи музея подразделяются на две основные категории: незарегистрированные пользователи («посетители») и зарегистрированные («специалисты»), которые различаются по уровню доступа к информационным ресурсам.

«Посетители» имеют только возможность просмотра информации, которая открыта для публичного доступа (например, в виде экскурсий и экспозиций). «Специалистам» доступны для просмотра все имеющиеся в музее информационные ресурсы, включая информацию запасников, закрытую для публичного доступа. Все «специалисты» подразделяются на две группы в зависимости от уровня доступа к ресурсам: группу «простых специалистов», работающих только в зале новых поступлений, и группу «музейных работников».

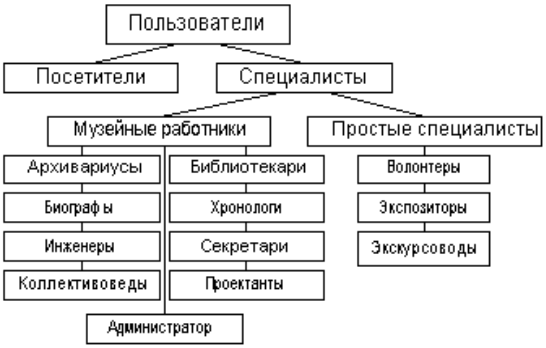


Рис. 2. Категории пользователей музея

В группе «простых специалистов» выделяются «волонтеры», имеющие права на добавление новых экспонатов, а также «экскурсоводы» и «экспозиторы», которые могут создавать собственные экскурсии и экспозиции. Добавленные или созданные ими объекты сначала помещаются в зал новых поступлений, впоследствии администраторы соответствующих ресурсов принимают решение об их включении в музей. Волонтеры, экскурсоводы и экспозиторы не имеют прав на редактирование информационных ресурсов музея.

Группу «музейных работников» можно представить в виде иерархической структуры, на самом верху которой находится «директор» (или «главный администратор»), обладающий полными правами на администрирование всех информационных ресурсов музея, включая администрирование пользователей музея. На втором уровне иерархии находятся администраторы отдельных ресурсов музея, которые назначаются «директором»: «главный экспозитор», «главный экскурсовод», «главный библиотекарь», «главный архивариус», «главный хронолог», «главный биограф», «главный коллективовед», «главный проектант», «главный инженер», «главный секретарь». Администраторы ресурсов имеют полные права на администрирование соответствующих типов ресурсов. В их полномочия также входит администрирование специалистов, работающих с соответствующими типами ресурсов. Третий уровень иерархической структуры включает «музейных работников», назначаемых администраторами соответствующих типов ресурсов: «библиотекарей», «архивариусов», «хронологов», «биографов», «коллективоведов», «проектантов», «инженеров», «секретарей». Они име-

ют ограниченные права на редактирование соответствующих типов ресурсов [4].

2. ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС СВМ

Компоненты пользовательского интерфейса музея можно условно подразделить на две основные группы: интерфейс управления информационными ресурсами, предназначенный для обеспечения механизма навигации и просмотра информации, поиска, ввода и редактирования информационных ресурсов, и интерфейс управления пользователями, служащий для регистрации, аутентификации, авторизации и администрирования пользователей.

2.1. Интерфейс управления информационными ресурсами

Интерфейс управления информационными ресурсами содержит компоненты для навигации и просмотра информации, поиска, ввода и редактирования ресурсов. Рассмотрим далее эти компоненты подробнее.

2.1.1. Навигация и просмотр информации

Виртуальное пространство музея

Виртуальное пространство музея включает в себя следующие залы: залы экскурсий и экспозиций, библиотеку, архив, зал хроники событий, зал ученых-информатиков, зал коллективов, зал проектов, зал вычислительной техники, зал конференций, зал новых поступлений и зал подготовки экспозиций и экскурсий.

Как уже говорилось выше, множество залов, доступных для посещения пользователем, может варьироваться в зависимости от категории пользователя. Незарегистрированным в музее пользователям (посетителям) открыты для просмотра только зал экскурсий и зал экспозиций, в то время как всем категориям зарегистрированных пользователей (специалистов) доступны все имеющиеся залы, включая запасники.

Виртуальное пространство музея представлено пользователю на главной странице музея, являющейся своего рода входом в музей. На главной странице, куда попадает при первом посещении музея пока еще не зарегистрированный пользователь, представлены только «открытые» залы (залы экскурсий и экспозиций). Навигационная структура этой страницы предоставляет пользователю гиперссылки для входа в зал экскурсий и зал экспозиций, а также для перехода к регистрации новых пользователей и для вхо-

да в музей уже зарегистрированных пользователей. О регистрации и аутентификации пользователей речь пойдет в следующем разделе статьи, посвященном интерфейсу управления пользователями.

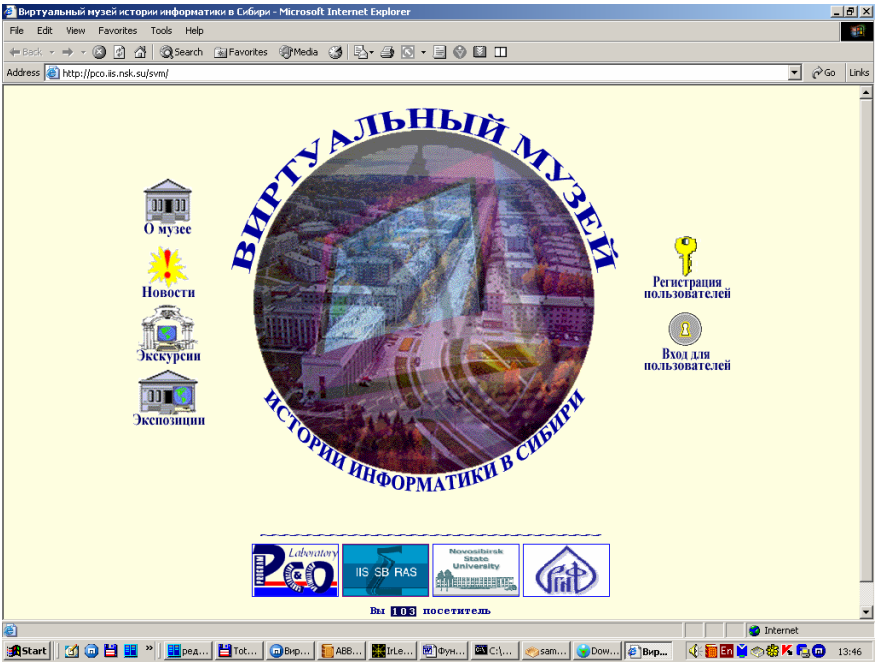


Рис. 3. Главная страница СВМ для категории «посетителей»

В отличие от незарегистрированных пользователей, главная страница музея, на которую зарегистрированный пользователь попадает сразу же после успешного прохождения аутентификации, отображает все виртуальное пространство музея. Отсюда пользователь может войти в любой из имеющихся залов-запасников — библиотеку, архив, зал хроники событий, зал ученых-информатиков, зал коллективов, зал проектов и т. д.

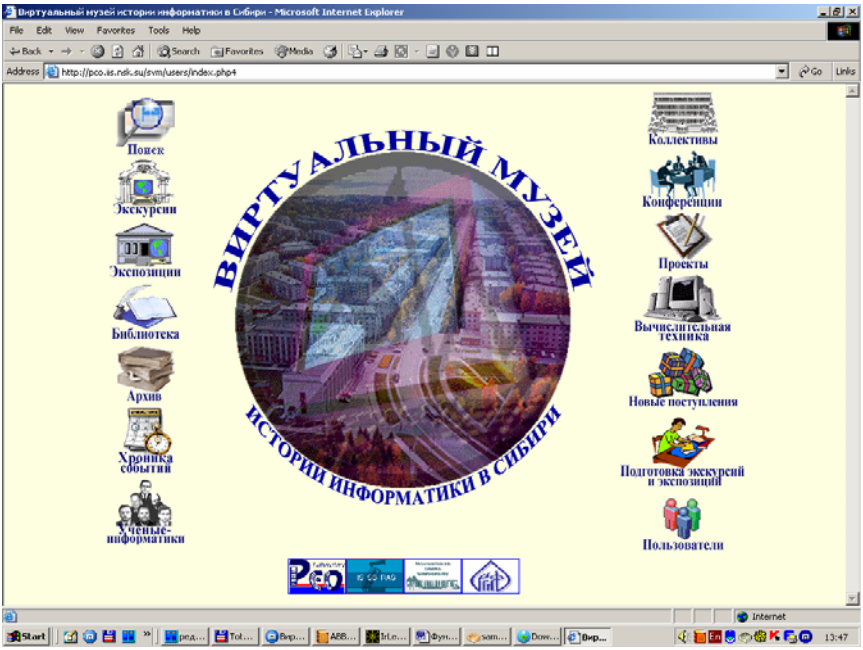


Рис. 4. Главная страница СВМ для категории «специалистов»

Как уже говорилось выше, навигация по залам виртуального пространства музея может осуществляться непосредственно с главной страницы при входе в музей. Также в процессе посещения залов и просмотра экспонатов, пользователь может перейти в любой интересующий его зал, используя главную навигационную панель. Эта панель содержит гиперссылки на все имеющиеся в музее залы, доступные пользователю с главной страницы музея, и всегда расположена в верхней части каждой страницы.

Просмотр залов

Теперь рассмотрим, как выглядят залы музея и каким образом составляющие их экспонаты могут демонстрироваться пользователю. Напомним, что каждый из вышеперечисленных залов музея предназначен для размещения экспонатов соответствующего типа. Так, в библиотеке размещены публикации, в архиве — документы архива, в зале проектов — проекты и т.д.

Каждый зал представлен пользователю в виде соответствующей главной страницы зала, имеющей специальную структуру. Рассмотрим далее панели навигации и панели инструментов, расположенные на каждой такой странице.

В самом верху страницы находится *главная навигационная панель*, содержащая гиперссылки для перехода во все имеющиеся залы и предназначенная для навигации по пространству залов.

Также на главной странице каждого зала расположена *локальная навигационная панель*, содержащая элементы «Поиск» и «Добавить» для перехода на соответствующие страницы. Перейдя по гиперссылке на страницу поиска, пользователь может производить поиск по различным критериям находящихся в данном зале экспонатов с помощью поискового интерфейса. В отличие от элемента «Поиск», элемент «Добавить» присутствует в локальной панели только для тех категорий пользователей, которые имеют права на ввод и редактирование экспонатов данного типа. Перейдя на страницу добавления нового экспоната, пользователь может добавлять в зал новые экспонаты, используя соответствующий интерфейс для ввода информации. Об интерфейсах поиска и редактирования информации будет подробно рассказано в следующих разделах.

Следующая панель, *панель выбора экспонатов,* присутствует во всех залах, кроме залов экскурсий, экспозиций, новых поступлений и подготовки экскурсий и экспозиций. Она выполняет функцию фильтрации имеющегося в зале множества экспонатов по алфавитному или временному критерию и представляет собой панель выбора экспонатов по годам или по алфавиту в зависимости от конкретного зала. Так, в библиотеке, архиве, хронике событий, зале проектов, зале вычислительной техники и зале конференций содержится панель выбора экспонатов по годам, в то время как в залах ученых-информатиков и коллективов находится панель выбора экспонатов по алфавиту.

Панель выбора экспонатов по годам включает следующие элементы, обозначающие соответствующие временные периоды: «<1950», «1950–1959», «1960–1969», «1970–1979», «1980–1989», «1990–1999», «2000<» и «ВСЕ ГОДА». При выборе, соответственно, одного из вышеперечисленных временных периодов пользователю будет представлен список всех экспонатов данного зала, относящихся к данному периоду. При выборе элемента «ВСЕ ГОДА» пользователь имеет возможность просмотреть сразу весь список имеющихся в зале экспонатов.

Панель выбора экспонатов по алфавиту содержит в качестве элементов, соответственно, все буквенные символы в алфавитном порядке, а так-

же элемент «ВСЕ». Выбрав, соответственно, один из вышеперечисленных элементов, пользователь получит список всех экспонатов данного зала, название которых начинается на заданную букву. Выбрав элемент «ВСЕ», пользователь имеет возможность просмотреть сразу весь список имеющихся в зале экспонатов.

Помимо вышеперечисленных трех панелей, на главной странице зала может присутствовать *панель выбора типов и критерия сортировки экспонатов.* Она предназначена для фильтрации имеющегося в зале множества экспонатов по типовому критерию и их сортировки. Панель выбора типов экспонатов позволяет пользователю отобрать для просмотра только интересующее его подмножество всех экспонатов, а панель выбора критерия сортировки позволяет отсортировать их требуемым образом.

Панель выбора типов экспонатов представляет собой чаще всего одну или иногда две группы кнопок с зависимой или независимой фиксацией. Кнопки с зависимой фиксацией (радиокнопки), или кнопки-переключатели, предназначены для взаимоисключающего выбора: пользователь может выбрать только одно значение для одного и того же свойства. Кнопки с независимой фиксацией (чекбоксы), или флаговые кнопки, позволяют пользователю выбрать несколько значений для одного и того же свойства. Итак, панель выбора типов экспонатов состоит из группы кнопок (радиокнопок или чекбоксов), соответствующих имеющимся типам экспонатов, отметив которые, пользователь может задать множество нужных для просмотра экспонатов. В том случае, когда пользователь не выбрал ни одного из типов экспонатов, по умолчанию отображается список всех экспонатов, содержащихся в зале.

Панель выбора критерия сортировки экспонатов представляет собой группу радиокнопок, обычно состоящую из радиокнопок «название» и «дата» в зависимости от конкретного зала. Таким образом, для отображаемого списка экспонатов обычно поддерживается сортировка по названию и дате. В случае, когда пользователь не указал критерий сортировки, экспонаты сортируются по названию.

Так, например, в библиотеке представлены публикации следующих типов: книги и монографии, полные сборники статей, труды конференций и журналы, а также отдельные статьи и тезисы, взятые из сборников, журналов и трудов конференций. В соответствии с этим, панель выбора типов публикаций состоит из двух групп: группы радиокнопок, позволяющих выбрать один из трех вариантов: «сборники», «статьи» или «книги», и группы чекбоксов, позволяющих отметить любые из трех вариантов: «сборники», «журналы» и «труды конференций». Таким образом, выбрав

соответствующие значения, пользователь может получить список или книг, или сборников (включая сборники статей, труды конференций и журналы), или статей (из сборников статей, трудов конференций и журналов). Панель выбора критерия сортировки позволяет отсортировать список публикаций по названию, дате или автору публикаций.

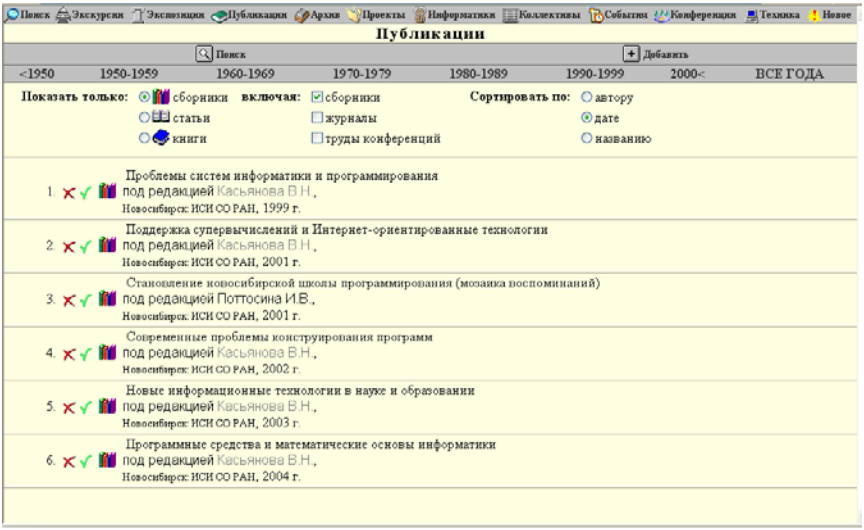


Рис. 5. Главная страница зала (библиотеки)

В архиве представлены документы следующих типов: текстовые, графические, аудио- и видеоматериалы. Согласно этому, панель выбора типов документов архива состоит из группы чекбоксов, позволяющей отметить любые из следующих пунктов: «текст», «графика» и «аудио» и «видео». Панель выбора критерия сортировки позволяет производить сортировку по названию и дате документов архива.

В сводной табл. 1 представлены параметры тех из вышеперечисленных панелей, которые различаются для некоторых залов, в частности, это: панель выбора экспонатов по годам/алфавиту и панели выбора типов и критерия сортировки экспонатов. Те панели, которые присутствуют во всех залах с одинаковыми параметрами (главная и локальная навигационная панель), не включены в данную таблицу.

Таблица 1

Параметры панелей отбора по годам/алфавиту, выбора типов и критерия сортировки экспонатов

Зал	Панель выбора по годам/алфавиту	Панель выбора типов экспонатов	Панель выбора критерия сортировки
Экскурсии	Нет	Нет	Нет
Экспозиции	Нет	Нет	Нет
Библиотека	По годам	Радиокнопки: сборники, статьи, книги Чекбоксы: сборники, журналы, труды конференции	Автор, дата, название
Архив	По годам	Чекбоксы: текст, графика, аудио, видео	Название, дата
Хроника событий	По годам	Нет	Название, дата
Ученые-информатики	По алфавиту	Нет	Нет
Коллективы	По алфавиту	Чекбоксы: институты, университеты, академии	Нет
Проекты	По годам	Нет	Название, дата
Вычислительная техника	По годам	Нет	Название, дата
Конференции	По годам	Нет	Название, дата
Зал новых поступлений	Нет	Нет	Нет
Зал подготовки экспозиций и экскурсий	Нет	Нет	Нет

Таким образом, указав в соответствующих панелях типы экспонатов, критерий сортировки и выбрав временной диапазон или первую букву в названии экспонатов, пользователь имеет возможность получить интере-

сующее его подмножество экспонатов, упорядоченных соответствующим образом. Полученное подмножество экспонатов представляет собой упорядоченный список, каждый элемент которого является отдельным экспонатом. Каждый экспонат в этом списке представлен кратким описанием, обязательно включающем в себя пиктограмму, специфицирующую тип экспоната, и название экспоната, а также необходимые атрибуты, варьирующиеся в зависимости от типа экспоната. Списки атрибутов для всех типов экспонатов перечислены в приведенной ниже табл. 2. Кроме этого, каждый элемент списка экспонатов обязательно содержит гиперссылку на страницу полного описания экспоната, речь о которой пойдет позже.

Таблица 2

Атрибуты экспонатов в зале

Залы	Атрибуты экспонатов
Библиотека	Название, список авторов, издательство, дата
Архив	Название, дата
Хроника событий	Название, даты начала и окончания
Ученые-информатики	Ф.И.О., ученая степень и ученые звания
Коллективы	Название, сокращенное название
Проекты	Название, даты начала и окончания
Вычислительная техника	Название, дата
Конференции	Название, даты начала и окончания, место проведения
Экскурсии	Название
Экспозиции	Название

Так, например, при посещении библиотеки, пользователь путем запроса, сформированного посредством выбора значений элементов вышеперечисленных панелей, получает требуемый список публикаций. Информация о публикациях в списке представляется следующим образом: название публикации, список авторов, издательство и год публикации. При этом названия публикаций, имена авторов и названия издательств содержат гиперссылки на страницы полных описаний соответствующих экспонатов музея — публикаций и ученых-информатиков.

В том случае, когда пользователь принадлежит к категории, имеющей права на редактирование экспонатов данного типа, каждый элемент списка снабжен двумя пиктограммами — «удалить» и «изменить». При нажатии на пиктограмму «удалить», соответствующий экспонат будет удален из музея,

а на пиктограмму «изменить» — в новом окне веб-браузера будет открыта страница редактирования экспоната, речь о которой пойдет позже.

Просмотр экспонатов

В предыдущих разделах было описано, как выглядит виртуальное пространство залов музея и как пользователь осуществляет по нему навигацию; как устроены составляющие его залы и каким образом в них пользователю предоставляется список интересующих его экспонатов. Теперь перейдем к рассмотрению того, какая информация и в каком виде предоставляется пользователю непосредственно о самих экспонатах. Для этого рассмотрим, как устроена страница экспоната, предназначенная для демонстрации экспоната пользователю.

Как уже говорилось, попасть на страницу экспоната пользователь может, непосредственно выбрав нужный экспонат из списка экспонатов, представленных на главной странице зала, и перейдя по связанной с ним гиперссылке. Перейти на страницу экспоната можно также по гиперссылкам, включенным в содержимое экскурсий и экспозиций и в полные описания экспонатов.

Страница экспоната, как и уже рассмотренная страница зала, содержит *главную навигационную* панель, содержащую гиперссылки для перехода во все имеющиеся залы и предназначенную для навигации по пространству залов.

Ниже на странице расположена *локальная навигационная панель*, содержащая элементы «Основная информация» и «Полный текст» (иногда — «Полная информация»). Эта панель служит для переключения между двумя вкладками (разделами) страницы, на которых, соответственно, представлены краткое и полное описания экспоната.

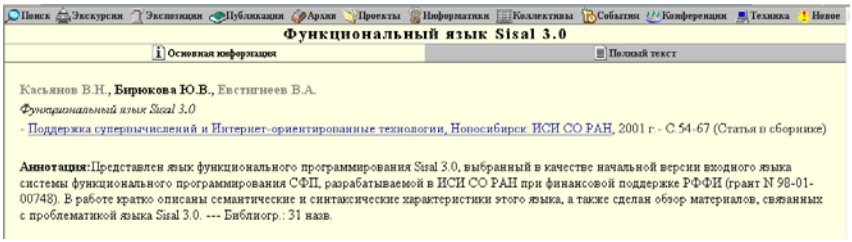


Рис. 6. Страница экспоната (публикации), вкладка «Основная информация»

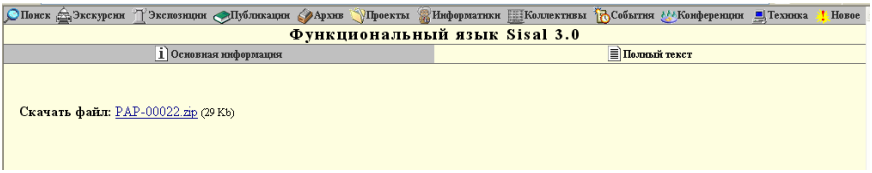


Рис. 7. Страница экспоната (публикации), вкладка «Полный текст»

Раздел под названием «Основная информация» служит для предоставления пользователю основной информации об экспонате в краткой форме и текстовом виде. Основная информация обычно включает в себя наиболее важные атрибуты экспоната, такие как название, дата, список авторов или разработчиков, а также краткое текстовое описание экспоната. Ряд атрибутов, представленных на вкладке основной информации, существенно различается в зависимости от типа демонстрируемого на странице экспоната. В табл. 3, представленной ниже, приведены наборы атрибутов, используемых для представления основной информации об экспонате для всех типов.

Рис. 8. Страница экспоната (коллектива), вкладка «Полный текст»

Раздел под названием «Полный текст» («Полная информация») предназначен для предоставления пользователю полного описания экспоната, представляемого в виде гипермедиа-страницы, которая может содержать

как гипертекст, снабженный форматированием и содержащий гиперссылки на другие информационные ресурсы, так и графику, видео- и аудиоматериалы. В некоторых случаях (для публикаций и документов архива) в этом разделе может быть расположен текстовый, графический, аудио- или видеофайл (или гиперссылка на него), представляющий содержимое экспоната.

В табл. 3 приведены наборы атрибутов, отображаемых в разделе основной информации, и указан вид представления информации в разделе «Полный текст» — гипермедиа-страница или файл — для всех типов экспонатов.

Таблица 3

Наборы атрибутов для страницы просмотра экспоната

Тип экспоната	Основная информация	Полный текст
Публикации	Список авторов, название, издательство, год выпуска, номера страниц, аннотация	Текстовый файл с полным текстом публикации
Документы архива	Название, дата, краткое описание	Файл (текст, графика, аудио, видео), представляющий содержание документа
События	Дата начала, дата окончания, название, краткое описание	Полное описание в виде гипермедиа-страницы
Ученые-информатики	Ф.И.О., фотография, дата рождения, ученая степень и ученые звания, занимаемые должности, данные об образовании (ВУЗ и годы обучения), названия и даты кандидатской и докторской диссертаций, научные интересы, почтовый адрес, e-mail и www-адреса	Полное описание в виде гипермедиа-страницы
Коллективы*	Название, сокращенное название, краткое описание, почтовый адрес, телефон/факс, e-mail и www-адреса	Полное описание в виде гипермедиа-страницы
Проекты	Название, дата начала, дата окончания, список разработчиков, краткое описание	Полное описание в виде гипермедиа-страницы

Вычислительная техника	Название, фотография, краткое описание, дата выпуска, список разработчиков	Полное описание в виде гипермедиа-страницы
Конференции	Название, дата начала, дата окончания, место проведения, тема, краткое описание, список организаторов	Полное описание в виде гипермедиа-страницы

* Страница просмотра информации о коллективах (помимо уже рассмотренных разделов «Основная информация» и «Полный текст») имеет раздел «Подразделения», содержащий список имеющихся в данной организации подразделений с гиперссылками на их полные описания.

Просмотр экскурсий и экспозиций

Рассмотрим далее интерфейс для просмотра экскурсий и экспозиций.

Определим точнее, что представляют собой экскурсии и экспозиции. В реальных музеях экспозиция — это зал, в котором размещены экспонаты, составляющие данную экспозицию. Экспонаты экспозиции посетитель может просматривать самостоятельно и в произвольном порядке. Экскурсия в реальном музее может проводиться экскурсоводом по какой-либо экспозиции. Таким образом, во время экскурсии посетитель просматривает экспонаты в определенной последовательности, заданной экскурсоводом. Можно представить себе экскурсию как один из путей навигации по экспонатам экспозиции.

По аналогии с реальными музеями, виртуальная экскурсия представляет собой демонстрацию экспонатов в определенной последовательности, без возможности свободной навигации по ним. Экскурсия может быть реализована, например, в форме презентации MS PowerPoint или видеоклипа. Таким образом, все содержимое экскурсии находится в одном файле, который может быть загружен и сохранен пользователем на локальный диск и впоследствии может просматриваться также в режиме офф-лайн.

Виртуальная экспозиция состоит из иерархически упорядоченных разделов (подэкспозиций или страниц). В отличие от экскурсии, экспозиция не предопределяет порядок просмотра пользователем составляющих ее экспонатов. Она предоставляет пользователю средства навигации для свободного перемещения по структуре экспозиции и просмотра экспонатов в произвольном порядке. Экспозиция может быть реализована, например, в виде набора (динамически генерируемых) гипермедиа-страниц, иерархически связанных между собой в соответствии со структурой экспозиции. При этом структура экспозиции в виде иерархии разделов в явном виде предоставляется пользователю и служит механизмом для навигации по разделам

экспозиции. Таким образом, пользователь может просматривать экспозицию только в режиме он-лайн.

Теперь рассмотрим, как выглядит интерфейс для просмотра экскурсий и экспозиций. Страница просмотра экскурсии (экспозиции) состоит из нескольких разделов: «Информация», «Оглавление» (только для экспозиции) и «Просмотр». В верху страницы расположена стандартная *главная навигационная панель*, предназначенная для навигации по пространству залов.

Ниже расположена *локальная навигационная панель*, содержащая элементы: «Информация», «Оглавление» (только для экспозиций) и «Просмотр». Эта панель служит для перехода между разделами, на которых, соответственно, представлены: краткая информация об экскурсии или экспозиции, структура экспозиции в виде списка составляющих ее разделов и непосредственно само содержимое экскурсии или экспозиции.

В разделе «Информация» содержится краткое описание экскурсии или экспозиции в текстовом виде: отображены название и краткое описание. Раздел «Оглавление» присутствует только на странице экспозиции, поскольку экспозиция имеет структуру, состоящую из упорядоченного множества разделов (страниц или подэкспозиций). Каждый раздел экспозиции имеет название, краткое описание и содержание, представленное отдельной гипермедиа-страницей.

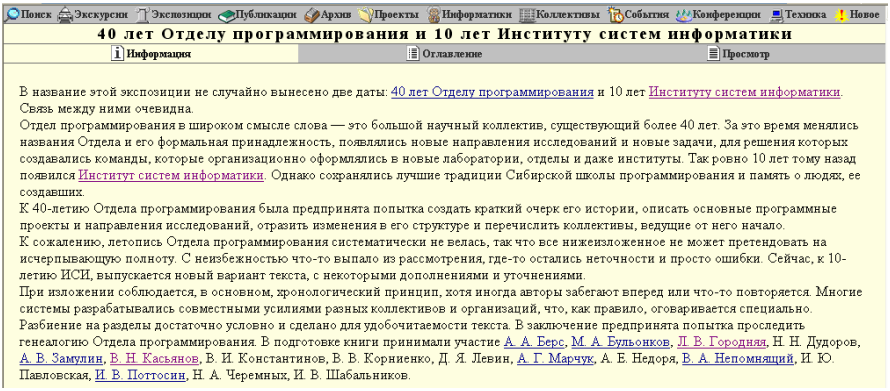


Рис. 9. Страница экспозиции, вкладка «Информация»

В разделе «Оглавление» отображена структура экспозиции, которую пользователь может использовать для свободной навигации по разделам экспозиции. Она представляет собой список разделов (или страниц), составляющих экспозицию: для каждого раздела в списке указано название и краткое описание, а также гиперссылка на содержимое раздела. При нажатии на эту гиперссылку содержимое раздела, представленное соответствующей гипермедиа-страницей, будет открыто во вкладке «Просмотр».

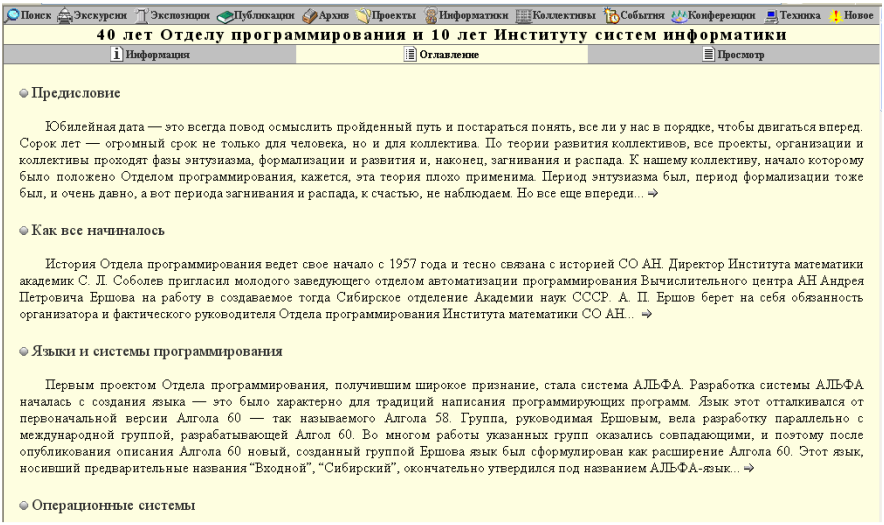


Рис. 10. Страница экспозиции, вкладка «Оглавление»

Вкладка «Просмотр» служит для просмотра содержимого экскурсии или экспозиции. При просмотре экскурсии она содержит гиперссылку на основной файл экскурсии, запускающийся в новом окне браузера. Для экспозиции на этой вкладке отображаются просматриваемые разделы экспозиции в виде гипермедиа-страниц. При этом в низу каждой страницы расположена панель навигации по разделам экспозиции (для перехода к следующему и предыдущему разделам).

Рис. 11. Страница экспозиции, вкладка «Просмотр»

2.1.2. Поиск информации

Поисковый интерфейс предназначен для поиска информационных ресурсов по различным видам запросов и просмотра выданной по запросам информации. Предоставляется возможность поиска всех имеющихся типов экспонатов: ученых-информатиков, коллективов, документов архива, публикаций, проектов, событий, конференций и вычислительной техники.

Существует два вида поиска: общий и локальный. Общий поиск осуществляется сразу по всему виртуальному пространству музея, т. е. поиск экскурсий, экспозиций и экспонатов. Локальный поиск проводится только в одном конкретном зале объектов соответствующего типа, например, поиск публикаций или ученых-информатиков.

Общий поиск

Гиперссылка на страницу общего поиска находится на главной странице музея, а также содержится на главной навигационной панели (элемент «Поиск»), присутствующей на каждой странице музея.

Страница общего поиска также содержит *главную навигационную панель*, предназначенную для навигации по пространству залов. Ниже расположена *локальная навигационная панель*, содержащая элементы «Запрос» и

«Результаты поиска», служащая для переключения на соответствующие вкладки страницы поиска.

Вкладка «Запрос» предоставляет пользователю средства для формирования поискового запроса. Запрос строится посредством заполнения формы поиска, содержащей поля ввода критериев поиска и управляющие элементы. Для проведения поиска требуется следующее: указать залы, в которых следует производить поиск, указать атрибуты экспонатов и ввести ключевые слова, по которым производить поиск. Соответственно, форма поиска разбита на две части и содержит: панель выбора залов и панель указания атрибутов и ввода ключевых слов.

Панель выбора залов имеет подзаголовок «Искать в залах:» и содержит группу чекбоксов, элементами которой являются названия имеющихся в музее залов: библиотека, ученые-информатики, коллективы, архив, проекты, хроника событий, конференции, вычислительная техника, экскурсии и экспозиции. По умолчанию в форме поиска отмечены все чекбоксы. Таким образом, отметив соответствующие чекбоксы и определив нужное подмножество залов, пользователь может производить по нему поиск экспонатов.

Панель указания атрибутов и ввода ключевых слов содержит строку ввода ключевых слов и группу чекбоксов, включающую элементы «в названии» и «в описании», для указания атрибутов, по которым будет производиться поиск. Таким образом, поиск по ключевым словам может проводиться только в названиях, только в описаниях или сразу в названиях и описаниях объектов. В строке ввода ключевых слов можно использовать метасимволы поиска ? и *, позволяющие производить поиск по образцу. Знак * специфицирует, что несколько произвольных символов могут появиться в позиции, представленной этим символом. Знак ? специфицирует одиночную позицию, в которой может появиться любой символ.

Также форма поиска содержит список выбора для задания количества выводимых на страницу результатов поиска: 10, 25, 50, 100, 500.

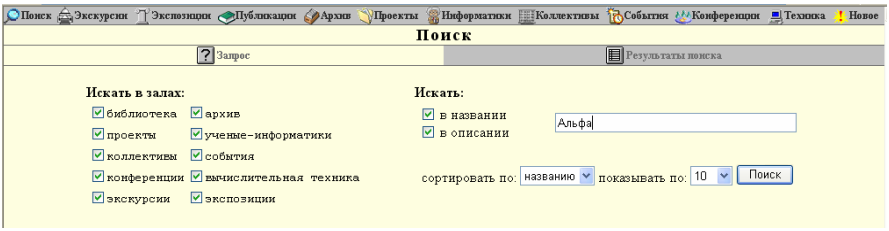


Рис. 12. Страница общего поиска, вкладка «Запрос»

При поиске отбираются только те объекты, атрибуты которых одновременно удовлетворяют всем заданным критериям поиска (т. е. полям, в которых указаны непустые значения). Незаполненные поля в поиске не участвуют.

Заполнив форму поиска и нажав на кнопку подачи запроса «Поиск», пользователь попадает на вкладку «Результаты поиска». В этой вкладке показывается количество найденных результатов и отображается список результатов поиска (экспонатов, экскурсий, экспозиций), удовлетворяющих заданным критериям поиска.

Каждый результат поиска в этом списке снабжен пиктограммой, соответствующей типу экспоната, и представлен кратким описанием, обязательно включающем в себя название экспоната, а также необходимые атрибуты, варьирующиеся в зависимости от типа экспоната. Кроме этого, каждый элемент списка экспонатов обязательно содержит гиперссылку на страницу полного описания экспоната, открывающуюся в отдельном окне браузера.

В случае, когда количество найденных результатов превышает указанное пользователем количество выводимых на страницу результатов поиска, весь список результатов разбивается на несколько страниц, каждая из которых содержит заданное количество результатов. Для навигации по списку результатов поиска в низу страницы расположена специальная панель навигации, состоящая из списка гиперссылок на пронумерованные страницы результатов. Также в панели навигации есть гиперссылки на предыдущую и следующую страницы результатов. Перемещаться по страницам можно посредством нажатия на номер соответствующей страницы.

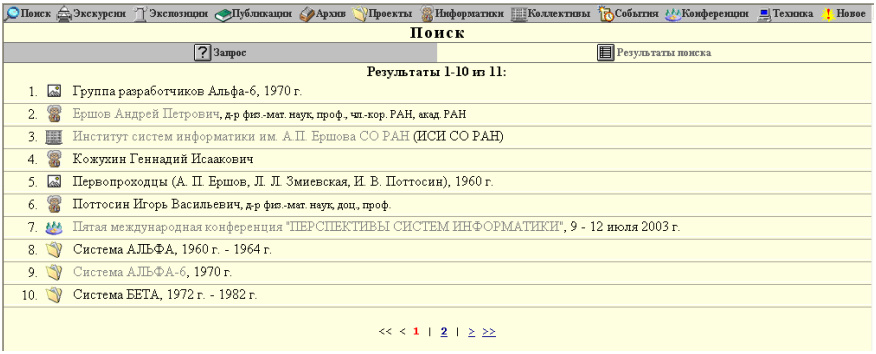


Рис. 13. Страница общего поиска, вкладка «Результаты поиска»

Локальный поиск

Для каждого типа экспонатов предусмотрен локальный поиск по соответствующим атрибутам, варьирующимся в зависимости от типа экспонатов. В каждом зале через локальную навигационную панель доступен переход на страницу поиска находящихся в данном зале экспонатов. Существуют, соответственно, страницы для поиска ученых-информатиков, коллективов, документов архива, публикаций, проектов, событий, конференций и вычислительной техники. Страницы поиска для разных типов экспонатов различаются набором полей ввода**,** соответствующих атрибутам, по которым может осуществляться поиск.

Страница поиска содержит *главную навигационную панель*, предназначенную для навигации по пространству залов. Ниже расположена *локальная навигационная панель*, содержащая элементы «Запрос» и «Результаты поиска», служащая для переключения на соответствующие разделы страницы поиска.

На вкладке «Запрос» расположена форма поиска, содержащая поля ввода**,** соответствующие атрибутам, по которым может осуществляться поиск. В формах поиска применяются три типа полей ввода: строки ввода, списки выбора и чекбоксы.

Строки ввода позволяют вводить алфавитно-цифровые данные непосредственно с клавиатуры. Во всех строках ввода на страницах поиска можно употреблять метасимволы поиска ? и *. Метасимволы поиска дают возможность осуществлять поиск по образцу, при котором одному образцу могут соответствовать сразу несколько значений. Списки выбора позволяют выбрать одно необходимое значение из заданного списка предопределенных значений. Чекбоксы позволяют выбрать (отметить) одно или несколько значений из заданного списка значений.

Форма поиска содержит панели ввода значений поисковых атрибутов, выбора критерия сортировки и выбора количества отображаемых на странице результатов поиска.

Каждому типу экспонатов соответствует своя *панель ввода значений поисковых атрибутов*, различающаяся наборами поисковых атрибутов. Например, при поиске ученого-информатика атрибутами поиска, в числе прочих, могут быть фамилия, ученая степень, ученое звание, дата рождения и т.д., при поиске публикации — имя автора и год издания. Наборы поисковых атрибутов формы поиска для экспонатов каждого типа приведены в табл. 4.

Таблица 4

Наборы поисковых атрибутов для каждого типа экспонатов

Типы экспонатов:	Наборы поисковых атрибутов
Публикации	Название, описание (аннотация), год издания, авторы
Документы архива	Название, описание, год
События	Название, описание, год
Ученые-информатики	Ф.И.О., год рождения, ученая степень, ученые звания, научные интересы
Коллективы	Название, описание
Проекты	Название, описание, год, участники
Вычислительная техника	Название, описание, год выпуска, разработчики
Конференции	Название, описание, год проведения, организаторы

Поля ввода для поиска экспонатов по названию и краткому описанию представляют собой чекбоксы «название» и «описание» и строку ввода ключевых слов, содержащихся в названии (описании). Чекбоксы «название» и «описание» предназначены для указания того, по каким атрибутам производить поиск. По умолчанию оба чекбокса уже отмечены, что предполагает поиск ключевых слов как в названиях, так и описаниях экспонатов. По желанию пользователь может сузить область поиска, ограничив ее, например, поиском ключевых слов только в названиях экспонатов.

Поля ввода для поиска экспонатов по дате представляют собой два списка выбора годов, соответствующих нижней и верхней границам временного диапазона, в котором должно находиться значение атрибута «дата» искомых экспонатов.

Поля ввода для поиска экспонатов по именам авторов (организаторов, разработчиков, участников) и по научным интересам представляют собой строки ввода.

Поля ввода для поиска ученых-информатиков по ученым степеням и званиям представляют собой группы чекбоксов, элементами которых являются, соответственно, наборы ученых степеней и званий.

Панель выбора критерия сортировки дает возможность пользователю указать, по какому атрибуту сортировать список найденных результатов. Она содержит список выбора критерия сортировки, различающийся для экспонатов разных типов и обычно включающий следующие элементы: «название» и «дата».

Как и для общего поиска, форма локального поиска включает *панель выбора количества отображаемых на странице результатов* поиска, со-

держащую список выбора со следующими значениями: 10, 25, 50, 100, 500 результатов.

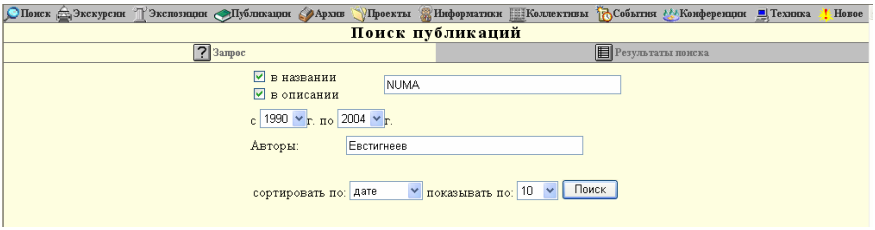


Рис. 14. Страница поиска публикаций, вкладка «Запрос»

Как и при общем поиске, при локальном поиске отбираются только те экспонаты, атрибуты которых одновременно удовлетворяют всем заданным критериям поиска.

Что касается вкладки «Результаты поиска», то она ничем не отличается от аналогичной вкладки страницы общего поиска, уже описанной выше. Отличие заключается только в том, что при общем поиске список результатов может содержать экспонаты различных типов, а при локальном — только экспонаты определенного типа.

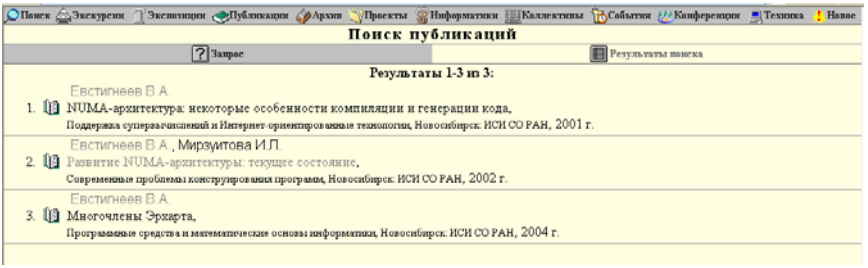


Рис. 15. Страница поиска публикаций, вкладка «Результаты поиска»

2.1.3. Ввод и редактирование информации

В основу организации СВМ положен принцип децентрализации, означающий, что информацию, хранящуюся в музее (т. е. экспонаты, экскурсии и экспозиции), могут загружать и сопровождать (т. е. добавлять, изменять или удалять) сами пользователи музея, обладающие правами доступа

согласно категориям. Информация может вводиться интерактивно через Internet посредством обычного веб-браузера.

Как уже говорилось, доступ к вводу и редактированию информации имеют только определенные категории пользователей. В табл. 5 приведены права на ввод, редактирование и удаление ресурсов для всех категорий пользователей, при этом права на ввод ресурсов обозначены символами «В», редактирование — «Р» и удаление — «У». Знак «-» означает отсутствие прав на ввод, редактирование и удаление ресурсов.

Таблица 5

Права доступа к информации для различных категорий пользователей

	Экскурсии	Экспозиции	Публикации	Доктыархива	Проекты	Информатики	Коллективы	События	Конференции	Выч. техника
Посетители	-	-	-	-	-	-	-	-	-	-
Волонтеры	-	-	В	В	В	В	В	В	В	В
Экскурсоводы	ВР	-	-	-	-	-	-	-	-	-
Экспозиторы	-	ВР	-	-	-	-	-	-	-	-
Библиотекари	-	-	ВР	-	-	-	-	-	-	-
Архивариусы	-	-	-	ВР	-	-	-	-	-	-
Проектанты	-	-	-	-	ВР	-	-	-	-	-
Биографы	-	-	-	-	-	ВР	-	-	-	-
Коллективоведы	-	-	-	-	-	-	ВР	-	-	-
Хронологи	-	-	-	-	-	-	-	ВР	-	-
Секретари	-	-	-	-	-	-	-	-	ВР	-
Инженеры	-	-	-	-	-	-	-	-	-	ВР
Администраторы	ВРУ	ВРУ	ВРУ	ВРУ	ВРУ	ВРУ	ВРУ	ВРУ	ВРУ	ВРУ

Для всех типов ресурсов, имеющихся в музее, т. е. экскурсий, экспозиций и экспонатов, реализован интерфейс для ввода и редактирования. Ввод данных осуществляется посредством заполнения соответствующих форм в зависимости от типа ресурсов, редактирование данных реализовано благодаря возможности редактирования данных, отображаемых в

соответствующих полях формы. Предусмотрен механизм для организации взаимосвязи ресурсов при вводе или редактировании информации.

Рассмотрим подробнее, как устроены страницы ввода и редактирования экспонатов, экскурсий и экспозиций.

Ввод и редактирование экспонатов

Попасть на страницу добавления экспоната определенного типа пользователь может, выбрав элемент «Добавить» в локальной навигационной панели, находящейся на главной странице зала, соответствующего типу экспоната. Например, для того чтобы добавить новую публикацию, пользователь должен войти в библиотеку и перейти по гиперссылке «Добавить», находящейся на соответствующей локальной навигационной панели.

Чтобы попасть на страницу редактирования информации об уже имеющемся экспонате, пользователь должен войти в соответствующий зал, выбрать из списка нужный экспонат и нажать на пиктограмму «изменить», расположенную рядом с кратким описанием экспоната. В случае, если пользователь хочет полностью удалить данный экспонат из музея, он должен нажать на пиктограмму «удалить», расположенную рядом с пиктограммой «изменить».

Упомянутые выше страницы добавления и редактирования экспоната на самом деле являются одной и той же страницей, только в случае добавления нового экспоната все поля ввода в форме являются пустыми, а в случае редактирования — содержат значения соответствующих атрибутов, доступные для редактирования. Исходя из этого, рассмотрим одновременно случаи добавления и редактирования информации, называя обе эти страницы обобщенно страницей ввода и редактирования экспоната.

Страница ввода и редактирования экспоната состоит из нескольких вкладок (разделов), содержащих соответствующие формы с полями ввода и редактирования информации. Для всех типов экспонатов на странице имеются следующие основные разделы: «Основная информация», «Дополнительная информация», «Авторы», («Разработчики», «Организаторы», «Участники»), «Полное описание» и «Данные об экспонате».

Страница ввода и редактирования содержит главную навигационную панель, уже описанную ранее, и локальную навигационную панель, содержащую в качестве элементов гиперссылки на вышеперечисленные разделы страницы («Основная информация», «Дополнительная информация», «Авторы» и т.д. в зависимости от типа экспоната). Эта панель служит для перехода на имеющиеся разделы страницы.

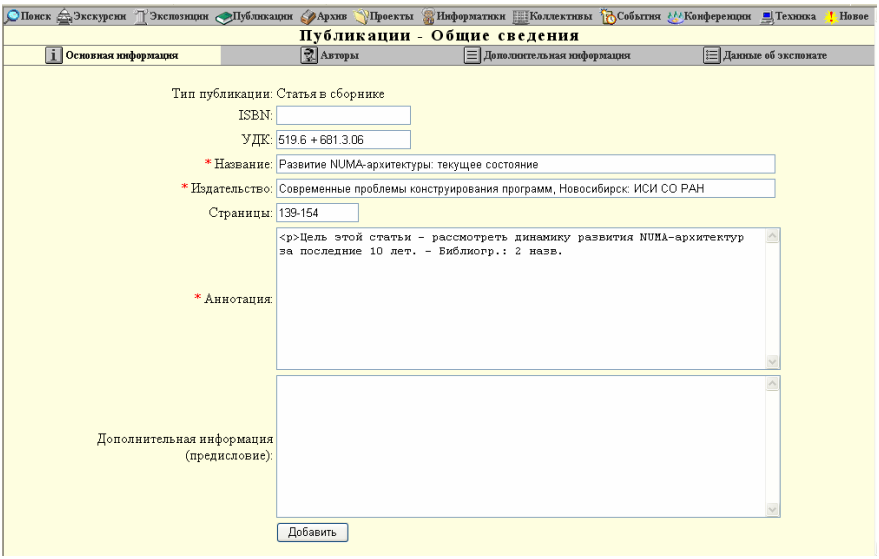


Рис. 16. Страница ввода и редактирования публикации, вкладка «Общие сведения»

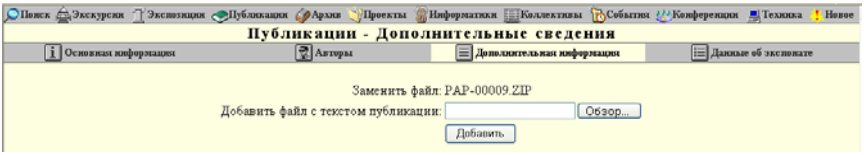


Рис. 17. Страница ввода и редактирования публикации, вкладка «Доп. сведения»

В табл. 6 приведены списки разделов страниц ввода и редактирования и списки расположенных на них полей ввода и редактирования атрибутов для экспонатов всех типов. Также указаны имена атрибутов и типы используемых полей ввода.

