671

ISSN 0136-3549
0320-3409

TALLINNA POLÜTEHNILISE INSTITUUDI

TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

ОБРАБОТКА ДАННЫХ.
ПОСТРОЕНИЕ ТРАНСЛЯТОРОВ.
ВОПРОСЫ ПРОГРАММИРОВАНИЯ

TALLINN 1988

671

ALUSTATUD 1937

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

TRANSACTIONS OF TALLINN TECHNICAL UNIVERSITY

УДК 681.3

ОБРАБОТКА ДАННЫХ. ПОСТРОЕНИЕ ТРАНСЛЯТОРОВ. ВОПРОСЫ ПРОГРАММИРОВАНИЯ

Труды экономического факультета LXVI

TALLINN 1988

ТАЛЛИНСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Труды ТПИ № 671

ОБРАБОТКА ДАННЫХ, ПОСТРОЕНИЕ ТРАНСЛЯТОРОВ.
ВОПРОСЫ ПРОГРАММИРОВАНИЯ
Труды экономического факультета LXУ1
На русском языке
Отв. редактор И. Амитан
Техн. редактор В. Ранник
Сборник утвержден коллегией Трудов ТПИ 29.06.88
Подписано к печати 09.12.88
МВ-05944
Формат 60x90/16
Печ. л. 11,0 + 0,5 приложение
Уч.-изд. л. 9,2
Тираж 400
Зак. № 789
Цена 1 руб. 80 коп.
Таллинский политехнический институт,
200108, Таллин, Эхитаяте тее, 5
Ротапринт ТПИ, 200006, Таллин, ул. Коскла, 2/9

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.06

Я.Я. Тепанди

ТЕСТИРОВАНИЕ БАЗ ЗНАНИЙ

Хорошо известно, что тестирование - наиболее трудоемкий и сложный этап разработки программного обеспечения [1, 2]. Стоимость тестирования системы программирования составляет 30-80 % от общей стоимости разработки. Тестирование программы требует более высокой квалификации, чем проектирование или написание этой программы [3].

Важность тестирования была полностью осознана только в связи с появлением программных систем, требующих повышенной надежности - систем реального времени, систем зарплаты, систем резервирования билетов и т.д. В области конструирования баз знаний (БЗ) наблюдаются аналогичные проблемы. Очень трудно показать (и тем труднее доказать), что БЗ действительно полностью соответствует своему назначению - сейчас даже не ясно, что значит "соответствует своему назначению". Между тем, все больше появляется экспертных систем, к которым предъявляются повышенные требования по надежности. Это системы управления полетами [4], конструирования СБИС [5] и многие другие. Для таких систем требуется методика проверки корректности БЗ.

Уточним сначала разницу между тестированием и отладкой. Под тестированием БЗ будем понимать деятельность, направленную на обнаружение ошибок в БЗ. Мы будем различать два вида тестирования: тестирование по содержанию БЗ (метод "белого ящика" по аналогии с программированием) и тестирование по назначению БЗ (тестирование по спецификации, по методу "черного ящика"). Отладка - это устранение выявленных в БЗ ошибок. Большинство работ, в которых затрагиваются вопросы качества и надежности БЗ, посвящены исследованию и разработке средств отладки БЗ [10].

Типичные ошибки в экспертных системах

Когда могут появляться ошибки в экспертной системе, а в частности - в БЗ? Система может ошибочно реагировать на любом шаге диалога. Мы будем различать следующие компоненты диалога: вопрос - задается либо системой пользователю, либо наоборот; ответ - сообщается системой пользователю или наоборот; объяснение - выдается системой; заключение - выдается системой как окончательный результат консультации.

В ходе диалога система может делать следующие ошибки, обусловленные недостаточным качеством БЗ.

I. Неправильное, не соответствующее действительности заключение. Причина этой ошибки - неправильно построенная БЗ.

2. Отсутствующее заключение (система зациклится) из-за ошибки в БЗ.

3. Правильное, но недостаточно информативное заключение. Причина - в БЗ мало сведений для выдачи нужного заключения, хотя на основе сообщенных пользователем ответов можно было бы выдать такое заключение. Частный случай этой ошибки - система заканчивает работу, но выдает заключение "не знаю".

4. Бесполезное, хотя и правильное и информативное заключение [6]. Например, если человеку необходимо вычислить логарифм от двойки и он задает вопрос "Сколько времени вычисляет ЕС-1055 логарифм от двойки?", то ответ " меньше 0,001 сек" и верный и информативный, но бесполезный - надо ответить что-то вроде "Используйте карманный калькулятор" или "Ответ равен 0.301".

5. Неэффективное заключение - система спрашивает слишком много или работает слишком долго, то же заключение можно было выдать быстрее или на основе меньшего количества вопросов. Причина ошибки - в БЗ мало сведений или они неудачно представлены.