Таблица 6

Список разделов и атрибутов для страниц редактирования экспонатов

Тип экспонатов	**Список разделов**			
Публикации	Основная информация	Авторы		Доп. информация
	Тип (список выбора) ISBN (строка ввода) УДК (строка ввода) Название (строка ввода) Издательство (строка ввода и список выбора) Год издания (список выбора) Страницы (строка ввода) Аннотация (поле ввода) Доп.информация (поле ввода)	Имя автора (список выбора и строка ввода)		Файл с текстом (выбор файла)
Документы архива	Основная информация	Полное описание		
	Название (строка ввода) Дата (списки выбора) Краткое описание (поле ввода)	Файл (выбор файла)		
События	Основная информация	Доп. информация		Полное описание
	Название (строка ввода) Дата начала (списки выбора) Дата окончания (списки выбора) Статус (список выбора)	Краткое описание (поле ввода)		Полное описание (поле ввода) Редактор для создания html-кода*
Ученые-информатики	Основная информация	Должности	Фотография	Полное описание
	Ф.И.О. (строка ввода) Дата рождения (списки выбора) Дата смерти (списки выбора) ВУЗ (список выбора и строка ввода) Годы обучения (строки ввода) Ученые степени	Организация, Подразделение, Должность (списки выбора и строки ввода)	Фотография (выбор файла)	Полное описание (поле ввода) Редактор для создания html-кода*

	(группа чекбоксов) Ученые звания (группа чекбоксов) Канд.диссертация (строки ввода) Докт.диссертация (строки ввода) Научные интересы (строка ввода) Адрес (строка ввода) E-mail (строка ввода) WWW (строка ввода)			
Коллективы	Основная информация	Подразделения		Полное описание
	Вид (список выбора) Название (строка ввода) Сокращение (строка ввода) Адрес (строки ввода) Телефон (строка ввода) E-mail (строка ввода) WWW (строка ввода) Краткое описание (поле ввода)	Название подразделения (строка ввода) Сокращение (строка ввода) Краткое описание (поле ввода)		Полное описание (поле ввода) Редактор для создания html-кода*
Проекты	Основная информация	Участники		Полное описание
	Название (строка ввода) Дата начала (списки выбора) Дата окончания (списки выбора) Краткое описание (поле ввода)	Участник (список выбора и строка ввода)		Полное описание (поле ввода) Редактор для создания html-кода*
Выч. техника	Основная информация	Разработчики		Полное описание
	Название (строка ввода) Дата выпуска (список выбора) Краткое описание (поле ввода) Изображение (выбор файла)	Разработчик (список выбора и строка ввода)		Полное описание (поле ввода) Редактор для создания html-кода*

Конференции	Основная информация	Организаторы	Полное описание
	Название (строка ввода) Тема (строка ввода) Место проведения (строка ввода) Дата начала (списки выбора) Дата окончания (списки выбора) Краткое описание (поле ввода) Статус (список выбора)	Организатор (список выбора и строка ввода)	Полное описание (поле ввода) Редактор для создания html-кода*

В таблицу не включен раздел «Данные об экспонате», поскольку он присутствует в одном и том же виде на страницах ввода и редактирования всех типов экспонатов. Соответствующая ему форма содержит следующие поля ввода: имя автора, дата создания, возможность модификации, возможность участия в выставках и права модификации. «Имя автора» и «дата создания» являются текстовыми строками ввода, содержащими соответственно идентификатор пользователя, добавившего данный экспонат, и дату добавления. Эти значения генерируется автоматически системой и не доступны для редактирования. Поля «возможность модификации» и «возможность участия в выставках» являются группами радиокнопок с элементом «разрешена», стоящим по умолчанию, и «не разрешена». Эти поля доступны для редактирования. Поле «права модификации» является строкой ввода и также доступно для редактирования.

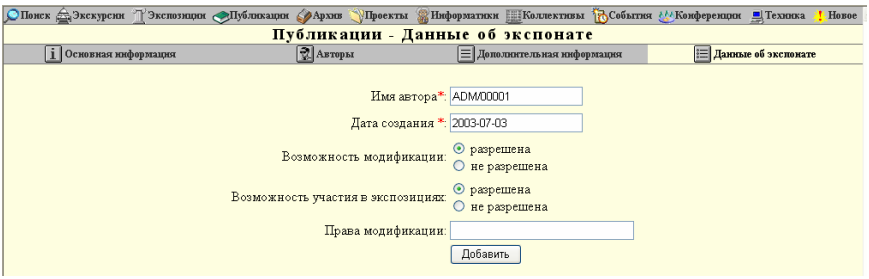


Рис. 18. Страница ввода и редактирования публикации, вкладка «Данные об экспонате»

Теперь рассмотрим подробнее разделы «Авторы», «Организаторы», «Разработчики» и «Участники». Все эти разделы имеют одинаковую структуру, поэтому для примера можно рассмотреть один из вышеперечисленных разделов, например, раздел «Авторы», находящийся на странице ввода и редактирования публикации.

Раздел «Авторы» содержит список уже имеющихся авторов и форму для добавления нового автора. В списке авторов перечислены все авторы публикации, снабженные порядковым номером и пиктограммой «удалить». Поле ввода «порядковый номер» представляет собой список выбора и доступно для редактирования, т. е. пользователь может изменить имеющийся порядок авторов в списке. Пиктограмма «удалить» служит для удаления выбранного автора из списка авторов публикации.

Форма для добавления нового автора содержит следующие поля ввода: список выбора, элементами которого являются экспонаты типа «ученые-информатики», и строку ввода. Если автор публикации уже есть в списке «ученых-информатиков», пользователь может выбрать его из списка. Если нет, пользователь может ввести его данные в пустой строке ввода. При вводе данных в поле ввода и нажатии на кнопку «Добавить», имя нового автора появится в списке авторов публикации, расположенной над формой добавления автора.

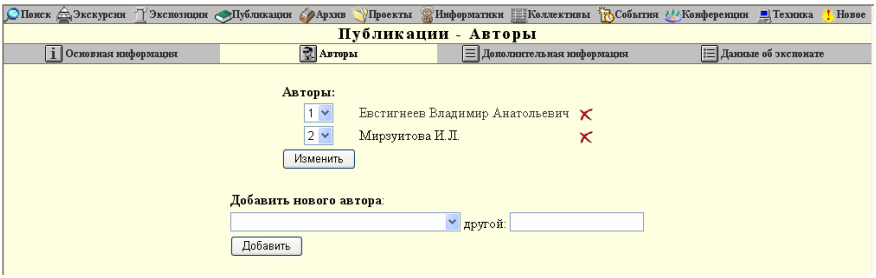


Рис. 19. Страница ввода и редактирования публикации, вкладка «Авторы»

Теперь рассмотрим раздел «Полное описание», содержащий html-редактор для создания html-кода для полного описания экспоната. Редактор поддерживает стандартные функции форматирования html-текста, включения графической, аудио- и видеоинформации, добавления гиперссылок и связей с другими экспонатами музея. Имеется возможность предварительного просмотра текста в процессе ввода и редактирования.

Редактор содержит форму, состоящую из многострочного поля ввода текста, нескольких панелей инструментов (панели форматирования, вставки и связывания экспонатов) и группы управляющих кнопок. Рассмотрим последовательно вышеперечисленные компоненты редактора.

Поле ввода текста предназначено для ввода и редактирования пользователем html-кода, создаваемого для полного описания экспоната. Html-код может вводиться пользователем в поле ввода вручную или генерироваться с помощью имеющихся панелей инструментов и автоматически вставляться в поле ввода. Рассмотрим имеющиеся панели инструментов редактора.

Панель форматирования содержит следующие элементы: списки выбора стиля (обычный, заголовок 1−заголовок 5), названия шрифта (Arial, Times New Roman, Courier, Tahoma, Verdana), размера шрифта (1−7) и цвета шрифта; кнопки для задания полужирного, курсива, подчеркнутого шрифтов; кнопки для создания упорядоченного и маркированного списков, кнопки для задания выравнивания абзаца (по левому и правому краю, по центру, по ширине).

Рассмотрим, как происходит работа с этой панелью инструментов. При выборе из выпадающего списка нужного типа заголовка открывается диалоговое окно со строкой ввода, в которую нужно ввести текст заголовка. После ввода и нажатия кнопки «ОК» в поле ввода добавляется html-код заголовка. При выборе из соответствующих списков выбора названия, размера или цвета шрифта в поле ввода сразу добавляется тэг с указанием соответственно названия, размера и цвета шрифта.

Теперь рассмотрим работу с кнопками для задания полужирного, курсива, подчеркнутого шрифтов и задания выравнивания абзаца. При нажатии любой из кнопок в поле ввода добавляется соответствующий открывающий тэг (,,<u>, и т.д.). Затем в поле ввода можно ввести текст, подлежащий форматированию. При отжатии уже нажатой кнопки в поле ввода добавляется соответствующий закрывающий тэг (,,</u>, и т.д.).

Кнопки для создания упорядоченного и маркированного списков работают следующим образом: при нажатии кнопки появляется диалоговое окно со строкой ввода первого элемента списка. После ввода текста и нажатия кнопки «ОК» в поле ввода полного описания добавляется html-код введенного элемента списка и появляется окно для ввода следующего элемента списка, и т.д. до нажатия кнопки «Отмена», по которой происходит добавление в поле ввода закрывающего тэга списка.

Панель вставки содержит кнопки для добавления гиперссылки, графики, аудио и видео. При нажатии на кнопку для добавления гиперссылки

сначала открывается диалоговое окно со строкой ввода URL гиперссылки, затем — ввода текста гиперссылки. После введения пользователем данных и нажатия кнопки в поле ввода текста добавляется html-код гиперссылки с указанным URL и текстом.

При нажатии любой из кнопок для добавления графики, аудио или видео, в панели инструментов появляется дополнительная панель, содержащая поле выбора файла, список выбора выравнивания (по левому и правому краям, по центру, по нижнему и верхнему краям, по середине) и отмеченный по умолчанию чекбокс «добавить в архив». Эта панель служит для вставки в полное описание графики, аудио- или видеофайлов.

Для вставки любого из объектов следует выбрать соответствующий файл, указать способ выравнивания объекта в тексте, выбрав его из списка, и отметить чекбокс «добавить в архив» в случае, когда нужно завести отдельный экспонат в архиве для этого объекта. После нажатия кнопки «Добавить» выбранный файл будет отправлен на сервер музея, а в поле ввода текста будет добавлен html-код для вставки соответствующего объекта — графики, аудио- или видеофайла. После добавления объекта, в случае помеченного чекбокса «добавить в архив», в отдельном окне браузера будет открыта страница для добавления этого объекта (графики, аудио- или видеофайла) в архив. Пользователю предлагается заполнить пустые поля в разделе «Информация» о документе архива, при этом в разделе «Полный текст» в качестве файла документа уже присутствует добавленный мультимедиа-файл.

Панель связывания экспонатов предназначена для вставки в полное описание связей с другими экспонатами. Предоставляется два способа связи с экспонатом: в виде *текста* и *гиперссылки*. В одних случаях может потребоваться вставить значение какого-нибудь атрибута экспоната просто в виде текста, без гиперссылки — например, краткое описание проекта. В других случаях требуется вставить гиперссылку на какой-нибудь экспонат. Например, в полном описании проекта вставить названия связанных с ним публикаций, снабженных гиперссылками на эти публикации в библиотеке. В качестве текста гиперссылки может выступать как значение выбранного пользователем атрибута, так и произвольный набор слов, введенный пользователем.

Панель связывания экспонатов содержит список выбора типов экспонатов и кнопку «выбрать». После выбора типа экспоната и нажатия кнопки появляется новая панель, содержащая список выбора экспонатов указанного типа, выбора атрибутов экспоната, строку ввода для произвольного тек-

ста гиперссылки, отмеченный по умолчанию чекбокс «создать гиперссылку» и кнопку «вставить».

Рассмотрим, как происходит добавление связи с экспонатом. Сначала следует выбрать нужный экспонат из списка. Затем пользователь может или выбрать из списка атрибут экспоната, значение которого он хочет вставить в текст, или ввести произвольный текст для гиперссылки на экспонат в строку ввода. В том случае, когда пользователь хочет вставить значение атрибута экспоната просто как текст, а не гиперссылку на экспонат, следует снять пометку с чекбокса «создать гиперссылку». После заполнения полей и нажатия кнопки «Вставить», в текстовое поле ввода будет добавлена связь с выбранным экспонатом, определяемая заданными параметрами.

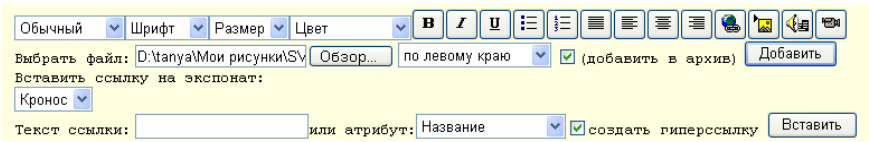


Рис. 20. Панели инструментов:
панель вставки (добавление графики) и панель связывания экспонатов

В html-код, расположенный в поле ввода, вставляются именно *связи* в виде специального кода, а не текстовые значения. При динамическом генерировании полного описания при просмотре этот код будет преобразован в определяемые им значения атрибутов экспоната.

Например, связь с экспонатом в html-коде может выглядеть следующим образом:

{CMP-00001.name}, где CMP-00001 — идентификатор экспоната, а name — имя его атрибута. Таким образом, код {CMP-00001.name} представляет собой название экспоната, имеющего идентификатор CMP-00001. В html-коде для полного описания присутствует только код {CMP-00001.name}, а при генерировании гипермедиа документа он преобразуется в конкретное значение — название заданного экспоната.

В том случае, когда нужно, чтобы название экспоната служило гиперссылкой на сам экспонат, требуется вставить связь в виде гиперссылки. В этом случае код связи будет выглядеть следующим образом:

{CMP-00001.name}

В случае гиперссылки на экспонат, в качестве текста гиперссылки может выступать значение выбранного атрибута этого экспоната или произ-

вольный текст, введенный пользователем, в качестве URL гиперссылки используется URL страницы описания экспоната.

Теперь рассмотрим группу управляющих кнопок, расположенных под полем ввода текста: «Расставить абзацы», «Preview» и стандартную кнопку отправления данных формы «Submit». Кнопка «Расставить абзацы» предназначена для автоматической расстановки тэгов начала абзаца <p> в местах начала новой строки текста, введенного в поле ввода. Кнопка «Preview» предназначена для предварительного просмотра в новом окне набранного в поле ввода html-кода в виде гипермедиа-страницы. Кнопка «Submit» служит для отправления введенного в поле ввода html-кода по окончанию редактирования.

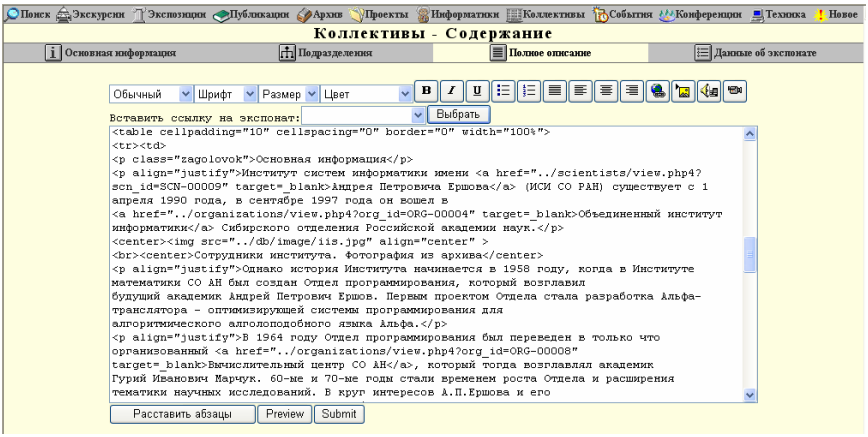


Рис. 21. Страница ввода и редактирования коллектива, вкладка «Полное описание»

Ввод и редактирование экспозиций

Так же, как и в уже рассмотренном случае добавления и редактирования экспонатов, перейти на страницу добавления и редактирования экспозиции пользователь может с главной страницы зала экспозиций. Для того чтобы добавить новую экспозицию, нужно перейти по ссылке «Добавить» в локальной навигационной панели. Чтобы попасть на страницу редактирования информации об уже имеющейся экспозиции, следует выбрать ее из списка представленных в зале экспозиций и нажать на соответствующую ей пиктограмму «изменить». В случае, когда нужно полностью удалить дан-

ную экспозицию из музея, следует нажать на пиктограмму «удалить», расположенную рядом с пиктограммой «изменить».

Страница ввода и редактирования экспозиции состоит из следующих разделов: «Основная информация», «Создание страницы» и «Индекс страниц». В верху страницы находится главная навигационная панель, уже описанная ранее, и локальная навигационная панель, содержащая в качестве элементов гиперссылки на вышеперечисленные разделы страницы. Эта панель служит для переключения между вкладками страницы.

Вкладка «Основная информация» содержит форму со строкой ввода названия экспозиции и многострочным полем ввода краткого описания экспозиции.

Вкладка «Создание страницы» предназначена для создания или редактирования раздела экспозиции, являющегося структурной единицей экспозиции и представляющего собой отдельную гипермедиа-страницу. Вкладка содержит строку ввода названия страницы (раздела), поле ввода краткого описания страницы и уже описанный html-редактор для ввода html-кода в поле ввода содержимого страницы. Поскольку функции и компоненты html-редактора были подробно рассмотрены в разделе, посвященном вводу и редактированию экспонатов, больше мы на нем останавливаться не будем. При редактировании уже существующей страницы отображается также строка ввода, содержащая порядковый номер страницы, также доступная для редактирования. Таким образом, пользователем может быть изменен существующий порядок страниц экспозиции.

Вкладка «Индекс страниц» содержит упорядоченный список страниц (разделов) экспозиции. Каждый элемент списка содержит порядковый номер, название страницы и пиктограммы «Изменить» и «Удалить». При нажатии на пиктограмму «Изменить» открывается вкладка «Создание страницы», поля ввода которой содержат значения соответствующих атрибутов выбранной для редактирования страницы, при этом эти поля доступны для редактирования. При нажатии на пиктограмму «Удалить» выбранная страница удаляется из списка страниц экспозиции.

Ввод и редактирование экскурсий

Доступ к странице добавления и редактирования экскурсии осуществляется из зала экскурсий таким же образом, как и в уже рассмотренном случае ввода и редактирования экспозиции. Подобным образом происходит добавление, редактирование и удаление экскурсии.

Страница ввода и редактирования экскурсии включает разделы «Основная информация» и «Файл». На странице находятся главная навигационная

панель и локальная навигационная панель, содержащая ссылки для перехода на соответствующие разделы страницы.

Как и для экспозиции, раздел «Основная информация» содержит строку ввода названия и поле ввода краткого описания экскурсии. Раздел «Файл» содержит поле выбора файла и предназначен для добавления в музей файла, содержащего экскурсию.

2.2. Интерфейс управления пользователями

Интерфейс управления пользователями включает компоненты для регистрации, аутентификации, авторизации и администрирования пользователей.

2.2.1. Регистрация пользователей

В музее предусмотрена регистрация пользователей. Незарегистрированные пользователи («посетители») имеют ограниченный доступ к просмотру ресурсов музея и не имеют прав на ввод и редактирование информации. Зарегистрированные пользователи («специалисты») получают доступ к просмотру всей имеющейся в музее информации, также они имеют возможность получить права для ввода и редактирования информации («музейные работники»).

Регистрация пользователей осуществляется на странице регистрации, перейти на которую можно, нажав на соответствующую пиктограмму на главной странице музея.

Процедура регистрации нового пользователя состоит в заполнении регистрационной формы, включающей обязательные и необязательные для заполнения поля ввода. Обязательные поля включают фамилию, имя, отчество, пароль и e-mail адрес, а необязательные — страну проживания, почтовый индекс, город и адрес.

Регистрационная форма также содержит группу чекбоксов со следующими элементами: «добавление экспонатов», «создание экскурсий», «создание экспозиций». Если пользователь желает осуществлять ввод информации в музей (добавлять экспонаты, создавать экскурсии или экспозиции), он должен отметить чекбоксы, соответствующие желаемым видам деятельности.

Если пользователь регистрируется с целью только просмотра ресурсов музея и, соответственно, не отмечает ни одного чекбокса, процедура регистрации происходит автоматически, при этом логин (имя пользователя для входа в систему) генерируется автоматически по специальному алгоритму

и сразу же высылается на указанный пользователем e-mail адрес. В этом случае пользователю автоматически присваивается категория «специалист».

В случае, когда пользователь планирует добавлять экспонаты в музей или создавать собственные экскурсии или экспозиции, и, соответственно, отмечает какой-нибудь чекбокс, процедура регистрации происходит не автоматически, а при участии администратора. Администратор рассматривает заявку пользователя и принимает решение о наделении данного пользователя соответствующими правами доступа к ресурсам. Пользователь получает письмо о том, что в течение недельного срока он получит логин для входа в систему. В этом случае пользователю присваивается, соответственно, категория «волонтера», «экскурсовода» или «экспозитора».

Что касается пользователей группы музейных работников, то их регистрация производится также администраторами в зависимости от категории.

2.2.2. Аутентификация и авторизация пользователей

Вход в музей зарегистрированных пользователей осуществляется с главной страницы музея посредством нажатия на пиктограмму «Вход для пользователей». При этом открывается диалоговое окно с полями ввода логина и пароля пользователя для входа в музей. После введения пользователем идентификационных данных происходит проверка их подлинности. При успешном прохождении аутентификации система определяет категорию, к которой относится пользователь. Затем происходит авторизация пользователя, во время которой пользователь получает определенные права доступа к ресурсам, соответствующие его категории, и затем автоматически перенаправляется на главную страницу музея для зарегистрированных пользователей. В случае трехкратного введения неверных идентификационных данных, пользователю выдается сообщение об ошибке аутентификации и предоставляется ссылка на прохождение регистрации.

2.2.3. Администрирование пользователей

Как уже говорилось ранее, главный администратор музея имеет права на администрирование пользователей музея, т. е. добавление новых пользователей, присвоение категорий пользователям, редактирование пользовательских данных, удаление пользователей. Вход на страницу администрирования пользователей разрешен только администратору и возможен с главной страницы музея посредством нажатия на пиктограмму «Пользова-

тели». Рассмотрим, что представляет собой интерфейс для администрирования пользователей.

Страница администрирования пользователей также содержит главную и локальную навигационные панели. На странице находятся две вкладки — «Все» и «Добавить».

На вкладке «Все» представлена информация обо всех зарегистрированных пользователях музея в табличной форме. Таблица имеет следующие поля ввода: uuid (идентификатор, он же логин пользователя) — нередактируемая строка ввода, название категории — редактируемый список выбора категорий, Ф.И.О., пароль, e-mail и почтовый адрес — редактируемые строки ввода. Также в каждой строке таблицы находятся управляющие кнопки «Изменить» и «Удалить».

Каждая строка таблицы представляет информацию об отдельном пользователе. Чтобы изменить информацию о пользователе, нужно ввести в соответствующие поля ввода новые данные и нажать на кнопку «Изменить». Для удаления пользователя нужно нажать на кнопку «Удалить».

Вкладка «Добавить» предназначена для заведения нового пользователя. Она содержит ту же самую форму, что и для самостоятельной регистрации пользователей, которая включает следующие поля ввода: фамилия, имя, отчество, пароль, e-mail адрес, страна проживания, почтовый индекс, город и адрес. Отличие между формами состоит в различии полей ввода, относящихся к данным о категории. В уже рассмотренной регистрационной форме пользователей содержится группа чекбоксов с элементами «добавление экспонатов», «создание экскурсий» и «создание экспозиций». В форме добавления нового пользователя вместо этой группы чекбоксов находится список выбора, элементами которого являются названия категорий пользователей, такие как администратор, архивариус, библиотекарь, биограф, волонтер, инженер, коллективовед и т.д. Таким образом, при заведении пользователя администратор может назначить ему любую категорию.

3. ЗАКЛЮЧЕНИЕ

В статье представлен разработанный и реализованный пользовательский интерфейс виртуального музея истории информатики в Сибири, предназначенного для накопления, систематизации и использования информации, относящейся к становлению и развитию информатики в Сибири.

Разработанный веб-интерфейс реализует функции управления информационными ресурсами и пользователями. Интерфейс управления инфор-

мационными ресурсами предназначен для обеспечения механизма навигации и просмотра информации, поиска, ввода и редактирования информационных ресурсов. Интерфейс управления пользователями служит для регистрации, аутентификации, авторизации и администрирования пользователей.

СПИСОК ЛИТЕРАТУРЫ

1. **Волянская Т.А.** Виртуальный музей истории информатики в Сибири: модель предметной области и модель пользователя // Новые информационные технологии в науке и образовании. — Новосибирск, 2003. — С. 124–146.
2. **Волянская Т.А.** Методы и технологии адаптивной гипермедиа // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 38–68.
3. **Волянская Т.А.** Применение адаптивной гипермедиа в виртуальном музее истории информатики в Сибири // Тез. докл. конф.-конк. работ студентов, аспирантов и молодых ученых «Технологии Microsoft в информатике и программировании». — Новосибирск, 2005. — С. 71–73.
4. **Касьянов В.Н., Несговорова Г.П., Волянская Т.А.** Виртуальный музей истории информатики в Сибири // Проблемы программирования. — Киев, 2003. — № 4.— С. 82–91.

Е.В. Касьянова*

ВВОДНЫЙ КУРС ПРОГРАММИРОВАНИЯ НА БАЗЕ ЯЗЫКА ZONNON

ВВЕДЕНИЕ

При обучении программированию наиболее важным представляется начальный этап, на котором обучаемый должен овладеть навыками точного формулирования алгоритмов на языках высокого уровня. Это невозможно сделать, прочитав несколько руководств или прослушав курс лекций по программированию. Необходима практика конструирования алгоритмов, и здесь невозможно обойтись без языка программирования, пригодного для целей начального обучения, и подходящего набора примеров и задач.

В статье кратко представлен вводный курс программирования на базе языка Zonnon, работа над которым ведется в Цюриховском институте информатики [6, 11, 12]; описаны две книги: «Введение в программирование» и «Практикум по программированию» [2, 3], поддерживающие этот курс.

Zonnon — это новый универсальный язык программирования в семействе языков Паскаль, Модула-2 и Оберон. Он сохраняет стремление к простоте, ясному синтаксису и независимости концепций, а также уделяет внимание параллельности и легкости композиции и выражения. Унификация абстракций является стержнем проектирования языка Zonnon, и она отражается в его концептуальной модели, основанной на модулях, объектах, определениях и реализациях. Язык Zonnon содержит такие новые черты, как активность в объектах, основанный на межобъектном взаимодействии диалог, перегрузка операций и обработка исключительных ситуаций. Язык Zonnon специально разрабатывается как платформно-независимый язык. Первая реализация языка Zonnon выполнена для платформы .NET [15]. Кроме того, предполагается интеграция компилятора в систему программирования Visual Studio .NET (в сотрудничестве с компанией Microsoft) [1].

Курс опирается на опыт преподавания основного курса по программированию для студентов механико-математического факультета (ММФ) Но-

* kev@iis.nsk.su

Работа выполнена при финансовой поддержке Microsoft Research (грант «Разработка вводного курса программирования»).

восибирского государственного университета (НГУ) [4]. Это семестровый курс, который предполагает еженедельно шесть часов аудиторных занятий:

- два часа лекционных занятий,
- два часа семинарских занятий у доски и
- два часа практических занятий за компьютером.

В основу курса положен принцип концентрического изложения материала. Предполагается, что с первых занятий студенты начинают упражняться в составлении программ, которые могут реально выполняться на доступных ЭВМ, и постепенно овладевают навыками разработки на языке Zonnon линейных, ветвящихся и итеративных алгоритмов, алгоритмов с массивами и процедурами. Также постепенно, одновременно с расширением класса решаемых задач, студенты углубляют свои знания о языке Zonnon.

К концу вводной части курса студенты овладевают основными конструкциями языка, образующими то естественно выделяемое его ядро для «начального обучения», которое условно можно назвать языком мини-Zonnon. В этом ядре нет ряда весьма важных возможностей языка Zonnon, таких как, например, поддержка объектно-ориентированного программирования, сужены средства языка Zonnon по структурированию данных и действий — например, нет множеств и бесконечных циклов. Поэтому, хотя мини-Zonnon и содержит все необходимые сведения о языке Zonnon для построения программ начального обучения, он является лишь базой для изучения программирования на языке Zonnon. Отметим, что в мини-Zonnon идентификаторы могут содержать наряду с латинскими буквами и буквы русского алфавита, хотя это и запрещено в языке Zonnon.

Следует отметить, что использование в качестве основы вводной части курса подмножества «живого» языка программирования на ММФ НГУ является традиционным. Первым использовалось подмножество языка Алгол-60, получившее название языка Минал [5]. Затем, когда в НГУ появились терминальные классы с диалоговым языком, являющимся вариантом языка Фортран, Минал был заменен языком мини-Фортран [8]. В начале 80-х гг. прошлого столетия в НГУ появилась одна из первых реализация языка Паскаль, тогда еще на машинах серии ЕС [7], и с тех пор вплоть до настоящего времени место первого учебного языка программирования для студентов ММФ НГУ бессменно занимает язык мини-Паскаль [4].

Статья начинается с изложения основных особенностей языка Zonnon (разд. 1). Разд. 2 содержит описание цели и основных принципов курса. Разд. 3 является кратким описанием книги «Введение в программирование», являющейся учебником по курсу. Цели и содержание компьютерного

практикума составляют разд. 4. Разд. 5 содержит описание книги «Практикум по программированию», предназначенной для поддержки практикума. В заключении подводятся некоторые итоги разработанного курса. Полный синтаксис языка Zonnon в состоянии на октябрь 2004 г. приведен в приложении.

1. ЯЗЫК ПРОГРАММИРОВАНИЯ ZONNON

Сам проект по разработке языка Zonnon возник как продолжение работы его авторов над языком Оберон (Oberon) в проекте, который был начат Microsoft Research в 1999 году с целью реализации значительного числа нестандартных языков программирования для платформы .NET [9, 10]. Язык Оберон [13, 14] является хорошо известным преемником языков Паскаль (Pascal) и Модула-2 (Modula-2). Мотивация авторов на продолжение работы по развитию языка Оберон связана со следующими двумя целями [11]:

a) экспериментально исследовать потенциальные возможности платформы .NET в комбинации с новой технологией CCI интеграции компиляторов для проектирования языков;
b) реализовать для платформы .NET язык Zonnon, являющийся эволюцией языка Oberon для .NET.

Поскольку язык Zonnon задуман как дальнейшая эволюция языка Оберон, авторы стремятся сохранить такие важные черты языка Оберон и его преемников, как компактность языка, ясность, недвусмысленность и ортогональность его основных понятий. Вместе с тем, чтобы создать современную альтернативу языку Оберон, авторы внесли в язык ряд изменений, основными из которых являются:

- более развитая модульная структура языка;
- продвинутая и одновременно простая и ясная объектная модель;
- концепция активных объектов.

«Общеалгоритмическая» часть языка Zonnon практически полностью повторяет соответствующие части языка Оберон, поэтому данный язык можно рассматривать как естественную замену Оберону там, где последний традиционно используется для обучения программированию.

Язык Zonnon возник как естественный результат исследований, проводившихся в течение последних нескольких лет в Федеральном Технологическом Институте (ETH) в Цюрихе. Непосредственными предшественни-

ками языка следует считать Active Oberon, реализованный как базовый язык ядра операционной системы BlueBottle, а также реализацию языка Oberon для платформы .NET (Oberon.NET). Язык Zonnon, имея ряд общих черт с указанными языками, вместе с тем существенно отличается от них по ряду принципиальных моментов. Первая реализация Zonnon выполнена для платформы .NET. Кроме того, предполагается интеграция компилятора в систему программирования Visual Studio .NET (в сотрудничестве с компанией Microsoft).

Хотя по размеру Zonnon уступает таким языкам, как C#, Java и Ada (см. приложение), он является универсальным языком программирования, пригодным для широкой области приложений. Типично она включает компонентно-ориентированную композицию, параллельные системы, алгоритмы и структуры данных, объектно-ориентированное и структурное программирование, графику, математическое программирование и низкоуровневое системное программирование. Zonnon предоставляет богатую объектную модель с инкапсулированным поведением и синтаксически-управляемыми диалогами, которые инкапсулируют состояние. Он может использоваться для написания программ в традиционном и объектно-ориентированном стилях, а также весьма подходит для целей обучения программированию от его базовых принципов до продвинутых концепций.

Унификация абстракций является стержнем проектирования языка Zonnon. Она отражается в его четырех столпах:

- *модуль (module)* — текстовый контейнер, а также объект композиции программ;
- *объект(object)* — типовой образец для определяемых объектов;
- *определение (definition)* — концепция абстракции и композиции для определяемых интерфейсов;
- *реализация (implementation)*— контейнер для переиспользуемых фрагментов объектных реализаций.

Модуль (*module*) имеет двойственную природу: он объявляет синтаксический контейнер для логически связанных программных деклараций и одновременно определяет объект, чей жизненный цикл управляется системой. Таким образом, модель предоставляет механизм для текстуального разбиения исходной программы, а также динамической загрузки на этапе выполнения некоторой части программы в виде созданного экземпляра объекта.

Любое число динамически созданных объектов может иметь свои жизненные циклы, управляемые системой, однако в любое заданное время

только единственный экземпляр каждого модульного объекта может быть создан системой. Поскольку модуль формирует единицу инкапсуляции и скрытия данных, он представляет собой также идеал как контейнер для реализации абстрактных типов данных.

Объектом (*object*) является типовой образец, включающий в себя поля, методы и активности. Поля представляют состояние объекта, методы — его функциональность, а активности — его параллельное поведение. Он может выставлять свой интерфейс к системному окружению двумя способами. Во-первых, с помощью присущего ему *интерфейса* (*intrinsic interface*), т.е. множества всех элементов, которые программист выбрал сделать публичными, а не сохранять как приватные, и, во-вторых, большим числом *определений* (*definitions*), каждое из которых выставляет свой *аспект* (*facet*), представляющий собой некоторый взгляд на службы объекта со стороны клиентов.

Определение (*definition*) задает некоторый свой *аспект* (*facet*) объекта в терминах абстрактного интерфейса, включающего определения полей и сигнатуры методов. Определения могут образовывать не только иерархию, но и *сеть связанных типов* (*a network of related types)*.

Реализация (*implementation*) определяет агрегат реализаций полей и методов, предназначенных для переиспользования, при присоединении к программе с помощью одного или нескольких объектных образцов. Объект, реализующий определение, должен реализовывать все его поля и методы. Однако если объект импортирует реализацию определения с тем же именем, то это неявно предполагает, что она будет его (возможно частичной) реализацией.

Программный текст (*program text*) состоит из модулей, объектов, определений и реализаций. *Внутренний интерфейс* (*intrinsic interface*) программы представляет собой множество деклараций, сделанных публичными всеми ее частями. П*рограмма периода исполнения* (*runtime program*) состоит из одного или более модулей и любых объектов, которые создаются динамически. *Система* (*system*) предоставляет механизмы динамической программной загрузки и выгрузки модулей и динамическим управлением объектных ресурсов во время выполнения, когда происходит исполнение программы.

Эти конструкции используются для формирования всей структуры программы в виде программных *единиц: module, object*, *definition* и *implementation*. Каждая конструкция может существовать как *отдельно скомпилированная единица* (*separately compiled unit*) или может текстуально встраиваться в содержимое другой конструкции. Указанные единицы обеспечи-

вают базис для композиции программ при «программировании в большом», а также для текстуального разбиения и раздельной компиляции во время разработки программы.

Объектная модель в языке Zonnon основывается на концепции, что «все есть объект». Она поддерживает три точки зрения на объекты: во-первых, как сущности с некоторым внутренним типом, использующие абстрактные операции способом, безопасным для типов, во-вторых, как провайдеры служб, доступные через определенные интерфейсы, и, в-третьих, как автоматические агенты, взаимодействующие через формальные диалоги. *Активности (activities)* используются как для добавления поведения объектам, так и для реализации диалога. Они внедряют естественным образом параллелизм в язык.

Многие из концепций языка Zonnon получены им по наследству. Цель состояла в том, чтобы предложить выразительные и связующие черты, которые уже доказали свою ценность. Язык Zonnon также вводит некоторые новые черты, такие как перегрузка знаков операций для естественного представления математических и других выражений и обработка ситуаций для повышения надежности. Некоторые черты были им реанимированы (взяты из ранних членов семейства паскалеподобных языков), например, пары *definition*, *implementation* и типы перечисления языка Модулы-2 и, по прагматическим причинам, основная форма операторов *read* и *write* языка Паскаль.

Когда осуществляется выбор языка для построения современных программных систем, важным соображением является достижение взаимодействия между программами, написанными на разных языках. Язык Zonnon специально проектировался как платформно-независимый язык, поддерживающий такое взаимодействие.

2. ЦЕЛЬ И ОСНОВНЫЕ ПРИНЦИПЫ КУРСА

Курс предназначен для обучения основным методам построения корректных, эффективных и надежных программ на базе языка Zonnon и платформы Microsoft.NET. Он ориентирован на широкий круг лиц, обучающихся методам программирования, в первую очередь, на студентов НГУ, других вузов и средних учебных заведений, а также школьников, желающих углубить свои знания по программированию. Курс предназначен главным образом для тех учебных заведений, в которых

- в настоящее время используется язык Паскаль в качестве языка начального обучения программированию и
- есть желание перейти к более современному курсу программирования, охватывающему концепции языков программирования нового поколения, таких как Java и C#, но осуществить этот переход плавно, без резкого изменения сложившегося стиля преподавания программирования.

Разработанный курс базируется на ряде методических и технологических принципов, основными из которых являются следующие.

1. Принцип концентрического изложения материала, когда обучаемый осваивает языковые средства и приемы программирования постепенно, слой за слоем. При этом с первых занятий студенты начинают упражняться в составлении программ, которые могут реально выполняться на доступных им компьютерах, а освоение нового слоя означает просто расширение круга задач, которые может решать обучаемый.

2. Принцип обучения конструированию программ на подробно комментированных образцах решения тщательно подобранных задач. Назначение примеров — не только дать образцы и описать основные схемы алгоритмов, но и на сравнительном анализе разных решений одной и той же задачи познакомить студента с такими понятиями, как эффективность, наглядность и надежность решения.

3. Принцип доказательного программирования, когда программа строится вместе с доказательством ее правильности. Для этого в курсе вводятся понятия промежуточных утверждений и инвариантов, а в разрабатываемых алгоритмах решения задач такие утверждения записываются в форме программных комментариев.

4. Принцип пошаговой разработки программ, когда программа строится из формальной спецификации задачи с помощью мелких формально проверяемых шагов преобразования.

5. Принцип модульного программирования, позволяющий проектировать, разрабатывать и собирать программу по частям и с использованием библиотек уже готовых частей.

6. Принцип объектно-ориентированного программирования, позволяющий разработчикам программ легко создавать все более сложные приложения с помощью инкапсуляции, наследования и полиморфизма.

3. ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

Книга [2] предназначена для изучения студентами основных понятий программирования и постепенного овладения навыками разработки на языке Zonnon линейных, ветвящихся и итеративных алгоритмов, алгоритмов с процедурами и со структурированными данными.

Книга содержит порядка 3000 задач и состоит из 6 глав.

Внутри одной главы задачи сгруппированы по методам решения и языковым конструкциям, на освоение которых они ориентированы. Каждая группа составит самостоятельный параграф, содержащий помимо текстов задач либо описание соответствующих языковых понятий, либо примеры решения некоторых типичных задач данной группы. Назначение примеров — не только дать образцы и описать основные схемы алгоритмов, но и показать, как алгоритм может быть выведен из рекурсивных соотношений — спецификации алгоритма; привести инварианты циклов и другие промежуточные утверждения, из которых выводится правильность программы. Здесь же на сравнительном анализе разных решений одной и той же задачи студент знакомится с такими понятиями, как эффективность, наглядность и надежность решения.

Главу завершает список заданий. Каждая из формулировок заданий представляет собой фактически схему для построения конкретных вариантов задания: индивидуальных задач на программирование для всех студентов учебной группы. Эти варианты получаются в результате подстановки в текст вместо номера варианта его значения и выбора по значению номера варианта подходящих частей текста, составляющего формулировку задания.

Например, 7-й вариант следующего задания:

«в заданной последовательности целых чисел найти

номер (при N **mod** 3 = 0),
модуль (при N **mod** 3 = 1),
квадрат (при N **mod** 3 = 2)

такого

первого (при N **mod** 4 = 0),
последнего (при N **mod** 4 = 1),
минимального (при N **mod** 4 = 2),
максимального (при N **mod** 4 = 3)

элемента, который является

четным (при N **mod** 2 = 0),
не кратным N (при N **mod** 2 = 1)

числом и совпадает с кодом некоторой
буквы (при N **mod** 8 > 3),
цифры (при N **mod** 8 ≤ 3)»

будет иметь вид: «в заданной последовательности целых чисел найти модуль такого первого максимального элемента, который является не кратным 7 числом и совпадает с кодом некоторой буквы» .

Гл. 1 содержит вводные понятия и начинается с ответов на такие вопросы, связанные с алгоритмами и функциями, как: что такое алгоритм, что такое компьютер, как можно определять функции и как можно определить язык Zonnon?

Методам построения простейших программ посвящена гл. 2. В ней рассматриваются стандартные типы данных и средства организации вычислений в линейных программах. Излагаются вопросы доказательства свойств программ: поясняется, зачем нужны доказательства правильности программ, вводятся понятия промежуточных утверждений, внешней спецификации программы, полной и частичной правильности программ, описываются методы задания внешней спецификации простых программ, и излагается метод промежуточных утверждений доказательства их свойств. Приводятся образцы разработки линейных программ на примере решения таких задач, как периметр и площадь прямоугольного треугольника, симметричная буква, возведение в степень и площадь треугольника.

Гл. 3 посвящена методам построения простых программ без циклов. Она начинается с описания средств для организации и описания ветвлений в программах: вводится блок-схемные представления программ и их фрагментов, описываются условные операторы и операторы выбора, излагаются методы доказательства свойств ветвящихся программ. Затем приводятся образцы разработки ветвящихся программ на примере решения таких задач, как точка в треугольнике, максимум из трех чисел, табличное задание функции, анализ квадратного уравнения и определение типа треугольника.

Гл. 4 посвящена методам построения итеративных программ. В ней описываются средства для организации и анализа свойств циклических вычислений: операторы цикла с условием на продолжение, операторы цикла с условием на окончание и цикла с параметрами. Излагаются методы пошаговой разработки программ. Приводятся образцы решения задач независимой обработки элементов последовательности (перекодировщик, выборка элементов последовательности), вычисления элементов последовательности (нахождение факториала, вычисление числа e, схема Горнера), реализации функций на последовательностях (подсчет вхождений, количество мак-

симумов), обработки последовательности последовательностей и реализации двойных циклов (обработка слов предложения, приближенное вычисление сумм рядов).

Вопросам построения программ обработки структурированных данных посвящена гл. 5. В ней описываются средства задания и анализа программ со структурами данных, приводятся образцы решения задач обработки векторов (векторная функция, поиск в упорядоченном векторе, подсчет количеств вхождений букв, свертка вектора) и программы обработки матриц (поиск максимума в матрице, произведение матриц, преобразование матриц, поиск номера строки-серии).

Гл. 6 посвящена классу программ с процедурами и функциями. В ней рассматриваются правила описания и вызовов процедур, блоки и локализация имен, изменение действий (входные параметры) и получение результатов (выходные параметры) процедур, а также вычисление единственного значения (функции). Излагаются методы использования процедур и функций для пошаговой разработки программ, рассматриваются рекурсивные подпрограммы и исследуется, когда и как нужно использовать рекурсию, описывается метод структурной индукции для доказательства свойств рекурсивных программ. Приводятся образцы решения задач с использованием процедур и функций, таких как обработка последовательности векторов, Ханойские башни, кратные суммы, быстрая сортировка и числа Фибоначчи.

4. ЦЕЛИ И СОДЕРЖАНИЕ ПРАКТИКУМА

Основной задачей компьютерного практикума является не только и не столько обучение студентов собственно записи (кодированию) известного алгоритма на языке Zonnon для платформы .NET, а практическое закрепление знаний, получаемых в ходе лекций и семинарских занятий по вводному курсу программирования, и овладение общими методами, приемами и навыками технологии решения задач на компьютере. Основная цель, которая ставится перед студентом при выполнении индивидуальных заданий, составляющих компьютерный практикум, — это практическое освоение всех этапов разработки надежной и наглядной интерактивной (диалоговой) Zonnon-программы для компьютерного решения несложной задачи, требующей разработки алгоритма, обработки сложных структур данных и создания дружественного интерфейса.

В силу этого, тематика индивидуальных заданий для компьютерного практикума определяется, в первую очередь, всеми видами работ, которые должен освоить студент, чтобы научиться создавать качественные (эффективные, наглядные и надежные) нетривиальные программы. В большинстве случаев задача, решаемая во время выполнения индивидуального задания, — это задача невычислительного характера, имеющая краткую и точную (содержательную) формулировку и допускающая большое разнообразие решений.

- Вместе с тем студенту и преподавателю нужно иметь в виду, что многие из методов и понятий, используемых в индивидуальных заданиях, являются неотъемлемой частью образования современного специалиста, использующего компьютер для решения своих задач. Поэтому предполагается, что при прохождении компьютерного практикума студент получит не менее пяти индивидуальных заданий, по одному из каждого тематического раздела.

Выполнение каждого задания, составляющего компьютерный практикум, в общем случае включает следующие виды работ:

- анализ условия задачи и выработка подхода к ее решению;
- пошаговая разработка (на основании выбранного подхода) алгоритма решения и его описание;
- обоснование алгоритма;
- выбор и обоснование представления для входных, выходных и промежуточных данных;
- кодирование алгоритма, т.е. его запись на языке Zonnon;
- выбор и обоснование набора тестов, на которых будет проверяться программа;
- отладка программы и демонстрация ее правильной работы на выбранном наборе тестов.

Это разбиение условно в том смысле, что фактически некоторые виды работ тесно переплетаются и выполнение их обычно составляет единый процесс. Например, строить набор тестов удобнее одновременно с построением самого алгоритма, а обосновывать правильность работы алгоритма удобно путем детальной демонстрации процесса его построения.

Выполнение каждого задания студент завершает составлением отчета, который включает:

- формулировку задачи;

- описание программы для пользователя, ее внешнюю спецификацию, т.е. описание способа задания входных данных, вида результатов программы при заданных входных данных и сценария диалога в процессе исполнения программы;
- словесное описание алгоритма и обоснование его правильности и эффективности;
- текст программы;
- описание тестового набора и его обоснование.

Предполагается, что студент, решая задачу, создает интерактивную (диалоговую) Zonnon программу для платформы .NET, которая за один запуск может обрабатывать не один входной набор, а последовательность входных наборов произвольной длины. Эти входные наборы либо заранее размещаются в специальных "входных" файлах программы, предъявляемых преподавателю вместе с разработанной программой, либо задаются пользователем программы в процессе ее исполнения. Таким образом, разработанная студентом программа после завершения обработки очередного входного набора в зависимости от указания пользователя может либо завершить свою работу (остановиться), либо продолжить ее и перейти к обработке следующего входного набора. В последнем случае программа вступает в диалог с пользователем, в процессе которого пользователь программы готовит и инициирует новый счет по программе. Для этого он либо вводит очередной входной набор с помощью клавиатуры и мышки, либо дает указание программе, из какого файла ей нужно взять необходимые данные.

При этом совсем не требуется, чтобы при переходе программы к обработке следующего входного набора этот набор каждый раз целиком задавался заново. Например, во многих заданиях удобно разделить входной набор на две части, выделяя более общие (как правило, более объемные) исходные данные задачи в отдельную часть, и предусмотреть специальный вид реализации счета по программе на последовательности входных данных, различающихся второй частью, без необходимости повторного задания первой. Например, в задаче нахождения кратчайшего пути между двумя заданными вершинами заданной системы дорог более общей (и более объемной) частью входного набора является система дорог. Поэтому естественно решение, при котором система дорог является общей для нескольких следующих друг за другом счетов по программе. В этом случае каждый раз, переходя к новому счету по программе, пользователь может выбрать один из двух вариантов продолжения работы:

- осуществить ввод очередной системы дорог из заданного файла с выводом его изображения на экран,

- запустить нахождение кратчайшего пути между двумя заданными городами в текущей системе дорог.

Важно, чтобы до выполнения заданий студент и преподаватель согласовали все требования, предъявляемые к их выполнению, в том числе, виды работ по каждому заданию, а также порядок и сроки сдачи заданий. В любом случае понимание условий задач студентом не должно отличаться от понимания преподавателем. Поэтому раньше, чем студент начнет разрабатывать алгоритм, студент и преподаватель должны вместе тщательно проанализировать условие задачи и обсудить особенности и трудности ее решения, зафиксировать интерфейс программы, включая представление входных и выходных данных.

Необходим также постоянный контроль текущего состояния решения заданий для координации установленных преподавателем требований и действительных намерений студента. Необходимо уделять особое внимание высокой надежности создаваемых программ. Программа должна содержать достаточное число так называемых стопоров ошибок — динамических проверок справедливости утверждений, характеризующих правильность функционирования программы и ее применения. Очень важно, чтобы программа не только давала правильные результаты для корректных исходных данных, но и осмысленно реагировала на некорректные данные и указания пользователя. Не менее важно, чтобы программа выдавала на экран (или в специальный файл) в удобном виде информацию о ходе вычисления по программе в таком объеме (и в таком виде), который позволяет легко идентифицировать ошибку (как в программе, так и во входных данных) и локализовать место ее возникновения.

Программа обязательно должна быть наглядной, т.е. хорошо структурированной, с достаточным количеством комментариев и т.д., а также содержать инструкцию по своему использованию («встроенный help»).

Особое внимание нужно уделить правильности программы и полноте тестового набора. Описание каждого теста помимо файла с входными данными должно включать информацию, достаточную для оценки правильности работы программы на этих входных данных. Минимальное требование к тестовому набору состоит в том, что каждый оператор программы должен быть достижим (т.е. выполняться) хотя бы на одном тесте из этого набора. Поэтому среди тестов набора должны быть представлены и некорректные исходные данные.

Входные и выходные данные разработанной программы должны быть естественными для человека. Степень естественности представления определяется объемом той работы, которая требуется для перехода от содержа-

тельного описания входных данных к их представлению (очень часто неестественность входного представления проявляется в его неоправданно большом размере) и от представления выходных данных к их содержательному пониманию.

Задачи, составляющие индивидуальные задания, допускают, как правило, целый спектр различных по эффективности решений. Предполагается, что студент должен выбрать и обосновать по возможности хорошее решение. Минимальным требованием к эффективности решения является успешная работа программы на согласованном с преподавателем тестовом наборе.

Для удобства все индивидуальные задания разбиты на три группы:

- обычная сложность (их формулировки не имеют пометки),
- повышенная сложность (с пометкой "↑"),
- пониженная сложность (с пометкой "↓").

При оценке сложности задания рассматривались три показателя:

- сложность структур данных,
- сложность вычислений,
- изобретательность.

В показатель «изобретательность» включались такие свойства задания, как непривычность для студента понятий, используемых в задании, сложность извлечения из определений тех свойств, на которых должен базироваться алгоритм решения задания, а также сложная связь между структурами данных и вычислениями. Каждый из указанных трех показателей задания оценивался в баллах 0, 1 или 2. Пометку "↓" получили задания, набравшие в сумме по трем показателям всего 1 балл, а пометку "↑" — задания, суммарный балл которых больше 3.

4. ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ

Книга [3] предназначена для проведения компьютерного практикума по курсу программирования. Она содержит 500 индивидуальных заданий различной сложности, ориентированных на приобретение студентами навыка практического решения задач создания пользовательских и Web-приложений для платформы Microsoft.NET, требующих разработки алгоритма, обработки сложных структур данных и разработки дружественного интерфейса. Каждое индивидуальное задание — это самостоятельная, как правило, комбинаторная или логическая задача с краткой и четкой форму-

лировкой, не содержащей описания алгоритма. Тематические задачи разбиты на пять разделов по 100 заданий в каждом разделе:

- графы и системы дорог;
- грамматики, языки и автоматы;
- формулы и программы;
- геометрия;
- игры и модели.

Основную часть книги составляют подробные рекомендации по выполнению разных видов работ, которые должен освоить студент, чтобы научиться создавать эффективные, наглядные и надежные нетривиальные приложения для платформы Microsoft.NET, в том числе для работы в среде Интернет.

Книга состоит из введения и 6 глав.

Нужно сказать, что при составлении сборника авторы стремились сделать изложение доступным для читателей, не обладающих специальными знаниями ни по математике, ни по программированию. Поэтому все необходимые для решения задач сведения по программированию, в том числе и полное описание языка Zonnon (гл. 1), включены в сборник, а условия большинства заданий и упражнений доступны для учащихся старших классов. Приведенные в сборнике (гл. 2) определения понятий, встречающихся в заданиях, вполне достаточны, для того чтобы можно было понимать формулировки заданий (гл. 6). Однако для построения эффективного решения некоторых заданий может оказаться полезным знание некоторых свойств определяемых здесь математических объектов. Поэтому во введении не только описываются цели и задачи практикума и даются общие рекомендации преподавателю и студенту, но и дается обзор дополнительной литературы, которая может помочь студенту при выполнении его заданий.

Гл. 3, 4 и 5 содержат подробные рекомендации по решению индивидуальных заданий, составляющих практикум. В гл. 3 даются общие рекомендации по разработке и обоснованию алгоритма, представлению данных, анализу алгоритма и его сложности, выбору и обоснованию тестов, начиная с анализа условия задачи и кончая отладкой созданной программы. Особенностью заданий данного практикума является то, что они связаны со сложными структурами данных, с организацией поиска или порождения тех элементов конечного (но, как правило, чрезвычайно большого) множества, которые удовлетворяют определенным ограничениям и требуют при своем решении анализа алгоритма и его сложности. В гл. 4 приводятся рекомендации по алгоритмическому решению задач, связанных с организацией вычислений на дискретных конечных математических структурах. В ней

описываются основные типы данных, возникающие при выполнении заданий (списки, стеки, очереди, деревья, графы, орграфы), приводится язык описания алгоритмов, рассматривается поиск с возвращением, описываются алгоритмы поиска с возвращением, обходов ордерева в глубину и в ширину, обходов графа в глубину и в ширину, дается усовершенствование поиска с возвращением, приводятся методы ветвей и границ, динамического программирования, а также порождения объектов, перестановок, подмножеств множества, сочетаний. В гл. 5 рассматриваются вопросы выбора представления для структур данных, присутствующих в формулировках заданий, и даются конкретные рекомендации по выбору представлений для таких структур данных, как множество точек на плоскости, прямая на плоскости, окружность и многоугольник, последовательность, вектор, матрица, тексты, слова и предложения, граф, корневое дерево, система дорог, грамматика, автоматная диаграмма, логическая формула, логический фрагмент, простое выражение, полином, игры.

ЗАКЛЮЧЕНИЕ

В статье представлен вводный курс по программированию на базе языка Zonnon, работа над которым ведется в Цюриховском институте информатики. Курс опирается на опыт преподавания основного курса по программированию для студентов механико-математического факультета НГУ с использованием языка Паскаль. Язык Zonnon задуман как дальнейшая эволюция хорошо известного и широко применяемого на западе в учебных целях языка Оберон, являющегося преемником языков Паскаль и Модула-2. Развивая язык Оберон, исходя из современных потребностей в программировании, авторы сохранили в языке Zonnon такие важные черты Оберона и его предшественников, как компактность языка, ясность, недвусмысленность и ортогональность его основных понятий. Поэтому можно ожидать, что язык Zonnon и данный вводный курс будут востребованы теми учебными заведениями, которые в настоящее время используют Паскаль в качестве языка начального обучения программированию и имеют желание перейти к более современному курсу программирования, охватывающему концепции языков программирования нового поколения, таких как Java и C#, но осуществить этот переход плавно, без резкого изменения сложившегося стиля преподавания программирования.

СПИСОК ЛИТЕРАТУРЫ

1. **Джонсон Б., Скибо К., Янг М.** Основы Microsoft Visual Studio .NET 2003. — М.: Русская Редакция, 2003.
2. **Касьянов В. Н., Касьянова Е.В.** Введение в программирование. — `http://pco.iis.nsk.su/ICP`
3. **Касьянов В. Н., Касьянова Е.В.** Практикум по программированию. — `http://pco.iis.nsk.su/ICP`
4. **Касьянов В.Н.** Курс программирования на Паскале в заданиях и упражнениях. — Новосибирск: НГУ, 2001.
5. **Касьянов В.Н.** Язык программирования Минал. — Новосибирск: НГУ, 1979.
6. **Касьянова Е.В.** Язык программирования Zonnon для платформы .NET // Программные средства и математические основы информатики. — Новосибирск: ИСИ СО РАН, 2004. — С.189–205.
7. **Основы** языка ПАСКАЛЬ-360 / Сост. Касьянов В.Н. — Новосибирск: НГУ, 1982.
8. **Программирование** на мини-Фортране / Сост. Касьянов В.Н. — Новосибирск: НГУ, 1981.
9. **Просиз Дж.** Программирование для .NET. — М.: Русская Редакция, 2003.
10. **Уоткинз Д., Хаммонд М., Эйбрамз Б.** Программирование на платформе .NET. — М.: Вильямс, 2003.
11. **Gutknecht J., Zueff E.** Zonnon Language Experiment, or How to Implement a Non-Conventional Object Model for .NET // OOPSLA'02. — Seattle, Washington, 2002.
12. **Gutknecht J., Zueff E.** Zonnon Language Report. — Zurich, Institute of Computer Systems ETH Zentrum, 2004.
13. **Wirth N.** From Modula to Oberon // Software — Practice and Experience. — 1988. — Vol. 18, N 6. — P. 661 — 670.
14. **Wirth N.** The programming language Oberon // Software — Practice and Experience. — 1988. — Vol. 18, N 6. — P. 671 — 690.
15. **Zonnon** Builder. — Дистрибутив системы доступен по адресу: `http://www.zonnon.ethz.ch.`

ПРИЛОЖЕНИЕ

Ниже при описании синтаксиса языка Zonnon используется Расширенный Бекуса—Наура Формализм (РБНФ), который характеризуется следующими свойствами.

- Альтернативы разделяются символом |.
- Скобки [и] обозначают факультативность выражения в скобках.
- Скобки { и } обозначают повторение содержимого (возможно 0

раз).

- Скобки (и) используются для формирования групп элементов.
- Нетерминальные символы начинаются с прописной буквы (например, *Statement*).
- Терминальные символы либо начинаются со строчной буквы (например, *letter)*, либо записывается целиком полужирным шрифтом (например, **begin**), или представляются в виде строк (например, ":=").
- Комментарии начинаются с символа // и продолжаются до конца строки.

```
// Синтаксис Zonnon в РБНФ
// Версия от 11 марта 2004

// 1. Программа и программные единицы (Program and program units)
CompilationUnit = { ProgramUnit "." }.
ProgramUnit = ( Module | Definition | Implementation | Object ).

// 2. Модули (Modules)
Module = module [ ModuleModifier] ModuleName [ ImplementationClause ] ";"
    [ ImportDeclaration ]
    ModuleDeclarations
    ( BlockStatement | end ) SimpleName.
ModuleModifier = "{" ident "}" // private или public; private по умолчанию
ModuleDeclarations = { SimpleDeclaration | NestedUnit ";" }
    { ProcedureDeclaration | OperatorDeclaration }
    { ActivityDeclaration }.
NestedUnit = ( Definition | Implementation | Object ).
ImplementationClause = implements DefinitionName { "," DefinitionName }.
ImportDeclaration = import Import { "," Import } ";".
Import = ImportedName [ as ident ].
ImportedName = ( ModuleName | DefinitionName | ImplementationName |
    NamespaceName |ObjectName).

// 3. Определения (Definitions)
Definition = definition [ DefinitionModifier] DefinitionName [ RefinementClause ] ";"
    [ ImportDeclaration ]
    DefinitionDeclarations
    end SimpleName.
DefinitionModifier = "{" ident "}" // private или public; public по умолчанию
RefinementClause = refines DefinitionName.
DefinitionDeclarations = { SimpleDeclaration } { { ProcedureHeading ";" }|
    ActivitySpecification }.
ActivitySpecification =
    activity "{" ProtocolEBNF "}" ActivityName "=" EnumType ";".
```

```
ProtocolEBNF = Specification of the protocol in EBNF based on the syntax
    alphabet.

// 4. Реализации (Implementations)
Implementation = implementation [ImplementationModifier]
    ImplementationName ";" [ ImportDeclaration ]
    Declarations
    ( BlockStatement | end ) SimpleName.
ImplementationModifier = "{" ident "}". // private или public; public по умолчанию
// 5. Объекты
Object = object [ ObjModifier ] ObjectName [ FormalParameters ]
    [ ImplementationClause ] ";" [ ImportDeclaration ]
    Declarations
    { ActivityDeclaration }
    ( BlockStatement | end ) SimpleName.
ObjModifier = "{" ident "}". // value или ref; value по умолчанию
    // private или public; private по умолчанию
ActivityDeclaration = activity ActivityName [ ImplementationClause ] ";"
    Declarations
    ( BlockStatement | end SimpleName ).

// 6. Описания (Declarations)
Declarations = { SimpleDeclaration } { ProcedureDeclaration }.
SimpleDeclaration = ( const [DeclModifier] { ConstantDeclaration ";" }
    | type [DeclModifier] { TypeDeclaration ";" }
    | var [DeclModifier] { VariableDeclaration ";" }
    ).
DeclModifier = "{" ident "}". // public или private или immutable
ConstantDeclaration = ident "=" ConstExpression.
ConstExpression = Expression.
TypeDeclaration = ident "=" Type.
VariableDeclaration = IdentList ":" Type.

// 7. Типы (Types)
Type = ( TypeName [ Width ] | EnumType | ArrayType | ProcedureType |
    InterfaceType ).
Width = "{" ConstExpression "}".
ArrayType = array Length { "," Length } of Type.
Length = ( ConstExpression | "*" ).
EnumType = "(" IdentList ")".
ProcedureType = procedure [ ProcedureTypeFormals ].
ProcedureTypeFormals = "(" [ PTFSection { ";" PTFSection } ] ")"
    [ ":" FormalType ].
PTFSection = [ var ] FormalType { "," FormalType }.
FormalType = { array "*" of } ( TypeName | InterfaceType ).
InterfaceType = object [ PostulatedInterface ].
PostulatedInterface = "{" DefinitionName { "," DefinitionName } "}".
```

```
// 8. Процедуры и знаки операций (Procedures & operators)
ProcedureDeclaration = ProcedureHeading [ ImplementationClause ] ";"
    [ ProcedureBody ";" ].
ProcedureHeading = procedure [ ProcModifiers ]
    ProcedureName [ FormalParameters ].
ProcModifiers = "{" ident { "," ident } "}". // private, public, sealed
ProcedureBody = Declarations BlockStatement SimpleName.
FormalParameters = "(" [ FPSection { ";" FPSection } ] ")" [ ":" FormalType ].
FPSection = [ var ] ident { "," ident } ":" FormalType.
OperatorDeclaration = operator [ OpModifiers ] OpSymbol
    [ FormalParameters ] ";" OperatorBody ";".
OperatorBody = Declarations BlockStatement OpSymbol.
OpModifiers = "{" ident { "," ident } [ "," Priority ] "}" | "{" Priority "}".
Priority = ConstExpression.
OpSymbol = string. // 1,2-литерная строка;
    // множество возможных символов ограничено

// 9. Операторы (Statements)
StatementSequence = Statement { ";" Statement }.
Statement = [ Assignment
    | ProcedureCall
    | IfStatement
    | CaseStatement
    | WhileStatement
    | RepeatStatement
    | LoopStatement
    | ForStatement
    | await Expression
    | exit
    | return [ Expression ]
    | BlockStatement
    | launch Statement
    | Send
    | BlockingReceive
    | NonBlockingReceive
    ].
Assignment = Designator ":=" Expression.
ProcedureCall = Designator.
IfStatement = if Expression then StatementSequence
    { elsif Expression then StatementSequence }
    [ else StatementSequence ]
    end.
CaseStatement = case Expression of
    Case { "|" Case }
    [ else StatementSequence ]
    end.
```

```
Case = [ CaseLabel { "," CaseLabel } ":" StatementSequence ].
CaseLabel = ConstExpression [ ".." ConstExpression ].
WhileStatement = while Expression do StatementSequence end.
RepeatStatement = repeat StatementSequence until Expression.
LoopStatement = loop StatementSequence end.
ForStatement = for ident ":=" Expression to Expression [ by ConstExpression ]
    do StatementSequence end.
BlockStatement = begin [ BlockModifiers ]
    StatementSequence
    { ExceptionHandler }
    [ CommonExceptionHandler ]
    end.
BlockModifiers = "{" ident { "," ident } "}". // locked, concurrent
ExceptionHandler = on ExceptionName { "," ExceptionName } do
    StatementSequence.
CommonExceptionHandler = on exception do StatementSequence.
Send = send expression [ "=>" activity].
BlockingReceive = receive [ activity "=>" ] variable.
NonBlockingReceive = accept [ activity "=>" ] variable.

// 10. Выражения (Expressions)
Expression = SimpleExpression
    [ ( "=" | "#" | "<" | "<=" | ">" | ">=" | in ) SimpleExpression ]
    | Designator implements DefinitionName
    | Designator is TypeName.
SimpleExpression = [ "+"|"-" ] Term { ( "+" | "-" | or ) Term }.
Term = Factor { ( "*" | "/" | div | mod | "&" ) Factor }.
Factor = number
    | CharConstant
    | string
    | nil
    | Set
    | Designator
    | new TypeName [ "(" ActualParameters ")" ]
    | new ActivityInstanceName
    | "(" Expression ")"
    | "~" Factor.
Set = "{" [ SetElement { "," SetElement } ] "}".
SetElement = Expression [ ".." Expression ].
Designator = Instance
    | Designator "{" Type "}" // Casting
    | Designator "^" // Dereference
    | Designator "[" Expression { "," Expression } "]" // Array element
    | Designator "(" [ ActualParameters ] ")" // Function call
    | Designator "." MemberName // Member selector
Instance = ( self | InstanceName | DefinitionName "(" InstanceName ")" ).
ActualParameters = Actual { "," Actual }.
```

```
Actual = Expression [ "{" [ var ] FormalType "}" ].
    // Аргумент с сигнатурой типа

// 11. Constants
number = (whole | real) [ "{" Width "}" ].
whole = digit {digit} | digit {hexDigit} "H".
real = digit { digit } "." { digit } [ ScaleFactor ].
ScaleFactor = "E" ["+" | "" ] digit { digit }.
HexDigit = digit | "A" | "B" | "C" | "D" | "E" | "F".
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9".
CharConstant = "'" character "'" | "'" character "'" | digit { HexDigit } "X".
string = '"' { character } '"' | "'" { character } "'".
character = letter | digit | Other.
Other = // Любая литера из алфавита за исключением тех,
    // которые используются…

// 12. Идентификаторы и имена (Identifiers & names)
ident = ( letter | "_" ) { letter | digit | "_" }.
letter = "A" | ... | "Z" | "a" | ... | "z" |
    // любая другая "культурно-определенная" буква
IdentList = ident { "," ident }.
QualIdent = { ident "." } ident.
DefinitionName = QualIdent.
ModuleName = QualIdent.
NamespaceName = QualIdent.
ImplementationName = QualIdent.
ObjectName = QualIdent.
TypeName = QualIdent.
ExceptionName = QualIdent.
InstanceName = QualIdent.
ActivityInstanceName = QualIdent.
ProcedureName = ident.
ActivityName = ident.
MemberName = ( ident | OpSymbol ).
SimpleName = ident.
```

Т.Г. Коновалова, В.М Комашко*

ОБРАБОТКА ДАННЫХ МИКРОЧИПОВЫХ ЭКСПЕРИМЕНТОВ ПРИ ПОМОЩИ ЯЗЫКА «R»

ВВЕДЕНИЕ

В решении тех многих вопросов, что встали перед биологами в постгеномную эру, существуют два основных подхода, предлагаемых функциональной геномикой. Первый подход основан на исследовании нуклеотидных последовательностей, приведший к обнаружению различных мотивов, сайтов и структурных доменов. Второй подход — для изучения функции гена — основан на исследовании модели его экспрессии (экспрессия — продукция гена, коррелирующая с конечным количеством белка, произведенным данным геном). Новым классом биотехнологий для одновременного отслеживания изменения уровня экспрессии десятков тысяч генов в разных экспериментах и под влиянием различных условий стала технология микрочипов.

В данной статье рассматривается подход к решению задачи выделения генов, изменивших с определенной достоверностью уровень своей экспрессии на основе данных из нескольких микрочиповых экспериментов. Эксперименты заранее разбиты на две группы. Над этими данными вначале проводится нормализация, а затем — кластеризация. В данной статье под кластеризацией подразумевается выделение двух кластеров — гены, изменившие и не изменившие свою экспрессию.

Существуют две основные технологии синтеза микрочипов: кДНК микрочипы и синтез *in situ (олигонуклеотидные чипы)* — чипы, состоящие из олигонуклеотидов (цепочек ДНК длины 10–20 нуклеотидов). кДНК микрочипы предоставляют относительный уровень экспрессии каждого отдельного гена, в то время как олигонуклеотидные чипы дают информацию об абсолютном уровне экспрессии.

После сканирования и преобразования изображения в цифры, отражающие уровень экспрессии каждого гена, исследователю предоставляется матрица размера $n \times l$, где n — это количество проведенных эксперимен-

* tiv_@ngs.ru

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 05-08-33448).

тов, будь то сравнение между тканью, пораженной раком, и такой же, но здоровой, или исследования экспрессии генов в ходе клеточного цикла, где каждый столбец представляет собой временную точку; *l* — обозначает количество проб (проба соответствует гену или части гена), причем *l* может варьировать от 100 до десятков тысяч. В такой матрице необходимо найти ключевые гены, экспрессия которых изменилась, для чего было разработано множество методов. Для достоверности результатов и возможности сравнения полученных данных из различных источников, результаты эксперимента до поиска ключевых генов должны быть нормализованы, т. е. приведены к единой шкале. Это, а также разработка единого языка описания, позволяют сделать данные доступными и сопоставимыми.

Во многих случаях целью микрочипового эксперимента является сравнение уровней экспрессии в двух различных образцах. В большинстве случаев один образец рассматривается как контрольный, а второй — как экспериментальный. При этом основной задачей является определение таких генов, экспрессия которых различается при сравнении двух исследуемых образцов. Несмотря на то что на первый взгляд эта проблема не кажется сложной, она становится такой, из-за того что на полученные значения интенсивности влияет множество эффектов и шума, связанных с технологическими ограничениями эксперимента.

Решение задач нормализации и кластеризации данных удобно выполнять в среде программирования языка *R GUI* [R Development Core Team, 2004] с использованием пакетов *Bioconductor* [Ihaka, Gentleman, 1996]. В данной статье описан процесс обработки данных с олигонуклеотидного чипа компании Affimetrix. В эксперименте рассматривалась экспрессия генов в клетках раковых опухолей мозга. Для анализа мы взяли результаты сканирования 12 чипов (по 6 для глио- и олигобластомы, являющихся опухолями вспомогательных тканей мозга).

Данная работа может быть использована как руководство по применению языка R и пакета BioConductor для обработки результатов микрочиповых экспериментов.

Список терминов

Проба — олигонуклеотид из 25 пар оснований, используемый для связывания РНК мишеней.

PM, perfect match — пробы, разработанные так, чтобы идеально совпадать с последовательностью мишени.

MM, mismatch — пробы, имеющие замену в одном нуклеотиде, для учета неспецифического связывания.

affyID — идентификатор набора проб, соответствующий гену или части гена.

CEL файл — файл простого текстового формата, полученный после цифровой обработки цветного изображения микрочипа и содержащий информацию об интенсивностях для PM/MM пар, а также физические координаты каждой клетки на слайде.

1. МЕТОДЫ НОРМАЛИЗАЦИИ

Пакеты используемого в данной работе модуля *Bioconductor* позволяют своим пользователям применять несколько широко используемых алгоритмов, а также комбинировать их с помощью функции *expresso*(). Одним из наиболее быстрых и эффективных является метод *RMA* [1], который включает в себя несколько этапов.

1. Коррекция по фону — *rma*. Так как, в отличие от кДНК микрочипов, олигонуклеотидный чип вообще не содержит фона, то коррекция осуществляется за счет соседних проб для учета неспецифического связывания и оптического шума. Интенсивности *PM* корректируются в глобальной модели для распределения интенсивностей проб. В данной модели предусматривается рассмотрение кривой эмпирического распределения интенсивностей проб. В частности, наблюдаемые *PM (Perfect Match)* пробы моделируются как сумма нормальной шумовой компоненты N (нормальной со средним μ и дисперсией σ^2) и экспоненциальной сигнальной компоненты S (экспоненциальная со средним α). Для избежания вероятности отрицательных значений нормаль округляется до нуля. Пусть O — это наблюдаемая интенсивность, тогда корректировка будет выполнена по следующей формуле:

$$E(s \mid O = o) = a + b \frac{\phi\left(\frac{a}{b}\right) - \phi\left(\frac{o-a}{b}\right)}{\Phi\left(\frac{a}{b}\right) + \Phi\left(\frac{o-a}{b}\right) - 1},$$

где $a = s - \mu - \sigma^2\alpha$ и $b = \alpha$. Здесь ϕ — плотность стандартного нормального распределения, а Φ — это распределение функций.

Необходимо заметить, что в этом методе для коррекции фона значения *MM* в расчет не принимаются.

2. В качестве метода нормализации был выбран метод квантилей [2]. Цель данного метода состоит в том, чтобы сделать распределение интенсивностей проб для каждого чипа одинаковым для всех остальных чипов. Этот метод мотивирован известным в статистике квантиль-графиком, показывающим, что распределение двух векторов данных одинаково, если график представляет собой прямую диагональную линию, и распределение разное, если график имеет другой вид. Увеличим количество векторов до n: если все n векторов имеют одинаковое распределение, то построение квантилей в размерности n дает прямую линию вдоль диагонали, заданную единичным вектором $\left(\frac{1}{\sqrt{n}},...,\frac{1}{\sqrt{n}}\right)$. Это предположение можно сделать для набора данных, имеющих одинаковое распределение, если спроектировать все точки n-размерного квантильного графика на эту диагональ.

Пусть $q_k = (q_{k1},...,q_{kn})$ для $k = \overline{1,p}$ будет вектором k-го квантиля для всех n чипов $q_k = (q_{k1},...,q_{kn})$ и $d = \left(\frac{1}{\sqrt{n}},...,\frac{1}{\sqrt{n}}\right)$ будет единичной диагональю. Чтобы трансформировать квантили так, чтобы они лежали на диагонали, рассмотрим проекцию q на d:

$$proj_d q_k = \left(\frac{1}{n}\sum_{j=1}^{n} q_{kj},...,\frac{1}{n}\sum_{j=1}^{n} q_{kj}\right).$$

Это подразумевает, что мы можем приписать каждому чипу одинаковое распределение, взяв средний квантиль и заместив им значения данных в исходных данных. Это оправдывается следующим алгоритмом.

1. Данные n чипов размера p представляют собой матрицу X размерности $p \times n$, где каждый чип — это столбец в данной матрице.
2. Сортируется каждый столбец матрицы X для получения X_{sort}.
3. Находится среднее значение по рядам матрицы X_{sort} и это значение присваивается каждому элементу в ряду для получения X'_{sort}.
4. Матрица с нормализованными значениями получается перестановкой столбцов матрицы X'_{sort} так, чтобы порядок был таким же, как и вначале.

Одной из возможных проблем этого метода может быть принуждение всех квантилей к равенству. Это может быть наиболее проблематично в наибольшей степени в хвостах распределения, где может случиться так, что

все пробы имеют одинаковое значение для всех чипов. Однако на практике, так как уровень экспрессии вычисляется с использованием всех величин множества проб, этот возможный недостаток не становится реальной проблемой.

Согласно методу расчета коррекции по фону, при коррекции на значения *PM*, значения *MM (MisMatch)* не учитываются вообще, хотя, например, у Affymetrix (2002) при расчете *PM* идеальное значение *MM* отнимается от значения *PM*, при этом *MM* всегда меньше *PM*.

3. Для вычисления меры экспрессии был использован метод *medianpolish*, также предложенный в [1]. Линейная мультичиповая модель состоит в корректировке данных из каждого набора проб. В частности, для набора k с пробами $i = 1,..,I_k$ и данными из $j = 1,...,J$ чипов строится следующая модель:

$$\log_2(PM_{ij}^{(k)}) = \alpha_i^{(k)} + \beta_j^{(k)} + \varepsilon_{ij}^{(k)},$$

где $\alpha_i^{(k)}$ — эффект пробы и $\beta_j^{(k)}$ — логарифм по основанию два величины экспрессии. Алгоритм *medianpolish* нужен для устойчивого прилаживания модели.

Таким образом, полный набор методов для обработки экспериментальных данных включает в себя несколько этапов: 1) коррекция по фону; 2) нормализация, т. е. приведение данных к единой шкале; 3) коррекция значений *PM*; 4) вычисление значения экспрессии.

2. МЕТОДЫ ВЫДЕЛЕНИЯ ГЕНОВ С ДИФФЕРЕНЦИАЛЬНОЙ ЭКСПРЕССИЕЙ

Задача идентификации таких генов разбивается на две части:

1) выбор статистики, которая ранжирует гены, для того чтобы доказать дифференциальную экспрессию — от самого строгого доказательства до самого слабого;

2) выбор критического значения статистики такого, что все значения больше этого критического будут считаться значимыми. Обычно при исследованиях выделяют около 100 генов, для которых возможно показать, что их экспрессия изменилась в сравнении между двумя условиями.

Метод выбора генов может быть характеризован в терминах положительной предсказанной величины (PPV), отрицательной предсказанной величины (NPV), специфичности и чувствительности. В общем, для любой

диагностики или метода классификации можно сравнить полученые результаты с экспериментальными. В ситуации, когда есть возможность принять всего лишь одно решение из двух, например, изменился или не изменился уровень экспрессии, результат всегда можно разделить на четыре категории: уровень действительно изменился и методом сообщен как изменившийся (true positives,TP); уровень не изменился, но метод сообщил о его изменении (false positives, FP, ошибка перепредсказания); действительно изменился, но метод сообщил как о неизменившемся (false negatives, FN, ошибка недопредсказания); действительно не изменился и методом сообщен как не изменившийся (true negatives,TN). Основываясь на этих категориях, можно выделить четыре количественных критерия: PPV, NPV, специфичность и чувствительность (рис.1).

Оценки этих значений обычно варьируют от 0 до 1, иногда их выражают в процентах. Наилучший метод характеризуется отсутствием ошибок недопредсказания и перепредсказания.

	Измененный	Не измененный	
Измененный (метод)	TP	FP	PPV= $\frac{TP}{TP+FP}$
Не измененный (метод)	FN	TN	NPV= $\frac{TN}{TN+FN}$
	Чувствительность $\frac{TP}{TP+FN}$	Специфичность $\frac{TN}{TN+FP}$	

Рис.1. Определение положительной и отрицательной предсказанных величин, специфичности и чувствительности. Для наилучшего метода PPV = NPV = специфичность = чувствительность = 1

Тестирование гипотез, коррекция для множественных сравнений

Другим подходом для выбора генов является использование одномерных статистических тестов. Пусть логарифмы отношений экспрессии будут принадлежать некоторому распределению.

Для данного порога и данного распределения уровень значимости, или величина *p,* — это вероятность того, что измеренная величина случайным

образом будет принадлежать заштрихованной области. Идея заключается в том, что ген, чье значение логарифма отношений попадает в некую область, находится далеко от среднего логарифма отношений уровней экспрессии и будет, таким образом, отнесен к дифференциально экспрессирующимся (экспрессия в данном случае будет повышена). Однако измеренный логарифм отношений может быть таким благодаря внешним факторам, например шуму. Вероятность этого обозначается величиной *p*. В этом случае отнесение гена к эспрессирующимся дифференциально будет ошибкой первого рода и величина *p* — это вероятность сделать такую ошибку.

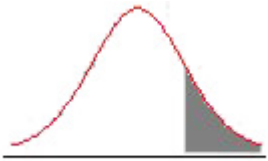


Рис. 2. Схематизированное изображение нормального распределения

Была принята нулевая гипотеза H_0, состоящая в том, что уровень экспрессии генов не поменялся при сравнении его в двух экспериментах. Соответственно, альтернативная или рабочая гипотеза H_1 предполагает наличие изменения уровня экспрессии, или, формализуя определение:

$$H_0: \mu_1 = \mu_2, \; H_1: \mu_1 \neq \mu_2 \; .$$

Для сравнения экспрессии применялась t-статистика:

$$t_j = \frac{\overline{x_{2j}} - \overline{x_{1j}}}{\sqrt{\frac{s_{1j}^2}{n_1} + \frac{s_{2j}^2}{n_2}}},$$

где $\overline{x_{ij}}$ — это среднее значение уровня экспрессии гена *j* в n_1 контрольных и n_2 исследуемых гибридизациях. В нашем случае $n_1 = n_2 = 3$; s_{1j}^2 и s_{2j}^2 — дисперсии.

Множественное тестирование, коррекция значения p

Вопрос множественной коррекции состоит в том, что необходимо полностью контролировать вероятность сделать ошибку первого рода. Эта вероятность равна вероятности сделать по крайней мере одну такую ошибку, и вычисляется она следующим образом:

$$P = 1 - (1 - p)^G, \quad [1]$$

где G — это количество генов. Данное выражение может быть переписано следующим образом:

$$\alpha_e = 1 - (1 - \alpha_c)^G, \quad [2]$$

где α_e — вероятность ошибки первого рода на уровне эксперимента, а α_c — вероятность ошибки для одного гена для одного сравнения. Задача состоит в вычислении такого уровня α, который нам нужно использовать для отдельных генов (α_c), для того чтобы быть уверенными, что глобальная или экспериментальная ошибка первого рода была меньше или равна выбранной α_e.

1. *Коррекция Шидака.* Из выражения [2] найдем значение α_c:

$$\alpha_c = 1 - \sqrt[G]{1 - \alpha_e},$$

где G — количество генов.

2. Коррекция Бонферрони. Для небольших значений p выражение [2] может быть аппроксимировано двумя первыми членами распределения бинома:

$$\alpha_e = 1 - (1 - \alpha_c)^G = 1 - (1 - G * \alpha_c + ...) \approx G * \alpha_c.$$

Используя эту аппроксимацию, вычисляем экспериментальный уровень α_c:

$$\alpha_e = \alpha_c * G \Rightarrow \frac{\alpha_e}{G}$$

3. Скачкообразная коррекция Холма. На первом этапе выбирается экспериментальный уровень значимости α_e. Гены упорядочиваются в порядке увеличения их значений p.

Сравниваются p-значения каждого гена с порогом, который зависит от позиции гена в упорядоченном списке. Порог вычисляется следующим об-

разом: $\frac{\alpha_e}{G}$ для первого гена, а для второго гена $\frac{\alpha_e}{G-1}$ и так далее. Пусть k будет максимальным i, для которой $p_i < \frac{\alpha_e}{G-i+1}$. Нулевая гипотеза будет отвергнута для всех i, меньших или равных k.

4. Метод отношения ошибочного предсказания. Алгоритм схож с методом скачкообразной коррекции Холма за исключением выбора порогов. Пусть k будет максимальным i, для которой $p_k < \frac{i}{G}\alpha_e$. Нулевая гипотеза будет отвергнута для всех i, меньших или равных k.

5. Перестановка. Как уже говорилось, данные после нормализации представляют собой матрицу, где столбцы обозначают различные эксперименты, а строки — гены. В этом методе на первом этапе производится перестановка столбцов матрицы или, что эквивалентно, меток каждого столбца. На каждой перестановке вычисляются новые значения p и корректируются с использованием метода скачкообразной коррекции Холма. Весь процесс — случайная перестановка меток столбцов и тестирование — повторяется сотни или десятки тысяч раз, что зависит от количества столбцов. В конце концов, величина p для гена i будет вычисляться следующим образом:

p для гена I = число перестановок, для которых $u_i^{(b)} \geq t_i$ */полное число перестановок,* где $u_i^{(b)}$ — это величины, скорректированные методом скачкообразной коррекции Холма.

3. ОБРАБОТКА ДАННЫХ В СРЕДЕ ПРОГРАММИРОВАНИЯ ЯЗЫКА *R*

1. Инсталляция R GUI и пакетов Bioconductor

С сайта `http://www.r-project.org/` скачивается и устанавливается файл rw1090.exe.

Установка пакетов Bioconductor осуществляется командами

```
R> source("http://www.bioconductor.org/getBioC.R")
R> getBioC() # Установка всех пакетов по умолчанию
```

или через меню *Packages/ Install package(s)* from *Bioconductor*.

Инсталляция пакетов производится один раз, но перед началом каждой рабочей сессии пакеты должны быть **загружены** командой library(“имя библиотеки”) или через меню *Packages/Load package*.

2. Подготовка среды

Установка директории, содержащей файлы в качестве рабочей (команда setwd("..") или в меню *File/Change Dir).*

Загрузка библиотек, необходимых для работы.

```
R>library(Biobase) ## содержит неспецифические
компоненты, требуется для работы других пакетов
Bioconductor
R>library(affy) ## обеспечивает считывание и
нормализацию данных
R>library(multtest)## функции для множественного
тестирования
R>library(annotate)## аннотация данных
R>library(hgu95av2)## описание платформы,
использованной экспериментатором
```

3. Загрузка данных

Для загрузки данных в R используется команда:

```
R> Data <- ReadAffy() ##считывает все данные в рабочей
директории
```

Чтобы выбрать часть файлов и избежать в дальнейшем отображения названий образцов как путей к файлам, целесообразно воспользоваться следующей последовательностью действий:

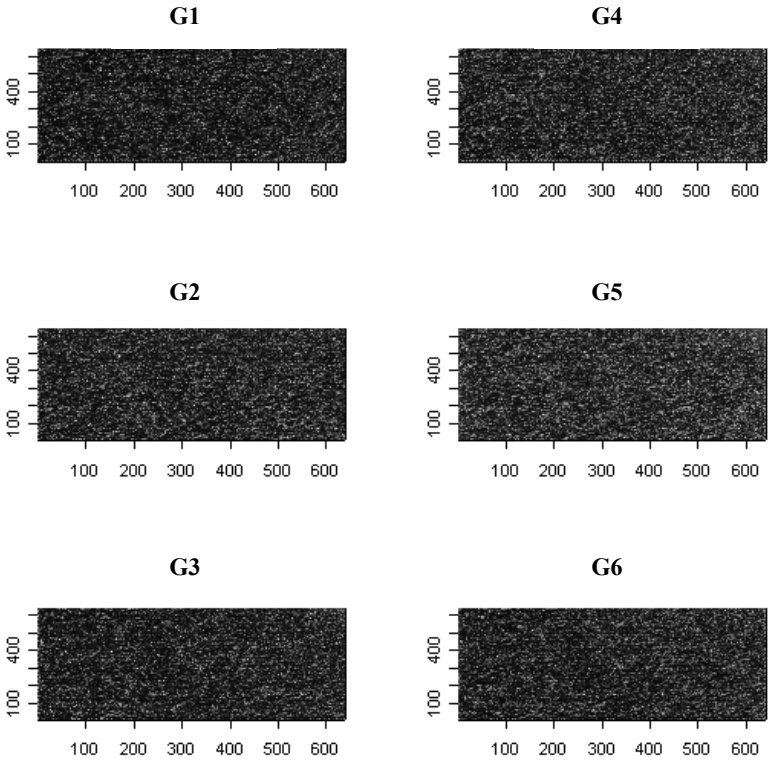
```
R> sample.files<-c("G1.cel","G2.cel","G3.cel",
"G4.cel","G5.cel","G6.cel","O1.cel","O2.cel",
"O3.cel","O4.cel","O5.cel","O6.cel")## первая буква
соответствует типу ткани
R>sample.names<-c("G1","G2","G3","G4","G5","G6","O1","O2",
"O3","O4","O5","O6")
R> data<-ReadAffy(filenames=sample.files)
```

Использование параметра widget (`ReadAffy(widget=TRUE))` позволяет воспользоваться окном браузера для выбора файлов и задания соответствующего описания.

4. Построение диагностических графиков, демонстрирующих необходимость стандартизации экспериментальных данных

Пространственное изображение

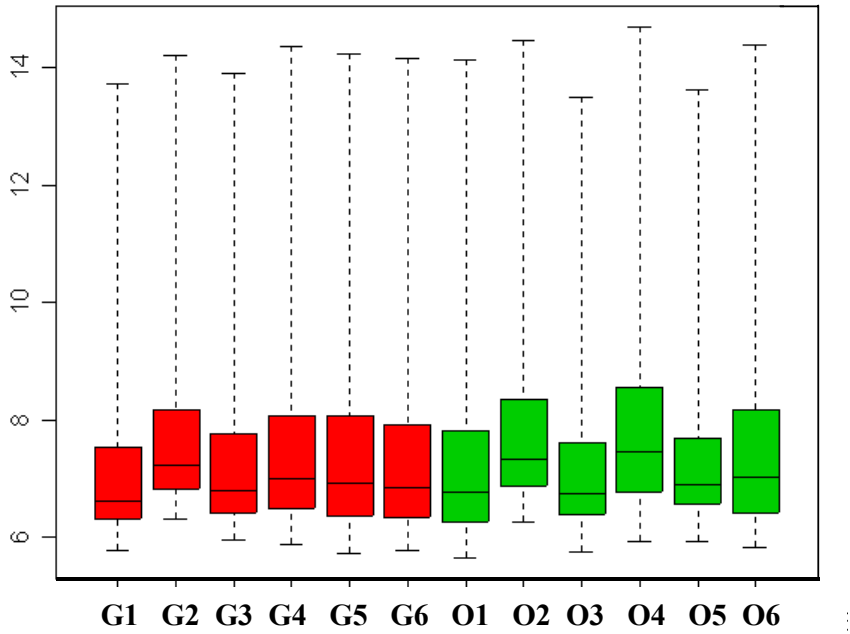
```
R>par(mfcol=c(3,2))#установка параметров графического
окна, (по умолчанию в новом окне будут рисоваться изо-
бражения для каждого файла по очереди)
R>image(data)
```



Данное изображение позволяет визуально оценить качество каждого отдельного чипа. Исследуется наличие царапин, пузырей, яркости и равномерности цвета чипа. Неравномерность цвета говорит о проблемах во время отмывки чипа от негибридизовавшейся ДНК. Как видно из этих изображений: яркость чипа равномерна, нет белых «вспышек», качество хорошее.

Диаграмма размаха

```
R> par(mfcol=c(1,1))
R> boxplot(vih,col=c(2,2,2,2,2,2,3,3,3,3,3,3,3) # col
— указывает цвета
```

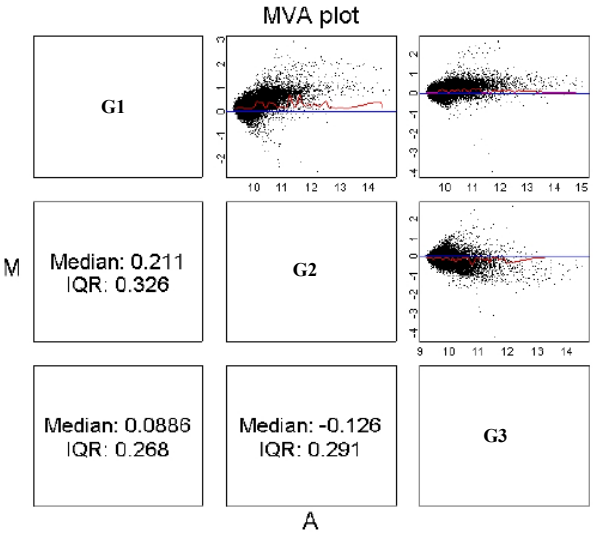


В данной диаграмме сравниваются значения медиан для всех чипов, объем данных для каждого чипа, попадающий в интерквартильное расстояние (50 % данных) и разброс самых крайних точек. На данной диаграмме для шести чипов видно, что значение медианы неодинаково, что говорит о необходимости нормализации данных.

MVA диаграмма

Строится для 1000 генов. Парные сравнения:

```
R> gn<-sample(geneNames(human),1000)
R> pms<-pm(data[,1:3],gn)
R> mva.pairs(pms)
```



Аналогично для остальных сравнений data[,4,6], data[,7,9].

MA диаграмма является наиболее информативной после диаграммы размаха; здесь M — это разница логарифмов экспрессии, а A — это среднее значение логарифма экспрессии. Для любых двух чипов i, j с интенсивностями проб x_{ki} и x_{kj}, где $k = \overline{1, p}$ представляет пробу, вычисляются

$$M_k = \log_2\left(x_{ki} \Big/ x_{kj}\right) \quad A_k = \frac{1}{2}\log_2\left(x_{ki} x_{kj}\right).$$

Диаграмма строится для значений интенсивности PM для всех возможных пар чипов. Для тех случаев, где мы ожидаем небольшое число генов с дифференциальной экспрессией, облако значений должно располагаться вдоль оси $M = 0$, и среднее значение, таким образом, должно быть небольшим. Однако вследствие нежелательных вариаций, связанных с ограничениями технологии, облако значений, как правило, имеет вид, близкий к характерному виду «запятой».

5. Стандартизация данных

Преобразование величин интенсивности проб в уровни экспрессии генов осуществляется следующей последовательностью действий:

- коррекция по фону;
- нормализация;
- коррекция, специфическая для пробы;
- вычисление значения экспрессии.

Можно выбрать один метод, осуществляющий все эти действия, например, метод RMA.

```
R> eset<-rma(data)
```

Также возможно комбинировать разные методы при помощи функции `expresso()`.

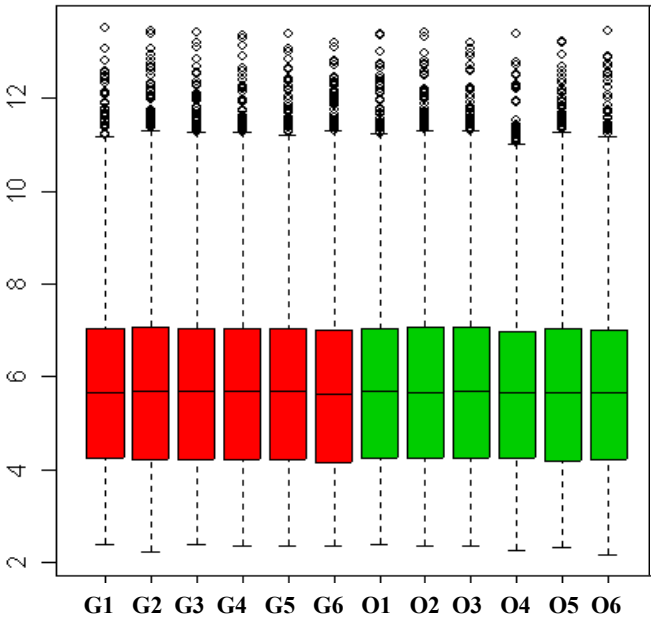
Запись результатов стандартизации в файл, формат .txt:

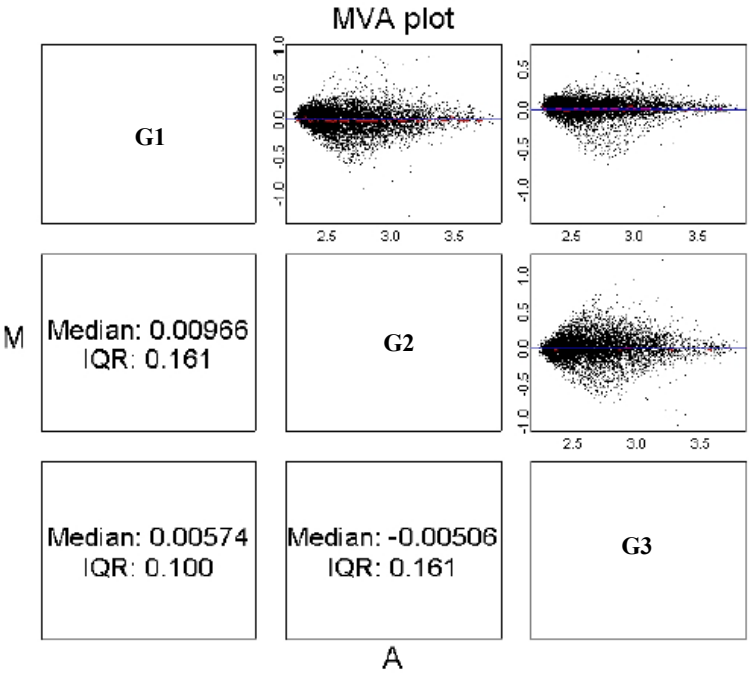
```
R>write.exprs(eset, file="mydata.txt")
```

6. Построение диагностических графиков, показывающих эффективность проведенной стандартизации данных

Аналогично пункту 4.

```
R>boxplot(eset,col=c(2,2,2,2,2,2,3,3,3,3,3,3))
```





7. Аннотация

Для облегчения восприятия результата создадим таблицу с описанием, включающую имена соответствующих генов, описание и, возможно, другую информацию из баз данных.

Включение столбца Probe в таблицу

```
R> col<-aaf.handler()[[1]]
```

Создание таблицы:

```
R> table<-aafTableAnn(affyID[1:12625],"hgu95av2”,col)
```

Описание генов:

```
R> descr<-multiget(affyID[1:12625],
env=hgu95av2GENENAME)
```

Создаем второй столбец с описанием генов:

```
R> col2<-aafTable(«Description»=descr)
```

Сливаем этот столбец с предыдущей таблицей:

```
R> table2<-merge(table, col2)
```

8. Определение генов, изменивших уровень своей экспрессии при сравнении двух различных экспериментов

Маркируем принадлежность арреев к одной из 2 групп:

```
R>data.c<-c(0,0,0,0,0,0,1,1,1,1,1,1)
```

Вычисляем t-статистику:

```
R>teststat<-mt.teststat(eset,data.c)
```

Вычисляем нескорректированное значение p:

```
R> rawp0<-2*(1-pnorm(abs(teststat)))
```

Коррекция значения P с использованием методов Бонферони, Холма, Шидака и т. д.

```
R>procs<-c("Bonferroni","Holm","Hochberg","SidakSS",
"SidakSD", "BH","BY")# методы коррекции
R>res<-mt.rawp2adjp(rawp0,procs)
R>adjp<-res$adjp[order(res$index),]#матрица с
количеством генов, изменивших экспрессию для
скорректированных значений p
R> mt.reject(adjp,seq(0,1,0.1))$r   # выводим матрицу
на экран
```

	rawp	Bonferroni	Holm	Hochberg	SidakSS	SidakSD	BH	BY
0	1	1	1	1	1	1	1	1
0.1	2771	97	97	97	98	98	766	220
0.2	4068	116	118	118	123	123	1291	305
0.3	5232	135	135	135	140	141	1781	359
0.4	6300	143	143	143	148	149	2416	425
0.5	7430	148	148	148	160	161	3107	474
0.6	8463	156	156	156	175	176	3903	546
0.7	9514	161	163	163	184	186	4908	602
0.8	10593	167	168	168	197	198	6265	658
0.9	11632	174	175	175	222	222	8553	713
1	12625	12625	12625	12625	12625	12625	12625	12625

Первый столбец — уровень значимости, второй — нескорректированные значения *p*, все остальные — значения *p*, скорректированные различными методами.

```
R>which<-mt.reject(adjp,0.01)$which[,2]# отбор с уров-
нем значимости 0.01 и по методу Bonferroni
R>  results<-table2[which,2]  #  отбираем  гены  по  2
столбцу таблицы с аннотацией
R> saveText(results,"sorted_bonf.txt")
```

Альтернативный вариант коррекции — коррекция методом перестановок:

```
R>resT<-mt.maxT(eset,data.c,B=1000)
R>ord<-order(resT$index)
R>rawp<-resT$rawp[ord]
R>maxT<-resT$adjp[ord]
R>teststat<-resT$teststat[ord]
R>mt.reject(cbind(rawp,maxT),seq(0,1,0.1))$r
```

	rawp	maxT
0	0	0
0.1	2462	6
0.2	3731	14
0.3	4909	19
0.4	6009	27
0.5	7111	35
0.6	8224	40
0.7	9309	52
0.8	10430	70
0.9	11563	91
1	12625	12625

```
R>test.maxT<-
mt.reject(cbind(rawp,maxT),seq(0,1,0.1))$r
R>write.table(test.maxT,file="mas_maxT.txt")
# записываем результаты в таблицу
R> which<-mt.reject(cbind(rawp,maxT),0.1)$which[,2]
R>results1<-table2[which,2]
R> saveText(results,"sorted_maxT.txt")
```

В результате с помощью метода Бонферони мы получили список из 57 генов, с достоверностью 99% изменивших свою экспрессию.

a disintegrin and metalloproteinase domain 22
A kinase (PRKA) anchor protein 1
adaptor-related protein complex 3, beta 2 subunit
adenylate kinase 2
ankyrin 3, node of Ranvier (ankyrin G)
apoptosis inhibitor 5
ATPase, aminophospholipid transporter (APLT), Class I, type 8A, member 1
ATP-binding cassette, sub-family C (CFTR/MRP), member 8
B/K protein
bassoon (presynaptic cytomatrix protein)
brain-specific protein p25 alpha
breakpoint cluster region
casein kinase 1, delta
centromere protein E, 312kDa
choline kinase-like
chromogranin A (parathyroid secretory protein 1)
chromogranin B (secretogranin 1)
creatine kinase, mitochondrial 1 (ubiquitous)
down-regulator of transcription 1, TBP-binding (negative cofactor 2)
Fas apoptotic inhibitory molecule 2
FK506 binding protein 12-rapamycin associated protein 1
G protein-coupled receptor 19
guanine nucleotide binding protein (G protein), gamma 5
H326
hypothetical protein DKFZp761N09121
hypothetical protein LOC157627
integrin beta 3 binding protein (beta3-endonexin)
internexin neuronal intermediate filament protein, alpha
KDEL (Lys-Asp-Glu-Leu) endoplasmic reticulum protein retention receptor 1
Kruppel-like factor 4 (gut)
LIM domain only 6
microtubule-associated protein 1A
MUF1 protein

p21 (CDKN1A)-activated kinase 3
peptidase D
phosphodiesterase 8B
potassium voltage-gated channel, KQT-like subfamily, member 2
protein kinase C, beta 1
protein phosphatase 1, regulatory (inhibitor) subunit 12B
protein tyrosine phosphatase, receptor type, N
protein tyrosine phosphatase, receptor type, N
protein tyrosine phosphatase, receptor type, N polypeptide 2
pyruvate dehydrogenase complex, component X
RaP2 interacting protein 8
ras homolog gene family, member C
regulating synaptic membrane exocytosis 2
ribosomal protein S12
SH3-domain GRB2-like 3
solute carrier family 31 (copper transporters), member 1
sorting nexin 7
synaptosomal-associated protein, 25kDa
synaptosomal-associated protein, 91kDa homolog (mouse)
transcription elongation factor A (SII)-like 1
transient receptor potential cation channel, subfamily C, member 1
ubiquitin-conjugating enzyme variant Kua
uroporphyrinogen III synthase (congenital erythropoietic porphyria)
uroporphyrinogen III synthase (congenital erythropoietic porphyria)

ЗАКЛЮЧЕНИЕ

В данной статье был рассмотрен подход к решению задачи выделения генов, изменивших с определенной достоверностью уровень своей экспрессии на основе данных из нескольких микрочиповых экспериментов.

Описан процесс обработки данных с олигонуклеотидного чипа компании Affimetrix. В эксперименте изучалась экспрессия генов в клетках раковых опухолей мозга глиобластомы и олигобластомы.

Нормализация и кластеризация данных выполнялись в среде программирования языка *R GUI* с использованием пакетов *Bioconductor*. В резуль-

тате был получен список из 57 генов, изменивших свою экспрессию с достоверностью 99%.

СПИСОК ЛИТЕРАТУРЫ

1. **Irizarry R.A., Hobbs B., Collin F., et al.** Exploration, normalization, and summaries of high density oligonucleotide array probe level data // Biostatistics. — 2003. — Vol. 4, No. 2. — P. 249–264.
2. **Bolstad B.M., Irizarry R.A., Astrand M., Speed T.P.** A comparison of normalization methods for high density oligonucleotide array data based on variance and bias // Bioinformatics. — 2003. — Vol. 2, N. 19. — P.185–193.

Ю. В. Малинина*

СЕМАНТИЧЕСКАЯ СЕТЬ КАК ФОРМАЛЬНЫЙ МЕТОД ОПИСАНИЯ И ОБРАБОТКИ ТЕКСТОВ ПО ПРЕОБРАЗОВАНИЯМ ПРОГРАММ

ВВЕДЕНИЕ

В конце 50-х гг. в области программирования была поставлена задача повышения эффективности программ с помощью выполнения над ними в процессе трансляции ряда преобразований. Эту задачу принято называть задачей оптимизации программ, а преобразования, которые выполняются над программами и могут повысить их эффективность, — оптимизирующими преобразованиями. К настоящему времени теория оптимизации программ обогатилась фундаментальными результатами (развит аппарат схем программ, выделен богатый набор оптимизирующих преобразований, найдены достаточные условия корректности применения ряда преобразований и т.д.), а практика — примерами эффективно работающих оптимизирующих компиляторов. Исследования в этой области расширяются. А прогресс информационных технологий привел к тому, что общение исследователей стало глобальным, более интенсивным, а информационная насыщенность научных сообщений значительно возросла. Мировая система научной коммуникации предназначена для объединения целенаправленной деятельности большого числа ученых, проживающих в разных странах, говорящих на разных языках и работающих в одной тематической области.

Однако, по-прежнему, публикации являются основным источником информации и непременной составляющей научного исследования, одновременно представляют собой и цель, и средство (или же важную составляющую часть последних). В каком-то смысле исследование начинается с научного текста (изучение литературы, постановка задачи) и им же и заканчивается (опубликование результатов). Научный текст — это форма представления, формализации и обобщения научных знаний и одновременно — это способ аргументации и экспликации логического вывода и правдоподобных рассуждений.

* malin@iis.nsk.su

Работа выполнена при финансовой поддержке Министерства образования РФ (научная программа «Университеты России», грант № УР.04.01.201).

Сегодня многие ученые осознают, что развитие любого научного направления во многом зависит от привлечения в эту тематическую область новых научных сил. Для этого молодые люди, начинающие свой путь в науке, должны получить максимально возможное представление о состоянии тематики и о задачах, которые предстоит решать. Известно, что наиболее продуктивными являются те ученые, которые осознанно сделали свой выбор в направлении научных исследований.

В силу существования большого количества языков программирования, теоретические результаты в области оптимизации программ обычно формулируются в терминах абстрактных математических моделей программ, независимых от конкретных алгоритмических языков, причем число таких моделей исчисляется десятками [2, 5].

Это приводит к тому, что оптимизирующие преобразования описываются в терминах различных моделей, что затрудняет изучение различных оптимизирующих преобразований. Для решения этой проблемы наиболее ценными являются попытки собрать наиболее полные библиографии документов, относящихся к тематической области преобразования программ, на протяжении всей истории развития данного научного направления. Многие ученые понимают важность такой информационной работы, позволяющей сохранить для следующих поколений исследователей результаты их научной работы.

На сегодня текстовый поиск — это простейший способ доступа к текстовым данным, предназначенный для подбора материалов по выбранной теме. «Классическая» поисковая машина умеет найти по запросу из нескольких слов все документы, в которые данные слова входят, и предъявить их пользователю, что, кстати, может сделать и читатель печатного учебника, сравнив по предметному указателю, на каких страницах одновременно встречаются нужные ему термины.

Этой простой возможности при росте объемов текстовых баз становится совершенно недостаточно, и возникает задача предварительной обработки наборов текстовых документов с целью предоставления информации о них в более компактной и информационно насыщенной форме, одновременно сохраняя ссылки на исходные источники.

В середине 20-го века для упрощения поиска материала по определенной тематике использовались периодические реферативные издания, названия и тематическая направленность рубрик которых изменялась в соответствии с тенденциями развития отдельных областей науки. Причем в них впервые реферировались не только отдельные издания, но и публикации в периодических и продолжающихся изданиях. Для поиска документов в ре-

феративных журналах создавался справочный аппарат: авторский и тематический указатели в каждом номере и кумулятивный — за весь год.

Для электронных публикаций существует аналог — уникальный реферативный журнал, созданный институтом информации Гарфильда "Science Citation Index", ориентированный на поиск новых научных публикаций в мировой системе периодических и продолжающихся изданий по системе научных ссылок. Появление этого издания использует естественную исторически сложившуюся систему классификации научных работ по ссылкам автора на работы его предшественников в определенной тематической области. В этом издании отсутствует разбиение статей на тематические рубрики, авторы указателя предлагают пользователям подготовленные ими тематические кластеры связанных между собой публикаций по системе цитирования общих предшественников. Как размеры, так и наименования кластеров постоянно корректируются ими в соответствии с тенденциями в науке.

При сжатых сроках и при большом объеме публикующейся информации классификации научных работ по ссылкам автора на работы его предшественников в определенной тематической области зачастую недостаточно для оценки важности каждой публикации для ведущихся исследований и определения необходимости ее внимательного изучения. Для оценки требуется бóльшее количество параметров.

Поскольку даже беглый просмотр большого количества статей требует больших затрат времени, то для облегчения этой задачи обычно прибегают к различным технологиям автоматического свертывания документов

АВТОМАТИЗИРОВАННОЕ СВЕРТЫВАНИЕ

Рассмотрим одно из направлений интеллектуальной обработки научного текста — автоматизированное свертывание, к которому относятся работы по автоматическому реферированию, с точки зрения его применения для извлечения фрагментов определенного смысла.

Это направление занимает как бы промежуточное положение между минимальным уровнем свертывания — переводом и максимальным — индексированием. Фактически автоматическое реферирование по своему характеру очень специфично, поскольку сводится к экстрагированию (извлечению) из документов минимальных релевантных фрагментов, некоторая совокупность которых и образует широкий спектр вторичных документов — различные виды аннотаций, рефератов, реферативных аннотаций, самостоятельных фрагментов, конспектов и их синтезированных производ-

ных — реферативных указателей, дайджестов, реферативных обзоров, квазихрестоматий и т.д. Эти вторичные документы, являющиеся результатом аналитико-синтетической переработки первичного документного потока, рассчитаны на удовлетворение как частных (индивидуальных), так и типовых (потенциальных) информационных потребностей различных категорий специалистов науки, техники и производства [1].

Главное различие между средствами аннотирования состоит в том, что они, по существу, формируют краткое изложение или набор выдержек. Дополнительно рефераты различаются по функции и целевым группам пользователей [1]. Так, например, реферат может быть повествовательным, информативным или критическим.

Повествовательные рефераты формируются по классическому принципу извлечения информации: они предоставляют достаточный объем информации, чтобы создать у пользователя представление о соответствующих источниках с тем, чтобы их можно было отобрать для более внимательного прочтения.

Информативные рефераты заменяют собой текст, в основном они содержат основную или новую фактическую информацию в сокращенной форме.

Критические рефераты (или обзоры) сообщают не только суть информации, но и предлагают определенное мнение о ней. Критические рефераты обладают дополнительной ценностью по сравнению с оригиналом, поскольку предлагают выводы, которых нет в самом тексте.

Уже сегодня существуют действующие системы автоматического аннотирования текстовых документов. Прежде всего нужно сказать, что фактически во всех известных системах машинное аннотирование является экстрагированием — программа не «пересказывает» смысл текста, а просто извлекает из него те фрагменты, которые считает важными, и объединяет их в аннотацию [10]. Важность конкретного предложения определяется по различным параметрам, в частности, по так называемым маркерам важности (например, «в заключение нужно сказать, что...»), количеству содержательных слов в нем и т. д.

В наиболее развитых средствах аннотирования учитывается также зависимость предложений друг от друга с тем, чтобы не вносить в аннотацию обрывки, начинающиеся, например, со слов: «К тому же...», «В-третьих...» и т. п. [1].

Другой способ свертывания текста — это представление его в форме семантической сети, которая инвариантна к форме описания фактов с точностью до выбранной автором структуры рассуждения.

Понятие «семантической сети» известно в математике начиная с конца 50-х гг. Это помеченный граф, вершины которого используются для представления объектов (предметов, событий, состояний), а дуги — для представления связей (отношений между объектами). Такая структура хорошо изучена с точки зрения математики и служит удобным средством представления знаний для дальнейшего анализа.

Сжатие информации при переходе от лексического к семантическому описанию документов приводит к ее *обобщению*, что эквивалентно получению некоторого *знания*. Ведь возможность более сжатого описания данных есть следствие скрытых в этих данных закономерностей. Сжатие информации как раз и сводится к выявлению этих закономерностей, выражающих наши знания о структуре данных.

Семантическая сеть — это множество понятий (слов и словосочетаний), связанных между собой. В семантическую сеть включаются наиболее часто встречающиеся слова текста, которые несут основную смысловую нагрузку. Для каждого понятия формируется набор ассоциативных (смысловых) связей, т.е. список других понятий, в сочетании с которыми оно встречалось в предложениях текста. При этом считается, что чем чаще встречаются вместе два понятия в предложениях текста, тем выше вероятность того, что они связаны по смыслу.

Статистическая информация об отдельных лексических единицах легко извлекается из текста, и есть все основания полагать, что она адекватно отражает его содержание в целом. Косвенное подтверждение этому можно найти в нейропсихологических исследованиях, которые установили, что анализ печатного текста, опираясь на зрительное пространственное (а не на линейное слуховое) восприятие, реализуется преимущественно правым полушарием мозга, использующим ассоциативную статистическую модель [3, 6]. Логический «левополушарный» анализ, моделированием которого, по сути, занимается формальная лингвистика, необходим лишь в отдельных «трудных» местах текста, несущих новую информацию и требующих детального осмысления.

АЛГОРИТМ РЕАЛИЗАЦИИ

Общая схема обработки текстов инвариантна по отношению к выбору естественного языка. Независимо от того, на каком языке написан исходный текст, его анализ проходит одни и те же стадии. Первые две стадии (разбиение текста на отдельные предложения и на слова) практически оди-

наковы для большинства естественных языков. Единственное, где могут проявиться специфичные для выбранного языка черты, — это обработка сокращений слов и знаков препинания (точнее, определение того, какие из знаков препинания являются концом предложения, а какие — нет). Последующие стадии: морфологический анализ; морфемный анализ; синтаксический анализ, напротив, сильно зависят от выбранного естественного языка. Последняя стадия (семантический анализ) также мало зависит от выбранного языка, но это проявляется только в общих подходах к проведению анализа [9].

В общем виде семантическую сеть можно представить как взвешенный граф, в вершинах которого находятся знаки, а ребра с весами отражают связи знаков между собой. Рассмотрим ассоциативную семантическую сеть. Для определения понятия близости двух знаков в таких сетях требуется для каждого из двух изучаемых знаков составить список часто встречающихся рядом с ним знаков. Таким образом, для каждого из двух знаков мы получим список «ассоциированных» с ним знаков. Сравнивая эти списки, можно сделать количественные выводы о том, насколько близки исходные два знака.

Текст будем рассматривать как множество соответствующих основ слов, входящих в него. Две основы слова объявим близкими, если они встречаются в тексте на расстоянии k слов. Знаки препинания и другие специальные символы и стоп-слова, встречающиеся в тексте, игнорируем. Степень близости двух слов — это число случаев, когда два слова, соответствующих данному знаку, встретились рядом. Последовательность обработки текста будет следующая.

1. Построить упорядоченный по очередности вхождения списка слов текст, с учетом пропуска так называемых стоп-слов. Это наиболее употребительные в данном языке слова, удаление которых не повлияет на качество идентификации (более того, может его улучшить). Например, в английском языке имеются стандартные доступные в Internet списки стоп-слов (несколько сотен слов), включающие в свой состав артикли, союзы, модальные и вспомогательные глаголы, числительные, местоимения.
2. Выделить основы слов (stemming), с помощью подходов, описанных в [2]. Это позволяет все однокоренные слова заменить корнем и не различать, например, слова task и tasks.
3. Исходя из того, как часто слова появляются рядом, для каждого слова составить списки близких к нему слов.

4. Сравнивая списки любых двух знаков, определить меру близости этих знаков.
5. По полученной информации построить взвешенный граф со знаками в качестве вершин и взвешенными ребрами в качестве связей между знаками.

Таким образом, для фразы «Неопределенность зависимостей между индексированными переменными» получаются следующие знаки: (определ, зависим, индекс, перемен). При k = 1 со знаком «зависим» будут рядом знаки (определ, индекс) с весами (1, 1) соответственно.

Описанный подход содержит упрощенный способ формирования семантической сети по тексту. Предполагается, что в дальнейшем он будет расширен вскрытием всех типов связей и взаимоотношений между понятиями. Основные виды таких отношений — это гипонимия (род—вид), соподчинение на одном уровне — парциация (часть—целое), ассоциация (локализация объекта, его назначения). В каждой системе знания любой объект в каком-то отношении нередко является и признаком, а почти каждый признак в другом отношении выступает как объект.

ЗАКЛЮЧЕНИЕ

Выше в краткой форме представлено состояние работы и сформулированы некоторые подходы к решению. Однако рамки задачи автоматизированного свертывания документного потока должны рассматриваться значительно шире. Речь идет о создании системы автоматической обработки потока публикаций с целью максимального раскрытия и использования его ресурсов для решения задач развития науки.

Не секрет, что в системе информационных коммуникаций наблюдается постоянное недоиспользование накопленных обществом знаний со всеми вытекающими отсюда негативными последствиями. Причина этого, прежде всего, в несовершенстве средств поиска информации (несмотря на широкое внедрение в эту сферу средств компьютерной техники) и методов аналитико-синтетической переработки первичного документального потока. Специалисту в действительности нужны не документы, а информация — факты, концепции и др. Однако информации очень много вообще, но крайне мало — в частности.

Такое положение обусловлено диалектическим противоречием между избыточностью конкретного документа за счет его многоаспектности и недостаточностью документного потока в целом за счет явления рассеяния.

Работы в области информационного анализа/синтеза и призваны, в известных рамках, снять это противоречие. Их конечная цель — максимальное использование когнитивных (познавательных) возможностей первичного документа за счет машинного «разбиения» его на самостоятельные минимальные релевантные фрагменты, утилизируемые затем в гипотетической пока еще базе знаний, обращение к которой позволило бы в значительной степени снизить необходимость использования первичного потока.

Предложенная работа является лишь предварительным этапом в реализации идеи компьютерного свертывания, которая должна постепенно трансформироваться в серию работ, направленных на получение репрезентативных результатов.

СПИСОК ЛИТЕРАТУРЫ

1. **Блюменау Д.И.** О некоторых направлениях формализации инфопроцессов // Проблемы инфовзаимодействия. — Новосибирск, 1993. — С. 206–223.
2. **Евстигнеев В. А. , Мирзуитова И. Л.** Анализ циклов: выбор кандидатов на распараллеливание. — Новосибирск, 1999. — (Препр. / ИСИ СО РАН; № 58).
3. **Евстигнеев В.А.** О некоторых формах промежуточного представления программ // Оптимизация и конструирование программ. — Новосибирск, 1993. — С. 52–59.
4. **Ермаков А.Е., Плешко В.В.** Семантическая сеть текста в задачах аналитика // Информатизация и информационная безопасность правоохранительных органов: Тр. XI Междунар. научной конф. — Москва, 2002. — С. 343–347.
5. **Ермолаев Д.С** Компьютерный морфологический разбор слов русского языка. — 2001 — `http://www.codenet.ru/progr/alg/morf.php`
6. **Иванко Е., Перевалов Д.** Использование ассоциативных семантических сетей для классификации звукозаписей // Компьютерная лингвистика и интеллектуальные технологии: Тр. Междунар. конф. «Диалог'2004». — М.: Наука, 2004.
7. **Касьянов В.Н**. Оптимизирующие преобразования программ. — М.: Наука, 1988.
8. **Киселев С.Л., Ермаков А.Е., Плешко В.В.** Поиск фактов в тексте естественного языка на основе сетевых описаний // Компьютерная лингвистика и интеллектуальные технологии: Тр. Междунар. конф. «Диалог'2004». — М.: Наука, 2004.
9. **Коваленко А.** «СТЕММЕР» морфологический анализ для небольших поисковых систем. —2002. — `http://samag.ru/img/uploaded/samag14649-0.pdf`
10. **Селезнев К**. Обработка текстов на естественном языке // Открытые системы. — 2003. — № 12.
11. **Хан У., Мани И**. Системы автоматического реферирования // Открытые системы. — 2003. — № 12.

Л. С. Мельников*, И. В. Петренко

ПУТЕВЫЕ ЯДРА И РАЗБИЕНИЯ В ГРАФАХ С МАЛЫМИ ДЛИНАМИ ЦИКЛОВ

1. ВВЕДЕНИЕ

Пусть $G = (V, E)$ — конечный граф. Количество вершин в самом длинном его пути будем обозначать через $\tau(G)$. Через $g(C)$ и $c(G)$ (*girth и circumference*) будем обозначать длину кратчайшего и длиннейшего циклов в G соответственно. Через C_n и P_n будем обозначать цикл порядка n и путь порядка n соответственно, где порядок это число вершин. Вершину $v \in V$ будем называть *P_n-терминальной вершиной* графа G, если она является конечной вершиной P_n, но при этом не является конечной для P_{n+1} в G.

Пусть S — некоторое подмножество множества вершин $V(G)$. Подграф графа G, порожденный множеством S, будем обозначать $G[S]$. *Открытой окрестностью вершины* $v \in V(G)$ назовем множество вершин вида $N(v) = \{u \in V(G) | (u, v) \in E(G)\}$. *Открытой окрестностью* подмножества A множества вершин $V(G)$ назовем множество вида $N(A) = \bigcup_{a \in A} N(a)$, а *замкнутой окрестностью* того же множества назовем множество вида $N[A] = N(A) \cup A$.

Для некоторой пары натуральных чисел (a, b) разбиение множества вершин $V(G)$ на два непересекающихся подмножества $\{A, B\}$ называется (a, b)-*разбиением*, если $\tau(G[A]) \leq a$, а $\tau(G[B]) \leq b$, а граф, для которого имеется такое (a, b)-разбиение называется (a, b)-*разбиваемым*. Если G является (a, b)-разбиваемым для любых a и b таких, что имеет место равенство $a + b = \tau(G)$, то такой граф будем называть τ-*разбиваемым*.

Похожее понятие о разбиениях было введено раньше для других характеристик теории графов. Граф G называется $\Delta(G)$-*разбиваемым* (где $\Delta(G)$ обозначает максимальную степень графа G), если для любой пары натуральных чисел (a, b) такой, что $a + b \geq \Delta(G) - 1$, существует разбиение множества вершин $V(G)$ на два непересекающихся подмножества $\{A, B\}$ такое, что $\Delta(G[A]) \leq a$ и $\Delta(G[B]) \leq b$. Л. Ловас в [15]

*omeln@math.nsc.ru

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (гранты № 05-01-00395 и 05-01-00816).

доказал, что каждый граф G является $\Delta(G)$-разбиваемым. М. Штибиц в [17] получил подобный результат, связанный с минимальной степенью $\delta(G)$. С.Капур, Х. Кронк и Д. Лик [13] также получили аналогичные результаты.

Является ли произвольный связный граф τ-разбиваемым? Этот вопрос впервые был поднят в [1], хотя похожие попытки были сделаны К. Томассеном [18]. Имеются результаты, подтверждающие справедливость этой гипотезы для некоторых классов графов. Ответ на этот вопрос для общего случая пока неизвестен. Также эта гипотеза тесно связана с открытой гипотезой П. Михока [16] о ядрах и работой Хайнала [11] и нашла определенное отражение в диссертациях [11, 19]. Краткое перечисление рассматриваемых задач и их связи с поднятым вопросом о τ-разбиваемости представлено в [5]. Вариант этого вопроса для ориентированных графов сформулирован в [14]. Подобные постановки рассматривались в [9, 10, 13].

Возникновение этого вопроса связано в частности с так называемым *k-хроматическим числом* графа G, обозначаемым $\chi_k(G)$, которое является наименьшим количеством множеств $\{V_1, V_2, \ldots, V_n\}$ в разбиении множества $V(G)$ таким образом, что имеет место $\tau(G[V_i]) \leq k$ для каждого i. В частности, Г. Чартрэнд, Д. П. Геллер и С. Хедетниеми в 1968 г. в [6] получили следующую верхнюю оценку для k-хроматического числа произвольного графа: $\chi_k(G) \leq \lfloor(\tau(G) - 1 - k)/2\rfloor + 2$. Ясно, что положительный ответ на вопрос о τ-разбиваемости произвольного графа улучшит оценку до $\chi_k(G) \leq \lceil\tau(G)/k\rceil$.

Для произвольного графа G подмножество K множества $V(G)$ будем называть *P_n-ядром* G, если выполнены следующие условия:

1) $\tau(G[K]) \leq n - 1$;

2) каждая вершина $v \in V(G - K)$ смежна с P_{n-1}-терминальной вершиной $G[K]$.

Так, например, максимальное независимое множество вершин образует P_2-ядро, а вершины максимального подграфа, не содержащего пути P_3, образуют P_3–ядро.

Для произвольного графа G подмножество S множества $V(G)$ будем называть *P_n-полуядром* G, если выполнены следующие условия:

1) $\tau(G[S]) \leq n - 1$;

2) каждая вершина $v \in N(S) - S$ смежна с P_{n-1}-терминальной вершиной $G[S]$.

В [7] подробно разобраны вопросы существования путевых ядер и τ-разбиваемости и, в частности, доказано, что из справедливости гипо-

тезы о существовании в произвольном графе P_n-ядра для $n \geq 2$ следует справедливость гипотезы о τ-разбиваемости. Там же доказано, что, если любой граф G из некоторого наследственного класса имеет P_n-полуядро, то любой граф из этого же наследственного класса имеет P_n-ядро. Связи иерархических свойств и длиннейших простых путей см. в [2, 3, 4].

Взаимосвязь путевых ядер и вопросов τ-разбиваемости ясна из следующего утверждения, доказанного в [7, 8].

Утверждение 1.1. *Пусть G — граф, $\tau(G) = a + b$, причем $1 \leq a \leq b$. Если в G существует P_{b+1}-полуядро, то граф G является (a, b)-разбиваемым.*

Наконец, в [7] была сформулирована и доказана теорема о том, что всякий граф G имеет P_7-ядро.

В первой [20] из двух наших статей доказана теорема о существовании P_8-ядра в любом графе G. Во второй нашей статье [21] доказана теорема о существовании P_n-ядра в графе G со свойством $c(G) = n - 2$.

В настоящей работе будет сделан следующий шаг, а именно, доказана теорема о существовании P_9-ядра в произвольном связном неориентированном графе G, доказательство которой будет приведено с некоторыми сокращениями, обусловленными переборным характером доказательства и, как следствие, его большим объемом.

2. ДЛИНЫ ЦИКЛОВ И ПУТЕВЫЕ ЯДРА

Взаимосвязь длин кратчайшего и длиннейшего циклов в графе была замечена и изучена в [7, 8], где были доказаны следующие теоремы.

Утверждение 2.1. *Пусть C — $(n-1)$-цикл в графе G, тогда C является P_n-полуядром графа G.*

Утверждение 2.2. *Если G — граф, причем $g(G) \geq n - 2$, то в G существует P_n-ядро.*

Авторам удалось несколько улучшить первый из цитируемых результатов. Это следующее утверждение было опубликовано в [21] и будет в дальнейшем использоваться как вспомогательное.

Утверждение 2.3. *Пусть G — связный граф, такой что длина наибольшего цикла $c(G) = n$, тогда в графе G существует P_{n+2}-ядро.*

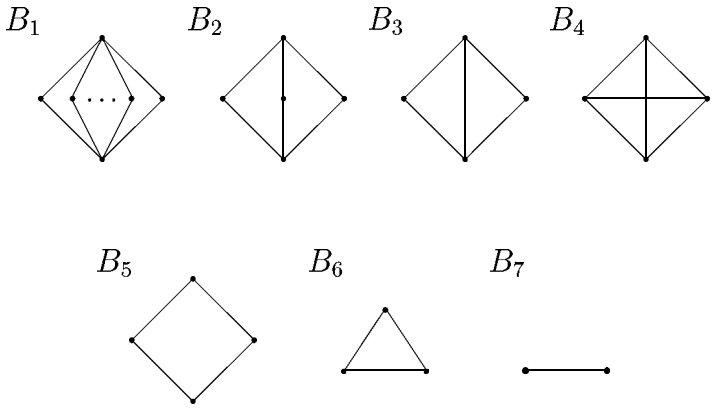


Рис. 1. Виды блоков в G при $c(G) \leq 4$

Продолжая исследование взаимосвязи длин кратчайшего и длиннейшего циклов и существования путевых ядер, авторы доказали следующее утверждение.

Теорема 2.4. *Пусть G — такой граф, что длина наибольшего цикла $c(G)$ не превышает 4, тогда в графе G существует P_n-ядро для любых $n \geq 2$.*

Доказательство.

Пусть $n \geq 2$. Доказательство производится индукцией по $|V(G)|$. Если $|V(G)| \leq n$, то $V(G)$ является P_n-ядром графа G. Предположим, $|V(G)| \geq n$. Поскольку имеет место $c(G) \leq 4$, то граф G состоит из блоков, которые представлены на рис. 1. Если в графе G имеется в точности один блок, то в нем, очевидно, имеется P_n-ядро, тем самым базис индукции доказан.

Предположим, граф G имеет более одного блока, пусть B — концевой блок графа G и v — некоторая вершина из блока B. Не ограничивая общности, можем предположить, что v не является точкой сочленения графа G.

По индукционному предположению, в графе $G \setminus v$ имеется P_n-ядро K. Если v смежна с (P_{n-1})-терминальной вершиной из K и v является

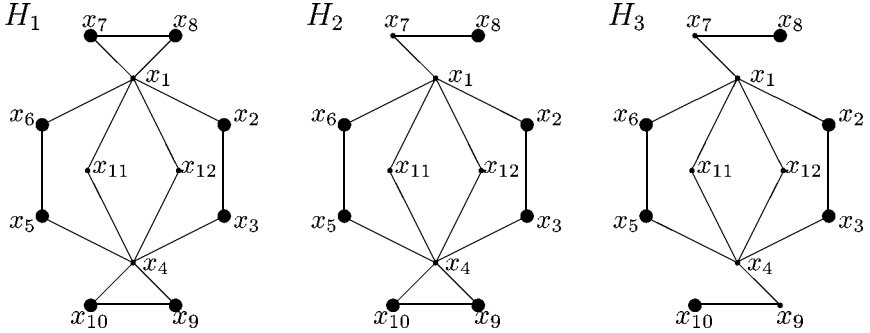


Рис. 2. Случай 1

P_n-терминальной вершиной в G, тогда множество вершин K является P_n-ядром графа G.

Предположим, v не является P_n-терминальной вершиной в G. В связи с тем, что B — концевой блок графа G, через который проходит путь P_n, на котором лежит вершина v, то в силу того, что $B = B_i$ для некоторого i, такого что $1 \leq i \leq 7$, и тогда множество вершин $K \cup \{v\}$ является P_n-ядром графа G. Теорема доказана.

Данная теорема носит вспомогательный характер и используется для доказательства теоремы о существовании P_9-ядра в произвольном графе G.

3. ТЕОРЕМА О СУЩЕСТВОВАНИИ P_9-ЯДРА

В данном разделе доказывается теорема о существовании P_9-ядра в некотором графе G. Доказательство теоремы приводится с сокращениями, фактически рассмотрен только случай, когда длина наибольшего цикла в графе G равна 6. Сокращения вызваны, в первую очередь, огромным объемом доказательства, которое носит переборный характер. С другой стороны, авторы полагают, что теорему, доказанную в предыдущем разделе, можно доказать и для случая, если длина наибольшего цикла в графе G не превышает 5. Исходя из этих соображений, а также из формата публикации, авторы позволили себе опустить один из случаев (случай 2), рассматриваемых в доказательстве нижеследующей теоремы.

Теорема 3.1. *В любом графе G имеется P_9-ядро.*

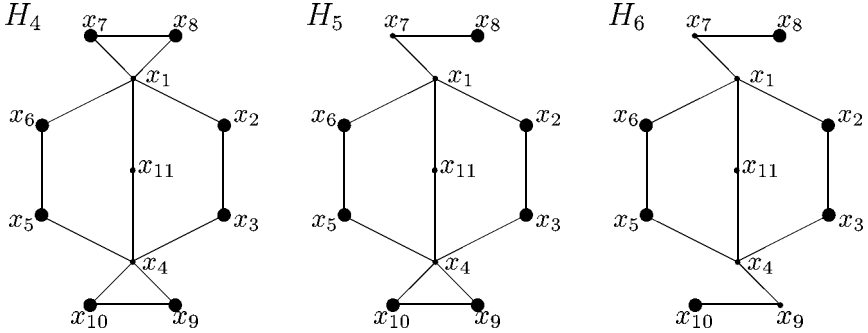


Рис. 3. Случай 2

Доказательство.

Если $\tau(G) \leq 8$, то $K = V(G)$ — P_9-ядро в графе G. Пусть G — такой граф, что $\tau(G) \geq 9$. Ввиду утверждения из [7], приведенного выше, достаточно показать, что в графе G имеется P_9-полуядро, при этом можно считать, что $g(G) \leq 7$ и что в G нет C_7. Рассмотрим следующие случаи.

1. $c(G) = 6$, т. е. в G имеется C_6, но отсутствует C_7.
2. $c(G) = 5$, т. е. в G имеется C_5 и отсутствуют C_6, C_7.
3. $c(G) \leq 4$, т. е. в графе G имеется C_4 и отсутствуют C_5, C_6, C_7.

Отметим, что в случае 3 мы находимся в условиях доказанной в предыдущем разделе теоремы, поэтому нам остается разобрать только случаи 1 и 2.

Используем следующий алгоритм.

Пусть $S = H_s$, где s — наименьший номер, такой что H_s является подграфом графа G.

Для некоторых $A, B \subseteq V(G)$ в начале работы алгоритма предполагается, что $B = V(G) \setminus V(S)$ и $A = \emptyset$.

Шаг 1. Все вершины из множества B, смежные с P_8-терминальными вершинами из S, перемещаются в A. Если $N(S) \cap B = \emptyset$, то алгоритм прекращает работу. В противном случае выполняется шаг 2.

Шаг 2. Если две вершины $x, y \in S$ смежны с некоторой вершиной b из B, то вершина b перемещается в S и осуществляется переход к шагу 1. В противном случае выполняется шаг 3.

Шаг 3. Пусть P_7-терминальная вершина x из S смежна с вершиной b из B. Тогда вершина b перемещается в S и выполняется шаг 1. В противном случае выполняется шаг 4.

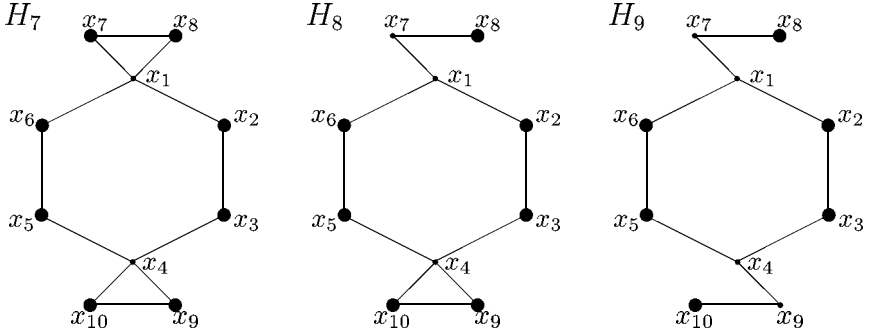


Рис. 4. Случай 3

Шаг 4. Пусть P_6-терминальная вершина x из S смежна с вершиной b из B. Тогда вершина b перемещается в S и выполняется шаг 1. В противном случае выполняется шаг 5.

Шаг 5. Пусть P_5-терминальная вершина x из S смежна с вершиной b из B. Тогда вершина b перемещается в S и выполняется шаг 1.

Конец.

Заметим, что в S нет P_n-терминальных вершин, таких что $n \leq 4$, и в процессе работы алгоритма они не появляются. Отметим, что на рис. 2–13 «жирные» вершины являются P_8-терминальными вершинами.

После того, как приведенный алгоритм завершит работу, т. е. когда множество $N(S) \cap B$ окажется пустым, множество S станет P_9-ядром графа G. Действительно, каждая вершина из A окажется смежной с P_8-терминальной вершиной из S. Поэтому нам остается доказать лишь только, что $\tau(S) \leq 8$, для чего, в свою очередь, достаточно показать, что в процессе работы алгоритма в S не возникнет путь длины 9.

Ясно, что путь длины 9 может возникнуть только в процессе перемещения вершин из B в S. Шаги 3, 4, 5 и 6 осуществляются только тогда, когда в B нет вершин, смежных более чем с одной вершиной из S. Поэтому при выполнении шагов 3, 4, 5 и 6 в S не может возникнуть путь длины 9. Следовательно, достаточно выяснить, что при выполнении шага 2 описанного алгоритма путь длины 9 в S не возникнет.

Рассмотрим случай 1. В графе G содержится один из подграфов, изображенных на рис. 2–13.

Заметим, что в случае, если $S = H_2, H_5, H_8$, вершина b из B не может быть смежна с вершинами x_1, x_7, так как это противоречит предполо-

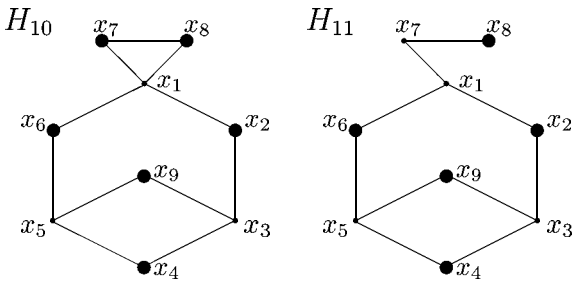


Рис. 5. Случай 4

жению о том, что s — минимально, а в случае, когда $S = H_3, H_6, H_9$ — с вершинами x_1, x_7 и вершинами x_4, x_8 по той же причине.

В случае, если $S = H_1, H_2, H_3$, то для вершины b из B возможны следующие варианты: если вершина b смежна с вершинами x_1, x_{11} или (симметричный вариант) x_4, x_{11}, тогда вершина b, будучи добавлена, станет P_8-терминальной, а вместе с ней P_8-терминальной станет и вершина X_{11}, при этом $\tau(S \cup b)$ не превысит 8. Аналогичные рассуждения проходят в случаях, если b из B смежна с x_1, x_{12} или x_4, x_{12}. Случай, когда b из B смежна с x_{11}, x_{12} невозможен, так как это противоречит предположению о том, что длина максимального цикла не превышает 6, а в этом случае вершины $x_1, x_2, x_3, x_4, x_{11}, b, x_{12}$ образуют цикл длины 7. И, наконец, если вершина b будет смежна с вершинами x_1, x_4, то при добавлении в S она не увеличит величину $\tau(S \cup B)$.

Рассмотрим случай $S = H_4, H_5, H_6$. Если предположить, что вершина b смежна с вершинами x_1, x_{11} или x_4, x_{11}, то вершины b и x_{11} становятся P_8-терминальными, $\tau(S)$ не возрастает, как и в случае, рассмотренном выше. Случай, когда b смежна с вершинами x_1, x_4 невозможен, так как противоречит предположению о минимальности s.

Если $S = H_7, H_8, H_9$, то к сказанному выше можем только добавить, что предположение о том, что b смежна с x_1, x_4 невозможно, так как противоречит предположению о минимальности s.

Для следующего семейства случаев, представленного на рис. 5–8, аналогично предыдущему семейству случаев заметим, что в случаях, когда $S = H_{11}, H_{13}, H_{15}, H_{17}$, предположение о том, что b смежна с x_1, x_7, приводит к противоречию с предположением о минимальности s, так как в этом случае оказывается, что в графе G содержится один

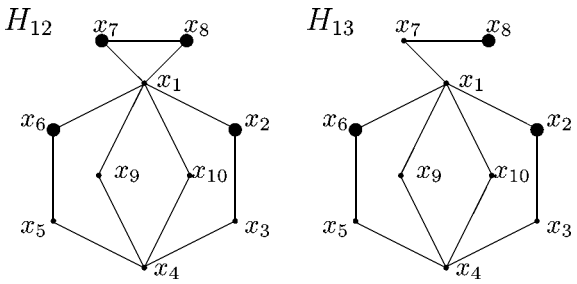


Рис. 6. Случай 5

из подграфов $H_{10}, H_{12}, H_{14}, H_{16}$ соответственно.

В случае, если $S = H_{10}, H_{11}$, с учетом замечания для вершины b, остается рассмотреть только вариант, когда вершина b окажется смежной с вершинами x_3, x_5. Заметим, что в этом случае вершина b может быть добавлена к S, при этом $\tau(S)$ не изменится. Аналогично в случае, если b смежна с x_1, x_3 или (симметричный случай) с x_1, x_5.

Случай, когда $S = H_{12}, H_{13}$, отчасти аналогичен случаю, когда $S = H_1, H_2, H_3$, так, предполагая, что b смежна с x_1, x_9 или с x_4, x_9, (симметричный случай x_1, x_{10} или x_4, x_{10}), получаем, что после добавления вершины b к подграфу S, вершины $b, x_9, (x_{10}), x_3, x_5$ становятся P_8-терминальными, и после этого ни одна вершина b_1 не может быть добавлена в S таким образом, чтобы $\tau(S)$ увеличилось. Аналогично в случае, если b смежна с x_1, x_3 или x_1, x_5, при этом P_8-терминальными становятся вершины b, x_9, x_{10}. Вершина b не может быть смежна с x_9, x_{10}, так как это противоречит предположению о том, что в G нет C_7, а с x_3, x_5 — потому что это противоречит предположению о минимальности s, так как при этом в графе G содержался бы подграф H_{10}, H_{11}. Случаи, если b смежна с x_5, x_9 или с x_5, x_{10} (симметричные случаи — x_3, x_{10} или x_3, x_9 соответственно), также невозможны в связи с предположением о длине максимального цикла в G.

Пусть $S = H_{14}, H_{15}$. Если b смежна с x_1, x_9 или x_4, x_9, то при ее добавлении в S вершины b, x_3, x_5, x_9 становятся P_8-терминальными и больше ни одна вершина b_1 не может быть добавлена в S таким образом, чтобы величина $\tau(S)$ изменилась. Аналогично с предыдущим случаем b не может быть смежна с x_3, x_5, так как это противоречит предположению о минимальности s. Предположение о том, что b смежна с x_1, x_4, отсылает нас к предыдущему случаю. Кроме того, вершина

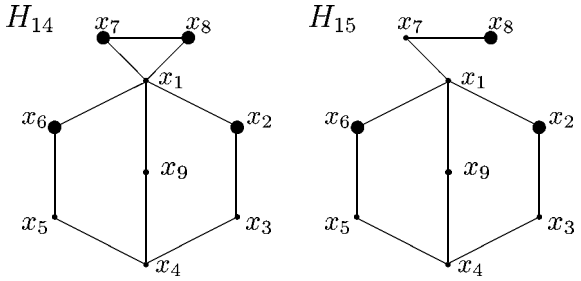


Рис. 7. Случай 6

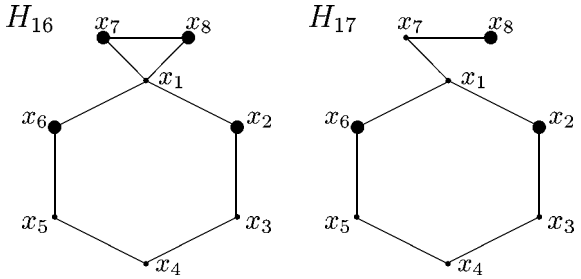


Рис. 8. Случай 7

b не может быть смежна с x_3, x_9 или x_5, x_9, так как в этом случае в G содержался бы C_7. Если же b окажется смежной с x_1, x_3 или x_1, x_5, то она и x_9 окажутся P_8-терминальными в S, причем при ее добавлении в S величина $\tau(S)$ не изменится.

Случай, если $S = H_{16}, H_{17}$, в целом полностью аналогичен предыдущим случаям этого семейства. Точно так же, если b смежна с x_1, x_3 или x_1, x_5, то она становится P_8-терминальной и при добавлении ее или ее дубликатов роста $\tau(S)$ не происходит. В этом случае b не может быть смежна ни с x_1, x_4, ни с x_3, x_5, так как при этом получаем противоречие с предположением о минимальности s. Отдельно заметим, что если допустить, что b смежна с x_3, x_4 или с x_4, x_5, то получим противоречие с предположением о том, что в G нет C_7, что, в общем, очевидно.

Особенный интерес представляют случаи $S = H_{18}, H_{19}, H_{20}$, так как они позволяют существенно сократить количество перебираемых случаев с учетом того, что при последовательном применении шага 4 и следом за ним шага 2 алгоритма, мы оказываемся в одной из рассмотренных

выше ситуаций, поэтому в этих трех случаях нам достаточно показать, что на шаге 2 ни в одном из случаев в подграфе S не возникает пути с длиной более 8.

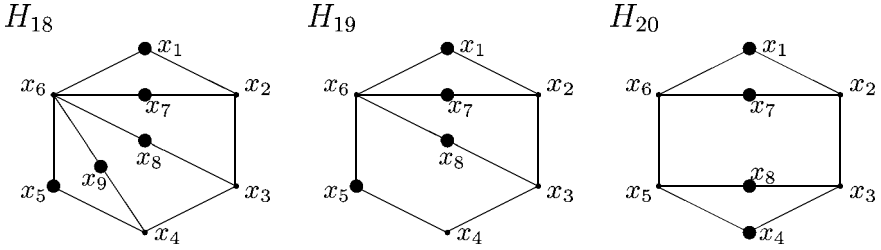


Рис. 9. Случай 8

Относительно случая $S = H_{18}$ достаточно заметить, что предположение о том, что b смежна с x_2, x_4 противоречит нашему предположению об отсутствии в G цикла длины 7, который в этом случае образовывали бы вершины $b, x_2, x_7, x_6, x_8, x_3, x_4$ (заметим сразу, что это же соображение справедливо и для следующего случая $S = H_{19}$), а при добавлении вершин из B, смежных с x_2, x_6 (симметричный вариант x_4, x_6) или x_3, x_6, значение $\tau(S)$ увеличиваться не будет.

Для случая $S = H_{19}$ добавим только, что, если b смежна с x_4, x_6, то мы оказываемся в условиях предыдущего случая, а для случая $S = H_{20}$ заметим, что, если b смежна с x_2, x_5 или x_3, x_6, то мы оказываемся в условиях случая $S = H_{19}$.

Рассмотрим случай $S = H_{21}$. Вершина b может быть смежна с вершинами x_1, x_3, при этом она является дубликатом вершины x_9 и при добавлении в S становится P_8-терминальной, как и в случаях, если вершина b окажется смежна с вершинами x_1, x_5 или с вершинами x_3, x_5. В любом из этих случаев величина $\tau(S)$ остается неизменной. Заметим, что b не может быть смежна с вершинами x_2, x_5, так как при этом в графе G имелся бы цикл длины 7.

Для случая $S = H_{22}$ заметим, что b не может быть смежна с вершинами x_2, x_6, так как при этом вершины $b, x_2, x_3, x_4, x_9, x_1, x_6$ образовывали бы цикл длины 7, а в случае, если допустить, что она смежна с x_3, x_6, то в графе имелся бы цикл длины 8. Ни одна вершина из B не может быть смежна ни с парой вершин x_2, x_4, ни с парой x_4, x_6, так как при этом получим противоречие с предположением о минимальности s,

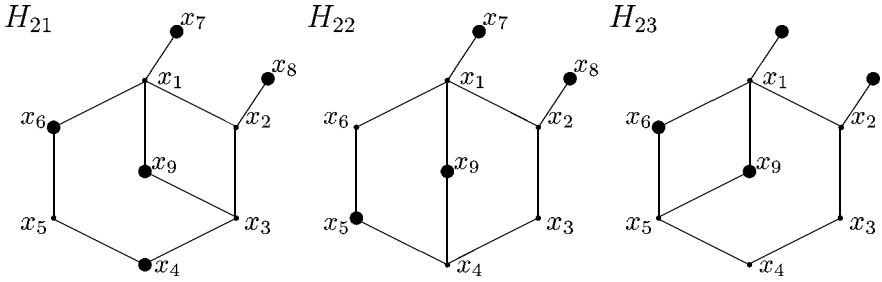


Рис. 10. Случай 9

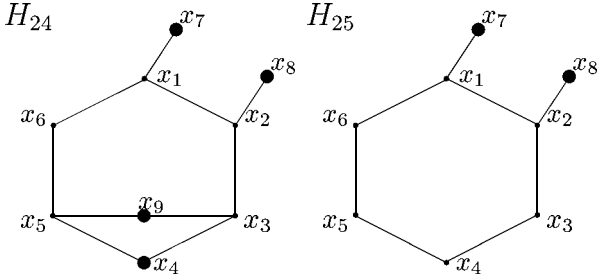


Рис. 11. Случай 10

так как в этом случае имело бы место $S = H_{19}$. Таким образом, при выполнении второго шага алгоритма на данном подграфе в S могут быть добавлены только вершины, смежные с x_1, x_4, что не может вызвать роста $\tau(S)$.

Аналогично, в случае $S = H_{23}$ в подграф могут быть добавлены только те вершины из B, которые смежны с x_1, x_5, при добавлении такие вершины станут P_8-терминальными и не изменят значение $\tau(S)$. Если бы b была смежна с x_1, x_3, то в G имелся бы подграф H_{21}, а если с x_1, x_4 или x_2, x_5, то в G имелся бы H_{22}. Наконец, если предположить, что b смежна с x_2, x_4, получим, что в G имеется H_{20}.

Рассмотрим случай $S = H_{24}$. Аналогично предыдущим случаям этого семейства, для вершины b из B возможен лишь вариант, когда b смежна с x_3, x_5, и тогда ее добавление в S не изменяет величину $\tau(S)$, причем сама она становится P_8-терминальной. Во всех остальных слу-

чаях получаем противоречие с нашим предположением о минимальности s. В случае $S = H_{25}$ содержательным является также только один вариант для b, а именно — вариант, когда она смежна с x_3, x_6. При этом и вершина b и вершины x_4, x_5 становятся P_8-терминальными.

Рассмотрим случай $S = H_{26}$. В этом случае для вершины b также имеется только один вариант, когда она смежна с x_2, x_6, при этом такая вершина, будучи добавлена в подграф S, не повлияет на величину $\tau(S)$. Во всех остальных случаях получим противоречие с нашим предположением о минимальности s. Аналогично и в случае $S = H_{27}$, предположение о том, что b смежна с x_1, x_3 или x_3, x_5, а также с x_2, x_6, приводит нас к тому, что в G имеется H_{19}, а полагая, что b смежна с x_1, x_5 или с x_2, x_5, мы придем к противоречию с предположением о том, что в G нет цикла длины 8. Поэтому вершина b может быть смежна лишь только с верщинами x_3, x_6, при этом ее добавление в S не изменит величину $\tau(S)$.

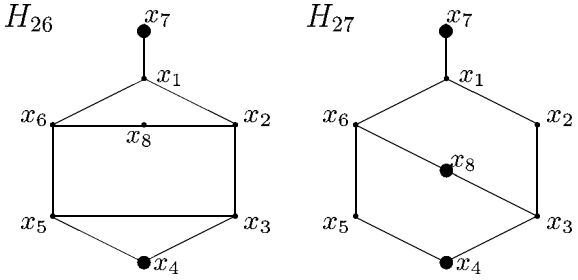


Рис. 12. Случай 11

И наконец, в случае, когда $S = H_{28}$ или $S = H_{29}$, ни одна вершина b из B не может быть добавлена в S при выполнении шага 2 алгоритма, так как при этом получим противоречие с предположением о минимальности s. Так, если предположить, что b смежна с x_2, x_6 или x_3, x_5, то окажется, что в G имеется H_{26}, а если бы b была смежна с x_2, x_5 или x_3, x_6, то в G имелся бы подграф H_{27}. Предполагая, что b смежна с x_1, x_4, в обоих случаях получим, что в G имелся бы цикл длины 7.

На этом разбор случая 1 может быть закончен. Как видно из вышеприведенных рассуждений, алгортим всякий раз заканчивает работу и S становится P_n-ядром графа G.

Как было сказано выше, в настоящей работе мы опустим разбор случая 2 в связи с большим количеством подслучаев, требующих перебора. Идея этого перебора полностью аналогична случаю 1, технически он осуществляется без каких бы то ни было проблем.

Теорема о существовании P_9-ядра доказана.

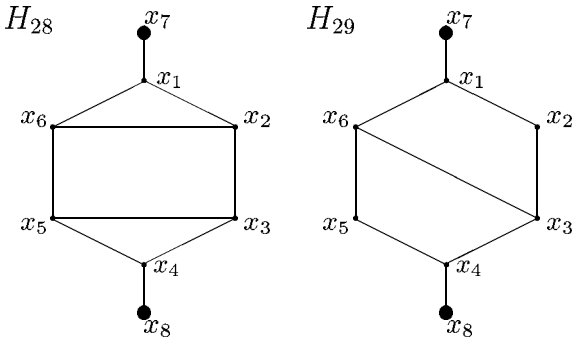


Рис. 13. Случай 12

4. ЗАКЛЮЧЕНИЕ

Как было сказано выше, наличие P_n-ядра в графе G означает, что граф G является $(\tau(G)-n+1, n-1)$-разбиваемым. Непосредственным следствием из теоремы о существовании P_9-ядра будет следствие о том, что граф G с $\tau(G) \geq 8$ является $(\tau(G)-8, 8)$-разбиваемым. Откуда немедленно вытекает

Теорема 4.1. *Граф G является τ-разбиваемым при любом $\tau(G) \leq 17$.*

Трудоемкость доказательства приводит авторов к необходимости искать контрпример графа G, который не является $\tau(G)$-разбиваемым, и следовательно, существует пара чисел (a, b), такая что $\tau(G) = a + b$ и граф G не является (a, b)-разбиваемым. Предположим $a \leq b$. Непосредственно из результатов данной работы, а также из результатов [2, 7, 20, 21] имеем следующие свойства, которыми обладает такой граф-контрпример.

1. В G отсутствует гамильтонов путь.

2. $a > 8$.

3. $17 < \tau(G) < |V(G)| - 1$.

4. $3 < \Delta(G) < |V(G)| - a - 1$, где $\Delta(G)$ обозначает максимальную степень графа G.

5. В G имеется циклический блок, не являющийся гамильтоновым, причем $c(G) \geq 5$.

6. В G имеются как четный, так и нечетный циклы, но нет циклов длины b.

7. В G имеются длинные циклы: имеет место неравенство $c(G) > a + 2$.

8. В G имеются и короткие циклы: имеет место $g(G) < a - 1$.

Кроме того, как уже говорилось, нам известно, что из существования P_n-ядра вытекает τ-разбиваемость графа. Однако нам неизвестно, есть ли обратная связь, т. е. следует ли из τ-разбиваемости графа существование путевого ядра некоторого порядка, и если ответ положительный, то какова связь порядка ядра с $\tau(G)$.

Авторы выражают свою искреннюю признательность А. В. Косточке, привлекшему внимание к этой теме и давшему много полезных советов в процессе работы.

СПИСОК ЛИТЕРАТУРЫ

1. **Borowiecki M., Broere I., Frick M., Mihók P., Semanišin G.** A survey of hereditary properties of graphs // Discussiones Mathematicae Graph Theory. — 1997.— Vol. 17, N 1. — P. 5–50.
2. **Broere I., Dorfling M., Dunbar J. E., Frick M.** A path(ological) partition problem // Discussiones Mathematicae Graph Theory. — 1998. — Vol. 18, N 1. — P. 113–125.
3. **Broere I., Dorfling M.** The decomposibility of additive of hereditary properties of graphs // Discussiones Mathematicae Graph Theory. — 2000 — Vol. 20, N 2. — P. 281–291.
4. **Broere I, Frick M., Semanišin G.** Maximal graphs with respect to hereditary properties // Discussiones Mathematicae Graph Theory. — 1997. — Vol. 17, N 1. — P. 51–66.
5. **Broere I., Hajnal P., Mihók P.**. Partition problems and kernels of graphs // Discussiones Mathematicae Graph Theory. — 1997. — Vol.17, N 2. — P. 51–56.
6. **Chartrand G., Geller D. P., Hedetniemi S. T.** A generalization of the chromatic number // Proc. Camb. Phil. Soc. — 1968. — Vol. 64, N 2. — P. 265–271.
7. **Dunbar J. E., Frick M.** Path kernels and partitions // J. Combin. Math. Combin. Comput. — 1999. — Vol. 31. — P. 137–149.

8. **Dunbar J.E., Frick M., Bullock F.** Path partitions and P_n-free sets // Discrete Math. — 2004. — Vol. 289, N 1–3. — P. 145–155.
9. **Frick M., Bullock F.** Detour chromatic numbers of graphs // Discussiones Mathematicae Graph Theory. — 2001. — Vol. 21, N 2. — P. 283–291.
10. **Györy E., Kostochka A.V., Luczak T.** Graphs without short odd cycles are nearly bipartite // Discrete Math. — 1997. — Vol. 163, N 1–3. — P. 279–284.
11. **Hajnal P.** Partitions of graphs with condition on the connectivity and minimum degree // Combinatorica. — 1984. — Vol. 3, N 1. — P. 95–99.
12. **Hajnal P.** Graph partitions // Ph.D. Thesis, Szeged Attila József University, 1984.
13. **Kapoor S. F., Kronk H. V., Lick D. R.** On detuurs in graphs // Canad. Math. Bull. — 1966. — Vol. 11, N 2. — P. 195–201.
14. **Laborde J. M., Payan C., Xuong N. H.** Independent sets and longest directed paths in digraphs // Graphs and other combinatorial topics (Prague, 1982). — Leipzig: Teubner, 1983. — P. 173–177.
15. **Lovász L.** On decomposition of graphs // Studia Sci. Math. Hungar. — 1966. — Vol. 1, N 1–2. — P. 237–238.
16. **Mihók P.** Problem 4 // Graphs, hypergraphs and matroids. — Zielona Góra: Higher College Engrg., 1985. — P. 86.
17. **Stiebitz M.** Decomposing graphs under degree constraints // J. Graph Theory. — 1996. —- Vol. 23, N 3. — P. 321–324.
18. **Thomassen C.** Graph decomposition with constraints on the connectivity and minimum degree // J. Graph Theory. — 1983. — Vol. 7, N 2. — P. 261–291.
19. **Vronka J.** Vertex sets of graphs with prescribed properties // Ph. D. Thesis, Košice, Šafárik University, 1986.
20. **Мельников Л. С., Петренко И. В.** О путевых ядрах и разбиениях в неориентированных графах // Дискретный анализ и исследование операций. — 2002. — Т. 9, Сер. 1, № 2. — С. 21–35.
21. **Мельников Л. С., Петренко И. В.** Путевые ядра и длины циклов в неориентированных графах // Современные проблемы конструирования программ. — Новосибирск, 2002. —- С. 222–231.

Г.П. Несговорова*

ОБЗОР ВИРТУАЛЬНЫХ МУЗЕЕВ В СЕТИ ИНТЕРНЕТ

ВВЕДЕНИЕ

Бурное развитие информационных технологий и их широкое использование во всех областях человеческой деятельности привело к изменению и модификации привычных реалий при их отражении в виртуальном пространстве. Примером этого, в частности, служат появившиеся и стремительно развивающиеся в сети Интернет так называемые виртуальные музеи, которые по сути своей представляют базы данных, содержащие музейные электронные экспонаты, цифровые фото-, аудио- и видеоматериалы, анимацию и многое другое. Понятие виртуальный музей вошло в нашу жизнь в конце прошлого века, начиная с середины 90-х годов. Сейчас в сети Интернет с помощью поисковых систем можно найти более тысячи электронных музеев с таким названием.

Что они представляют собой, что скрывается под вывеской «**Виртуальный музей»** и попытаемся выяснить в этой статье, целью которой является обзор и некоторая попытка классификации того множества явлений, представленных в сети, которые носят название виртуальный музей.

1. РЕАЛЬНЫЙ МУЗЕЙ VERSUS ВИРТУАЛЬНЫЙ МУЗЕЙ

Классически учреждение культуры под названием музей определяется следующим образом.

Музей (лат. museum — храм муз) — это учреждение, собирающее, хранящее и выставляющее для обозрения произведения искусства, предметы истории, науки, быта, промышленности, сельского хозяйства, материалы из жизни и деятельности великих людей и т.д., расположенные по определенной системе, с целью их сохранения, наглядного изучения, а также распро-

* gpn@iis.nsk.su

Работа выполнена при финансовой поддержке Российского гуманитарного научного фонда (грант № 02-05-12010).

странения знаний среди широких масс населения [1]. Впервые упоминания о музеях относятся к XV–XVI вв.

Определение и все вышеперечисленные функции реально существующих музеев присущи в той или иной степени и виртуальным музеям, существующим в сети Интернет. В чем же их разница, какие особенности, отличия в принципе организации и общие черты имеются у реальных и виртуальных музеев все это будет рассмотрено ниже.

1.1. Виртуальный музей — новая реальность?

По ключевым словам «виртуальный музей» любая поисковая система в Интернете выдаст множество сайтов, где эти слова задействованы. При просмотре этих сайтов возникает желание разобраться в том, «что есть что», и навести некоторый порядок. В современном Интернете существуют два типа музейных сайтов, которые делятся на

1) представительства реально существующих музеев и

2) собственно виртуальные музеи,

но называются все они пока одинаково: виртуальные музеи.

Сайты на музейную тематику **реальных музеев** (РМ) весьма многочисленны и популярны и обеспечивают Интернет-пространство информацией о многих музеях мира, их экспозициях, выставках, каталогах, месте расположения, времени работы и разных мероприятиях, т.е. традиционный сайт РМ есть лишь техническое средство для распространения музейной информации и в некотором роде реклама. Сайт реального музея надо бы называть (в отличие от виртуального музея) **виртуальный мир** такого-то реального музея, чтобы не путать с собственно виртуальным музеем, например, **Виртуальный мир Эрмитажа** или **Русского музея**.

Собственно **виртуальные музеи** (ВМ) встречаются в Интернете реже, их куда меньше, чем представительств реально существующих музеев, но они набирают силу и их становится в последнее время все больше и больше. Далее займемся рассмотрением особенностей музеев второго типа в Интернете, которые и будем называть **виртуальными**.

Общим для реальных и виртуальных музеев является то, что любой музей — это место, где история продолжается, они не являются хранилищем древностей. История здесь встречается с современностью, так как экспонаты музеев пополняются отражениями событий текущего момента (книги и биографии, документы и вещи, проекты и фотоматериалы и т.д.).

1.2. Отличительные черты виртуального музея

Особенность виртуального музея заключается в том, что такого музея (в отличие от реального) физически не существует (вернее сказать — организационно), но все же это музей, так как соответствует данному выше определению. Он хоть и расположен в сети Интернет, но основан на реальных экспонатах и имеет свою собственную структуру. Причем каждый организатор виртуального музея выбирает ту структуру и организацию, которая кажется ему наиболее удобной и наглядной. В чем-то прообразом для ВМ служит РМ и его структурная организация (экспонаты, выставки, экспозиции, запасники, каталоги и т.д.), а что-то каждый привносит свое, создавая свой особый виртуальный музей. Надо отметить, что идея создания виртуального музея столь же проста, сколь сложна ее техническая реализация.

При кажущейся аналогии с обычным реальным музеем, виртуальный музей — это все-таки новая реальность, которая выходит за рамки традиционного представления о реальном музее с его постоянной экспозицией и временными выставками, в то время как экспозиция виртуального музея постоянна лишь в своем развитии, а время работы выставок виртуального музея может длиться годами, и их количество связано лишь с новыми идеями, интересными проектами, а ограничено только тематикой данного музея. Экспонаты реального музея со временем приходят в негодность, коллекция же виртуального музея снимает вопрос о сохранении своих образцов.

Особенностью виртуального музея является и то, что зритель (он же пользователь) «посещает виртуальный музей на своем компьютере, общается с ним один на один и сам устанавливает с ним (ВМ) личные отношения, погружается в новую реальность, которую он сам воссоздает в своем сознании. Именно в «новой реальности» виртуального музея человек из зрителя превращается в участника этой «новой реальности», здесь ему никто не мешает: ни другие посетители, ни служители музея" [2]. Кроме того, посещать виртуальный музей можно в любое время дня и ночи, нет никаких очередей за билетами и ограничений на время пребывания в музее для просмотра экспонатов.

Виртуальный музей будет работать долгие годы, не прерываясь ни на минуту, даже в праздники и выходные, днем и ночью, попасть в него можно из любого уголка мира, и число посетителей виртуального музея гораздо больше, чем у реального. Конечно, у виртуального музея, как у любого сайта, тоже есть свой «сценарий»: это его структура, план, карта, но инициати-

ва при посещении виртуального музея принадлежит все-таки самому человеку.

Таким образом, виртуальный музей — это не просто сайт реально существующего музея, а созданный в сети оригинальный сайт, не имеющий своего аналога в реальности и представляющий любую тематику, если по ней находятся реальные материалы, имеющие свое физическое или идейное воплощение в реальном мире. Есть даже мнение, что виртуальный музей — это зеркало будущего реального музея, а не наоборот, т.е. на основе виртуального музея можно создать реальный музей, если это кому-нибудь надо.

Представительства реальных музеев в сети Интернет и виртуальные музеи — это разные организации. ВМ отличаются от виртуальных представительств РМ прежде всего тем, что они являются не только носителями той или иной информации, но и ее первоисточником.

2. ОТНОШЕНИЕ К СЕТИ ИНТЕРНЕТ РЕАЛЬНЫХ И ВИРТУАЛЬНЫХ МУЗЕЕВ

У реальных и виртуальных музеев разные отношения к сети Интернет. Первый — рассматривает сеть Интернет как способ представительства своей деятельности, а второй — как способ ведения самой деятельности, т.е. способ существования. РМ размещает в сети Интернет справочную информацию о работе музея, анонсы временных выставок, обзор основных разделов музея. Его задача — заинтересовать человека через Интернет в реальном посещении музея.

ВМ размещает в Интернете собственно экспозицию музея и собственно выставки. Его задача — показать посетителю виртуального музея **здесь** и **сейчас** то, что в другом месте (реальном или виртуальном) он увидеть не сможет. Состав экспозиций и выставок и их количество определяются лишь концепцией виртуального музея.

Реальный музей не использует в полную силу Интернет, у него, как правило, отсутствует страница ссылок, для него Интернет лишь одна из рекламных площадок своей деятельности. Для ВМ само его существование обусловлено максимальным включением в сетевые ресурсы, электронные конференции, совместные проекты и т.п., без участия в которых о нем никто не узнает. Поскольку виртуальный музей не связан с конкретным помещением (зданием), то для него Интернет — это сфера и жизнедеятельности и среда обитания.

Даже внешне сайт реального музея отличается от сайта виртуального: сайт РМ делает акцент на **оригинальном дизайне и внешнем эффекте**, он создается по образу рекламного проспекта, путеводителя, каталога, руководствуясь требованиями музейных работников.

В сайте виртуального музея главное внимание уделяется **актуальности информации и оригинальности материалов** (часто в ущерб оригинальности дизайна). Наконец, частота обновления информации на сайте ВМ в несколько раз выше, чем на сайте РМ, так как для формирования круга своих постоянных посетителей виртуальный музей непрерывно обновляет сайт, размещая на нем новостную информацию, организуя форумы, конференции. Виртуальная выставка в значительно меньшей степени ограничена временем и пространством по сравнению с реальной.

3. НОВЫЕ ВОЗМОЖНОСТИ ВИРТУАЛЬНОГО МУЗЕЯ

Виртуальный музей — это идеальная площадка для его создателя (куратора), который для размещения своего музея не связан ни помещением, ни физическим местонахождением экспонатов либо необходимостью их транспортировки, к примеру. Существуют, например, конкретные частные коллекции по какой-то определенной тематике, они разобщены, каждая имеет своего хозяина, но нет помещения для их демонстрации. При этом владельцы хотели бы объединиться и ознакомить с ними тех, кому это интересно, не сдавая свои экспонаты в существующие реальные музеи для организации выставки. В этом с**лучае виртуальный музей частных коллекций** — это именно та структура, в которой они могут реализовать свое желание, организовав совместный проект. Их идея в этом случае найдет в виртуальной среде свое адекватное воплощение.

Виртуальный музей позволяет сохранить уникальную архивную информацию, которая по старинке записана на бумажных или магнитных носителях и которая со временем может быть утрачена, т.е. виртуальный музей снимает вопрос о сохранности своих образцов, что так актуально для реальных музеев (учитывая различные природные катаклизмы и стихийные бедствия, от которых никто не застрахован). При этом место для сохранности любой информации в ВМ практически не ограничено.

4. ОСНОВНЫЕ АСПЕКТЫ СУЩЕСТВОВАНИЯ ВИРТУАЛЬНЫХ МУЗЕЕВ

4.1. Информационные аспекты

Все принадлежит всем — это главный феномен Интернета. Здесь всеобщая «халява» соседствует со всеобщим альтруизмом. Виртуальный музей не просто открыт для всех, он принадлежит всем! Лозунг виртуального музея: «Посторонним вход разрешен!» Каждый практически свободно может использовать любую информацию виртуального музея (картинки, статьи, документы, фото и пр.) и совершенно бесплатно.

Коммуникативная функция виртуального музея — один из самых важных аспектов «новой реальности». Виртуальный музей можно рассматривать как окно в мир: вы свободно можете зайти в любой из виртуальных музеев различной тематики, поучаствовать в конференциях, оставить свои впечатления и пожелания в гостевой книге. «Сетевой человек» живет ощущением причастности ко всему, что происходит в мире.

Актуальность — это важный аспект, включающий в себя обновляемость информации и своевременность её появления в сети. Если за месяц-другой не открылось ни одной новой выставки, это не будет способствовать посещаемости сайта. Если же в виртуальном музее регулярно появляются новости, анонсы будущих выставок и обсуждается на форуме интересная проблематика, то «к нему не зарастет народная тропа».

4.2. Социальные аспекты

Свобода — это значит, что у виртуального музея нет нужды в помещении, финансировании, штате сотрудников и проверяющих — цензорах, как в реальном музее. Достаточно оригинальной идеи и желания реализовать ее не на словах, а в реальности, хоть и виртуальной. Единственное условие — наличие компьютера с выходом в Интернет и умение воплотить свою идею в виртуальном виде технически.

Равенство в сети состоит в том, что все виртуальные музеи в какой-то степени равны. Как и в реальной жизни, здесь важна «различимость», т.е. оригинальность тематики сайта и его жизненность, иными словами обновляемость, а не размер и количество вложенных затрат. Важно, чтоб было интересно, ново и в хорошей компании.

Братство заключается в том, что виртуальный музей есть часть ресурсов сети, которые, в свою очередь, дробятся на более специализированные сообщества (представительства реальных музеев, виртуальные выставки и

музеи, различные сетевые проекты и т.д.). Все эти сообщества организуются стихийно, этому способствуют страницы ссылок, которые создает всякая виртуальная организация, так как именно страница ссылок и определяет тот контекст, в котором она «работает», а также участие в электронных конференциях и сетевых проектах.

Демократия присуща сетевой жизни, так как она более непосредственна и дружественна, чем реальная жизнь. Вы можете связаться по e-mail с любым корреспондентом и можете рассчитывать на ответ, лишь бы предмет общения был интересен для обеих сторон. Виртуальный музей инициирует связи с близкими ему виртуальными организациями и отдельными личностями и получает письма от самых разнообразных корреспондентов, которые тоже рассчитывают на ответную реакцию.

Независимость виртуального музея — это прежде всего то, что он не зависит в своей деятельности от чиновников. Над ним нет ни управления культуры, ни другой вышестоящей организации, не надо ни с кем согласовывать свою деятельность и выбивать финансирование.

4.3. Технические аспекты

Время: у виртуального музея свое отношение со временем. Он может жить и развиваться многие годы, а может быть «закрыт» в один момент, если что-то сломалось в сети. Для этого часто используют свободные ресурсы и организуют там «зеркала» на случай технических неполадок.

Пространство: это один из основных аргументов в пользу «новой реальности» виртуального музея. Во-первых, виртуальный музей, в отличие от реального, свободен в выборе своей структуры и спокоен за возможность ее развития в случае расширения экспозиции и открытия новых выставок. Во-вторых, само пространство виртуального музея отличается от залов реального музея, оно пронизано гиперссылками, реализующими взаимосвязь и многоаспектность представления информации. В-третьих, это «пространство» — распределенное, так как физически части виртуального музея могут размещаться на разных ресурсах, а главное, что сам виртуальный музей может представлять собой ассоциацию разных проектов: музейных, выставочных, информационных и в том числе совместных.

Мультимедийность: виртуальный музей позволяет использовать и объединять в проектах различные способы представления информации: тексты, рисунки, фото, аудио, видео, анимацию и т.п., создавая тем самым «новую реальность» самого виртуального музея.

5. ТЕМАТИКА ВИРТУАЛЬНЫХ МУЗЕЕВ

Ко всем указанным особенностям и преимуществам виртуальных музеев перед реальными можно добавить также и широту их тематики. Конечно, выбранная для создания виртуального музея тема может иметь ограниченный материал в реальном мире, но интересна она не своим объемом, а оригинальностью экспонатов и своей познавательностью. Путешествуя по «бескрайним просторам Интернета», можно натолкнуться на виртуальные музеи редкой и необычной тематики, для демонстрации экспонатов которой вряд ли были бы открыты реальные музеи или даже временные выставки в них. Из обнаруженных в Интернете виртуальных музеев хочется отметить редкие по своей тематике: «Дурацкий музей», «Музей печали», «Реалии стран победившего социализма», «Музей родовой травмы», раздел «Рвотные пакетики» в виртуальном музее Аэрофлота, «Музей радио и телевидения», «Музей спичечных этикеток (филуминистики)». Наряду с «Музеем Симона Петлюры» можно найти «Музей Coca-Cola» и зайти в «Музей палехской миниатюры», посетить «Виртуальный музей находок», «Виртуальный музей мошенничества» и т.д. и т.п. Большое количество виртуальных музеев посвящено истории развития компьютерной техники и информационных технологий в мире, бывшем Советском Союзе, России, что является интересным и важным для тех больших и малых городов страны, которые имеют только, как правило, краеведческие музеи и не имеют в них даже экспозиций на тему развития науки и техники. Время же любой временной тематической выставки в реальном музее ограничено. Иногда она находится в другом городе или в другой стране, поэтому реально посетить ее и узнать что-то новое на интересующую тему или приобщиться к прекрасному бывает затруднительно, и тогда на помощь приходит виртуальный музей.

6. ОБРАЗОВАТЕЛЬНАЯ И ПРОСВЕТИТЕЛЬСКАЯ ФУНКЦИИ ВИРТУАЛЬНОГО МУЗЕЯ

Задача виртуального музея, как и реального, — помочь людям, оторванным географически от музейных центров, стать творческой личностью и сформировать свою систему ценностей вне зависимости от того, где они проживают. Виртуальный музей несет не только информационную функцию, но также и образовательную (правда, для этого, как минимум, надо иметь под рукой компьютер с выходом в Интернет, будь это дома или в учебных классах).

Виртуальный музей может стать отличным учебным полигоном не только для студентов разных специальностей, но и для школьников, учащихся средних специальных учреждений, а также интеллигенции и особенно жителей сельских местностей, так как их тематика очень обширна.

Перед «входом» в виртуальный музей его посетитель (пользователь) должен дать обязательство сохранять авторское право его создателя, учитывая особенности правого статуса электронных библиотек и виртуальных музеев в виртуальном пространстве. Это оговорено не во всех ВМ, хотя само собой подразумевается, и необходимо делать ссылки на создателя ВМ, хотя юридически авторские права в Интернете не всегда бывают защищены.

Посещая виртуальный музей в Интернете, все полученные знания можно «унести» с собой, сохранив на свой жесткий диск, а полученные впечатления время от времени освежать, заглядывая в ВМ в поисках новинок. Таким образом, «взяв» материалы виртуального музея с собой, можно показать их знакомым, родственникам и друзьям с помощью лазерного диска, расширив тем самым число посетителей ВМ.

Однако при посещении виртуального музея возникает вопрос, адекватно ли впечатление от виртуальной экскурсии впечатлению от реальной? Опрос на тему: «Считаете ли Вы, что виртуальный музей может заменить обычный?» — дал следующие результаты:

ДА — 25%
НЕТ — 61%
НЕ УВЕРЕН — 12%
НЕ ЗНАЮ — 2% .

Прежде всего эта статистика относится к виртуальным музеям, посвященным живописи. Несомненно, что смотреть картины лучше в реальном музее, чем в виртуальном: восприятие не то (это как разница между подлинником картины и ее копией). Но ВМ в данном случае может явиться источником информации о существовании конкретного произведения искусства и вызвать желание ознакомиться с ним в реальном варианте.

Для улучшения качества восприятия в проектах виртуальных музеев могут использоваться возможности анимации объектов и видеоизображений. Можно реализовать интерфейс с пользователем через события взаимодействия с окружающими объектами. Вы можете подвигать стулья, повернуть экран виртуального компьютера, послушать голосовые сообщения о

выбранном экспонате, включить музыку сопровождения, что создаст иллюзию реальной экскурсии.

В некоторых виртуальных музеях [3] предлагается смоделировать виртуальный музей, как реальный: например, ВМ располагается на фоне озера и лесных полян, по которым можно пройтись. У входа в музей находится площадка с виртуальным рекламным щитом, перед входом стоит виртуальный охранник, который укажет направление дальнейшего осмотра при обращении.

Вход в музей осуществляется путем нажатия «мышкой» на дверь, после чего открывается зал посетителей. При визуальном просмотре можно послушать звуковые сопровождения на каждый экспонат, используя технологии визуального и речевого сопровождения виртуального мира. В конце зала — дверь в другой зал, можно посмотреть в окно, сделав панорамный обзор. При этом открывается природный ландшафт с изображением бьющего фонтана, клумб, озера, деревьев, где виртуально можно погулять, выйдя из музея. Такими средствами сближаются виртуальный и реальный миры и улучшается восприятие виртуальной реальности.

Для сравнения виртуальных музеев выработаны следующие коэффициенты по оценке рейтинга ресурсов:

1 — КП — количества посещений
2 — КЦ — количества цитирования
3 — ШОТ — широты охвата темы
4 — КИ — качества информации
5 — КПИ — качества представления информации, и разработана методика их расчета [4].

7. ОСОБЕННОСТИ ВИРТУАЛЬНОГО МУЗЕЯ ИСТОРИИ ИНФОРМАТИКИ В СИБИРИ

Виртуальный музей истории информатики в Сибири (CBM/SVM) создается в лаборатории конструирования и оптимизации программ ИСИ СО РАН [5]. Музей строится в виде информационно-поисковой, справочной и обучающей адаптивной гипермедиа-системы, доступной в сети Интернет. Подавляющее большинство из представленных в настоящее время в Интернете виртуальных музеев основано на использовании традиционных гипермедиа-технологий, одним из ограничений которых является то, что они предоставляют одно и то же информационное содержание и один и тот

же механизм навигации всем пользователям. Вместе с тем, виртуальный музей предназначен для посещения его различными категориями пользователей с различными предпочтениями, целями, знаниями, интересами, которых могут интересовать разные части имеющейся информации. Для этого они могут использовать разные пути для навигации. Поэтому при создании СВМ особое внимание уделялось вопросам адаптации. Цель адаптивной гипермедиа и состоит в том, чтобы увеличить функциональные возможности гипермедиа, сделав ее индивидуализированной.

Для адаптивного представления информации используются такие методы, как дополнительные и предварительные объяснения и сортировка, методы адаптивной навигационной поддержки (полное руководство), а также сортировка, сокрытие, аннотирование и генерирование ссылок. Модель зарегистрированного пользователя состоит из моделей категорий, знаний и предпочтений.

Смысл методов адаптивной навигационной поддержки состоит в том, чтобы помочь пользователям найти путь в гиперпространстве с помощью адаптации способа представления ссылок к целям, знанию и другим характеристикам каждого индивидуального пользователя.

В СВМ реализован дружественный интерфейс для регистрации и аутентификации пользователей музея и ведения ими электронной конференции.

А в основном СВМ включает те же составляющие структурные единицы, которые присущи и реальным музеям. Базы данных виртуального музея истории информатики предусматривают хранение и обработку информации о публикациях, документах архива, проектах, ученых-информатиках, коллективах, событиях, конференциях и вычислительной технике, являющихся экспонатами виртуального музея.

Основная цель создания этого виртуального музея — сохранение историко-культурного наследия, связанного с созданием и развитием информационных ресурсов в Сибири, являющихся важнейшим национальным богатством, а также обеспечение свободного повсеместного доступа к ним с целью повышения общеобразовательного и культурного уровня широких слоев населения.

ЗАКЛЮЧЕНИЕ

Обобщая все сказанное выше, можно утверждать, что виртуальный музей представляет собою компонент виртуального культурно-информационного пространства, расположенного в сети, и позволяет соби-

рать, сосредотачивать и связывать воедино разнородную информацию (текстового, графического, звукового, видео, анимационного и других форматов) по определенной тематике, как правило, не отображенной ни в одном из реально существующих музеев.

Основные критерии, которым должны удовлетворять виртуальные музеи:

1) репрезентативность и содержательность виртуальной экспозиции, исключающие искажение фактов, которые могут привести к предвзятому представлению об истории, эпохе;

2) многослойность представленной информации, подходящей для разных профессиональных, возрастных и образовательных категорий пользователей;

3) интуитивно ясный и дружественный пользовательский интерфейс.

Эти критерии, кстати, применимы и для реальных музеев, где в качестве дружественного интерфейса может выступить хороший экскурсовод.

Таким образом, «виртуальный музей не памятник, а коммуникативный очаг, обеспечивающий открытый доступ каждому человеку к новым территориям знания, опыта, выражения» [6]. При этом не стоит сбрасывать со счетов и роль представительств реальных музеев в Интернете, поскольку они, как и виртуальные музеи, служат одному делу: просвещению и обогащению народа знаниями.

Автор с благодарностью примет любые конструктивные критические замечания и предложения по обсуждаемой тематике.

СПИСОК ЛИТЕРАТУРЫ

1. **Словарь** иностранных слов. — М: Советская энциклопедия, 1964.
2. **Кононыхин Н.** Музеи в Интернет и виртуальные музеи. — http://www.russ.ru/
3. http://www.marstu.mari.ru8101
4. **Поваляев Е.** Виртуальные музеи в Интернет // КомпьютерПресс. — 2000. — № 9. — http://www.compress.ru
5. **Касьянов В.Н., Несговорова Г.П., Волянская Т.А.** Виртуальный музей истории информатики в Сибири // Современные проблемы конструирования программ. — Новосибирск, 2002. — С. 169–181.
6. **Могилевская Т.** Искусство в Интернете. Динамика в России // Взгляд с Востока. — М: MediArtLab, 2000. — С. 214–217.

Р. А.Осмонов*

МЕТОД РАСПАРАЛЛЕЛИВАНИЯ АЛГОРИТМОВ УНИМОДУЛЯРНЫМИ ПРЕОБРАЗОВАНИЯМИ

1. ВВЕДЕНИЕ

Применение оптимизирующих преобразований направлено на повышение рабочих характеристик программы. Сохранение смысла программы, т.е. эквивалентность исходной программе, является необходимым условием применения оптимизирующих преобразований.

Система оптимизирующих преобразований — это набор преобразований в совокупности со стратегией их применения.

Наибольший эффект от оптимизации достигается при применении ее к участкам повторяемости, таким как циклы, определяющие время работы программы.

В качестве программной модели берется гнездо из двух циклов с постоянными границами цикла. Нам дан фиксированный набор экземпляров тела цикла, которые следует выполнять последовательно в определенном порядке. Целью оптимизационных преобразований является определение группы экземпляров, которые могут быть распараллелены.

Метод волнового фронта (Wavefront Method) используется для выполнения гнезда цикла на параллельных и векторных компьютерах в том случае, когда ни один из циклов не может быть выполнен в векторной форме. Метод волнового фронта создает «волну», которая проходит через пространство итераций. Все итерации на линии волнового фронта исполняются параллельно. Этот метод выявляет максимум параллелизма, при этом сохраняя все отношения зависимости по данным [1].

* rafhat_iis@gorodok.net

Работа выполнена при финансовой поддержке Министерства образования РФ (научная программа «Университеты России», грант № УР.04.01.201).и Российского фонда фундаментальных исследований (грант № 04-07-90441).

2. ОПРЕДЕЛЕНИЕ УНИМОДУЛЯРНЫХ ПРЕОБРАЗОВАНИЙ

В случае с целочисленными матрицами операции сложения, вычитания, умножения, скалярного умножения на целое число и транспонирования всегда приводят к целочисленной матрице. Однако вычисление обратной матрицы для несингулярной матрицы не всегда приводит к целочисленной матрице. Важным же свойством унимодулярной матрицы является то, что обратная к ней — также целочисленна и унимодулярна [2].

Определение 1. Квадратная целочисленная матрица U — *унимодулярна*, если $|det(U)| = 1$.

Важный момент использования унимодулярных матриц состоит в том, что, переходя к новым индексным переменным в цикле, мы получаем целочисленный вектор тогда, когда исходный вектор индексных переменных целочисленный, и — наоборот. Преобразованный цикл будет состоять из тех же экземпляров операторов, но экземпляры в новом цикле будут исполняться в порядке возрастания новых индексных переменных.

Унимодулярные преобразования входят как в класс *оптимизирующих преобразований*, так и в класс *реструктурирующих преобразований* — преобразований, увеличивающих степень параллелизма в программе за счет изменения структуры программы [3].

Другим классом преобразований программ являются *несингулярные преобразования,* включающие перестановку, скашивание, обращение и масштабирование цикла. Несингулярные матрицы включают унимодулярные матрицы как частный случай. Преимуществом использования несингулярных матриц является то, что их проще сгенерировать и работать с ними, чем с унимодулярными матрицами.

Сгенерированная унимодулярная матрица может представлять собой как явное преобразование, так и совокупность преобразований в силу данных ей свойств. В случае, когда $det(U)$ элементарной унимодулярной матрицы равен –1, то она соответствует преобразованию — обращению цикла.

Определение 2. Между итерациями в гнезде существует зависимость в том случае, если есть итерации $S(i_1, i_2)$ и $S(j_1, j_2)$, такие что:

- итерация $S(i_1, i_2)$ выполняется раньше $S(j_1, j_2)$, экземпляр $H(i_1, i_2)$ исполняется до экземпляра $H(j_1, j_2)$ тогда и только тогда, когда $(i_1, i_2) < (j_1, j_2)$;
- обе итерации обращаются к одной и той же ячейке памяти, и по крайней мере, одно из обращений есть запись;

- между итерациями в ячейку памяти не заносится никакая информация [4].

Определение 3. *Индексный вектор* гнезда — это упорядоченная пара (I_1, I_2).

Определение 4. *Пространство итераций* гнезда — это набор всех возможных значений (I_1, I_2), т.е. множество точек с целочисленными координатами.

Определение 5. Если существуют две итерации (i_1, i_2) и (j_1, j_2), удовлетворяющие вышеприведенным условиям, то говорят, что в гнезде существует зависимость с *дистанционным вектором* (D_1, D_2), определяющимся как $(D_1, D_2) = (j_1 - i_1, j_2 - i_2)$.

Определение 6. В гнезде существует зависимость с *вектором направления* (s_1, s_2), если есть зависимость с любым дистанционным вектором (D_1, D_2), где $s_1 = sig(D_1)$ и $s_2 = sig(D_2)$. Обычно удобно говорить, что гнездо имеет определенный дистанционный вектор зависимости или вектор направления.

Определение 7. *Зависимость на уровне k* — это зависимость с вектором направления соответствующего вида:

- *зависимость на уровне* 1, в случае вектора направления вида $(1, s_2)$;
- *зависимость на уровне* 2, в случае вектора направления $(0, 1)$.

Структура унимодулярного преобразования включает в себя три базовых преобразования: перестановка, скос и обращение цикла.

Перестановка циклов в гнезде — это метод выявления параллелизма, являющийся также мощным средством векторной оптимизации. Применение преобразования перестановки циклов к гнезду из двух циклов соответствует изменению порядка обхода двумерного пространства итераций, а так как между итерациями могут существовать зависимости по данным, то перестановка циклов легальна, если не будут нарушены эти зависимости [4].

Скашивание цикла — это модификация формы итерационного пространства цикла, т.е. преобразование границ цикла. Применение скашивания цикла может изменять векторы направлений, предотвращающие перестановку циклов. Цикл может быть скошен фактором различной величины. Скашивание цикла в комбинации с перестановкой цикла порождает метод волнового фронта.

Обращение цикла меняет первоначальное направление обхода циклом пространства итераций на противоположное. Обращение приводит к изменению заголовка цикла с уменьшением итерационной переменной до значения нижней границы.

3. ПРОЦЕДУРА ПОСТРОЕНИЯ УНИМОДУЛЯРНЫХ ПРЕОБРАЗОВАНИЙ

Если взять в качестве элементарной матрицы единичную матрицу, также являющуюся унимодулярной, то можно получить элементарные матрицы унимодулярных преобразований. Существуют следующие классы унимодулярных матриц размерностью 2×2, которые непосредственно используются в преобразованиях:

- $\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}$ и $\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$ — обращающие матрицы;
- $\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$ — перестановочная матрица;
- $\begin{bmatrix}1 & p\\ 0 & 1\end{bmatrix}$ — множество верхних скашивающих матриц и $\begin{bmatrix}1 & 0\\ p & 1\end{bmatrix}$ — множество нижних скашивающих матриц, где p — целое число, называемое фактором искажения.

Унимодулярная матрица может быть выражена как произведение обращающей, перестановочной и верхней скашивающей матриц.

Определение 8. Гнездом циклов является множество циклов, вложенных один в другой. Гнездо называется *совершенным*, если тело каждого следующего цикла, отличного от самого внутреннего, состоит только из следующего цикла гнезда [3].

Тело цикла содержит ряд экземпляров операторов $H(I_1, I_2)$, которые исполняются в направлении увеличения индексных переменных (I_1, I_2). Каждый такой экземпляр при преобразовании описывается уникальным значением новых переменных (K_1, K_2). Преобразование двойного цикла (L_1, L_2) под действием унимодулярной матрицы U приводит к новому циклу, который состоит из тех же экземпляров H, как в (L_1, L_2), но где экземпляры исполняются в порядке возрастания (K_1, K_2).

Если взять в качестве U элементарную матрицу, то получим специальное унимодулярное преобразование. Преобразование $(L_1, L_2) \to (L_1^U, L_2^U)$ называется:

- обращением цикла, если U — обращающая матрица;
- перестановкой цикла, если U — перестановочная матрица;
- скашиванием цикла, если U — скашивающая матрица.

Скашивание внутреннего цикла внешним получается, если U — верхняя скашивающая матрица, нижняя скашивающая матрица U вызывает скашивание внешнего цикла внутренним циклом.

Условием распараллеливания может служить наличие зависимости на уровне k.

Определение 9. Внешний цикл может быть распараллелен тогда и только тогда, когда нет зависимости на уровне 1. Внутренний цикл может быть распараллелен тогда и только тогда, когда нет зависимости на уровне 2.

Если ни один из этих случаев не выполняется, тогда необходимо переупорядочить экземпляры путем замены индексных переменных. При наличии унимодулярной матрицы, позволяющей перейти к новым индексным переменным и переупорядочить экземпляры тела цикла, можно сформулировать следующие теоремы.

Теорема 1. Пусть дана 2×2 целочисленная унимодулярная матрица U такая, что преобразованное гнездо двойного цикла (L_1^U, L_2^U) эквивалентно исходному гнезду (L_1, L_2). Внешний цикл $L^U{}_1$ может быть исполнен параллельно тогда и только тогда, когда есть фиксированный целочисленный вектор $(a_1, a_2) > 0$ такой, что каждый дистанционный вектор зависимости в (L_1, L_2) имеет вид $\alpha(a_1, a_2)$, где α — положительное целое число.

Доказательство. Предположим, что существует целочисленный вектор $(a_1, a_2) > 0$ такой, что любой дистанционный вектор зависимости в (L_1, L_2) имеет вид $\alpha(a_1, a_2)$, где α — положительное целое число. Пусть $g = НОД(a_1, a_2)$. Тогда $g > 0$, так как a_1 и a_2 не могут быть оба нулевыми. Определим целочисленную матрицу 2×2

$$U = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix},$$

с $u_{11} = a_2/g$, $u_{21} = -a_1/g$ и (u_{12}, u_{22}) равно любой паре целых чисел, для которых $a_1u_{12} + a_2u_{22} = g$.

Эта матрица унимодулярная, так как

$$det(U) = (u_{11}u_{22} - u_{21}u_{12}) = (a_2u_{22} + a_1u_{12}) / g = 1.$$

Для каждого дистанционного вектора $(D_1, D_2) = \alpha(a_1, a_2)$ в (L_1, L_2) мы имеем

$$(D_1, D_2)*U = \alpha(a_1, a_2)*U = \alpha(a_1u_{11} + a_2u_{21}, a_1u_{12} + a_2u_{22}) = (0, \alpha g) >_2 0.$$

Данное выражение показывает отсутствие зависимости на уровне 1, следовательно, внешний цикл может быть распараллелен.

Теорема 2. Всегда существует унимодулярная матрица U 2×2 такая, что преобразованное гнездо (L^U_1, L^U_2) эквивалентно исходному гнезду (L_1, L_2), и внутренний цикл L^U_2 может быть исполнен параллельно.

Доказательство. Если есть дистанционный вектор зависимости (D_1, D_2) в (L_1, L_2) такой, что $D_1 > 0$ и $D_2<0$, тогда обозначим через $\mu = \lfloor max\{-D_2/D_1\} + 1 \rfloor$, где максимум берется из всех дистанционных векторов (D_1, D_2) в исходном гнезде таких, что $D_1 > 0$ и $D_2<0$. Иначе берем $\mu = 1$.

Возьмем унимодулярную матрицу

$$U = \begin{bmatrix} \mu & 1 \\ 1 & 0 \end{bmatrix}.$$

Рассмотрим знак выражения $(\mu D_1 + D_2)$ для положительного дистанционного вектора (D_1, D_2) в (L_1, L_2). Так как $\mu > 0$, знак очевидно положителен, если $D_1 > 0$ и $D_2 > 0$ или если $D_1 = 0$ и $D_2 > 0$. Знак также положителен, если $D_1 > 0$ и $D_2<0$, так как $\mu > (-D_2/D_1)$. Тогда следует результат

$$(D_1, D_2)*U = (\mu D_1 + D_2, D_1) >_1 0.$$

Отсутствие зависимости для каждого дистанционного вектора на уровне 2, позволяет распараллелить внутренний цикл.

Чтобы проверить корректность применения унимодулярного преобразования, необходимо знать дистанционные вектора исходного цикла, поскольку информации о векторах направления недостаточно в этом случае. Однако еще остается вопрос о выборе корректных преобразований, способных привести к распараллеливанию цикла, и о последовательности их применения.

Приведем некоторые свойства преобразований, позволяющие сделать такой выбор.

Перестановка циклов используется во многих случаях, например, для векторизации внутреннего цикла, если внешний цикл должен исполняться последовательно.

Применение скашивания цикла всегда корректно, поскольку оно не влияет на численные результаты программы. Скашивание также не меняет порядок выполнения итераций, итерации в скошенном пространстве итераций исполняются в том же самом порядке, как и соответствующие итерации в исходном пространстве итераций. Однако это преобразование изменяет вектора направлений: при скашивании внутреннего цикла элементы дистанционных векторов изменяются, причем положительный вектор всегда остается положительным.

Пространство итераций представляется на плоскости в виде точек, координаты которых определяют соответствующие итерации. Так, обращение внешнего цикла есть отражение индексного пространства относительно вертикальной оси координат, а обращение внутреннего цикла — отражение индексного пространства относительно горизонтальной оси координат. Значения элементов дистанционных векторов остаются неизменными по абсолютному значению, меняются только вектора направлений [5].

Цикл в преобразованном гнезде может быть исполнен параллельно тогда и только тогда, когда не существует дистанционного вектора в исходном цикле, удовлетворяющего условию $dU >_k 0$, где k — уровень вложенности цикла. Унимодулярное преобразование гнезда цикла может быть выполнено путем последовательного применения конечной последовательности элементарных преобразований, соответствующих элементарным унимодулярным матрицам.

Задачей нижеприведенного алгоритма является распараллеливание внутреннего или внешнего цикла посредством получения унимодулярной матрицы. С помощью полученной унимодулярной матрицы можно получить преобразованный цикл и проверить эквивалентность исходному. В случае эквивалентности можно определить, какой из циклов в преобразованном гнезде может быть исполнен параллельно. Важно, чтобы интервал распараллеливаемого цикла был максимальным.

4. ЭКСПЕРИМЕНТЫ И РЕЗУЛЬТАТЫ

Для экспериментов выбирались совершенные гнезда циклов двойной вложенности. Исходный код программы представлен на языке C + +. Форма записи цикла представляется выражением

«*for* ($I_1 = n_1$; $I_1 < N_1$; I_1 + +)
for($I_2 = n_2$; $I_2 < N_2$; I_2 + +)
$H(I_1, I_2)$;»,

где *i, j* — индексные переменные, n_1, N_1, n_2, N_2 — постоянные границы цикла. Алгоритм состоит из четырех основных шагов: нахождение индексных переменных и границ исходного гнезда цикла; вычисление дистанционных векторов; определение векторов направлений и зависимостей на уровне; генерация унимодулярной матрицы для распараллеливания внешнего или внутреннего цикла.

Алгоритм направлен на выявление параллелизма внешнего или внутреннего цикла. Каждый из случаев предполагает наличие соответствующих дистанционных векторов, допускающих данное распараллеливание. Множество дистанционных векторов определяется по индексам массивов, входящих в тело цикла, и записывается в виде матрицы *D*. Для распараллеливания внешнего цикла достаточно наличие дистанционных векторов, для распараллеливания же внутреннего цикла необходимо определить вектора направлений по множеству дистанционных векторов. Зависимость на уровне *k* может быть определена как по множеству векторов направления, так и по множеству дистанционных векторов. Определение зависимости на уровне *k* может однозначно указать, какой из циклов может быть распараллелен. Однако, если стоит задача о распараллеливании цикла, первоначально не подлежащего параллельному исполнению, тогда мы прибегаем к системе преобразований.

Параллельное исполнение внешнего цикла проверяется возможностью представления всех дистанционных векторов гнезда цикла в виде $\alpha(a_1, a_2)$, где $\alpha > 0$ и $(a_1, a_2) > 0$. Затем находятся *НОД*(a_1, a_2) и целые числа (x_0, y_0) такие, что $a_1x_0 + a_2y_0 = g$, по которым и вычисляются элементы унимодулярной матрицы. В случае, если матрица оказывается единичной, мы объявляем внешний цикл параллельным. В противном случае матрица включа-

ет в себя одно из преобразований перестановки, скашивания и обращения цикла или их комбинации. Прибегая к преобразованию гнезда цикла полученной унимодулярной матрицей, действуем на индексные переменные, таким способом переходя к новым переменным, при этом меняются и границы циклов. Таким образом, в случае возможности исполнения внешнего цикла параллельно, первый столбец матрицы $D * U$ получаем нулевым, согласно определению зависимости на уровне k.

Рассмотрим следующее гнездо

L_1: *for* ($I_1 = 5$; $I_1<100$; $I_1 + +$)
L_2: *for*($I_2 = 16$; $I_2<80$; $I_2 + +$)
S: $A[I_1][I_2] = A[I_1\text{-}2][I_2\text{-}4] + A[I_1 - 3][I_2 - 6];$

Есть два дистанционных вектора в гнезде: (2, 4) и (3, 6). Данные вектора показывают зависимость на уровне 1, следовательно, внешний цикл не может быть распараллелен. Заметим, что (2, 4) = 2(1, 2) и (3,6) = 3(1, 2). Так как дистанционные вектора имеют вид $\alpha(a_1, a_2)$, где $\alpha > 0$ и $(a_1, a_2) > 0$, то существует унимодулярная матрица U такая, что после того как данное гнездо преобразовано матрицей U, внешний цикл в новом гнезде может быть распараллелен (Теорема 1). Найдем $НОД(a_1, a_2) = НОД(1, 2) = 1$, и два целых числа $u_{12} = 1$ и $u_{22} = 0$ такие, что $1 * u_{21} + 2 * u_{22} = 1$. Затем вычисляются целые числа $u_{11} = a_2/НОД(a_1, a_2) = 2$ и $u_{21} = -a_1/НОД(a_1, a_2) = -1$. Получаем матрицу

$$U = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix}.$$

Преобразованный цикл:

for ($K_1 = \text{-}70$; $K_1<184$; $K_1 + +$)
for ($K_2 = \lceil max\{5,\ 8 + K_1/2\} \rceil$; $K_2<\lfloor min\{100,\ 40 + K_1/2\} \rfloor$; $K_2 + +$)
S:$A[K_2][-K_1 + 2*K_2] = A[K_2 - 2][\text{-}K_1 + 2*K_2 - 4] + A[K_2 - 3][-K_1 + 2*K_2 - 6];$

Внешний цикл K_1 в новом гнезде может быть исполнен параллельно. Дистанционные вектора (2, 4) и (3, 6) преобразованы в дистанционные вектора (0, 2) и (0, 3).

Если не удается распараллелить внешний цикл, следующим шагом мы попытаемся произвести преобразования, приводящие к параллельному исполнению внутреннего цикла, так, чтобы число итераций внешнего цикла

было минимизировано. Для этого находятся все дистанционные вектора и вектора направлений. Унимодулярная матрица генерируется путем выбора вектора направления из начального списка векторов $\{(1, 0)^T, (0, 1)^T, (0, -1)^T, (1, 1)^T, (\mu, 1)^T\}$ в качестве первого столбца унимодулярной матрицы. Условием выбора является $(D_1, D_2) * U > {}_1 0$. Если лучший выбор для первого столбца есть $(1, 0)^T$, то просто отмечаем внутренний цикл данного гнезда параллельным. В противном случае в качестве второго столбца берем вектор $(1, 0)^T$ и преобразуем исходное гнездо полученной унимодулярной матрицей.

Рассмотрим пример:

```
L1:  for (I1 = 5; I1<100; I1 + + )
L2:       for(I2 = 5; I2<100; I2 + + )
   S:          A[I1][I2] = A[I1]][I2 – 1] + A[I1 – 2][I2 + 3] + A[I1 – 3][I2 + 7];
```

Гнездо имеет дистанционные вектора (0, 1), (2, –3) и (3, –7), которые соответствуют наличию зависимости как на уровне 1, так и на уровне 2. Преобразуем его в гнездо, где внутренний цикл может быть исполнен параллельно и количество итераций внешнего цикла минимизировано. Так как гнездо имеет вектор направления (0, 1), то вектора $(1, 0)^T$ и $(0, –1)^T$ отбрасываются как возможные выборы для первого столбца U. Присутствие вектора направления (1, –1) отбрасывает заодно вектора $(0, 1)^T$ и $(1, 1)^T$.

Затем подсчитываем для всех дистанционных векторов

$$\mu \leftarrow \lfloor max\{–D_2/D_1\} + 1 \rfloor,$$

где максимум берется из всех дистанционных векторов (D_1, D_2) таких, что $D_1 > 0$ и $D_2 < 0$:

$$\mu = \lfloor max\{–(–3)/2, –(–7)/3\} + 1 \rfloor = \lfloor 7/3 + 1 \rfloor = 3$$

Добавляем вектор $(\mu, 1)^T$ в начальный список. После проведенных операций оказывается, что вектор $(\mu, 1)^T$ является лучшим выбором в качестве первого столбца унимодулярной матрицы.

Искомая матрица $U = \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix}$. Преобразованный цикл имеет вид:

for ($K_1 = 20$; $K_1 < 400$; $K_1 + +$)
for ($K_2 = \lceil max\{5, (K_1 - 100)/3\} \rceil$; $K_2 < \lfloor min\{100, (K_1 - 5)/3\} \rfloor$; $K_2 + +$)
S: $A[K_2][K_1 - 3 * K_2] = A[K_2][K_1 - 3 * K_2 - 1] + A[K_2 - 2][K_1 - 3 * K_2 + 3] + A[K_2 - 3][K_1 - 3 * K_2 + 7]$;

Внутренний цикл в новом гнезде может быть распараллелен.

Рассматривая сгенерированные матрицы, можно заметить, что при распараллеливании внешнего цикла применялись все три преобразования, а при преобразовании внутреннего — только скашивание и перестановка. Альтернатива параллельного исполнения внешнего или внутреннего цикла может исходить из условий задачи или архитектуры целевой машины. В общем, если параллелизм существует в гнезде, преобразования, рассмотренные в этой статье, найдут его.

5. ЗАКЛЮЧЕНИЕ

Данная работа обобщает простейшие случаи применения унимодулярных преобразований. Представленный метод приводит циклы к более удобному виду для применения и приспособления векторных конструкций. Вопрос о выполнении цикла в параллельной форме сводится к преобразованию, вызванному определенной унимодулярной матрицей, действующей на индексные переменные. Пригодность преобразований обращения, перестановки, скашивания цикла определяются природой проблемы. Исходной информацией о применении того или иного преобразования может явиться множество векторов направления. Однако множество всех дистанционных векторов в гнезде, очевидно, несет больше информации, чем множество всех векторов направления. В процессе преобразования не должна нарушаться структура зависимости данной программы, что и подразумевает принцип преобразования программы: значение дистанционного вектора должно оставаться положительным.

СПИСОК ЛИТЕРАТУРЫ

1. **Bacon D. F., Susan L.** Compiler transformations for High-Performance computing // ACM Computing Surveys. — Vol. 26, N 4. — 1994. — P. 345–420.
2. **Гантмахер Ф. Р.** Теория матриц. — М.: Наука, 1998.

3. **Евстигнеев В. А., Касьянов В. И.** Оптимизирующие преобразования в распараллеливающих компиляторах // Программирование. — №6. — 1996. — С. 12–26.
4. **Векторизация** программ: теория, методы, реализация. Сб. статей / Под ред. Г. Д. Чинина. — М.: Мир, 1991.
5. **Banerjee U.** Loop transformations for restructuring compilers. The foundations. — Kluwer Academic Publishers. — 1993.

К. А. Пыжов*

БЛОК РЕДУКЦИИ В КОМПИЛЯТОРЕ SISAL 3.0

ВВЕДЕНИЕ

Функциональное программирование — парадигма, значительно отличающаяся от последовательной модели программирования: функциональная программа является по сути композицией функций, в которой каждая функция сама по себе есть композиция функций или примитивных операторов (арифметические операции, условные выражения и т.д.). Это означает, что разработчик не должен задумываться над явным описанием параллельных процессов; разбиение программы на процессы определяется иерархией функций и зависимостями по данным в программе.

SISAL — функциональный язык, дающий широкие возможности для автоматического анализа и управления параллелизмом на этапе трансляции. Имена в SISAL определяют «идентификаторы» значений, а не ячейки памяти. Значения, вычисляемые и используемые в программе, являются динамическими объектами, т.е. идентификаторы связаны с конкретными ячейками памяти только во время «существования» этих значений при выполнении программы. Это определяет динамический характер графа потока данных в представлении программы, при этом вершины графа представляют операторы, а дуги — значения, передаваемые между вершинами. «Пространство существования» значения — это множество дуг, принадлежащих всем путям от определения этого значения до последнего его использования. Значения, определяемые дугами графа, могут иметь связанные с ними имена, а могут и не иметь.

Значения всех семантических элементов языка SISAL определяются только значениями формальных аргументов и вложенных выражений. Это исключает сторонние эффекты и позволяет производить более эффективный анализ программ, чем в случае императивных языков. Кроме того, поскольку SISAL является языком однократного присваи-

[0]*pyjov@ngs.ru

Работа выполнена при финансовой поддержке Министерства образования РФ (научная программа "Университеты России", грант № УР.04.01.201).

вания, он не содержит операций записи в память. В связи с этим отпадает необходимость исследования таких зависимостей по данным, как антизависимость и зависимость «по выходу».

С другой стороны, получение эффективного кода при трансляции таких программ вызывает некоторые трудности, связанные с необходимостью применения специфических оптимизаций, с задачей автоматического распределения потока управления, а также с невысокой популярностью языков «потока данных» вообще, и вследствие этого с очень небольшим по сравнению с императивными языками количеством разработанных алгоритмов и технологий трансляции. Тем не менее, языки типа SISAL вполне могут конкурировать с традиционными языками при исполнении на параллельных архитектурах ([4, 6, 7]). При этом особое внимание необходимо уделять оптимизации программы на разных уровнях.

1. ТРАНСЛЯТОР SISAL 3.0 В СИСТЕМЕ SFP

Транслятор языка SISAL, рассматриваемый в данной статье, является частью системы программирования SFP, использующей в качестве входного языка функциональный язык Sisal 3.0. Система SFP должна предоставить программисту на его рабочем месте удобную среду для разработки функциональных программ, предназначенных для последующего исполнения на ЭВМ с параллельной архитектурой. В рамках этой среды программист должен иметь возможность, с одной стороны, создавать программу без учета целевой параллельной архитектуры, а с другой — производить настройку программы на ту или иную целевую параллельную архитектуру с целью достижения высокой эффективности исполнения разработанной программы на суперЭВМ. Процедура настройки состоит в реализации оптимизирующей трансляции разработанной функциональной программы в программу на языке Си, в процессе которой программа подвергается необходимым оптимизирующим преобразованиям под управлением пользователя. Система SFP включает следующие компоненты: интерфейс, отладчик, front-end транслятор, блоки промежуточных представлений IR1, IR2 и IR3, блоки анализа, преобразования и визуализации IR1-, IR2- и IR3-программ, конверторы промежуточных представлений, back-end трансляторы. Представления IR1 и IR2 соответствуют по своим функциям и возможностям известным промежуточным представлениям IF1 и IF2 для функциональных программ, а IR3 приближено к внутреннему представлению импера-

тивных программ. Кроме того, для визуализации внутренних представлений и преобразований программ используется система визуализации иерархических графов HIGRES [3].

Блок оптимизации на представлении IR1 выполняет такие высокоуровневые преобразования, как удаление общих подвыражений, свертка констант, упрощение условных выражений, удаление «мертвого» кода и т.д. Поскольку на этапе первого внутреннего представления отсутствует информация о конкретной архитектуре и распределении потока управления, все преобразования IR1 являются редуцирующими, т. е. гарантированно не снижают эффективность программы [1]. Такие специфические преобразования, как удаление копирований, выполняются на следующем представлении IR2. Оптимизации, связанные с потоком управления, и машинно-зависимые оптимизирующие преобразования относятся к представлению IR3.

1.1. Внутреннее представление IR1

Для представления программ, написанных на языке SISAL, требуется внутреннее представление, отражающее особенности языка, такие как машинно-независимый параллелизм и функциональность. Для одного из первых трансляторов SISAL, системы OSC (Optimizing SISAL compiler [4]), был разработан язык IF1 [8]. Этот язык основывался на ациклических графах и идеально подходил для поставленной цели в рамках языка SISAL 1.0. Развитием IF1 является представление IR1, используемое в настоящий момент в системе SFP [2].

Первоначально в системе SFP использовалась реализация представления IF1, написанная на языке C++ и фактически являвшаяся частью транслятора SISAL 3.0. В свою очередь, блок оптимизации IF1 являлся частью этого представления. Однако впоследствии было принято решение реализовать блок внутреннего представления в виде COM-интерфейсов, что позволило облегчить разработку отдельных частей компилятора и повысить его надежность. В связи с этим потребовалось изменить реализацию блока оптимизаций первого внутреннего представления.

IR1, как и IF1, базируется на потоковых графах. Это наиболее удобная и подходящая форма представления SISAL-программ. Вершины графа представляют операции, операторы и некоторые стандартные конструкции; дуги представляют значения, передающиеся от вершины к вершине. Каждой дуге приписан тип значения, которое она несет. На-

пример, граф для функции

```
function F(i : integer   returns real)
 let
     a := i;
     b := 2
 in
     (a+b)/5

 end let
end function
```

в представлении IR1 имеет вид, представленный на рис. 1.

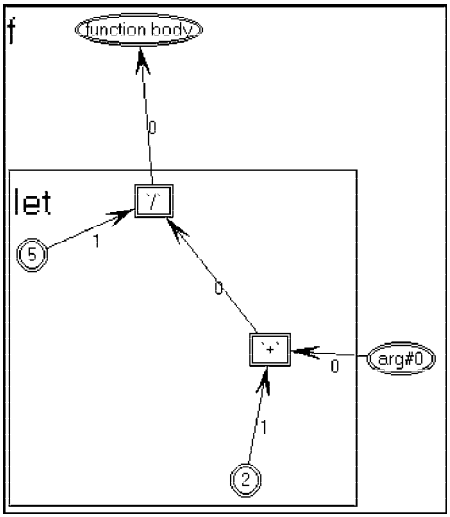


Рис. 1.

Литералы в IR1 представляются специальными вершинами. Эти вершины содержат тип литерала и значение. Каждая вершина имеет входные и выходные порты. Количество портов соответствует количеству аргументов и результатов. Функция является подграфом основного графа программы, содержащим собственно тело этой функции. Каждая функция-подграф имеет входы-аргументы и выходы-результаты.

Пример.

```
function Pythag(x, y : real   returns real)
    sqrt(x*x + y*y)
end function

function main(r : real  returns real)
   Pythag(r, 2.0)
end function
```

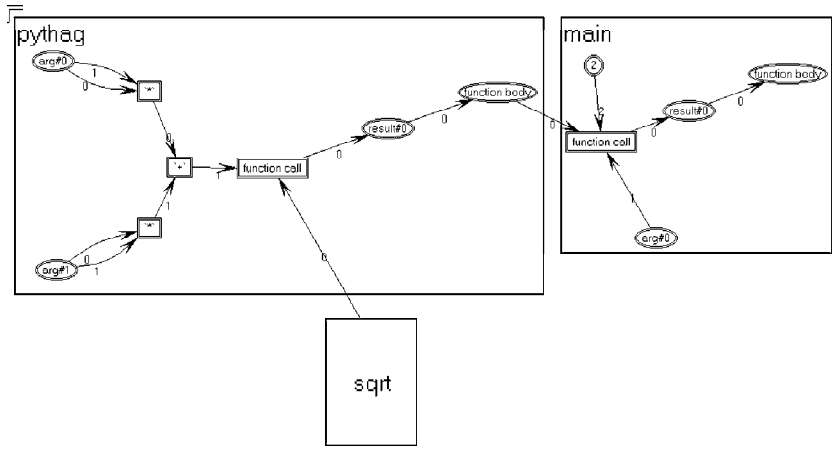


Рис. 2.

2. НЕКОТОРЫЕ ПРЕОБРАЗОВАНИЯ IR1

Как уже отмечалось, блок оптимизации выполняет некоторые редуцирующие оптимизирующие преобразования графа представления IR1. Некоторые из них являются локальными (например, свертка константных выражений), другие требуют выполнения нелокальных контекстных условий.

Подстановка параметров функции.

Это преобразование заменяет в графе внутреннего представления функции вершину-параметр на вершину-константу.

Пример.

```
function  f(x, y  : real   returns real)
  let
       s := if  x > y then x*x - y*y
                      else 0
  in
       s
  end let
end function

function first(x, r : real   returns real, real)
        f(x, 0.25),
        second(x, r)
end function

function second(x, r: real   returns real)
   f(x, r)*r
end function

function main(r: real    returns real, real)
 let
      x := 20
 in
     first(x, r)
 end let
end function
```

Внутреннее представление этого фрагмента представлено на рис. 3.

Функция first вызывается с константным значением первого параметра : 20, поэтому к ней применима подстановка параметра. Аналогичные преобразования применяются к функциям f и second. После выполнения этой редукции внутреннее представление рассматриваемого фрагмента примет вид, изображенный на рис. 4.

Редукция условного выражения.

Данное преобразование применяется к условному выражению с тождественно истинным либо тождественно ложным условием. Осуществляются анализ статической вычислимости условия и замена конструкции условного выражения на ветвь, соответствующую тождественно истинному условию.

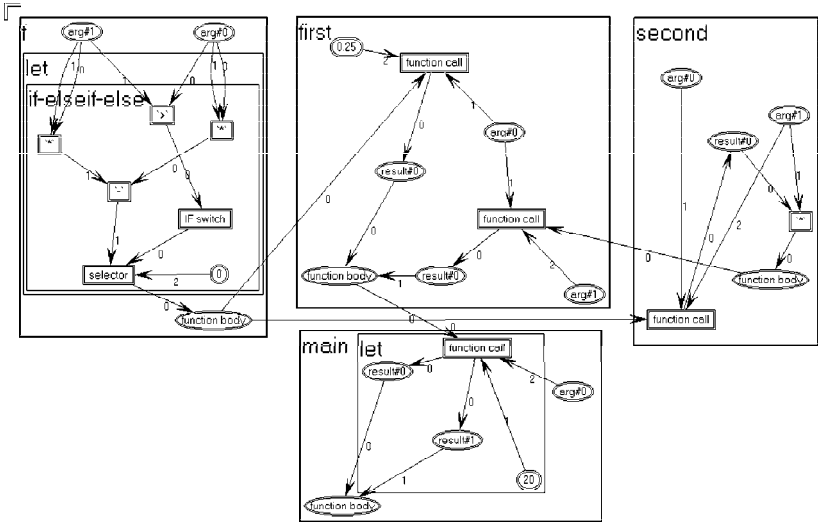

f
arg#1
arg#0
let
if-elseif-else
IF switch
selector
function body
first
0.25
function call
result#0
arg#0
second
arg#1
main
let
result#1
20

Рис. 3.

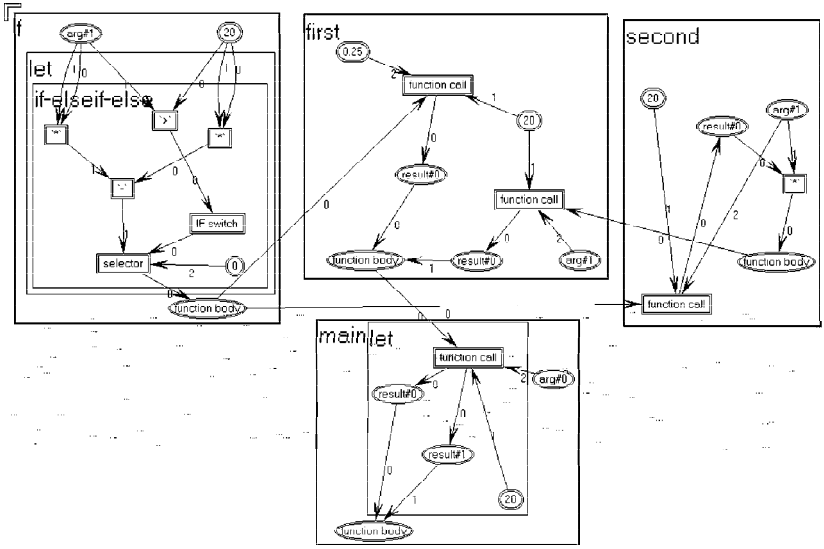

f
arg#1
20
let
if-elseif-else
IF switch
selector
function body
first
0.25
function call
result#0
second
main
let
result#1

Рис. 4.

Пример.

```
function f(x:real   returns real)                %x=0
 let
      p := 0.5 - sin(x);
      s := if p >=0 then  sqr(p)
                          else 0
           end if
in
 sin(s)  end let
end function
```

или на внутреннем представлении:

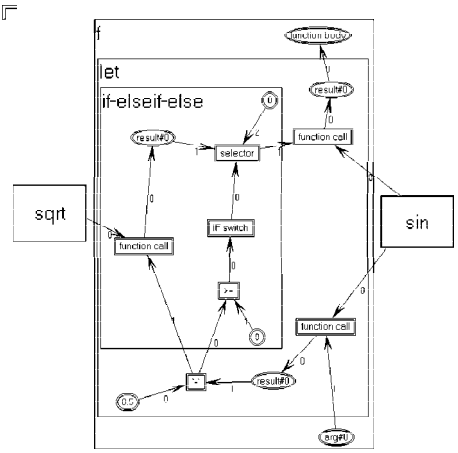


Рис. 5.

Условное выражение в присваивании переменной s имеет недостижимую else-ветвь (в контексте $x = 0$), поэтому ее можно отсечь. Фрагмент, полученный в результате такой редукции, представлен на рис. 6.

Редукция выражения CASE.

Редукция заключается в замене конструкции case на ветвь, соответствующую истинному логическому значению. Производится проход по всем условным ветвям и сравнение управляющего выражения (которое является константой) с тестовыми значениями ветвей. Если тестовыми значениями является диапазон, происходит проверка принадлежности управляющего выражения этому диапазону значений.

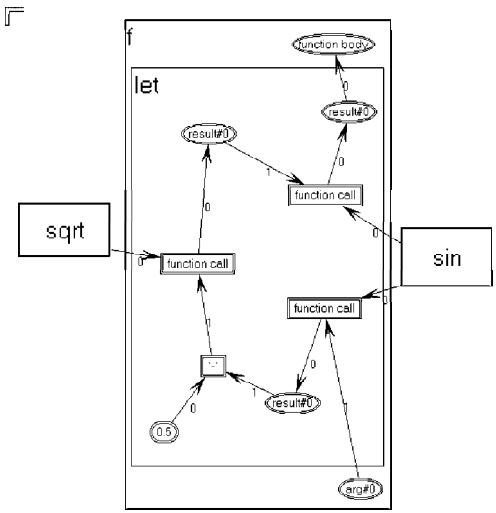


Рис. 6.

Редукция пустых циклов.

Если удается статически доказать, что выражение for ни разу не выполняет тело цикла, такой цикл удаляется.

Удаление «мертвого» кода.

В результате выполнения некоторых преобразований (например, редукции условного выражения) в графе внутреннего представления IR1 могут появиться функции, нигде не вызываемые, т.е. не связанные дугами с другими частями графа программы. Данное редуцирующее преобразование удаляет такие подграфы-функции из графа программы.

Свертка константных выражений.

Производится вычисление и подстановка значений, которые могут быть вычислены статически.

Рассмотрим следующий фрагмент программы:

```
function f(i, j: integer   returns integer)
    let
        p:=2
    in
        4*p + 3.5*i + j
```

```
        end let
    end function
```

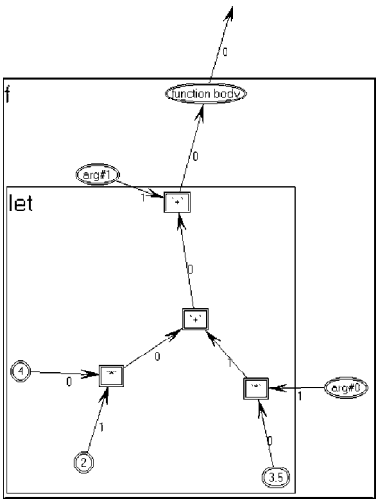


Рис. 7.

Выражение $4*p$ является в данном случае константным, поэтому его естественно вычислить на стадии трансляции. Фрагмент, полученный в результате такого преобразования, представлен на рис. 8.

Алгебраические упрощения.

Реализована редукция выражений вида E*1 и E+0, где E — любое допустимое выражение языка Sisal.

Упрощение условий.

В ряде случаев в условных операторах часть условия или все условие целиком является статически вычислимым. Такие конструкции возникают не только непосредственно из транслируемой программы, но и в результате применения других редукций, скажем, тех же константных вычислений и подстановок.

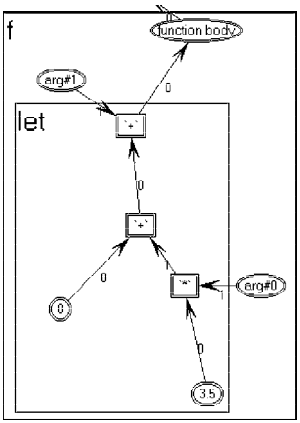


Рис. 8.

ЗАКЛЮЧЕНИЕ

Был реализован блок редуцирующих преобразований для системы SFP, включающий описанные выше оптимизирующее преобразования. Блок являлся частью реализации представления IF1, которое использовалось в первоначальной версии транслятора. Сейчас стоит задача переноса оптимизмрующих преобразований на новое представление IR1.

СПИСОК ЛИТЕРАТУРЫ

1. **Касьянов В. Н.** Оптимизирующие преобразования программ. — М.: Наука, 1986.
2. **Стасенко А. П.** Внутреннее представление системы функционального программирования SISAL 3.0. — Новосибирск, 2004. — 54 с. — (Препр. / ИСИ СО РАН; № 110).
3. **Лисицын И. А.** Применение системы HIGRES для визуальной обработки иерархических графовых моделей // Проблемы систем информатики и программирования. — Новосибирск, 1999.
4. **Sarkar V., Cann D.** POSC — a Partitioning and Optimizing SISAL Compiler. — ACM Digital Library, 1990.
5. **Simpson R.** SISAL Compiler User Manual. — Livermore, CA, 1986.

6. **Gaudiot J.-L., Bohm W., Najjar W., DeBoni T., Feo J.** The SISAL Model of Functional Programming and its Implementation. — IEEE, 1997.
7. **Gharachorloo K., Sarkar V., Hennessy J.L.** A Simple and Efficient Implementation Approach for Single Assignment Languages. — ACM Digital Library, 1988.
8. **Skedzielewski S., Glauert J.** IF1: an Intermadiate Form for Applicative Languages. — Livermore, CA, 1985.

А. И. Синяков*

АНАЛИЗ МОДУЛЬНОГО ПОДХОДА И ЕГО ПРИМЕНЕНИЕ В РАЗЛИЧНЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

ВВЕДЕНИЕ

На сегодняшний день существует несколько подходов в программировании: модульный, объектно-ориентированный, структурный. Кратко рассмотрим все эти подходы и более подробно остановимся на модульном, а во второй части рассмотрим, как реализована поддержка модулей в некоторых языках программирования.

1. МЕТОДЫ ПРОГРАММИРОВАНИЯ

1.1. Модульный подход

Модульное программирование — это искусство разбиения задачи на некоторое число подзадач, реализуемых в виде отдельных модулей, а также умение широко использовать стандартные модули путем их параметрической настройки.

Модульность является одним из основных принципов построения программных систем. В общем случае программный модуль — это отдельная функционально законченная программная единица, некоторым образом идентифицируемая и объединяемая с другими. Понятие модульности и модульного программирования появилось давно, и до сих пор основные принципы и понятия модульного подхода не устарели и лежат в основе практически всех других подходов.

Вначале основным принципом выделения модуля было разбиение алгоритма или программы на функционально замкнутые фрагменты, используемые, как правило, неоднократно. Развитие понятия модульности сказа-

* sinal@ngs.ru

Работа выполнена при финансовой поддержке Министерства образования РФ (научная программа «Университеты России», грант № УР.04.01.201).

лось в том, что модуль стал средством декомпозиции не только структур управления, но и структур данных. Такому представлению о модуле способствовало развитие понятия типа данных. Модуль понимается не только как единица компиляции и хранения, но и как единица проектирования и раздельной разработки программной системы большим коллективом разработчиков. Таким образом, модуль понимается как средство определения логически связанной совокупности объектов, средство их выделения и изоляции.

Современный подход рассматривает модульное программирование как метод декомпозиции, в основе которого лежат структуры данных. Модуль включает в себя описание структуры данных и базовые действия над ней. Развитию такого представления о модуле в значительной мере способствовало появление понятия типа данных.

Основные аспекты модульности в языках программирования, поддерживающих модульный подход, будут рассмотрены немного позже. Композиции модулей строятся на основе информации об интерфейсе модуля. При этом под интерфейсом понимается совокупность описаний и соглашений о средствах и способах передачи объектов модулей и их подобъектов. Средства и соглашения относительно задания интерфейса модуля различны, широко используется область видимости объектов модулей путем задания списков импорта и экспорта.

Сведения об интерфейсе модуля, а также характеристики его объектов, необходимые при использовании этих объектов другими модулями, и составляют спецификацию модуля в отличие от его реализации, в которой описывается представление и алгоритмы обработки, связанные с теми или иными объектами модуля.

Рассмотрим несколько подробнее программу модульного подхода, круг решаемых им проблем.

1. Изоляция определяемого понятия

Первое и наиболее очевидное назначение модуля — выделение и изоляция некоторого понятия. Поэтому модуль в том или ином языке определяется некоторой замкнутой конструкцией, имеющей уникальное имя, для которой определены некоторые правила преобразования.

2. Введение абстрактных объектов

Введенное с помощью модуля понятие можно рассматривать как объект, имеющий определенную семантику на некотором уровне абстракции, независимо от его функционального назначения: алгоритм, тип данных, механизм распределения памяти, структура данных. Природа этих объектов

несущественна, важно, что введя такое понятие, можно пользоваться им, не вдаваясь в его представление и реализацию. Модуль хорошо подходит для описания абстрактного типа данных. Под абстрактным типом данных понимается определение некоторого понятия в виде набора из одного или нескольких объектов с некоторыми свойствами и операциями. Абстрактный тип данных задается через множество допустимых операций, поэтому главной и неотъемлемой частью модуля, описывающего абстрактный тип, должны быть перечень и описание этих операций (чаще всего в виде функций). Объекты такого типа могут использоваться как аргументы допустимых типом операций и будут тем самым защищены от некорректного или несанкционированного доступа.

3. *Модуль как средство поэтапной разработки программных систем.*

Процесс разработки систем складывается из этапов, основными из которых являются этап проектирования и этап кодирования, или программирования. На первом из них производится модуляризация проектируемой системы, т.е. разбиение ее на взаимодействующие объекты, средством описания которых и являются модули. На этом этапе каждый из модулей может быть представлен только своей краткой характеристикой или спецификацией. Это дает возможность описать сразу всю систему, оставив детальное определение объектов на более поздний срок.

Коллективность разработки крупных программных продуктов усиливает требования к описанию интерфейсов модуля, к точной фиксации не только входных и выходных данных модуля, но и таких их свойств, как тип, структура, допустимые значения. Описание входных и выходных данных модуля, условий его использования, перечень действий модуля — все это принято называть спецификацией модуля. Таким образом, в процессе разработки программных систем первостепенной задачей является именно спецификация модуля.

Полное описание модуля состоит из трех разделов: спецификация, представление и реализация (как правило, раздел представления отсутствует). Разбиение на разделы обеспечивает нечувствительность одного модуля к внутренним изменения других.

В разделе *представление* описывается внутренняя спецификация модуля. Это в основном описание структур данных, используемых при представлении объектов модуля. Выделение их в отдельный раздел позволяет ограничить к ним доступ пользователя, исключает зависимость других модулей от представления объектов данного модуля.

В разделе *реализация* дается описание действий, реализуемых модулем. Например, для модуля, определяющего некоторый абстрактный тип, это описание операций, допустимых над объектами этого типа.

4. Создание проблемно-ориентированного контекста.

Проблемно-ориентированный контекст определяется множеством объектов, необходимых для задания проблемной области, в которой ставятся задачи, решаемые данной программной системой.

5. Локализация машинной зависимости программ.

Одно из применений модульной концепции — локализация машинной зависимости внутри только отдельных модулей. Этим достигается возможность быстрой подмены машинно-зависимых фрагментов программы при переносе ее на другую платформу.

1.2. Структурный подход

Структурный подход базируется на систематическом использовании абстракций для управления массой деталей и способе документирования, который помогает проектировать программу.

В структурном подходе важна форма и дисциплина. Она нацелена на повышение продуктивности программирования за счет увеличения надежности программ и облегчения их модификации. Одним из главных способов повышения надежности является улучшение структуры программы, что позволяет понимать, сопровождать и модифицировать программу без участия авторов.

Составными частями структурного подхода являются: нисходящая разработка, структурное программирование, сквозной структурный контроль.

1. Нисходящая разработка.

Традиционно проектирование программной системы делается сверху вниз. При использовании метода нисходящей разработки детальное проектирование программ, кодирование, отладку и документирование можно делать параллельно. Нисходящая разработка призвана уменьшить сложность программы.

2. Элементы структурного программирования.

Одним из основополагающих принципов структурного программирования является то, что все создаваемые программы и их фрагменты должны быть структурированными, иначе говоря, подчиняться таким правилам композиции конструкций языка, чтобы процесс исполнения этих конструк-

ций был бы легко видим и понимаем по их текстуальному представлению: структура текста должна соответствовать структуре исполнения. Естественное требование для этого — это требование того, чтобы композиция осуществлялась такими конструкциями, исполнение которых имеет одну входную точку в тексте (с которой начинается исполнение конструкции), и одну выходную точку (исполнением которой завершается исполнение конструкции). Большинство управляющих конструкций языков построено именно таким образом. Структурированная программа не может содержать операторы перехода в произвольную точку текста: в таком случае структура текста уже не будет соответствовать структуре возможного исполнения.

3. Сквозной структурный контроль.

Сквозной структурный контроль является неотъемлемой частью структурного подхода и регламентирует организационную и контролирующую деятельность руководителей проекта. Термин «сквозной» указывает на способ проверки — все контролируемые элементы выполняются шаг за шагом.

Термин «структурный» подчеркивает, что контроль является составной частью рабочего цикла, он заранее предопределен и продуман.

Контроль ориентирован на то, чтобы обнаружить ошибки в принятых решениях и создать атмосферу, при которой сам проектировщик и другие члены проекта стремились бы найти ошибки как можно раньше.

Сквозной контроль осуществляется на всех этапах разработки, начиная с этапа определения требований.

1.3. Объектно-ориентированный подход

В основе объектно-ориентированного подхода лежит рассмотрение объекта в качестве главного понятия при решении всего круга проблем, связанных с разработкой сложных систем. Объектно-ориентированный подход (ООП) является результатом эволюционного процесса развития методологий, в частности, модульного и структурного подходов. В отличие от структурного подхода ООП подразумевает другой взгляд на процесс декомпозиции разрабатываемой системы. Как и другие парадигмы, ООП затрагивает процессы проектирования и программирования, но в отличие от других подходов он распространяется и на такие виды деятельности и типы обработки информации, как проектирование пользовательского интерфейса и архитектуры компьютеров, базы данных и базы знаний.

Главными компонентами объектного подхода является абстрагирование, ограничение доступа, модульность и иерархия.

Абстрагирование — это выделение таких существенных характеристик некоторого объекта, которые отличают его от всех других видов объектов и таким образом четко определяют особенности данного объекта с точки зрения дальнейшего его рассмотрения и анализа.

Ограничение доступа — это процесс защиты отдельных элементов объекта (структуры объекта, реализации его методов).

Модульность — это свойство системы, определяемое возможностью декомпозиции ее на ряд тесно связанных модулей. Модули выполняют роль физических контейнеров, в которые помещаются определения классов и объектов при проектировании системы.

Абстрагирование, ограничение доступа и модульность являются средствами упрощения сложных систем. Иерархия является эффективным механизмом сокращения сложности. В ООП структура классов является иерархией по номенклатуре, а структура объектов — иерархией по составу. Основным видом иерархии по номенклатуре является наследование. Оно означает такое соотношение между классами, когда один класс использует структурную или функциональную часть одного или нескольких классов (простое и множественное наследование).

Любой язык объектно-ориентированного программирования характеризуется тремя основными следующими свойствами.

1. *Инкапсуляция*: определение объекта путем не только описания его структуры, но и всего множества операций с ним.
2. *Наследование*: возможность построения иерархии порожденных объектов благодаря предоставлению доступа каждого из порожденных объектов к коду и данным предка.
3. *Полиморфизм*: возможность идентифицировать одним и тем же именем множество аналогичных операций (действий), аргументами которых являются разные объекты некоторой иерархии объектов.

Объектно-ориентированное программирование по существу включает в себя все понятия модульного программирования. К ним относится и понятие межмодульного интерфейса. Каждый модуль определяется как пара — определяющий раздел (или интерфейс) и реализующий раздел.

2. СРЕДСТВА МОДУЛЬНОСТИ В ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

Рассмотрим более подробно особенности модулей в современных языках программирования и их использование.

2.1. Язык Модула 2

Модула 2 — язык программирования общего назначения, разработанный, прежде всего, для реализации систем программирования. Основывается на языках Паскаль и Модула.

Модуль предназначен для отделения спецификации от реализации, раздельной компиляции, определения новых типов и операций над ними, реализации низкоуровневых операций.

Модуль можно представить парой: определяющий и реализующий модули. Они имеют одно и то же имя. Определяющий модуль задает спецификацию доступных извне объектов модуля: констант, типов, переменных, процедур. Перечень этих объектов, кроме того, задается в списке экспорта данного модуля. Объекты, используемые в данном модуле и внешние по отношению к нему, перечисляются в списке импорта.

Синтаксис определяющего модуля имеет следующий вид:

```
definition module <имя модуля>;
        <список импорта>;
        <список экспорта>;
        <спецификация объектов>;
end <имя модуля>
```

Реализующий модуль содержит описание локальных объектов модуля и полное описание процедур и функций, специфицированных в соответствующем определяющем модуле. Все объекты, описанные в определяющем модуле (в том числе импортируемые), доступны в его реализующем модуле. Реализующий модуль может иметь тело-блок, состоящий из операторов. Если модули вложены в процедуру, а не являются единицами компиляции, то их тела исполняются в порядке следования модулей и служат для инициализации локальных переменных этой процедуры.

Синтаксис реализующего модуля таков:

```
implementation module < имя модуля >;
          <описание процедур и функций> <тело модуля>;
end < имя модуля >;
```

Определяющий модуль является интерфейсным по отношению к реализующему и использующим его модулям.

Область действия модуля определяется уровнем его описания. Если модуль задан независимо, т.е. как единица компиляции, его можно использовать в любом месте программы. Если же модуль описан внутри процедуры, он локален в теле этой процедуры. Внешние объекты модуля видимы в той же области, что и сам модуль. Внешними (доступными извне) становятся объекты, специфицированные в модуле и перечисленные явно в списке его экспорта. Модуль может потребовать уточненного (qualified) использования экспортируемых объектов заданием списка экспорта в виде

export qualified <список имен>

При использовании имена этих объектов должны предваряться именами данного модуля. Уточненный экспорт необходим при разработке модулей, которые придется использовать совместно с другими модулями, заранее не известными. Он позволяет избежать конфликтов имен объектов, экспортированных из разных модулей. В модуле возможен лишь один список экспорта, поэтому все экспортируемые объекты могут быть либо уточнены, либо нет.

Помимо описанных в модуле объектов в нем доступны также объекты, экспортируемые другими модулями, либо видимые на уровне описания модуля, когда модуль описан в некоторой процедуре. Такие объекты перечисляются в списке импорта. Объекты, импортируемые из модуля *M* и уточненные в нем, должны использоваться с префиксом *M*. Префикс можно упомянуть лишь в списке импорта, указав этим связь импортируемых объектов с именем их модуля, например,

from M *import* S_1, S_2

Рассмотрим пример модуля для работы с памятью. Пусть в программе описаны

```
type T = …;
var УК : pointer to T;
```

Операторы *new*(УК) и *dispose*(УК), служащие для создания объекта и отказа от него, будут переведены транслятором в процедуры

```
ALLOCATE(УК, TSIZE(T))
DEALLOCATE(УК, TSIZE(T)),
```

где первый параметр — указатель на размещаемый или убираемый объект, а второй — его длина.

Вместо стандартного модуля ПАМЯТЬ, где определены эти процедуры, можно описать свой модуль ПАМЯТЬ, определяющая часть которого имеет вид:

```
definition module ПАМЯТЬ;
   from SYSTEM import ADDRESS;
   export qualified ALLOCATE, DEALLOCATE, РЕЖИМ;
   procedure ALLOCATE(var УК: ADDRESS; ДЛИНА: CARDINAL);
   procedure DEALLOCATE(var УК: ADDRESS; ДЛИНА: CARDINAL);
   procedure РЕЖИМ (РЖ: CARDINAL);
      (* РЖ=1: когда не хватает свободного места,
                  ALLOCATE обрывает работу (по умолчанию);
               2: когда не хватает свободного места, ALLOCATE выделяет nil *)
                     . . .
end ПАМЯТЬ;
```

Аналогично можно определить системные или пользовательские модули управления процессами, работы с файлами и т.п.

2.2. Язык Ада

Язык Ада — универсальный язык программирования высокого уровня. Прежде всего, Ада используется для построения больших систем, к которым предъявляются достаточно высокие требования по надежности.

Пакет (модуль) — это средство, которое позволяет сгруппировать логически связанные вычислительные ресурсы и выделить их в единый самостоятельный программный модуль, реализация которого защищена от пользователя. Под вычислительными ресурсами в этом случае подразумеваются данные (типы данных, переменные, константы и т.д.) и подпрограммы, которые манипулируют этими данными.

Пакет в общем случае задается парой: спецификация и тело. Обе части имеют один и тот же идентификатор.

Спецификация определяет интерфейс с вычислительными ресурсами (сервисам) пакета, доступными для использования во внешней, по отношению к пакету, среде. Другими словами — спецификация показывает, что доступно при использовании этого пакета.

Тело является приватной частью пакета и скрывает в себе все детали реализации предоставляемых для внешней среды ресурсов, т.е. тело хранит информацию о том, как эти ресурсы устроены.

```
<спецификация модуля> ::=
    package <имя модуля> is
        <видимая описательная часть>
        <приватная описательная часть>
    end <имя модуля>

<тело модуля> ::=
    package body <имя модуля> is
        <описательная часть>
        begin <последовательность операторов>
        exception <реакция на исключительные ситуации>
    end <имя модуля>
```

Тело пакета может отсутствовать, если он предназначен только для описания данных или новых типов.

Если в видимой части пакета специфицированы функции или другой пакет, то тела их должны входить в описательную часть тела этого пакета. Здесь же могут быть описаны локальные объекты пакета, необходимые для реализации внешних объектов.

В конце пакета или его компонента может задаваться реакция на исключительные ситуации в виде:

```
exeption
    when <имена ситуаций> <операторы>
    when others <операторы>
```

Операторы обработки реакции имеют доступ ко всем данным компонента, в частности, к параметрам функции, и могут выполнять от имени этой функции оператор возврата.

Область видимости объектов пакета за его пределами экранируется охватывающей конструкцией (блок, программа, другой пакет). В то же время объекты, объявленные в объемлющих программных сегментах, видимы внутри данного сегмента.

Ограничить доступ можно предложением ограничения видимости (restricted), запрещающим видимость всех объектов некоторого сегмента либо разрешающим видимость указанных объектов.

Доступ к объектам видимого пакета осуществляется через составное имя:

```
<имя пакета>.<имя объекта>
```

Такой способ еще называют *использование объектов как поименованных компонентов*.

Все внешние объекты пакета можно сделать непосредственно видимыми (указанием только их имени), если в области видимости этого пакета поместить декларацию использования:

use <имя пакета>

Пример:

```
package RATIONAL NUMBERS is
    type RATIONAL is
    record
        NUMERATOR : INTEGER;
        DENOMINATOR : INTEGER range 1..INTEGER'LAST;
    end record;
    function "=" (X,Y : RATIONAL) return BOOLEAN;
    function "+" (X,Y : RATIONAL) return RATIONAL;
    function "*" (X,Y : RATIONAL) return RATIONAL;
package body RATIONAL_NUMBERS is
    procedure SAME_DENOMINATOR (X,Y : in out RATIONAL) is;
    begin
        --Задается приведение X и Y к общему знаменателю
    end;
    function "=" (X,Y : RATIONAL) return BOOLEAN is U,V: RATIONAL;
    begin
        U := X; V := Y; SAME_DENOMINATOR (U,V);
        return U.NUMERATIOR = V.NUMERATOR;
    end "=";
    function "+" (X,Y : RATIONAL) return RATIONAL is
        . . .
    end "+";
    function "*" (X,Y : RATIONAL) return RATIONAL is
        . . .
    end "*";
end RATIONAL_NUMBERS;
```

2.3. Язык Legos

Язык Legos был разработан на основе языка Паскаль для модульных систем программирования.

Модуль служит для структурирования программы, а также для указания связей между отдельными ее компонентами. Каждый модуль компилируется отдельно в контексте, создаваемом объектами, объявленными в списке видимых модулей.

В зависимости от роли в программе выделяют три типа модулей: простой, структурный и интерфейсный.

Простой модуль описывается конструкцией

module <имя модуля> *use* <имена видимых модулей>
<список деклараций> <список команд>
end <имя модуля>

В списке с ключом use перечисляются имена модулей, используемых данным, они составляют контекст для компиляции.

Структурный модуль создается с целью защитить некоторые внутренние возможности программы от модификаций. Он состоит из программы, которую необходимо закрыть, и ассоциированного с ней интерфейса. Заголовок структурного модуля имеет вид:

module <имя модуля> = K : R,

где K — специфицирующий (или интерфейсный) модуль; R — реализующий модуль.

Модуль интерфейса, являясь специфицирующим модулем, делает доступными извне объекты, реализуемые другими модулями, и определяет правила доступа к ним. Заголовок модуля интерфейса имеет следующий вид:

module <имя модуля> *of* <список имен модулей>

Модуль реализации должен содержать в своем use-списке имя своего специфицирующего модуля.

В модуле *A*, использующем модуль *B* (module *A* use *B*), могут применяться все объекты, определенные в модуле *B*.

При желании скрыть те или иные объекты модуля *B* (в различных его использованиях), может быть описано несколько его интерфейсных модулей $B_1 \ldots B_n$, содержащих спецификации только нужных объектов. Нужный для *A* интерфейс модуля *B* указывается в его списке use (т.е. module *A* use B_i). При этом видимыми становятся все специфицированные в B_i объекты.

Модули могут объединяться в конструкции, называемые программами и являющимися средством формирования контекста из объектов нескольких модулей:

program <имя программы> *of* <список имен модулей>

Для каждой такой программы можно описать один или несколько модулей интерфейса. Каждый модуль интерфейса тогда предоставляет особую видимость программы.

В то же время один и тот же модуль интерфейса может использовать различные программы (например, если каждая из них является специфической реализацией объектов, определенных в интерфейсном модуле).

Делая сборку программы, можно одновременно сопоставлять модуль и его интерфейс для данной программы. Например,

program P *of* $J_1 : M_1, J_2 : M_2, M_3$

Пример. Описание модулей для выделения и освобождения памяти в «куче».

Модуль спецификации:

```
module allocator;
    action
    allocate = procedure (inout p: pointer, size: integer);
    (* предоставление памяти длины size, заданной указателем p *)
    deallocated = procedure (inout p: pointer; size: integer);
    (* освобождение памяти длинны size с адреса p *)
end allocator;
```

Модуль реализации:

```
module allocreal use allocator, errormodule
    const m = 100;
    n = 1000;
    var first, last: array [1..m] of integer, current: integer;
    heap: array [1..n] of unspecified;
    body allocate;
        var i, j: integer;
        begin
            . . .
```

```
    end (* allocate *)
    body deallocate;
        var i, j: integer;
        begin
            . . .
    end (* deallocate *)
    begin (* инициализация *)
        current := 1;
        first [current] := 1;
        last [current] := 1000;
end allocreal;
```

2.4. Язык Clu

Основным побудительным мотивом разработки языка Clu явилась необходимость поддержки абстракции данных. Использование абстракций данных ведет к объектно-ориентированному стилю программирования, при котором данные рассматриваются как главные структуры, и это определяет организацию структуры программы. Программа на языке Clu состоит из группы модулей трех видов: процедур, итераторов (абстракция управления), кластеров (абстракция данных).

Особенность процедур языка Clu заключается в том, что в заголовке явно специфицируются исключительные условия окончания, например:

```
square_root = proc (x : real) returns (real)
    signals (no_real-result)
```

Итератор — это модуль, поставляющий последовательность элементов по их аргументам. Итератор вызывается оператором for. Поставляемый итератором элемент присваивается переменной цикла, выполняется тело цикла, управление возвращается итератору, поставляющему следующий элемент и т. д. Цикл заканчивается с завершением итератора.

Кластер используется для спецификации и реализации новых типов данных. Он реализует абстракцию данных, которая представляет собой набор объектов и элементарных операций создания этих объектов и манипулирования ими. Кластеры и итераторы могут содержать локальные собственные переменные, сохраняющие значения от вызова к вызову, но не доступные извне.

Все три вида модулей языка Clu допускают параметры. Параметризация дает возможность определять целый класс абстракций средствами одного

модуля. Параметр задается именем и типом. Помимо стандартных типов вводится тип type, означающий, что формальному параметру может соответствовать фактический параметр любого типа и модуль обеспечивает соответствующие операции с параметром произвольного типа.

Определение модуля имеет вид

имя = $\left\{\begin{matrix} proc \\ iter \\ cluster \end{matrix}\right\}$ [<параметры>] *is* <список допустимых операций>
[*where* <спецификация операций>]
<раздел представления>
<раздел описания подпрограмм>
end <имя>

Факультативная спецификация where для модуля, имеющего параметр типа type, описывает требования к соответствующим этому параметру фактическим параметрам, выраженные в терминах ограничений на операции.

Поскольку кластер описывает абстрактный тип, в его теле содержится раздел «представление» (rep), в котором в терминах некоторого конкретного типа дается описание конкретного представления этого абстрактного типа. Оно должно быть единственным для данного кластера, но может быть параметризованным.

Операции в кластере описываются в терминах конкретного представления, недоступного пользователям. Каждая операция, определяемая кластером, реализуется с помощью подпрограммы (процедуры или итератора).

Модули языка Clu никогда не вкладываются друг в друга. Программа является одноуровневой структурой, в которой каждый модуль может быть использован в любом модуле программы. Проверка типа и межмодульный интерфейс выполняются на основе информации, имеющейся в заголовке модуля, а в случае кластеров — еще и в заголовках процедур и итераторов, реализующих операции.

Каждый модуль представляет собой отдельную текстуальную единицу и компилируется независимо от других модулей.

Основными элементами семантики языка Clu являются объекты и переменные. Объектами называются единицы данных, создаваемые и обрабатываемые программами. Каждый объект имеет тип, определяющий набор элементарных операций по созданию и обработке объектов данного типа. Объект может быть создан и обработан только через эти операции.

Память под объекты выделяется динамически. Теоретически объекты существуют на протяжении всего времени исполнения программы.

В отличие от многих других языков программирования переменные в Clu не имеют значений, а используются только для ссылок на конкретные объекты во время выполнения (в частности, две переменные могут ссылаться на один и тот же объект).

Пример. Реализация типа данных «комплексное число», представимое в кластере координатами x и y.

```
complex = cluster is create, add, get_x, get_y, equal
    rep = struct [x, y: real]
    add = proc (a, b: cvt) returns (cvt)
        signals (overflow, underflow)
        return (rep $ {x: a.x+b.x, y: a.y+b.y})
        resignal overflow, underflow
    end add
    get_x = proc (c: cvt) returns (real)
        signals (overflow, underflow)
        return (c.x)
    end get_x
    get_y = proc (c: cvt) returns (real)
        signals (overflow, underflow)
        return (c.y)
    end get_y
    . . .
end complex
```

2.5. Язык Симула 67

Симула 67 — универсальный язык программирования, являющийся расширением языка Алгол 60. При его создании стремились устранить противоречия между универсальностью и специфическими требованиями к языкам для различных проблемных областей.

Понятию модуль в языке Симула 67 эквивалентно понятие класс. Центральным является понятие объект. Объект — это экземпляр блока, имеющий собственные локальные данные и действия, описанные в декларации класса. Декларация класса, не содержащая никаких действий, определяет структуру данных. Объекты создаются динамически в процессе выполнения программы.

Имя класса может быть префиксом в декларации другого класса, тогда последний становится подклассом первого. Префиксация означает, что все понятия и префиксы, определенные в префиксном классе, становятся доступными для объектов класса, имеющего этот префикс. Каждый из классов может иметь последовательность префиксов произвольной глубины. Возможности языка можно расширять, описывая так называемые системные классы. Для некоторой прикладной области можно создать системный класс, описывающий присущие ей понятия, методы и свойства.

Декларация класса имеет вид:

```
<префиксы>  class <имя класса>
              <совокупность формальных параметров>
              <спецификация параметров>
              <спецификация виртуальных величин>
              <тело класса>
            end <имя класса>
```

Декларация класса с префиксом C и идентификатором класса D (C *class* D) определяет некоторый подкласс D класса C. В общем случае, если C_1, C_2,…, C_n — классы, причем C_1 не имеет префикса, а C_k имеет префикс C_{k-1}(k = 2, 3,…), то индекс «k» называется префиксальным уровнем класса C_k; C_i является подклассом класса C_j при i > j, а префиксальный уровень подкласса C_i считается выше префиксального уровня класса C_j.

Объекты модуля называются атрибутами класса.

Рассмотрим в качестве примера модуль, описывающий составное понятие «интегрирование по Гауссу»:

```
class Гаусс(n); integer n;
   begin array W, X [1 : n];
      real procedure интеграл (F, a, b);
      real procedure F : real a, b;
      .  .  .
   end Гаусс;
```

Если переменные G5, G10 описать в виде ссылки на класс Гаусс

ref (Гаусс) G5, G10;

то после выполнения операций

G5 : — *new* Гаусс (5);
G10 : — *new* Гаусс (10)

их значениями будут объекты класса Гаусс. Тогда G5.интеграл (<параметры>) и G10. интеграл (<параметры>) называются дистанционными идентификаторами, имеющими атрибут класса.

Понятие класса равносильно понятию абстрактных типов данных. Объект конкретного типа создается операцией генерации объекта, имеющей вид

new <имя класса> (<список фактических параметров>)

Ниже представлен пример описания класса, выражающего понятие «расстановка» (функция расстановки). Процедура «оценка» объявлена виртуальной и может быть подменена другой процедурой.

```
class расстановка (n); integer n;
   virtual : integer procedure оценка;
   begin integer procedure оценка (T);
      value T; text T;
      . . .
   end оценка;
. . .
расстановка class АЛГОЛ оценка;
   begin integer procedure оценка (T);
      value T; text T;
   end (АЛГОЛ оценка);
```

Механизм виртуальных величин позволяет реализовать механизм инкапсуляции и подмены реализации объектов.

2.6. Язык Оберон 2

Оберон 2 — язык программирования общего назначения, продолжающий традиции языков Паскаль и Modula 2. Его основные черты — блочная структура, модульность, раздельная компиляция, статическая типизация со строгим контролем соответствия типов (в том числе межмодульным), а также расширение типов и связанные с типами процедуры.

Модуль — совокупность объявлений констант, типов, переменных и процедур вместе с последовательностью операторов, предназначенных для присваивания начальных значений переменным. Модуль представляет собой текст, который является единицей компиляции.

```
module = MODULE indent ";" [ImportList] DeclarationSequence
        [BEGIN StatementSequence] END ident "."
ImportList = IMPORT import {"," import} ";"
Import = ident [":=" ident]
```

Список импорта определяет имена импортируемых модулей. Если модуль *A* импортируется модулем *M* и *A* экспортирует идентификатор *x*, то *x* упоминается внутри *M* как *A.x*. Если *A* импортируется как *B:=A*, объект *x* должен вызываться как *B.x*. Это позволяет использовать короткие имена-псевдонимы в уточненных идентификаторах. Модуль не должен импортировать себя. Идентификаторы, которые экспортируются (т.е. должны быть видимы в модулях-импортерах) нужно отметить экспортной меткой в их объявлении.

Последовательность операторов после символа BEGIN выполняется, когда модуль добавляется к системе (загружается). Это происходит после загрузки импортируемых модулей. Отсюда следует, что циклический импорт модулей запрещен. Отдельные (не имеющие параметров и экспортированные) процедуры могут быть активированы из системы.

Оберон 2 поддерживает динамическую загрузку модулей. Загруженный модуль может вызывать команду незагруженного модуля, задавая ее имя как строку. Специфицированный модуль при этом динамически загружается и выполняется заданная команда. Динамическая загрузка позволяет пользователю запустить программу как небольшой набор базисных модулей и расширять ее, добавляя последующие модули во время выполнения по мере необходимости.

Модуль M_0 может вызвать динамическую загрузку модуля M_1 без того, чтобы импортировать его. M_1 может, конечно, импортировать и использовать M_0, но M_0 не должен знать о существовании M_1. M_1 может быть модулем, который спроектирован и реализован намного позже M_0.

Пример:

```
MODULE Trees; (* экспорт: Tree, Node, Insert, Search, Write, Init *)
   IMPORT Texts, Oberon; (* экспорт только для чтения: Node.name *)
   TYPE
      Tree* = POINTER TO Node;
      Node* = RECORD
         name-: POINTER TO ARRAY OF CHAR;
         left, right: Tree
```

```
        END;
    VAR w: Texts.Writer;
    PROCEDURE (t: Tree) Insert* (name: ARRAY OF CHAR);
        VAR p, father: Tree;
    BEGIN p := t;
        REPEAT father := p;
            IF name = p.name^ THEN RETURN END;
            IF name < p.name^ THEN p := p.left ELSE p := p.right END
        UNTIL p = NIL;
        NEW(p);
        p.left := NIL;
        p.right := NIL;
        NEW(p.name, LEN(name)+1);
        COPY(name, p.name^);
        IF name < father.name^ THEN father.left := p ELSE father.right := p END
    END Insert;
    PROCEDURE (t: Tree) Search* (name: ARRAY OF CHAR): Tree;
        VAR p: Tree;
    BEGIN p := t;
        WHILE (p # NIL) & (name # p.name^) DO
            IF name < p.name^ THEN p := p.left ELSE p := p.right END
        END;
        RETURN p
    END Search;
    PROCEDURE (t: Tree) Write*;
    BEGIN
        IF t.left # NIL THEN t.left.Write END;
        Texts.WriteString(w, t.name^);
        Texts.WriteLn(w);
        Texts.Append(Oberon.Log, w.buf);
        IF t.right # NIL THEN t.right.Write END
    END Write;
    PROCEDURE Init* (t: Tree);
    BEGIN NEW(t.name, 1);
        t.name[0] := 0X;
        t.left := NIL;
        t.right := NIL
    END Init;
BEGIN Texts.OpenWriter(w)
END Trees.
```

2.7. Язык Perl

Perl — интерпретируемый язык, приспособленный для обработки произвольных текстовых файлов, извлечения из них необходимой информации и выдачи сообщений. Perl также удобен для написания различных системных программ. Этот язык прост в использовании, эффективен, но про него трудно сказать, что он элегантен и компактен. Perl сочетает в себе лучшие черты C, shell, sed и awk, поэтому для тех, кто знаком с ними, изучение Perl не представляет особого труда. Синтаксис выражений Perl близок к синтаксису C. Хотя Perl приспособлен для сканирования текстовых файлов, он может обрабатывать также двоичные данные и создавать .dbm файлы, подобные ассоциативным массивам. Perl позволяет использовать регулярные выражения, создавать объекты, вставлять в программу на C или C++ куски кода на Perl, а также позволяет осуществлять доступ к базам данных, в том числе Oracle.

Хотя в языке Perl и существует понятие модуль (package), синтаксис соответствующей конструкции и, вообще, само понятие определить четко (в виде РБНФ, например) в общем случае довольно сложно. Дело в том, что, в отличие, скажем, от языков с традиционно развитыми средствами модульного программирования, вроде Modula и Oberon, в которых модуль — это статическая конструкция, где каждый элемент имеет свое, четко определенное назначение и место, в Perl модуль — это, скорее, даже не законченная синтаксическая конструкция, а некий конгломерат объявлений, функций, операторов, выражений, директив импорта и тэгов.

Не следует думать, что модуль — это обязательная конструкция для оформления программ на Perl. Если о языке Oberon можно сказать, что любая программная система, написанная на нем, есть совокупность модулей, то в отношении языка Perl такого сказать нельзя: программная система на нем в общем случае состоит из модулей, классов (рассматриваемых как некая разновидность модуля) и скриптов (файлов, содержащих тело модуля без заголовка).

Синтаксис оформления модуля на Perl выглядит следующим образом:

```
package SomeModule;
  # Все, что следует после заголовка модуля, считается телом модуля,
  # которое оканчивается заголовком другого модуля или концом файла
  # Можно, хотя это необязательно, в конце модуля поставить тэг __END__
  # или тэг __DATA__, после которых и до конца файла любой текст будет
  # проигнорирован компилятором.
```

В модулях Perl синтаксисом не предусмотрены, как это сделано в Modula и Oberon, специальные разделы для описания списков импорта и экспорта, типов, констант, переменных, функций, инициализации. Блоки инициализации и завершения, директивы импорта, операторы и выражения, описания функций могут следовать в любом порядке, практически без ограничений. Переменные могут использоваться до их объявления и инициализации, а функции — до их описания, и такие ситуации не контролируются компилятором — все они обрабатываются на этапе выполнения.

Perl допускает множественное определение функции с одним и тем же именем в области видимости одного модуля. При вызове функции по этому имени управление будет передано той, что была описана последней, а сообщение об ошибке не появится ни на этапе компиляции, ни на этапе выполнения. Данная "особенность", совместно с неспособностью Perl проконтролировать наличие вызываемой функции на этапе компиляции, способна несколько осложнить отладку.

Данную ситуацию не следует путать с механизмом перегрузки функций в стиле C++: существование данного механизма, основанного на различиях в описании принимаемых перегруженными функциями параметров, исключено в Perl, поскольку средства описания функций Perl слишком примитивны.

В Perl существует несколько имен, которые имеют предопределенное назначение и используются интерпретатором во время выполнения для тех или иных целей. Здесь рассмотрим три функции, предназначенные для описания: `BEGIN`, `END` и `AUTOLOAD`. Поскольку ключевое слово `sub` при этом указывать необязательно, то они часто называются не функциями, а блоками.

Блоки `BEGIN` и `END` — это соответственно блоки инициализации («конструкторы») и завершения («деструкторы») модуля. «Конструкторы» выполняются сразу же, как только они обработаны компилятором, даже не дожидаясь разбора остальной части модуля. Это, как правило, происходит на этапе загрузки модуля в память при запуске программы (если модуль импортирован директивой `use`) или при динамической компиляции и загрузке в память на этапе выполнения программы (если модуль импортирован директивой `require`). «Деструкторы» выполняются при завершении работы интерпретатора. В одном модуле может быть описано множество блоков `BEGIN` и `END`, но, в отличие от пользовательских функций, все они в нужное время будут вызваны интерпретатором, причем блоки `BEGIN` вызываются в порядке их описания, а блоки `END` — в порядке,

обратном порядку их описания в модуле. Такой порядок вызова принят для того, чтобы соответствующие «конструкторы» и «деструкторы» можно было группировать парами и обеспечить при этом вызов «деструкторов» в обратном «конструкторам» порядке:

```
begin {
   # вызван первым
}
end {
   # вызван последним
}
begin {
   # вызван вторым
}
end {
   # вызван предпоследним
}
. . .
```

Блоки AUTOLOAD были введены в язык как средство борьбы с неэффективностью запуска программы, возникающей при большом количестве модулей в ней. Дело в том, что Perl не позволяет откомпилировать модули предварительно в псевдокод, а затем использовать уже откомпилированные версии для запуска и выполнения программы. Вместо этого компиляция модулей происходит при каждом запуске программы (если модуль импортирован директивой `use`) или в процессе выполнения (если модуль импортирован директивой `require`). Если определение вызываемой функции в модуле не найдено, то управление будет передано блоку AUTOLOAD (если он есть, в противном случае произойдет ошибка этапа выполнения). В качестве параметров ему будут переданы те, что были указаны при вызове функции, а встроенная переменная $AUTOLOAD будет содержать квалифицированное имя вызываемой функции.

Блок AUTOLOAD может обрабатывать такие вызовы по своему усмотрению, в частности, загружая на этапе выполнения определения необходимых функций из внешних файлов. На механизме автозагрузки основана работа стандартных модулей `AutoLoader` и `SelfLoader`, которые позволяют, разделив описания функций одного модуля по разным файлам, загружать их по требованию.

Данный механизм, по сути, является избыточным, поскольку в Perl уже есть средство динамической компиляции и загрузки модуля — это директива `require`. Автозагрузка просто позволяет распространить этот подход на уровне отдельных функций модуля, что может быть оправданным только в том случае, когда модуль содержит большое количество больших функций.

Синтаксисом модуля языка Perl не предусмотрена возможность описания списка экспортируемых объектов. Модуль в Perl может экспортировать лишь переменные и функции, поскольку возможность не то что экспорта, но даже описания констант и типов отсутствует. По умолчанию все переменные и все функции являются экспортируемыми, т. е. доступными из внешних модулей, импортирующих данный модуль.

Для объявления частных переменных модуля можно использовать встроенную функцию `my`, поскольку модуль неявно определяет блок, в котором содержится его тело и которому соответствует своя область видимости. Переменные, объявленные с помощью `my`, доступны только внутри данного модуля. А вот для описания частных функций модуля соответствующих средств нет — приходится описывать частную переменную-ссылку на анонимную функцию и вызывать ее по ссылке:

```
my $PrivateFunction = sub {
   # тело функции
};
```

Для импорта модулей предназначены две директивы: `use` и `require`. Первая используется для статического импорта (на этапе компиляции), а вторая — для динамического (на этапе выполнения). Директива `use`, кроме имени импортируемого модуля, позволяет указать список импортируемых объектов, например:

```
use SomeModule ("SomeVariable", "SomeFunction");
```

Если список указан и он не пуст либо если список не указан вовсе, то модуль будет откомпилирован, загружен, и будет вызвана функция `import`, которая может быть определена в импортируемом модуле и обязана самостоятельно реализовать импорт запрашиваемых объектов, имена которых ей будут переданы в качестве параметров. Собственно Perl эту возможность не поддерживает, а если функция `import` в модуле не оп-

ределена, то ошибки не возникнет ни на этапе компиляции, ни на этапе выполнения. При этом функция `import` фактически может реализовать не импорт, а любое другое действие. Если указан пустой список, то ни один объект не будет импортирован, а функция `import` вызвана не будет.

Чтобы облегчить жизнь разработчику модулей, в стандартную библиотеку Perl включен модуль `Exporter`, который реализует функцию `import`. Ее и следует использовать в подавляющем большинстве случаев. Здесь мы не будем подробно останавливаться на этом модуле. Заметим лишь, что для того чтобы воспользоваться всеми преимуществами модуля `Exporter`, ваш модуль должен наследовать модуль `Exporter`, рассматриваемый в данном случае как класс.

Надо отметить, что под словом «импорт» в Perl понимается нечто отличное от того, что принято понимать под этим словом в других языках. «Импорт» означает, что при обращении к импортированным объектам можно не использовать квалифицированных идентификаторов, а работать с ними так, будто они описаны в данном модуле. При этом сохраняется возможность обращения к прочим доступным объектам (не my-переменным и всем функциям) с помощью квалифицированных идентификаторов, например:

```
SomeModule::SomeFunction.
```

Таким образом, импорт не позволяет на самом деле управлять видимостью объектов при разработке модуля.

Наконец, отметим, что все имена, определенные в модуле, хранятся в хеше с именем, совпадающим с именем модуля, с «присоединенным» к нему символом «::». Причем этот хеш и его элементы доступны как для чтения, так и для записи. Возможностью «пополнения» таблицы символов модуля активно пользуются такие стандартные модули Perl, как `Exporter` и `overload`.

2.8. Язык Haskell

Haskell является одрим из наиболее мощных функциональных языков. Наиболее важными возможностями Haskell являются следующие.

- Haskell — ленивый (non-strict) язык.

- Haskell — чисто функциональный (т.е. не содержащий конструкций, неявно зависящих от состояния среды или изменяющих ее, например, он не содержит оператор присваивания).
- Haskell содержит простой и логичный механизм перегрузки функций (известный как «классы типов»). Возможные применения этого механизма выходят далеко за рамки решения непосредственной задачи. К примеру, классы типов обеспечивают возможности близкие (в целом, превосходящие) возможностям шаблонов C++, сохраняя при этом возможность раздельной трансляции.
- Императивные возможности реализуются в Haskell при помощи так называемых монад. Понимание этой конструкции является одной из наибольших проблем при изучении Haskell, но она стоит того, чтобы с ней разобраться.

В Haskell модули несут двоякое назначение — с одной стороны, модули необходимы для контроля над пространством имен (как, собственно, и во всех других языках программирования), с другой стороны, при помощи модулей можно создавать абстрактные типы данных.

Определение модуля в Haskell достаточно просто. Именем модуля может быть набор любых символов; имя начинается только с заглавной буквы. Дополнительно имя модуля никак не связано с файловой системой (как, например, в Pascal и Java), т.е. имя файла, содержащего модуль, может быть не таким же, как и название модуля. На самом деле, в одном файле может быть несколько модулей, так как модуль — это всего лишь декларация самого высокого уровня. Как известно, на верхнем уровне модуля в Haskell может быть множество деклараций (описаний и определений) — типы, классы, данные, функции. Однако один вид деклараций должен стоять в модуле на первом месте (если этот вид деклараций вообще используется). Речь идет о включении в модуль других модулей — для этого используется служебное слово *import*. Остальные определения могут появляться в любой последовательности.

Определение модуля должно начинаться со служебного слова *module*. Например, ниже приведено определение модуля Tree:

```
module Tree (Tree (Leaf, Branch), fringe) where
  data Tree a = Leaf a
  Branch (Tree a) (Tree a)
  fringe :: Tree a -> [a]
  fringe (Leaf x) = [x]
  fringe (Branch left right) = fringe left ++ fringe right
```

В этом модуле описан один тип; вполне нормально, что имя типа (Tree) совпадает с названием модуля, в данном случае они находятся в различных пространствах имен) и одна функция (fringe). В данном случае модуль Tree явно экспортирует тип Tree (вместе со своими подтипами Leaf и Branch) и функцию fringe — для этого имена типа и функции указаны в скобках после имени модуля. Если наименование какого-либо объекта не указывать в скобках, то он не будет экспортироваться, т.е. этот объект не будет виден извне текущего модуля.

Использование модуля в другом модуле выглядит следующим образом:

```
module Main where
import Tree (Tree(Leaf, Branch), fringe)
main = print (fringe (Branch (Leaf 1) (Leaf 2)))
```

В приведенном примере видно, что модуль Main импортирует модуль Tree, причем в декларации import явно описано, какие именно объекты импортируются из модуля Tree. Если это описание опустить, то импортироваться будут все объекты, которые модуль экспортирует, т.е. в данном случае можно было просто написать: import Tree.

Бывает так, что один модуль импортирует несколько других (надо заметить, что это обычная ситуация), но при этом в импортируемых модулях существуют объекты с одним и тем же именем. Естественно, что в этом случае возникает конфликт имен. Чтобы этого избежать, в Haskell существует специальное служебное слово *qualified*, при помощи которого определяются те импортируемые модули, имена объектов в которых приобретают вид: <Имя Модуля>.<Имя Объекта>, т.е. для того чтобы обратиться к объекту из квалифицированного модуля, перед его именем необходимо написать имя модуля:

```
module Main where
import qualified Tree
main = print (Tree.fringe (Tree.Leaf 'a'))
```

Использование такого синтаксиса полностью лежит на совести программиста. Некоторым нравится полная определенность, которую привносят квалифицированные имена, и они используют их в ущерб размеру программ. Другим нравится использовать короткие мнемонические имена, и они используют квалификаторы (имена модулей) только по необходимости.

В Haskell только с помощью модуля можно создать так называемые абстрактные типы данных, т.е. такие, в которых скрыто представление типа, но открыты только специфические операции над созданным типом, набор которых вполне достаточен для работы с ним. Например, хотя тип Tree является достаточно простым, его все-таки лучше сделать абстрактным типом, т.е. скрыть то, что Tree состоит из Leaf и Branch. Это делается следующим образом:

```
module TreeADT (Tree, leaf, branch, cell, left, right, isLeaf) where
  data Tree a = Leaf a
  Branch (Tree a) (Tree a)
  leaf = Leaf
  branch = Branch
  cell (Leaf a)    = a
  left (Branch l r) = l
  right (Branch l r) = r
  isLeaf (Leaf _) = True
  isLeaf _ = False
```

Видно, что внешний пользователь (программист) может получить доступ к внутреннему содержанию типа Tree только при помощи использования определенных функций. Впоследствии, когда создатель этого модуля захочет изменить представление типа (например, оптимизировать его), ему необходимо будет изменить и функции, которые оперируют полями типа Tree. В свою очередь, программист, который использовал в своей программе тип Tree, ничего менять не будет, так как его программа все также останется работоспособной.

В декларации импорта (import) можно выборочно спрятать некоторые из экспортируемых объектов (при помощи служебного слова *hiding*). Это

бывает полезным для явного исключения определений некоторых объектов из импортируемого модуля.

При импорте можно определить псевдоним модуля для квалификации имен экспортируемых из него объектов. Для этого используется служебное слово *as*. Это может быть полезным для укорачивания имен модулей.

Все программы неявно импортируют модуль Prelude. Если сделать явный импорт этого модуля, то в его декларации возможно скрыть некоторые объекты, чтобы впоследствии их переопределить.

Все декларации *instance* неявно экспортируются и импортируются всеми модулями.

Методы классов могут быть так же, как и подтипы данных, перечислены в скобках после имени соответствующего класса во время декларации экспорта/импорта.

2.9. Язык Sisal 3.0

Sisal 3.0 является функциональным языком программирования. Он ориентирован на поддержку научных вычислений и представляет собой дальнейшее развитие языка Val. Обладая всеми качествами, присущими функциональным языкам программирования, Sisal способствует разработке корректных детерминированных программ, которые свободны от совмещения имен, побочных эффектов и ошибок, зависящих от реального времени. Результаты детерминированы, невзирая на архитектуру, операционную систему или обстановку исполнения.

Модуль в языке Sisal представляет собой независимую единицу компиляции, состоящую из двух частей: интерфейса (interface) и реализации (implementation). Интерфейсная часть модуля предназначена для определения функций типов и редукций. Реализационный раздел содержит полное описание функций и редукций из интерфейса модуля. Кроме того, в нем могут определяться дополнительные объекты (функции, типы, константы), необходимые для реализации.

Синтаксис интерфейса выглядит следующим образом:

```
interface <имя интерфейса>
        <объявление функций>
        <определение типов>
        <объявление редукций>
```

Реализация имеет синтаксис

```
implementation <имя реализации>
        <полные определения функций и редукций из соответствующего интерфейса>
        <другие определения>
```

Интерфейс определяет связи модуля с внешним миром. Подключение модуля к программе (другому модулю) выполняется при помощи ключевого слова uses

```
uses <имя модуля>
```

Программы на языке Sisal — это совокупность модулей, связанных друг с другом при помощи объявлений экспорта и импорта. Среди совокупности функций из всех задействованных модулей должна быть одна, с которой начинается исполнение программы.

Объявление импорта производится после имени интерфейса перед всеми определениями либо после имени реализации, перед описанием всех объектов. При этом вызов какого-либо объекта из подключенного модуля осуществляется обращением к имени этого объекта. Если в программе уже присутствует такое имя, то при трансляции будет сообщено о семантической ошибке. В таком случае в языке предусмотрена возможность составного имени для вызова объектов модуля

```
<имя модуля>.<имя объекта>
```

Такой синтаксис позволяет избежать проблемы совпадения имен при использовании модулей. Использование расширенного синтаксиса разрешается даже в случае однозначности имени, но это будет переопределением.

Если модуль написан на другом языке программирования, то для его подключения необходимо выделить из dll-файла этого модуля его интерфейс. При написании подобных модулей необходимо учитывать специфику типов языка Sisal. Наглядным примером является ошибочное значение, которым дополнено множество значений каждого типа данных в языке.

Модуль может храниться как исходный файл (здесь и далее под файлом модуля подразумевается оба файла, описывающих данный модуль), как внутреннее представление (файл содержит запись откомпилированного исходного модуля) либо в виде dll-файла.

При подключении модуля разрешается использовать любую из вышеперечисленных форм. Предполагается использовать динамический способ загрузки модулей, т.е. модуль подгружается только в момент его вызова, а не при начальном запуске программы. При вызове функции из модуля может произойти одно из следующих действий: если функция берется из файла внутреннего представления, то производится проверка версии исходного модуля и объектного файла (при обнаружении несоответствия производится перекомпиляция этого модуля) и после этого создается новый объект внутреннего представления со ссылкой на этот метод. Либо, если имеется только исходный файл модуля, вначале производится его трансляция в объектный код, а затем вызов, как и в предыдущем случае. При использовании dll-файла из его интерфейса извлекаются входные и выходные параметры запрашиваемой функции, по которым и создается объект внутреннего представления.

СПИСОК ЛИТЕРАТУРЫ

1. **Бежанова М.М., Чилингарова Т.П**. Модуляризация в современных языках программирования. — Новосибирск, 1983. — 45 с. — (Препр. / АН СССР, Сиб. отд-ние ВЦ; N 453).
2. **Математическое** обеспечение ЭВМ: окружения и интерфейсы / Сост. И.В. Поттосин, М.М. Бежанова; Новосиб. гос. ун-т. — Новосибирск, 1994. — 74 с.
3. **Programming** in Ada 95 / J.Barnes — Beading etc.: Addison-Wesley; 2nd ed., 1998. — 720 p.
4. **Nikitin E.** Into the realm of Oberon: An introduction to programming and the Oberon-2 programming language. — Berlin etc.: Springer-Verlag, 1998. — 197 p.

5. **Wall L., Christiansen T., Schwartz R**. Programmig Perl. — O'Reilly, 2nd ed., 1996. — 670 p.
6. **Haskell**: The craft of functional programming / S.Thompson. — Beading etc.: Addison Wesley; 2nd ed., 1999. — 512 p.
7. **Поддержка** супервычислений и интернет-ориентированные технологии: Сб. статей / Под ред. В.Н. Касьянова; АН СССР. Сиб. отд-ние. Ин-т систем информатики. — Новосибирск, 2001. — 264 с.
8. **Стасенко А.П.** Внутреннее представление системы функционального программирования Sisal 3.0. — Новосибирск, 2004. — 84 с. — (Препр. / СО РАН. Ин-т систем информатики; № 110).

А. П. Стасенко

СИСТЕМА ИНТЕРФЕЙСОВ ТРАНСЛЯТОРА ВО ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ IR1*

ВВЕДЕНИЕ

Разработанный в виде COM-библиотеки транслятор из языка Sisal [1] во внутреннее представление IR1 [2] требует описания системы интерфейсов, с помощью которых происходит взаимодействие с клиентом библиотеки.

1. ТРЕБОВАНИЯ К СИСТЕМЕ ИНТЕРФЕЙСОВ

При разработке системы COM-интерфейсов библиотеки txt2ir1.dll, описывающих трансляцию из некого текстового представления программы во внутреннее представление IR1, требовалось обеспечить следующие свойства.

1. Отсутствие жесткой привязки к языку текстового представления. Это требование закладывает возможность использования данной системы интерфейсов для трансляции во внутреннее представление IR1 отличных от Sisal языков программирования.
2. Полная независимость от реализации транслятора, за исключением явного выделения хорошо формализуемого перевода потока символов в поток лексем (лексического анализа). Скрытие деталей реализации в данном случае повышает наглядность интерфейсов и не является существенным недостатком, так как результат трансляции обычно более важен, чем её процесс.
3. Исключение явного выделения сущности (интерфейса) синтаксического дерева, как результата работы синтаксического анализатора и

* stasenko@nm.ru

Работа выполнена при финансовой поддержке Министерства образования РФ (научная программа «Университеты России», грант № УР.04.01.201).

вытекающее из этого совмещение этапов синтаксического и семантического анализов. Данная особенность следует из графовой природы внутреннего преставления IR1, которое можно рассматривать в качестве обобщения понятия синтаксического дерева.

4. Поддержка UTF-8 кодировки уникода (Unicode-16) во входном потоке символов лексического анализатора и дальнейшее оперирование только уникод-строками. Уникод позволяет избежать трудностей с поддержкой иностранных алфавитов, а UTF-8 кодировка была выбрана из-за компактного текстового (ASCII) представления синтаксических конструкций большинства языков программирования (включая Sisal).
5. Ориентация на анализ потоков символов и лексем. Такое требование обеспечивает на уровне интерфейсов однопроходный, последовательный лексический и синтаксический анализы.
6. Возможность рассматривать исходное представление в качестве части (модуля) полного исходного представления программы (проекта), связанной с другими частями. Необходимость этого условия вытекает из модульной структуры программ на языке Sisal 3.0 [3].
7. Поддержка реализации лексического анализатора в виде самостоятельной фазы трансляции и в виде подпрограммы синтаксического анализатора, возвращающей очередную лексему. Первый вариант удобен при дополнительной предварительной обработке препроцессором списка разбираемых лексем, а второй — снижает емкостную сложность лексического анализа до константной, в случае, если на входе используется поток символов файла, считываемых с диска по мере необходимости.
8. Наличие развитого механизма сообщения об ошибках и предупреждениях на этапах лексического, синтаксического и семантического анализов. Под «развитым механизмом» подразумевается возможность получить описания и места возникновений произошедших событий и достаточно оперативно (до следующего разбираемого символа или лексемы потока) на них отреагировать.
9. Использование множества строк внутреннего представления в качестве таблицы имен лексического анализатора. При выполнении

данного условия отпадает необходимость дублировать строки на этапе разбора лексем.

10. Упрощение вспомогательных интерфейсов, указатели на которые требуются при трансляции. Более простые интерфейсы легче реализовать в случае, если требуется реализация, отличная от стандартной. Также ограничение интерфейсов уменьшает количество побочных эффектов реализации, их использующей.
11. Полная совместимость со скриптовыми клиентами библиотеки, наподобие VBScript и JavaScript. В этом случае диапазон и удобство применения разрабатываемой библиотеки существенно увеличиваются за счёт широкого распространения интернет-технологий.

2. АНАЛОГИЧНЫЕ РАЗРАБОТКИ

К сожалению, в открытых источниках практически не встречаются хорошо разработанные системы интерфейсов трансляторов. Это связано, прежде всего, с тем, что для разработки трансляторов широко применяются компиляторы компиляторов [4, 5], создающие встраиваемый исходный код, который не нуждается в хорошо проработанном внешнем интерфейсе. К тому же проблемой многих генераторов лексических анализаторов является отсутствие встроенной поддержки уникода. Положение также несколько усугубляется нетипичной для императивных языков природой цели (back-end) трансляции, представляющей собой поток вычислений, а не абстрактное синтаксическое дерево.

Тем не менее, несмотря на неприменимость существующих разработок в непосредственном виде, можно выделить следующие существующие системы интерфейсов транслятора.

1. GOLD Parser ActiveX Engine, ClearParse ActiveX/COM Interface и ProGrammar ActiveX Control. Обладают сходной функциональностью и позволяют аккуратно контролировать процесс последовательного выделения нетерминалов, заданной грамматики, из входного потока символов. Данная функциональность является достаточно низкоуровневой и не может быть обобщена на уровне интерфейсов с реализацией транслятора, которая не имеет в основании автоматной модели, построенной по некоторой грамматике. К

недостаткам указанных интерфейсов также относится отсутствие концепции потоков символов и лексем. Входные данные лексического анализатора непосредственно задаются с помощью имени файла или строкой символов, что в некоторых случаях может быть неоптимальным.

2. JavaCC [6]. Генерирует транслятор для языка Java. Использует стратегию нисходящего разбора [7], в отличие от стандартной для генераторов типа *yacc* стратегии восходящего разбора. Данный подход является более интуитивным и упрощает отладку транслятора. Рекурсивная природа алгоритма трансляции допускает естественное обобщение на уровне системы вложенных интерфейсов (понятий). Также такой подход позволяет использовать более развитую и точную систему сообщений об ошибках трансляции.
3. SabreCC [8]. Является в некотором роде развитием системы JavaCC в области использования объектно-ориентированного подхода для лучшего разделения автоматически генерируемого и пользовательского кода транслятора. SableCC генерирует классы прохода дерева, используя расширенную версию шаблона «посетитель» («visitor design pattern») [9], позволяющего выполнять действия над узлами абстрактного синтаксического дерева, используя наследование.
4. Система TeachLisp. Представляет собой интерпретатор языка Lisp, встроенный в Internet-учебный курс для обучения языку Лисп (в настоящий момент не функционирует).

3. ОПИСАНИЕ СИСТЕМЫ ИНТЕРФЕЙСОВ

На рис. 1 приведена UML [10] диаграмма всех интерфейсов библиотеки за исключением интерфейсов, связанных с представлением текста и лексем. На этой диаграмме у каждого интерфейса в алфавитном порядке указаны его атрибуты и операции, причём те или другие могут быть опущены. Атрибуты, помеченные символом @ и словом {frozen}, доступны только для чтения.

Основной интерфейс библиотеки *ITXT2IR1*, отвечающий за трансляцию, в ней не реализован, так как его реализация является языково-специфичной и вынесена в отдельную библиотеку. Метод *TextToLexemes* осуществляет

лексический разбор, а метод *ModuleLexemesToIR1* отвечает за синтаксический и семантический анализ. Аргумент *type* метода *ModuleLexemesToIR1* является языково-зависимым идентификатором понятия, задаваемого потоком лексем (например, интерфейс или реализация модуля).

Интерфейс *IModuleSearch* служит для обеспечения поиска интерфейса модуля по его имени, что может потребоваться для семантического анализа имён текущего модуля. Интерфейс модуля возвращается как указатель на интерфейс *IUnknown*, и его дальнейшая идентификация зависит от реализации метода *ITXT2IR1* → *ModuleLexemesToIR1*. Обычно интерфейс модуля можно извлечь из потока лексем (*ILexemeStream*), его задающих, или внутреннего представления IR1 (интерфейс *IIR1*).

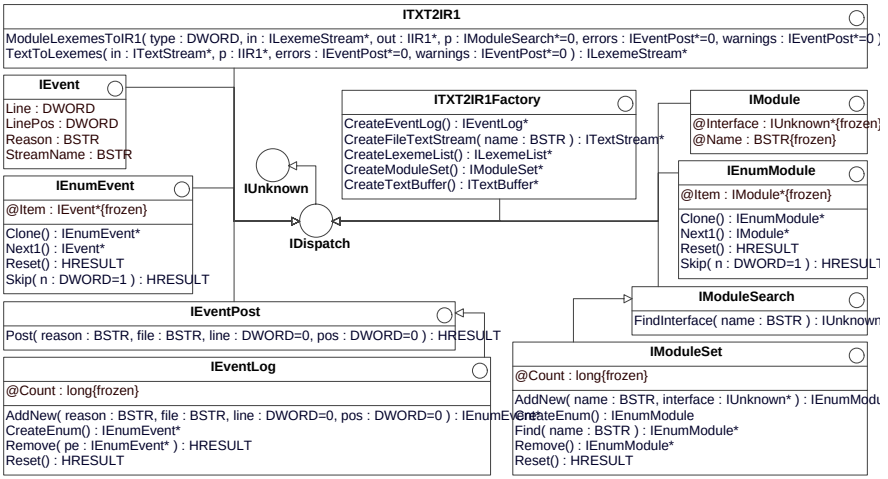


Рис. 1. Диаграмма основных интерфейсов

Простота интерфейса *ITXT2IR1* повышает его универсальность, так как этот интерфейс не зависит ни от входного языка, ни от реализации транслятора. Допускается использование утилиты типа *lex* для порождения лексического анализатора, а утилиты типа *yacc* для порождения синтаксического и семантического транслятора. Для реализации библиотеки sis2ir1.dll

транслятора из языка Sisal 3.0 был использован автомат, описанный с помощью графического метаязыка [11].

В библиотеке реализован интерфейс *ITXT2IR1Factory*, служащий в качестве фабрики классов для остальных интерфейсов библиотеки. На рис. 2 приведена UML диаграмма всех остальных интерфейсов библиотеки, не вошедших на рис. 1.

К интерфейсам, непосредственно участвующим в процессе трансляции, относятся *ITextStream*, *ILexemeStream*, *IEventPost*, *IModuleSearch*, *ILexeme*, *IEnumLexeme* и *IEnumText*. Все остальные интерфейсы являются вспомогательными и служат для обеспечения полноценной работы скриптовых клиентов с библиотекой.

К необходимым для эффективной трансляции методам относится метод *ITXT2IR1Factory → CreateFileTextStream*, который возвращает поток символов (*ITextStream*), связанный с файлом, с указанным именем. Непосредственное чтение байтов файла происходит по мере необходимости в момент перечисления элементов потока символов (*IEnumText*). Файл рассматривается как последовательность UNICODE символов в UTF-8 кодировке, поэтому метод *ITextStream→Count*, возвращающий количество элементов коллекции, в этом случае не реализован.

Предполагается, что при реализации метода *ITXT2IR1 → TextToLexemes* лексический анализ производится только при перечислении (*IEnumLexeme*) элементов возвращаемого потока лексем (*ILexemeStream*). Соответственно, тогда же и происходит возникновение возможных событий (ошибок и предупреждений). Поэтому временная и емкостная сложности работы самого метода константны.

Интерфейс *IEventPost* дает транслятору возможность сообщить о произошедшем событии. Стандартная реализация этого интерфейса (*IEventLog*) позволяет накапливать произошедшие события. Их просмотр возможен после получения следующей лексемы потока *ITXT2IR1 → TextToLexemes* или завершения процесса трансляции *ITXT2IR1 → ModuleLexemesToIR1*. Во втором случае для более тонкой реакции можно переопределить стандартную реализацию интерфейса входного потока лексем для проверки произошедших событий после каждой лексемы. Также возможно расширение реализации самого интерфейса *IEventPost* для ещё более оперативной реакции на событие.

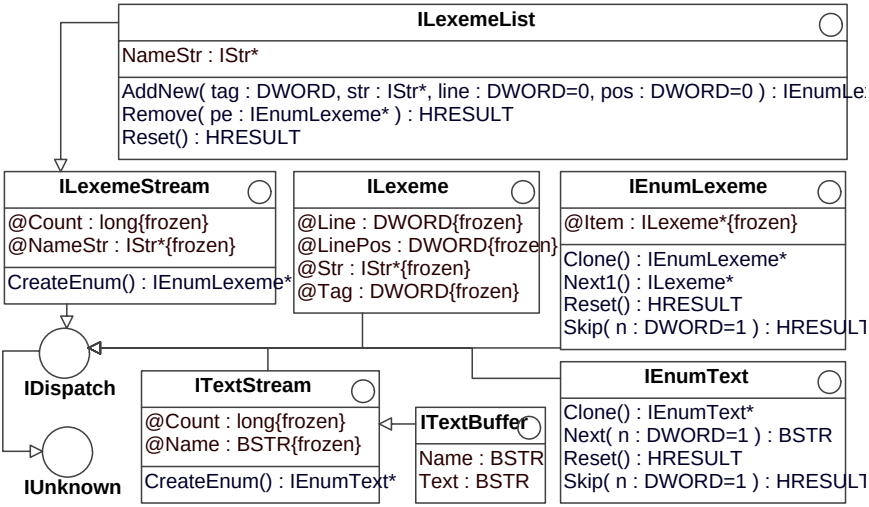


Рис. 2. Диаграмма интерфейсов потоков

К поддержке совместимости со скриптовыми клиентами относятся:

1) наследование всех интерфейсов от интерфейса *IDispatch*;
2) использование только совместимых типов данных: указатели на интерфейс, *long*, *HRESULT = long*, *DWORD = unsigned long*, *BSTR*;
3) отсутствие передачи параметров по ссылке (исключая возвращаемое значение);
4) наличие стандартной реализации всех вспомогательных интерфейсов;
5) все коллекции поддерживают скрытое свойство *_NewEnum* для стандартных средств их итерирования;
6) возможность регистрации библиотеки в качестве сервера автоматизации.

4. ПУТИ ДАЛЬНЕЙШЕГО РАЗВИТИЯ

Дальнейшее развитие системы интерфейсов планируется в основном за счёт увеличения функциональности интерфейса *ITXT2IR1*. Разумным выглядит добавление методов, ответственных за анализ сущностей на уровне модуля программы: типов, функций и выражений. Имеет смысл в качестве расширения концепции аргумента *type* метода *ModuleLexemesToIR1* произвести добавление интерфейса, задающего параметры трансляции, которые зависят от её реализации и входного языка.

Вероятным выглядит появление дополнительного интерфейса, реализующего сущность транс-литератора из потока байтов в последовательность уникод-символов. Также по мере развития других частей системы функционального программирования возможно расширение интерфейсов *ILexeme* и *IEvent*. В частности, планируется расширить интерфейс лексемы следующими свойствами.

1. Номер позиции начала текстового представления лексемы в потоке символов. Это поможет обнаруживать её местоположение в потоке за константное время без предварительного разделения потока на строки символов. Такое свойство также важно и для интерфейса *IEvent*.
2. Окончание (номер позиции, строка и столбец) текстового представления лексемы в потоке символов. Данная информация необходима для указания точных границ синтаксических конструкций во внутреннем представлении, что, возможно, потребуется при их «подсвечивании» в процессе отладки.
3. Строка с текстовым представлением лексемы в потоке символов. Может оказаться полезным в качестве дополнительной отладочной информации.

Интерфейс *IEvent* разумно расширить свойствами, дополнительно классифицирующими и уникально идентифицирующими произошедшее событие. В качестве классификации можно использовать:

1) более тонкую степень серьезности события: совет, предупреждение, ошибка, «фатальная» ошибка;
2) уровень важности события;

3) лексическую, синтаксическую или семантическую природу события.

С введением подобной классификации событий, станет ненужным существующее в интерфейсе *ITXT2IR1* жесткое деление событий на ошибки и предупреждения. Для этого в интерфейс *IEventLog* необходимо внедрить возможность перечисления событий, принадлежащих одному классу.

ЗАКЛЮЧЕНИЕ

В рамках описанной системы интерфейсов реализован транслятор из языка Sisal 3.0 во внутреннее представление IR1. С точки зрения разработки этого конкретного транслятора предложенная система интерфейсов оказалась достаточно удобной. Отсутствие привязки интерфейсов к языку Sisal позволяет надеяться, что рассмотренная система интерфейсов будет приемлемой и для других языков программирования.

СПИСОК ЛИТЕРАТУРЫ

1. **Бирюкова Ю. В.** Sisal-90: руководство пользователя. — Новосибирск, 2000. — 84 с. — (Препр. / РАН. Сиб. Отд-ние. ИСИ; № 72).
2. **Стасенко А. П.** Внутреннее представление системы функционального программирования Sisal 3.0. — Новосибирск, 2004. — 54 с. — (Препр. / РАН. Сиб. Отдние. ИСИ; № 110).
3. **Касьянов В. Н., Бирюкова Ю. В., Евстигнеев В. А.** Функциональный язык Sisal 3.0 // Поддержка супервычислений и интернет-ориентированные технологии. — Новосибирск, 2001. — С. 54-67.
4. **Lesk M. E.** Lex — a lexical analyzer generator. — Computing Science Technical Report 39 — AT&T Bell Laboratories, Murray Hill, N.J., 1975.
5. **Johnson S. C.** YACC — yet another compiler compiler. — Computing Science Technical Report 32 — AT&T Bell Laboratories, Murray Hill, N.J., 1975.
6. **Galles D.** Modern Compiler Design. — Scott Jones Publishers, 2005.
7. **Евстигнеев В. А., Касьянов В. Н.** Теория графов: алгоритмы обработки деревьев. — Новосибирск: Наука, 1994.
8. **Gagnon E.** SableCC, An Object Oriented Compiler Framework. — M.S. Thesis — McGill University, 1998.

9. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб.: Питер, 2001 — 368 с. — ISBN: 5-272-00355-1.
10. **Stevens P., Pooley R.** Using UML: software engineering with objects and components. — Object Technology Series. — Addison-Wesley Longman, 1999. — 280 p. — ISBN: 0-201-64860-1.
11. **Стасенко А.П.** Графический метаязык для описания транслятора // V Всерос. конф. молодых ученых по математическому моделированию и информационным технологиям / Программа и тезисы докладов. — Новосибирск, 2004. — С. 53.

Ю. Хан

ОБЗОР СРЕДСТВ ОТЛАДКИ ПРОГРАММ НА ФУНКЦИОНАЛЬНЫХ ЯЗЫКАХ[1]

ВВЕДЕНИЕ

В процессе создания программного обеспечения немаловажную роль играет отладка. По оценкам экспертов, она занимает от половины до двух третей всего времени разработки [1, 2]. Следовательно, необходимы эффективные отладочные средства, приспособленные к используемой парадигме.

Функциональная парадигма вносит свои характерные особенности в процесс отладки. Так, например, порядок выполнения действий в программе может существенно отличаться от того порядка, который можно предположить, читая её исходный текст, особенно в случае «ленивых» вычислений. Некоторые проблемы, часто возникающие в императивных программах, не свойственны функциональным программам (например, использование неинициализированной переменной или утечка памяти).

Таким образом, разные методики отладки имеют различную применимость в зависимости от парадигмы. В данной работе делается попытка проанализировать существующую ситуацию в отношении функционального программирования.

1. СУЩЕСТВУЮЩИЕ СИСТЕМЫ ПРОГРАММИРОВАНИЯ НА ФУНКЦИОНАЛЬНЫХ ЯЗЫКАХ

1.1. Caml и Objective Caml

Caml и его объектно-ориентированное расширение — это диалекты языка ML, разработанные в Национальном исследовательском институте информатики и автоматики (INRIA) (Франция). Поддерживаются платформы

[1] iis@centaur.mailshell.com

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 04-01-715) и Министерства образования РФ (научная программа «Университеты России», грант № УР.04.01.201)

UNIX, Windows и Macintosh. Существуют две реализации — Caml Lite и Objective Caml.

Язык обладает, помимо чисто функциональных свойств, также некоторыми чертами императивных языков. В частности, в нём есть многократно присваиваемые массивы, циклы и исключительные ситуации.

В Caml Lite основным средством отладки является трассировка. В режиме трассировки при входе в отлаживаемую функцию выводятся её аргументы, а при выходе — результат. Для более детального исследования работы программы авторы рекомендуют использовать отладочный вывод. Также предоставляется пошаговый отладчик, который, однако, реализован только под UNIX [3].

Система Objective Caml включает в себя отладчик, работающий под UNIX и Windows (только в версии под Cygwin). Отладчик предоставляет пользователю традиционные возможности:

- пошаговое исполнение;
- точки останова;
- стек вызовов;
- просмотр (но не изменение) значений переменных.

Также поддерживается удалённая отладка, когда программа и отладчик работают на разных машинах.

Отладчик не имеет собственного интерфейса пользователя. Управление производится через командную строку. Кроме того, пользователь может подключить отладчик к редактору Emacs, что позволит соотносить текущую позицию в программе с её исходным текстом [4].

1.2. Mercury

Язык Mercury, разработанный в Мельбурнском университете, сочетает в себе черты логических и функциональных языков. В нём отсутствуют побочные эффекты, используется строгая типизация. Язык поддерживает модульное программирование. Компилятор работает под UNIX-подобными системами [5].

Система включает в себя отладчик, работающий автономно (с использованием интерфейса командной строки) или в интеграции с редактором Emacs. Поддерживаются следующие возможности:

- пошаговое исполнение;
- просмотр значений переменных;
- стек вызовов;
- точки останова (с возможностью временного отключения);

- декларативная отладка.

В режиме декларативной отладки система задаёт пользователю вопросы о правильности результатов функций, начиная с самой внешней. Если функция даёт (согласно пользователю) неверный ответ, а все вложенные вызовы правильные, отладчик заключает, что ошибка находится в этой функции.

1.3. Scheme

Scheme — упрощённый диалект языка LISP. Были рассмотрены две реализации: EdScheme версии 5.0 (Schemers Inc.) и DrScheme версии 208 (группы PLT). Обе работают под управлением Windows и Macintosh; DrScheme также поддерживает Linux, FreeBSD и Solaris.

В EdScheme средства отладки ограничиваются трассировкой. Пользователь определяет подмножество функций, которые будут трассироваться, и запускает вычисление выражения. После окончания вычислений система отображает дерево вызовов отмеченных функций. В каждом узле отображаются аргументы функции; результат по умолчанию скрыт, но может быть показан по команде пользователя. Также есть режим, когда вычисление приостанавливается при достижении трассируемой функции [6].

DrScheme включает средство Stepper, позволяющее шаг за шагом проследить редукции выражения. На каждом шаге вызов функции разворачивается в её результат либо определение. Цель Stepper'а — скорее обучающая, чем отладочная. Также DrScheme поддерживает трассировку, которая ограничивается выводом всего дерева вызовов пользовательских функций с подставленными значениями аргументов [7].

Следует упомянуть встроенный механизм модульного тестирования. Вместе с определением функции программист может задать выражения и их ожидаемые результаты. При перекомпиляции модуля тесты выполняются, и рядом с ними отображается символ успешного или неуспешного выполнения, что позволяет программисту сразу определить момент, когда изменение программы приводит к нарушению её работы. Методика модульного тестирования подробно описана в книге [8].

Кроме того, система следит за тем, чтобы все пути управления были протестированы; фрагменты программы, которые не покрываются тестами, выделяются цветом.

1.4. Erlang

Язык Erlang разработан в фирме Ericsson для создания надёжных распределённых систем реального времени. Он поддерживает явные конструкции параллелизма, интерфейс к другим языкам программирования, сборку мусора в реальном времени. Компилятор работает под управлением Windows и UNIX-подобными системами.

Среда предоставляет традиционные средства пошаговой отладки — точки останова, пошаговое выполнение, просмотр и изменение значений переменных. Отладчик реализует собственный графический интерфейс пользователя [9].

1.5. Oz

По заявлениям разработчиков, язык Oz объединяет в себе возможности функционального, объектно-ориентированного и логического программирования и позволяет создавать параллельные многопоточные и распределённые системы. Виртуальная машина, исполняющая программы на Oz'е, работает под основными вариантами UNIX и под Windows [9].

Возможности отладчика включают:

- отображение дерева (строго говоря, леса) всех работающих в данный момент потоков программы. Для каждого потока отображается его состояние — работает, заблокирован, завершился успешно, завершился аварийно. Здесь пользователь может выбрать поток для отладки;
- стек вызовов (с значениями параметров функций) для каждого потока;
- пошаговую отладку;
- точки останова;
- просмотр значений переменных;
- удалённую отладку.

Отладчик интегрируется с редактором Emacs.

1.6. Haskell

Haskell — последовательный чисто функциональный язык с отложенными («ленивыми») вычислениями. Существует несколько реализаций, работающих на различных платформах.

Были рассмотрены системы: Buddha, Freja, HOOD и HsDebug.

Buddha [11] и Freja [12] реализуют механизм декларативной отладки. При этом используются различные способы сокращения выводимой информации:

- Freja приводит аргументы функций к наиболее упрощённому виду;
- обе системы выводят длинные структуры только частично, предоставляя пользователю возможность развернуть их при необходимости;
- обе системы не отображают подвыражения, которые в ходе работы программы не вычислялись;
- библиотечные функции считаются правильными и не участвуют в трассировке и отладке.

Поскольку полный след программы может занимать значительный объём памяти, система Freja производит частичную трассировку. При этом сохраняется только та часть следа, на которую хватает памяти, а если процесс отладки выходит за границы этого фрагмента, программа перезапускается с новыми параметрами трассировки.

Другой подход, используемый в системе HOOD (Haskell Object Observation Debugger) [12], состоит в выводе промежуточных значений выражений. Отслеживаемые выражения оборачиваются в функцию, которая эквивалентна тождественной, за исключением того, что она выводит свой аргумент в отладочный вывод.

Наконец, третий вариант — отказ от семантики отложенных вычислений. Система HsDebug заменяет «ленивые» вычисления строгими, что позволяет использовать традиционную методику отладки [13]. Однако такая замена не всегда бывает корректна. В частности, в случае отложенных вычислений программа может использовать бесконечные структуры. Попытка строго вычислить такую структуру приведёт к зацикливанию или исчерпанию свободной памяти. В HsDebug вводятся ограничения по времени для вычисления каждого выражения.

2. МЕТОДИКИ ОТЛАДКИ

Вышеописанные наблюдения позволяют выделить три основных направления.

2.1. Трассировка

При компиляции в программу добавляется код, собирающий информацию о ходе выполнения. После того как программа отработает, программист может использовать собранные данные для нахождения ошибок.

Преимущества:

- *простая реализация*.

Недостатки:

- *отсутствие наглядности*. Часто протокол состоит из серии имён функций с значениями параметров. Соотнесение их с исходным текстом программы возлагается на пользователя;
- *пассивность*. Выполнение и отладка программы разнесены во времени; во время отладки нельзя повлиять на выполнение;
- *замедление*. При больших объёмах отладочной информации её запись может занимать значительное время.

Метод непосредственно применим для строгих языков без параллелизма, в том числе не только функциональных, но и императивных. В случае параллельных программ, где потоки влияют друг на друга, может требоваться сопоставление протоколов отдельных потоков между собой.

2.2. Пошаговое выполнение

Отладка ведётся одновременно с выполнением программы. Как правило, пользователь видит текущее состояние программы и может влиять на степень подробности рассмотрения. Часть отладчиков также позволяют вмешиваться в работу программы, изменяя значения переменных.

Преимущества:

- полная информация о процессе выполнения;
- контроль количества выводимой информации;
- возможность влияния на исполнение программы.

Недостатки:

- метод фокусируется на операционной семантике программы — *из каких шагов состоит* вычисление, а не *что именно* вычисляется;
- реализация требует разработки механизма сопоставления внутреннего представления программы с исходным текстом. В случае отладки программ, скомпилированных в машинный код процессора, дополнительно требуется поддержка со стороны компилятора — сохранение соответствия между именами и адресами переменных, документированное представление данных и т. п.

Пошаговое исполнение применяется к императивным и строгим функциональным языкам.

2.3. Декларативная отладка

Метод декларативной, или алгоритмической, отладки был первоначально разработан для логических языков программирования. Позже его перенесли на функциональные языки с отложенными вычислениями.

В «ленивых» языках вычисления откладываются до того момента, когда их результат непосредственно потребуется, поэтому промежуточные данные могут представлять собой довольно громоздкие выражения, а порядок действий может сильно отличаться от интуитивного. Это считается причиной плохой применимости традиционных пошаговых методов к отладке программ с отложенными вычислениями [14].

Алгоритмическая отладка состоит из двух фаз.

- Пользовательская программа выполняется в специальном режиме трассировки. При этом собирается *след* — дерево вызовов пользовательских функций, где сохраняются аргументы и результаты.
- След обходится в глубину под контролем пользователя, который выносит суждения о правильности или ошибочности результатов применения той или иной функции.

Процесс завершается, когда найдена вершина дерева, давшая неверный результат, в то время как все её потомки вычислились корректно.

Метод имеет те же основные недостатки, что и трассировка — пассивность и замедление.

ЗАКЛЮЧЕНИЕ

Рассмотренные системы демонстрируют три основных подхода к отладке функциональных программ — трассировку, пошаговое выполнение и декларативную (алгоритмическую) отладку. При этом пошаговая отладка преобладает в системах программирования на строгих языках, в то время как для языков с отложенными вычислениями применяется алгоритмическая отладка.

СПИСОК ЛИТЕРАТУРЫ

1. **Brooks F. P. Jr.** The Mythical Man-Month: Essays on Software Engineering. — Boston: Addison–Wesley Professional, 1995. — 322 p.
2. **Spolsky J.** Painless Software Schedules. — http://www.joelonsoftware.com/articles/fog0000000245.html

3. **Weis P.** Frequently asked questions about Caml Light. — http://pauillac.inria.fr/caml/FAQ/index-eng.html
4. **Leroy X. et al.** The Objective Caml system release 3.08. Documentation and user's manual. — http://pauillac.inria.fr/caml/ocaml/htmlman/index.html
5. **Henderson F. et al.** The Mercury User's Guide. — http://www.cs.mu.oz.au/research/mercury/information/doc-release/user_guide_toc.html
6. **The EdScheme** for Windows Knowledge Base. — http://www.schemers.com/
7. **PLT** DrScheme: Programming Environment Manual. — http://download.plt-scheme.org/doc/drscheme/
8. **Beck K.** Extreme Programming Explained: Embrace Change. — Boston: Addison–Wesley Professional, 1999. — 224 p.
9. **Erlang** Debugger User's Guide. — http://www.erlang.se/doc/doc-5.4/lib/debugger-2.3/doc/html/part_frame.html
10. **Lorenz B., Kornstaedt L.** The Mozart Debugger. — http://www.mozart-oz.org/documentation/ozcar/index.html
11. **Pope B.** buddha Version 1.2 User's Guide. — http://www.cs.mu.oz.au/~bjpop/buddha/onlinedocs/UserGuide/UserGuide.html
12. **Nilsson H.** How to Look Busy while Being as Lazy as Ever: The Implementation of a Lazy Functional Debugger // J. of Functional Programming. — 2001. — N 11(6). — P. 629–671.
13. **The Haskell** Object Observation Debugger User Manual. — http://www.haskell.org/hood/documentation.htm
14. **Ennals R., Jones S. P.** HsDebug: Debugging Lazy Programs by Not Being Lazy // ACM SIGPLAN Haskell Workshop'03, August 28th, 2003, Uppsala, Sweden. —2003. — P. 84–87.
15. **Chitil O., Runciman C., Wallace M.** Freja, Hat and Hood — A Comparative Evaluation of Three Systems for Tracing and Debugging Lazy Functional Programs // Proc. of 12th Internat. Workshop on Implementation of Functional Languages 2000, Aachen, Germany, Sept. 4–7, 2000. — Berlin etc.: Springer-Verlag, 2001. — P. 176–193. — (Lect. Notes Comput. Sci.; N 2011).

Е. С. Черемушкин

АНАЛИЗ РАЗЛИЧНЫХ УЧАСТКОВ ДНК С ПОМОЩЬЮ АВТОКОРРЕЛЯЦИОННОЙ ФУНКЦИИ[1]

ВВЕДЕНИЕ

Функционально различные участки ДНК имеют различную структуру. В настоящее время общие закономерности различных функциональных участков ДНК подробно изучаются. В данной статье проведен анализ таких участков ДНК с помощью автокорреляционной функции и выявлены некоторые интересные особенности структуры различных функциональных участков.

Обычно целью данного изучения ставится распознавание неизвестных участков ДНК. Нашей целью было изучение структуры ДНК: найти некоторые различия, которые возможно не будут способствовать распознаванию неизвестных участков, но которые характеризуют качественные различия ДНК разных функций.

Первый этап такого исследования — это изучение автокорреляционной функции сигнала, образованного последовательностью ДНК.

1. СРАВНЕНИЕ АКФ РАЗЛИЧНЫХ ТИПОВ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

Для распознавания неизвестных участков в ДНК наилучшими оказались методы скрытых марковских моделей, методы нейросетей и частотный анализ [1]. Но эти методы не дают достаточного представления об особенностях сигналов в этих участках. Теряются даже очень простые, но в то же время интересные закономерности.

Для первого эксперимента были выбраны три типа последовательностей: регуляторные районы (районы, с которыми связываются специфические белки — транскрипционные факторы — и которые отвечают за регу-

[1] cher@bionet.nsc.ru

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 05-08-33448).

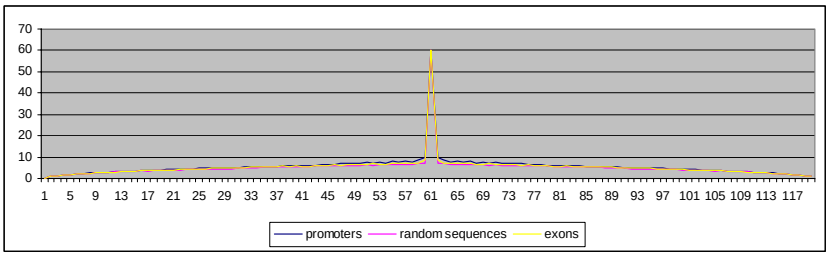
ляцию процессов в клетке), случайные последовательности и кодирующие районы (районы, которые кодируют белок).

Преобразуем ДНК в комплексную определяющую последовательность согласно таблице 1. Разобьем участки на районы длины 60. Теперь построим для каждого района $s_i = a_1, ..., a_N$ АКФ по формуле $F_i(t) = |sum(a_k * a_{k+t}^)|/L$, где ^ — комплексное сопряжение, L — длина последовательности, в данном случае равная 60. Усредним $F_i(t)$ по всем районам данного типа. Полученная средняя АКФ, представлена на рис. 1.

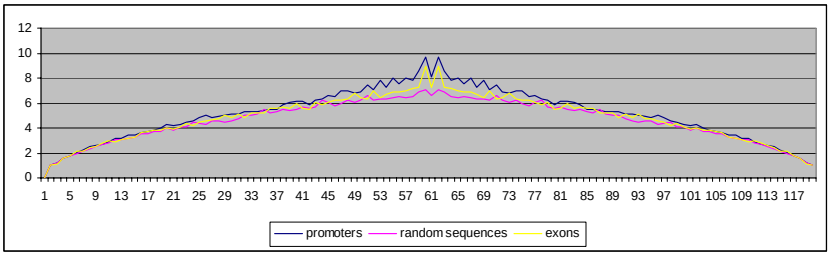
Таблица 1

Преобразование последовательности ДНК в определяющую последовательность сигнала

A	(1,0)	G	(0,-1)
C	(0,1)	T	(-1,0)



а)



б)

Рис. 1. а) АКФ различных районов ДНК; б) АКФ без нулевого пика

Видно, что АКФ различных участков практически совпадают, но видимые различия статистически значимы. Средняя корреляция этих районов: регуляторные районы (промоторы) — 5.007, случайные последовательности — 4.543, кодирующие районы (экзоны) — 4.707. Таким образом, видим, что регуляторные районы более скоррелированны, чем экзоны, а те более скоррелированны, чем случайные последовательности. Для того чтобы проверить достоверность этих данных, получим аналогичные значения для половины выборок: регуляторные районы (промоторы) — 4.988, случайные последовательности — 4.529, кодирующие районы (экзоны) — 4.765. Таким образом, можно утверждать, что данные о различиях в корреляции достоверны.

2. АНАЛИЗ РАЗБРОСА АКФ В РАЗЛИЧНЫХ УЧАСТКАХ ДНК

Рассмотрим дисперсию АКФ различных районов ДНК. Дисперсия характеризует разнообразие участков ДНК внутри одного функционального класса. Если АКФ внутри участков одного типа сильно варьирует, то это означает функциональную насыщенность фрагментов данного типа. Природе «необходимо», чтобы некоторые участки были крайне уникальны, а другие наоборот — очень похожи.

Посчитаем среднее значение E_i боковых пиков АКФ всех последовательностей данного типа. Затем посчитаем дисперсию по выборке $\{E_i\}$. Получим следующие значения: регуляторные районы (промоторы) — 26.03, случайные последовательности — 8.12, кодирующие районы (экзоны) — 11.66. Таким образом, промоторы гораздо более разнообразны по своей информационной насыщенности, чем другие последовательности. Это объясняется их назначением: они содержат участки, которые должны распознаваться специфическими белками. Причем определенные белки должны связываться только с конкретными промоторами. Если более подробно взглянуть на участки с маленькой и большой АКФ, можно заметить их отличие от случайных последовательностей (рис. 3).

Низкая АКФ

```
aactgaggtgcacagaggaaacctagttaagtactaagtggtggacttagaattttgaag
aatggttgtaagccaccaggtggttgctgggatttgaactccgggacctctggaagagca
agagaactctatataaatcaggttatttggcaggggtccaagatataacccaagccgaa
ccccggttatctaccggggctgggggttggtgttaaaattctcaagctagtatgtgctca
ccgttttctagggcaggaacccagggagggaagggctctggctggtatactgtaagtagc
cggtgttttagtgggtacatttcaacacaagcagcatttgggattacatcagcaaataca
cgtctggaggcttggttctgaagatgtggttttatcctcagctcagctcaaaattcgctg
ctaacgttaggctggaggtttcccgatagagaattcccctcacaccctctgccaatcctc
ctgggaaccaaacttgtatcttatgaaatagtagtaaagctctcattacagttacagctg
gcaggaacgtgcgcaccgccggagcctggcggcgacagtcacctgctcagcgggtacaga
gcgcagggaccacgctccctacccgccccgctggcagccgcagacagggcgccccgagaa
ggtcttggttttaggtcttttcaaagactaagcaatcttgttccgagctagcttttgagg
taacctgctgaaatccgaggtcgggctgttgctaggatgggcacctctccctctagtgtt
tatagacagcgaagagcccctgcagccgagctaggccagaagaacccatggctggaactc
ttaaactttgtgtgcctcagtttccttatctgtaaagtgggattaataatgtcatcctga
tttaaaattacagttgacctgggacgaagttcccataaagacctgcaagtcatccaccat
```

Высокая АКФ

```
aaaaaaaaaaaaaaaaagttcaaattctaactctgtaacagaaaccgcgaaaaaaaaccca
aaaaggaggtgggctatttctgtttatccacctgtctctacaaaagcaaataaaactgtg
acacacacacacacacacacaggctcttagtgcatgtatccacctctctctctctctc
acagagaaaccctgtctcaaacaaacaaacaaacaaacaaacaaacaaacaaacaacacc
atcacctttaatgtatttgcattttattatatacacatatacaaacatatatacatacat
cccagctctccatagccctgccccagctccccacagccctgccccagctccccagaccca
ccccagttccccacagccctgccccagctctccatagccctgccccagctctccatagcc
ctctttctttctttctttctttctctctctctttccttctttctttctttctttct
gagaaagaagtgagaggagggaagacaggggaagggaagagagaaaagagaggagagttg
gcccgcgcccgagcccaggccgagccttgccctggtccccggccgggccgaggccgcgcc
ggggcggtgcgggggggggggggaagggagtggtctccataagggggaggggagaaggcag
gtgcctagctctccgttgttgcaggggggaaaaaaaagtattgggaaaaaaaaaaaaccat
tacaagagaaaaaaaacaaacaaacaagaaacagctacaacccggcaaacgaaccacgtg
tatgttttcttttgctctctttctccctttctccctttctttctttctctttttctttct
tcgcgctccgcccttctccgtcctcttagcctcgctccttccctcagccccgccctcaat
tctctcacacacacacacacacacacacacacacacacacacaccacaaatgcattggtgtt
tctctctctttctttctctttctcatatgcacacatgcacacacacacataactctgaac
ttccttttctttttctttctctttttttccttttccaaaaggggtcacacacacacccgcc
ttctttctttctttctttctttctttctttctttctccttttttttgtttgtttggctttt
tttacattttgttatatacatacataatacacacatacacatatatacatacatataact
```

Рис. 3. Участки промоторов с низкой и высокой АКФ

3. ИЗУЧЕНИЕ ЗАВИСИМОСТИ СКОРРЕЛИРОВАННОСТИ РЕГУЛЯТОРНЫХ УЧАСТКОВ ОТ ПОЛОЖЕНИЯ В ГЕНЕ

Для предварительного анализа были взяты промоторные участки ДНК в районе старта транскрипции от −1000 до 80. В данном эксперименте участки были разбиты на 3 района: R1 = [−1000, −640], R2 = [−640, −280], R3 = [−280, 80]. Значения среднего и дисперсии в этих районах A(R1) = 5.04, A(R2) = 5.10, A(R3) = 4.86, D(R1) = 18.34, D(R2) = 15.52 и D(R3) = 9.88.

Таким образом, ближе к старту транскрипции средняя АКФ у промоторов понижается, и это говорит о том, что в районе старта больше достаточно уникальных участков. Из этого можно сделать вывод: низкая АКФ участка указывает на то, что этот участок несет функциональную нагрузку. Из этого следует, что природе «выгодно» поддерживать общее однообразие ДНК, и только функционально важные участки имеют свой «уникальный» паттерн.

ЗАКЛЮЧЕНИЕ

Суммируя результаты исследований, можем заключить, что АКФ различных участков практически совпадают, но видимые различия статистически значимы. А именно, регуляторные районы более скоррелированны, чем экзоны (кодирующие районы), а те более скоррелированны, чем случайные последовательности.

Промоторы (регуляторные участки) — гораздо более разнообразны по своей информационной насыщенности, чем другие последовательности. Это объясняется их назначением: они содержат участки, которые должны распознаваться специфическими белками. Причем определенные белки должны связываться только с промоторами одного типа.

Ближе к старту транскрипции средняя АКФ у промоторов понижается, и это говорит о том, что в районе старта больше достаточно уникальных участков. Из этого можно сделать вывод, что низкая АКФ участка указывает на то, что этот участок несет функциональную нагрузку. Таким образом, природе «выгодно» поддерживать общее однообразие ДНК и только функционально важные участки имеют свой «уникальный» паттерн.

СПИСОК ЛИТЕРАТУРЫ

1. **Azad R.K., Borodovsky M.** Probabilistic methods of identifying genes in prokaryotic genomes: connections to the HMM theory // Brief Bioinform. — 2004. — Vol. JU-5, N 2. — P. 118–130.

Д.Н. Штокало, Е.С Черемушкин

ПОСТРОЕНИЕ ПРОГРАММНОГО КОМПЛЕКСА “REGULATORY SEQUENCES ANALYZER” ДЛЯ РАСПОЗНАВАНИЯ ЦИС-ЭЛЕМЕНТОВ В ПОСЛЕДОВАТЕЛЬНОСТЯХ ДНК[1]

ВВЕДЕНИЕ

Перед авторами стояла следующая задача — наглядная, удобная для пользователя реализация алгоритмов **Match** и **CoMatch**, т.е. алгоритмов распознавания потенциальных цис-элементов последовательности ДНК с использованием весовых матриц. Поиск потенциальных цис-элементов должен производиться на прямой либо обратной цепи последовательности.

Наглядная реализация подразумевает возможность динамически в процессе выполнения программы осуществлять выбор искомых сайтов, выбор порогового значения, мгновенный и понятный вывод результатов в виде прокручиваемой ДНК последовательности и выделение цветом найденных сайтов.

1. АЛГОРИТМЫ И ИСПОЛЬЗУЕМЫЕ ДАННЫЕ

Match и **CoMatch** относятся к алгоритмам весовых матриц. Основная идея алгоритма весовых матриц [1, 2] заключается в приписывании четырех весов каждой позиции сайта в соответствии с четырьмя нуклеотидам A, T, G и C. Эти веса связаны с частотой встречаемости конкретного нуклеотида в конкретной позиции.

1.1. Алгоритм Match

Пусть $F = \left| f_{ij} \right|$ — l×4-матрица нуклеотидных частот, f_{ij} — абсолютная частота встречаемости i-го нуклеотида на j-ой позиции в обучающей

[1] cher@bionet.nsc.ru

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 05-08-33448).

выборке выровненных нуклеотидных фрагментов, кодирующих известные сайты связывания ($I = 1, .., l; j = 1, .., 4$). Далее определяется весовая матрица W. Существует много способов определения элементов весовой матрицы w_{ij}. Рассмотрим способ, реализованный в “Regulatory Sequence Analyzer”:

$$w_{ij} = I(i) * f_{ij}, \quad \text{где} \quad I(i) = \left| \sum_{B \in \{A,C,G,T\}} f_{i,B} * \ln(4 * f_{i,B}) \right|.$$

Таблица

Пример частотной(F) и весовой (W) матрицы

		позиции сайта							
		123	124	125	126	127	128	129	130
	‘A’	49	0	288	26	77	67	45	50
F:	‘C’	48	303	0	81	95	118	85	96
	‘G’	69	0	0	116	0	46	73	56
	‘T’	137	0	15	80	131	72	100	101
	‘A’	9.7	19.4	30.9	0	75	0	0	0
W:	‘C’	19.4	9.7	0	74.9	0		12.5	0
	‘G’	19.4	19.4	10.2	0	0	49.9	50	0
	‘T’	0	0	10.2	0	0	12.5	0	74.9

Процедура распознавания функционального сайта (характеризуемого весовой матрицей W) в произвольном нуклеотидном фрагменте длины L заключается в сопоставлении величины *match* и заранее заданного порогового значения $match^{(crit)}$:

$$match = \frac{x - x_{\min}}{x_{\max} - x_{\min}},$$

где $x_{\max} = \sum_{i=1}^{L} \max_j(w_{ij})$, $x_{\min} = \sum_{i=1}^{L} \min_j(w_{ij})$, а значение x оценивает степень близости тестируемого фрагмента и обучающей выборки:

$$x = \sum_{i=1}^{L} w_{i,B}, \quad B \in \{A, C, G, T\}.$$

Все потенциальные сайты в заданной нуклеотидной последовательности распознаются с помощью применения вышеизложенного алгоритма к каждому скользящему окну из этой последовательности.

1.2. Алгоритм распознавания двойных сайтов CoMatch

Сайты связывания некоторых транскрипционных факторов состоят из двух полусайтов с варьирующимся расстоянием между ними. Расстояние между полусайтами зависит от типа фактора, узнающего этот сайт. Полусайты могут иметь схожую структуру. Так как сайт состоит из 2-х консервативных доменов с варьирующим расстоянием между ними (рис. 1.1), то зададим double-core модели распознавания M_k следующим образом[4]:

$$M_k = \langle m_1, m_2, d_1, d_2 \rangle,$$

где m_1 и m_2 — весовые матрицы [2], d_1, d_2 — минимальное и максимальное расстояния между половинками сайтов. Пусть $w_1(i)$ и $w_2(j)$ веса m_1 и m_2 в позиции i и j соответственно на последовательности. Сайт считается распознанным, если вес $w = \frac{w_1(i) + w_2(j)}{2}$ больше заданного порога C и расстояние между половинками сайтов — $d \in [d_1, d_2]$.

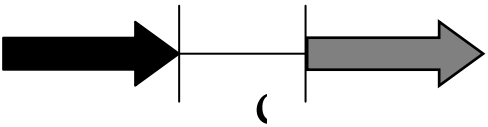


Рис. 1.1. Сайт состоит из 2-х консервативных доменов с варьирующим расстоянием между ними

Распознавание двойных сайтов будем производить следующим образом. Если на последовательности был распознан характерный полусайт, рассмотрим, какой максимальный вес w_k распознавания дает каждая из моделей M_k. Если модель M_k не распознана в данном районе, то считаем $w_k = 0$. Рассмотрим алгоритм получения моделей M_k. Пусть $S = (S_1, ..., S_m)$ — обучающая выборка последовательностей сайтов. Для каждого подмножества $S' = (S_1', ..., S_m')$ множества S зададим два набора подпоследовательностей $S^1 = (s_1^1, ..., s_n^1)$ и $S^2 = (s_1^2, ..., s_n^2)$, $s_i^1, s_i^2 \in S_i'$, длина s_i^j равна 6.

Найдем с помощью классической процедуры гиббс самплинга [5] S^1 и S^2 такие, что s_i^1 похожи друг на друга и s_i^2 похожи между собой. По S^1 и S^2 создадим соответствующие матрицы m_1 и m_2. Затем выберем расстояния $d_1 = \min_i \left(d(s_i^1, s_i^2) \right)$ и $d_2 = \max_i \left(d(s_i^1, s_i^2) \right)$. Выберем начальное подмножество $S_{[0]}$, называемое базовой выборкой. Теперь построим модель $M_{[0]}$ и добавим в $S_{[0]}$ последовательность из $S \setminus S_{[0]}$, для которой вес $w_{[0]}$ модели $M_{[0]}$ максимален. Таким образом получим модель $M_{[1]}$. Будем продолжать процедуру добавления до тех пор, пока вес $w_{[k]}$ превышает изначально заданный порог C (рис. 1.2).

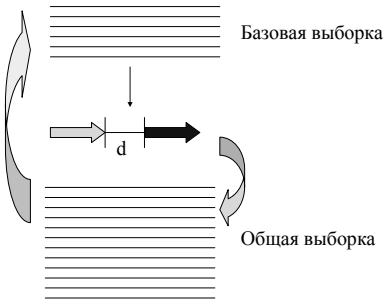


Рис 1.2

После окончания процедуры получим модель M, описывающую выборку S. Таким образом получим различные модели $M_1,...,M_T$ для различных классов сайтов.

1.3. Библиотека матриц, используемая программой при реализации алгоритма Match

Файл с матричной библиотекой [3], открываемый в главном окне программы, имеет расширение .dat и содержит в себе записи, перечисляющие частотные матрицы F, снабженные комментарием.

Пример записи, взятой из библиотеки матриц:

```
//
AC  M00028
XX
ID  I$HSF_01
XX
DT  18.10.1994 (created); ewi.
DT  16.10.1995 (updated); ewi.
CO  Copyright (C), Biobase GmbH.
XX
NA  HSF
XX
DE  heat shock factor (Drosophila)
XX
BF  T00386; HSTF; Species: fruit fly, Drosophila melanogaster.
XX

P0      A     C     G     T
01     31    14     4     1     A
02      0     0    50     0     G
03     48     2     0     0     A
04     43     1     5     1     A
05     23     1    14    12     N
XX
BA  50 functional genomic HSEs
XX
CC  not included are sequences with more than 2 mismatches at positions 2 to 4;
CC  the matrix only describes the properties of the basic 5-bp unit of which
CC  three have to be present to constitute a minimal HSE
XX
RN  [1]
```

```
RX  MEDLINE; 94167242.
RA  Fernandes M., Xiao H., Lis J. T.
RT  Fine structure analyses of the Drosophila and Saccharomyces heat shock
RT  factor-heat shock element interactions
RL  Nucleic Acids Res. 22:167-173 (1994).
XX
//
```

Необходимо, чтобы запись содержала строки, начинающиеся с управляющих символов:

```
AC          — начало записи
ID          — имя матрицы
P0          — начало матрицы
XX          — конец матрицы
//          — конец записи
```

1.4. Библиотека двойных сайтов, используемая программой при реализации алгоритма CoMatch

Пользователю необходимо открыть два библиотечных файла в главном окне программы. Первый — с расширением .clib, второй — формата .matrixlib.

Файл формата .clib содержит информацию о двойном сайте, т.е. его имя, имена составляющих матриц, номера матриц в файле .matrixlib и др.

Файл формата .matrixlib должен перечислять матрицы, упомянутые в файле .clib. В отличие от .dat, файл .matrixlib содержит не частотные матрицы F, а уже весовые, т.е. W.

Открыв файл .clib, программа автоматически попросит открыть файл .matrixlib.

Фрагмент файла в формате .clib:

```
AC  E00001
ID  V$C_AR_G01
I1  M00001
M1  V$AR_HG01
S1  0.0
I2  M00002
M2  V$AR_HG02
S2  0.0
OR  > >
DI  0 6
//
```

Фрагмент файла в формате .matrixlib:

```
AC  M00001
ID  V$AR_HG01
NA
MATR_LENGTH  6
CORE_START  1
CORE_LENGTH 5
MAXIMAL 5421.14879048849
MINIMAL 0
THRESHOLD 0.5
WEIGHTS
1 A:654.318988544113 C:0 G:373.89656488235 T:0
2 A:821.353332948717 C:0 G:308.007499855769 T:0
3 A:144.009999807692 C:108.007499855769 G:144.009999807692 T:0
4 A:120.281615774509 C:420.98565521078 G:0 T:120.281615774509
5 A:1398.72445341174 C:0 G:0 T:139.872445341174
6 A:0 C:0 G:1981.75636056544 T:0
//
AC  M00002
ID  V$AR_HG02
NA
MATR_LENGTH  6
CORE_START  1
CORE_LENGTH 5
MAXIMAL 8008.98919012654
MINIMAL 5.33833323717916
THRESHOLD 0.5
WEIGHTS
1 A:0 C:0 G:0 T:1981.75636056544
2 A:0 C:0 G:1398.72445341174 T:139.872445341174
3 A:0 C:0 G:0 T:1981.75636056544
4 A:8.00749985576874 C:5.33833323717916 G:5.33833323717916
T:10.6766664743583
5 A:0 C:1981.75636056544 G:0 T:0
6 A:373.89656488235 C:0 G:0 T:654.318988544113
//
```

1.5. Поиск цис-элементов на комплиментарной цепи данной последовательности

Если пользователь пометил «галочкой» пункт Reverse DNA на главном окне, то после нажатия кнопки «Calculate» программа произведёт поиск цис-элементов на комплиментарной цепи ДНК, т.е. программа вычислит комплиментарную цепь и примет её за основную. Более эта пометка на работу программы не повлияет.

2. РАБОТА С ПРОГРАММОЙ

При запуске программы проследите за наличием в той же директории папки Pictures. Запущенная программа предоставляет *Главное окно* для выбора желаемого алгоритма и файлов с биологическими данными.

2.1. Главное окно

Настройки Главного окна повлияют на результат работы программы только после нажатия кнопки “Calculate”.

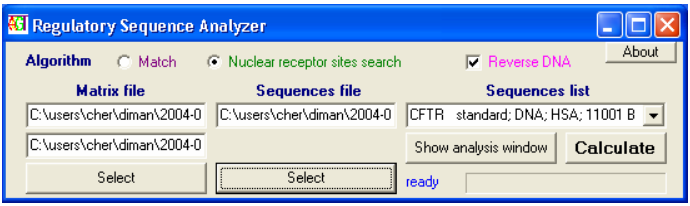


Рис. 2.1. Главное окно программы

2.1.1. В колонке “Algorithm” производится выбор алгоритма Match (см. 1.1) либо CoMatch (см. 1.2), а также, при желании произвести поиск на комплиментарной цепи последовательности, помечается «галочкой» пункт “Reverse DNA” (см. 1.5).

2.1.2. В колонке “Matrix file” производится открытие библиотек с данными о сайтах. Если отмечен пункт Match, программа позволяет открыть файлы с расширением .dat (см. 1.3). Второе редакционное поле колонки “Matrix file” при этом блокируется. Если отмечен пункт CoMatch, возможно открытие файлов композиционной библиотеки формата .clib, а также при

необходимости открытие файлов матричной библиотеки .matrixlib (см. 1.4).

2.1.3. В колонке “Sequences file” производится выбор файла библиотеки последовательностей формата .embl. После выбора файла происходит автоматическое построение списка имён последовательностей ДНК, содержащихся в embl файле. Этот список приготовлен для выбора в “Sequences list”.

2.1.4. При открытии необходимых файлов становится возможен обсчёт данных и немедленный вывод результатов в автоматически открываемом *Окне результатов*. Для этого нажмите кнопку “Calculate”. Стадия готовности результата будет отражена заполняющейся полосой “ready”.

2.1.5. Кнопка “Show analysis window” предназначена для показа *Окна результатов*. Если до момента нажатия этой кнопки они ещё не были получены программой, нажатие не произведёт никакого эффекта.

2.2. Окно результатов

На любое действие пользователя с элементами этого окна программа реагирует немедленно и полностью отражает результаты, соответствующие статусу интерфейса.

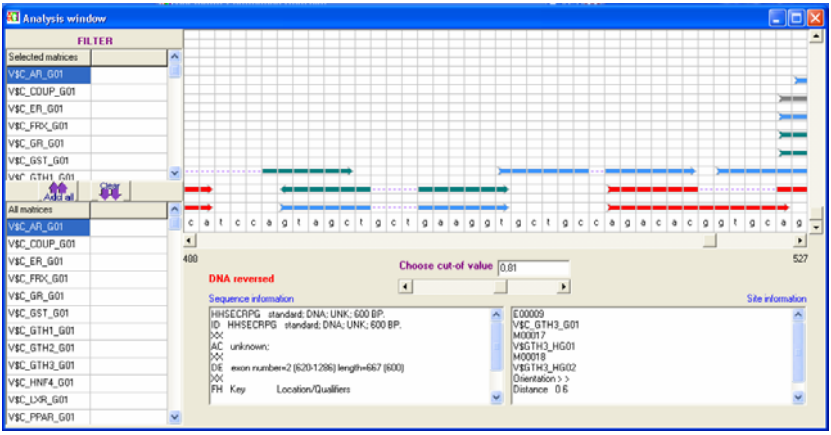


Рис. 2.2. Окно результатов

В окне расположены:

- две таблицы слева для фильтрации сайтов;
- два поля внизу для отображения информации о последовательности и каком-либо выбранном сайте;
- центральный скроллер для динамического выбора порога “cut-off value”;
- поле для наблюдения последовательности и стрелок, соответствующих найденным программой сайтам.

2.2.1. Таблица “All matrices” содержит список всех матриц (моделей), которые искала программа. “Клик” на ячейку левой колонки добавляет соответствующую матрицу (модель) в таблицу “Chosen matrices”. Если “Chosen matrices” уже имеет данную матрицу (модель), добавления не происходит. Нажатие кнопки “Add all” влечёт занесение в “Chosen matrices” списка всех имеющихся матриц (моделей). “Клик” на ячейку правой колонки выводит на поле “Site information” информацию о данной матрице (модели).

2.2.2. Таблица “Chosen matrixes” содержит список матриц (моделей), результат поиска которых желает видеть пользователь. Пользователю видны только стрелки, соответствующие матрицам (моделям) из данной таблицы. Клик на ячейку левой колонки удаляет из таблицы соответствующую матрицу (модель), кнопка “Clear” очищает таблицу. “Клик” на ячейку правой колонки выводит на поле “Matrix information” информацию о данной матрице (модели) и, кроме того, выделяет стрелки соответствующего сайта красным цветом.

2.2.3. Центральный скроллер выбора порога “cut-off value” позволяет выбрать порог в интервале [0.5, 1]. Результат изменения порога немедленно виден на поле результатов. Повышающийся порог уменьшает количество выводимых стрелок, понижающийся — увеличивает.

2.2.4. На поле результатов виден участок последовательности, как правило, длиной в 40 символов. Выше расположены изображения сайтов в виде стрелок. Покрываемый стрелкой интервал последовательности соответствует найденному цис-элементу. “Клик” по стрелке выделит красным цветом все стрелки, соответствующие этой матрице (модели), и заполнит “Site information” информацией. Стрелка, обращённая вправо, означает положение найденного цис-элемента на прямой цепи ДНК, обращённая влево — на обратной цепи. Поле вывода снабжено скроллерами прокрутки, так что пользователю сразу легко доступна итоговая информация обо всей последовательности.

ЗАКЛЮЧЕНИЕ

Реализована программная система для визуализации результатов работы алгоритмов **Match** и **CoMatch**, т.е. алгоритмов распознавания потенциальных цис-элементов последовательности ДНК с использованием весовых матриц. Поиск потенциальных цис-элементов производится на прямой либо обратной цепи последовательности. Наглядная реализация позволяет в процессе выполнения программы динамически осуществлять выбор искомых сайтов, выбор порогового значения, мгновенный и очевидный вывод результатов в виде прокручиваемой ДНК последовательности и отмеченных цветной стрелкой найденных сайтов.

СПИСОК ЛИТЕРАТУРЫ

1. **Schneider T., Stephens R.** Sequence logos: a new way to display consensus sequences // Nucleic Acids Res. — 1990. — Vol. 18. — P. 6097–6100.
2. **Kel A.E., Gossling E., Reuter I., et al.** MATCH: A tool for searching transcription factor binding sites in DNA sequences // Nucleic Acids Res. — 2003. — Vol. 31, N 13. — P. 3576–3579.
3. **Wingender E., Chen X., Fricke E., et al.** The TRANSFAC system on gene expression regulation // Nucleic Acids Res. — 2001. — Vol. 29, N 1. — P. 281–283.
4. **Черемушкина Е., Черемушкин Е., Чекменев Д., Кель О.** Метод идентификации сайтов ядерных рецепторов // Тез. конференции-конкурса «Технологии Microsoft в информатике и программировании», 21–23 февраля 2004, Новосибирск. — Новосибирск, 2004. — С. 137–139.
5. **Lawrence, C.E., Altschul, S.F., Bogouski, M.S., et al.** Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment // Science. — 1993. — Vol. 262, Iss. 5131. — P. 208–214.

СОДЕРЖАНИЕ

CONTENTS

УДК 519.68 + 681.3.06

Исследовательская система для анализа текстов на естественном языке / Батура Т.В., Корда О.В., Мурзин Ф.А., Позименко А.А. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 7–20.

В статье кратко описывается исследовательская система для анализа текстов на естественном языке, разрабатываемая авторами. — Библиогр.: 12 назв.

Exploratory system for analysis of texts in a natural language / Batura T.V., Korda O.V., Murzin F.A.,Pozimenko A.A. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 7–20.

In the paper, the exploratory system developed by the authors for analysis of texts in a natural language is briefly described. — Refs: 12 titles.

УДК 519.68 + 681.3.06

Сравнительный анализ методов поиска регуляторных модулей в последовательностях ДНК, использующих данные микроэррэев / Валеев Т.Ф. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 21–28.

Данная статья посвящена проблеме поиска регуляторных модулей в последовательностях ДНК. Описана постановка задачи и произведён краткий сравнительный обзор трёх разработок, направленных на её решение: пакета TOUCAN, системы TELiS и Composite Module Analyst, отмечены достоинства и недостатки различных подходов. — Библиогр.: 8 назв.

Comparative analysis of the search techniques using microarray data for regulation modules in the DNA sequences / Valeev T.F. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 21–28.

This paper describes the problem of regulation modules search in the DNA sequences. The statement of the problem is given, as well as the review of three projects aimed at the problem solution: TOUCAN package, TELiS system and Composite Module Analyst. Advantages and shortcomings of different approaches are pointed out. — Refs: 8 titles.

УДК 519.68 + 681.3.06

Расчет коэффициента нефтенасыщенности по результатам ядерного каротажа / Винокуров А.А., Ильин И.В., Мурзин Ф.А., Семич Д.Ф. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 29–54.

В статье рассматриваются алгоритмы для расчета нефтенасыщенности по данным ядерного каротажа. — Библиогр.: 2 назв.

The calculation of the oil saturation coefficient based on results of nuclear oil-wells carotage / Vinokurov A.A., Ilyin I.V., Murzin F.A., Semich D.F. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 29–54.

The paper considers the algorithms for calculation of oil saturation based on data of nuclear oil-wells carotage. — Refs: 2 titles.

УДК 519.68 + 681.3.06

Интерфейс пользователей виртуального музея истории информатики в Сибири / Волянская Т.А. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 55–94.

Описывается пользовательский интерфейс виртуального музея истории информатики в Сибири. Разрабатываемый музей предназначен для накопления, систематизации и использования информации, относящейся к становлению и развитию информатики в Сибири. Музей создается в виде информационно-поисковой, справочной адаптивной гипермедиа-системы, доступной в Интернет. Рассматривается интерфейс управления информационными ресурсами: механизм навигации и просмотр информации, поиск, ввод и редактирование информации. Описывается интерфейс управления пользователями: регистрация, аутентификация, авторизация и администрирование пользователей. — Библиогр.: 4 назв.

User Interface of Virtual museum of informatics history in Siberia / Volyanskaya T.A. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 55–94.

The user interface of the Virtual museum of informatics history in Siberia is presented. The museum is intended for acquisition, systematization and use of information about formation and development of informatics in Siberia. The museum is developed as an on-line information-retrieval adaptive hypermedia system. Information management interface and its components (navigation structure, information view, search, insert and update) are described. The user management interface and its components (user registration, authentication, authorization and management) are considered. — Refs: 4 titles.

УДК 519.68 + 681.3.06

Вводный курс программирования на базе языка Zonnon / Касьянова Е.В. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 95-116.

Кратко представлен вводный курс программирования на базе нового языка Zonnon, работа над которым ведется в Цюриховском институте информатики. Разрабатываемый язык задуман как современная альтернатива хорошо известному языку Оберон, являющемуся преемником языков Паскаль и Модула-2.

Курс предназначен главным образом для тех учебных заведений, в которых в настоящее время используется язык Паскаль в качестве языка начального обучения программированию и в которых есть желание плавно перейти к новому курсу программирования, охватывающему концепции современных языков программирования, таких как C#, Java и Ada. — Библиогр.: 15 назв.

Introductory course of programming based on the Zonnon language / Kasyanova E.V. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 95–116.

The paper presents an introductory course of programming based on a new language Zonnon being under development at the Institute of Computer Systems in Zurich. The language is designed as a modern alternative to the well-known language Oberon being an evolution of Pascal and Modula-2 languages.

The course is intended for use in that Russian universities and education organizations which now use the Pascal language in their introductory courses of programming and wish to move to a new course covering the concepts of modern languages, such as C#, Java and Ada. — Refs: 15 titles.

УДК 519.68 + 681.3.06

Обработка данных микрочиповых экспериментов при помощи языка «R» / Коновалова Т.Г., Комашко В.М.// Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 117–136.

В данной статье рассматривается подход к решению задачи выделения генов, изменивших с определенной достоверностью уровень своей экспрессии на основе данных нескольких микрочиповых экспериментов.

На примере обработки данных с олигонуклеотидного чипа компании Affimetrix описан процесс нормализации и кластеризации данных в среде программирования языка *R GUI* с использованием пакетов *Bioconductor*.

Данная работа может быть использована как руководство по применению языка R и пакета BioConductor для обработки результатов микрочиповых экспериментов. — Библиогр.: 2 назв.

Processing of data of microarray experiments with the use of the R language / Konovalova T.G., Komashko V.M. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 117–136.

The aim of this work is to describe an approach to identification of genes that changed their expression in several microarray experiments.

By the example of data from Affimetrix oligonucleotide chip, we describe the process of normalization and clasterization in the R GUI environment using Bioconductor packages.

This work can be used as a tutorial on application of R language and BioConductor package for processing the results of microarray experiments. — Refs: 2 titles.

УДК 519.68 + 681.3.06

Семантическая сеть как формальный метод описания и обработки текстов по преобразованиям программ / Малинина Ю.В. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 137–144.

Электронные вычислительные машины в состоянии обрабатывать только формализованные языки. Так как естественный язык не формализован, то вычислительная система не в состоянии его обрабатывать. Автоматическая обработка смысла текста естественного языка возможна только при наличии его формализованного представления. В статье рассматриваются подход к извлечению и унификации информации, содержащейся в публикациях по преобразованиям программ. За основу берется автоматическое формирование смыслового портрета текста в виде ассоциативной (семантической) сети. Полученная в итоге семантическая сеть может быть использована в дальнейшем для навигации и реферирования текста. — Библиогр.: 11 назв.

Semantic network as a formal method of description and processing of texts about program transformations / Malinina Yu.V. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 137–144.

Computers are able to process only formalized languages but it is not the case with a natural language. This paper discusses the mechanisms used to extract, unify, and express information about program transformations in network notations. A semantic network or net, which is a graphic notation for representing information in patterns of interconnected nodes and arcs, is proposed as a formalized presentation of a text. — Refs: 11 titles.

УДК 519.68 + 681.3.06

Путевые ядра и разбиения в графах с малыми длинами циклов / Мельников Л.С., Петренко И.В.// Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 145–160.

Через $\tau(G)$ будем обозначать количество вершин в наиболее длинном пути графа G. Для некоторой пары натуральных чисел (a,b) такой, что $a+b=\tau(G)$, граф G является (a,b)-*разбиваемым*, если его множество вершин $V(G)$ можно разбить на два класса $\{A,B\}$ таким образом, что $\tau(G[A]) \leq a$, а $\tau(G[B]) \leq b$. Подмножество K множества $V(G)$ называется P_n-*ядром*, если $\tau(G[K]) \leq n-1$ и каждая вершина $v \in V(G) \setminus K$ смежна с вершиной, которая является конечной для пути длины $n-1$ в графе G. Известно, что наличие P_n-ядра в графе G означает, что он является $(\tau(G)-n+1, n-1)$-разбиваемым. В настоящей статье доказана теорема о том, что каждый граф имеет P_9-ядро. — Библиогр.: 21 назв.

Path kernels and partitions in graphs with small lengths of cycles / Mel'nikov L.S., Petrenko I.V. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 145–160.

Let $\tau(G)$ denote the number of vertices in the longest path of a graph G. For some pair of integers (a,b) with property $a+b=\tau(G)$ the graph G is (a,b)-*decomposable,* if the set of vertices $V(G)$ has such partition on two subsets $\{A,B\}$ that $\tau(G[A]) \leq a$ and $\tau(G[B]) \leq b$. The subset K of the set of vertices $V(G)$ is called P_n-*kernel,* if $\tau(G[K]) \leq n-1$ and every vertex $v \in V(G) \setminus K$ are adjacent with the terminal vertex of the path of length $n-1$ in the graph G. It is known that the existence of P_n-kernel in the graph G implies that the graph G is $(\tau(G)-n+1, n-1)$-decomposable. Here the theorem on existence of P_9-kernel in any graph G has been proved. — Refs: 21 titles.

УДК 519.68 + 681.3.06

Обзор виртуальных музеев в сети Интернет / Несговорова Г.П. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 161–172.

Дается обзор существующих в сети Интернет так называемых виртуальных музеев и делается попытка установить различия между сайтами — представительствами реальных музеев и собственно виртуальными музеями. — Библиогр.: 6 назв.

Review of virtual Internet-museums / Nesgovorova G.P. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 161–172.

A review of the so-called virtual Internet-museums is given and an attempt is made to tell the difference between the sites-representations of real museums and virtual museums themselves. — Refs: 6 titles.

УДК 519.68 + 681.3.06

Метод распараллеливания алгоритмов унимодулярными преобразованиями / Осмонов Р.А. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 173–184.

Цель данной статьи — представить теорию унимодулярных матриц, применяемую для распараллеливания циклов. Рассматриваемый в статье метод волнового фронта (Wavefront) включает в себя такие преобразования, как перестановка цикла, обращение цикла и скашивание цикла. В качестве программной модели берется гнездо из двух циклов с постоянными границами цикла. В статье изучается преобразование гнезда циклов посредством унимодулярной матрицы, действующей на индексные переменные. Показываются корректность применения перечисленных выше преобразований; сопоставление матриц соответствующим преобразованиям; возможность параллельного исполнения внешнего или внутреннего преобразованных циклов; существование матриц, позволяющих параллельное исполнение; вычисление границ цикла нового гнезда. Описываемые преобразования меняют относительный порядок исполнения итераций гнезда цикла и используются как для выявления параллелизма, так и для повышения его степени. — Библиогр.: 5 назв.

A method of algorithm parallelization by unimodular transformations / Osmonov R.A. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 173–184.

The aim of this paper is to present the theory of unimodular matrices used for loop parallelization. The wave front method here considered includes conversions, such as loop interchange, loop reverse and loop skewing. A nest of two loops with constant boundaries is taken as a program model. A transformation of the loop nest by means of a unimodular matrix that operates on index variables is studied in the paper. The following results are shown: correctness of application of the listed above transformations; assignment of matrices to appropriate transformations; possibility of parallel execution of the outer or inner transformed loop; existence of matrices that allow parallel execution; calculation of boundaries of a loop of a new nest. The described transformations change the relative order of execution of iterations of the loop nest and are used both to find out parallelism and to increase its degree. — Refs: 5 titles.

УДК 519.68 + 681.3.06

Блок редукции в компиляторе SISAL 3.0 / Пыжов К.А. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 185–196. Описан блок редуцирующих оптимизаций на внутреннем представлении IR1 для языка Sisal 3.0, т. е. оптимизирующих трансформаций IR1, гарантированно не ухудшающих каких-либо свойств программы. Реализованный блок оптимизаций выполняет такие высокоуровневые преобразования, как удаление общих подвыражений, свертка констант, протяжка констант, упрощение условных выражений, удаление мертвого кода и т.д. — Библиогр.: 8 назв.

A block of reducing optimizations for Sisal 3.0 compiler / Pyzhov K.A. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 185–196.

The article describes a block of high-level reducing optimizations for Sisal 3.0

compiler. The reducing optimizing transformations are transformations which do not decrease any qualities of a program. The block performs the following high-level optimizations: Common Subexpression Elimination, Constant Folding, Constant Propagation, IF-transformations, Dead Code Elimination, etc. — Refs: 8 titles.

УДК 519.68 + 681.3.06

Анализ модульного подхода и его применение в различных языках программирования / Синяков А.И. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 197–228.

В данной работе рассматриваются несколько основных подходов в программировании: модульный, структурный и объектно-ориентированный. Во второй части работы анализируется реализация модульности в некоторых языках программирования. В заключение приводится реализация модульности в функциональном языке Sisal 3.0. — Библиогр.: 8 назв.

Analysis of a module approach and its implementation in different programming languages / Sinyakov A.I. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 197–228.

This work considers several basic programming methods, such as modular, structured and object-oriented. In the second part of the paper, we analyze implementation of a module in some programming languages and illustrate it with a functional language Sisal 3.0. — Refs.: 8 titles.

УДК 519.68 + 681.3.06

Система интерфейсов транслятора во внутреннее представление IR1 / Стасенко А.П. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 229–238.

В статье кратко описывается система COM (Component Object Model) интерфейсов, задающих трансляцию из некого текстового представления программы во внутреннее представление IR1, основанное на графовой модели IF1. Приводятся требования к желаемой функциональности системы интерфейсов и возможные направления её расширения. — Библиогр.: 11 назв.

The interface system for a translator to the internal representation IR1 / Stasenko A.P. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 229–238.

This paper describes the COM (Component Object Model) interface system which is used to represent translation from the generic textual program representation to internal representation IR1 based on the IF1 graph model. Requirements to the functionality of the interface system are presented, as well as possible ways of its enhancement. — Refs: 11 titles.

УДК 519.68 + 681.3.06

Обзор средств отладки программ на функциональных языках / Хан Ю. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 239–246.

Функциональная парадигма вносит свои особенности в процесс отладки программ. К некоторым языкам традиционные методы отладки практически неприменимы. В данной статье рассматривается ряд систем программиро-

вания на функциональных языках и предпринимается попытка классифицировать их по применяемым методикам отладки. — Библиогр.: 15 назв.

Overview of debugging techniques for functional languages / Khan Yu. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. 239–246.

The functional paradigm introduces its special features to the debugging process. Traditional debugging techniques are not applicable to some functional languages. In this article, a series of functional programming systems are reviewed and an attempt is made to classify the systems according to the debugging techniques employed. — Refs: 15 titles.

УДК 519.68 + 681.3.06

Анализ различных участков ДНК с помощью автокорреляционной функции / Черемушкин Е.С. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 247–252.

Целью данной работы было изучение структуры ДНК с помощью автокорреляционного анализа: найти некоторые различия, которые возможно не будут способствовать распознаванию неизвестных участков, но которые характеризуют качественные различия ДНК разных функций.

В результате исследований выяснилось, что АКФ различных участков практически совпадают, но видимые различия статистически значимы. А именно, регуляторные районы более скоррелированны, чем экзоны (кодирующие районы), а те более скоррелированны, чем случайные последовательности.

Промоторы (регуляторные участки) — гораздо более разнообразны по своей информационной насыщенности, чем другие последовательности. Природе «выгодно» поддерживать общее однообразие ДНК и только функционально важные участки имеют свой «уникальный» паттерн. — Библиогр.: 1 назв.

Analysis of different types of DNA with autocorrelation function / Cheryomushkin E.S. // Tools and techniques of program construction and optimization. — Novosbirsk, 2005. — P. 247–252.

The aim of this work is to study the DNA structure with autocorrelation function (AKF) analysis: to find some qualitative differences for DNA that serves different biological function.

The result of our research is that AKF of different DNA regions are similar, but differences are statistically significant. Regulatory regions are more correlated than exons (coding regions) which, in turn, are more correlated than randomly generated sequences.

The class of promoters (regulatory regions) has much more variable informational content than other sequences. Nature gets profit from supporting common informational similarity of DNA, but only functionally necessary parts have their own unique pattern. — Refs: 1 titles.

УДК 519.68 + 681.3.06

Построение программного комплекса “Regulatory Sequences Analyzer” для распознавания цис-элементов в последовательностях ДНК / Штокало Д.Н., Черемушкин Е.С. // Методы и инструменты конструирования и оптимизации программ. — Новосибирск, 2005. — С. 253–264.

В данной статье представлена документация к программному комплексу "Regulatory Sequences Analyzer", разработанного авторами, для визуализации поиска потенциальных цис-элементов последовательности ДНК. Приводится краткое описание используемых программным комплексом алгоритмов Match и CoMatch, основанных на привлечении весовых матриц для поиска простых и двойных сайтов соответственно. Приводится описание матричных библиотек для данных алгоритмов.— Библиогр.: 5 назв.

Construction of a programme complex "Regulatory Sequences Analyzer" for recognition of cis-elements in the DNA sequence / Shtokalo D.N., Cheryomushkin E.S. // Tools and techniques of program construction and optimization. — Novosibirsk, 2005. — P. .

This article presents documentation for the program complex "Regulatory Sequence Analyzer" developed by the authors for visualization of a search for potential cis-elements in the DNA sequence. The article provides a brief description of "Match" and "CoMatch" algorithms that are based on using weight matrices in a search for singular and double sites, correspondingly. The description of matrix libraries for these algorithms is given. — Refs: 5 titles.

МЕТОДЫ И ИНСТРУМЕНТЫ КОНСТРУИРОВАНИЯ И ОПТИМИЗАЦИИ ПРОГРАММ

Под редакцией
проф. Виктора Николаевича Касьянова

Рукопись поступила в редакцию 15. 02. 2005
Ответственный за выпуск Г.П. Несговорова
Редактор З.В. Скок

Подписано в печать 14. 07. 2005
Формат бумаги 60 × 84 1/16 Объем 15,8 уч.-изд.л., 17,3 п.л.
Тираж 75 экз.

ЗАО РИЦ "Прайс-курьер" 630090, г. Новосибирск, пр. Акад. Лаврентьева, 6,
тел. (383) 330-72-02