6. Ошибки типа 3 и 4, а может быть и остальные, могут также встретиться в объяснениях.

Причины и выявление ошибок

Наиболее просто (на синтаксическом уровне) обнаруживаются ошибки типа 2. Проверка на существование циклов в БЗ имеется во многих экспертных системах [7, 8]. Как правило, эта проверка проводится статистически, учитывая только содержание БЗ. Это проще, чем динамическая проверка во время работы с БЗ, но ограничивает типы допустимых правил (запрещается иметь цепь правил вида $A \rightarrow B$, $B \rightarrow C$, ..., $E \rightarrow A$, хотя для конкретной БЗ и в конкретной ситуации обращения к системе такая цепь не обязательно привела бы к зацикливанию работы с БЗ). На синтаксическом уровне можно обнаружить также частный случай ошибок типа 3 (никакого заключения не будет выдано). Такая проверка будет реализовываться в системе ТЕСТСПЕЦ [9].

Построение системы, в которой можно избежать ошибок типа 4, требует весьма тонкого анализа действительной цели пользователя. В построенных до настоящего времени экспертных системах такой анализ, как правило, не проводится. Поэтому эти системы способны реагировать на запрос, но не могут распознавать необходимость активной перестройки намерений пользователя. Вследствие этого, о тестировании для нахождения этого типа ошибок пока рано говорить - даже если такая ошибка обнаружится, то в системе нет средств для ее предотвращения. Другими словами, об устранении таких ошибок пока следует говорить не на уровне БЗ, а на уровне самой экспертной системы.

Ошибки типа 6 часто встречаются в экспертных системах. Как пользователи, так и разработчики согласны, что объяснения, данные в экспертных системах, с одной стороны, слишком распространенные, а с другой стороны, трудные для понимания. Поскольку текст объяснения строится на основе текстов в БЗ и вывода заключения, построенного системой, то улучшением содержания БЗ можно добиться значительного улучшения качества объяснений, данных системой. Однако весьма возможно, что этого недостаточно - необходимо радикальное изменение принципов выдачи объяснений. Предпринимаются попытки выдачи информации (и объяснений) в графическом виде, в виде сводных таблиц и т.д. [11].

Ошибки типа I, 3, 5 могут также являться результатом ошибок в БЗ или результатом несоответствия самой экспертной системы решаемым задачам. Консультант, поддерживающий тестирование БЗ, должен обнаруживать обе причины. Однако техника обнаружения этих причин совершенно различная. Если в первом случае можно генерировать тестовые данные и испытанием системы установить ее (не)соответствие поставленным задачам, то во втором случае необходим анализ системы и ее предметной области, с тем чтобы установить их несоответствие. Кстати, для тестирования программ этот вопрос тоже не решен. Вопрос о возможном несоответствии языка программирования или операционной системы не ставится и не решается в системах, поддерживающих тестирование программ. Причина в том, что такой анализ требует особого подхода, например, на основе экспертных систем.

Таким образом, ошибки в работе экспертной системы могут оказаться результатом несоответствия инструментальной системы решаемым задачам или результатом ошибок в БЗ. Вопросы о типах и особенностях решаемых экспертными системами задач, а также о принципах выбора инструментальной системы рассматриваются, например, в [I2]. Первое приближение к решению проблемы оценки выбора инструментальной системы может быть таковым: если обнаружена ошибка в БЗ, а средства данной инструментальной системы недостаточны для исправления этой ошибки, то приходится рассматривать предположение о неадекватности инструментальной системы. Рассмотрим далее возможности нахождения ошибок (тестирования) БЗ.

Тестирование базы знаний

По методу "белого ящика" обнаруживаются ошибки типа 2 и частный случай ошибок типа 3. Для того, чтобы обнаружить остальные ошибки, необходимо сравнение заключения системы, хода выбора заключения или объяснений с некоторыми "правильными" действиями системы. Делать это можно только тогда, когда нам известны "правильные" действия. В большинстве случаев это не так, поэтому о полной автоматизации тестирования БЗ в общем случае вряд ли можно говорить. Что можно - это генерировать тестовые наборы ответов пользова-

теля и консультировать по вопросам тестирования БЗ. Для генерации тестовых наборов можно исходить из самой БЗ (опять метод "белого ящика") или решаемой задачи (метод "черного ящика"). В последнем случае необходимо иметь спецификацию задачи дополнительно к БЗ. Здесь можно возразить, что раз уже есть спецификация задачи, то БЗ уже не нужна - достаточно оттранслировать спецификацию и использовать ее для решения задачи. Второе возражение - кто гарантирует, что спецификация верна? На оба эти возражения ответ один - спецификация не должна быть детальным и полным описанием задачи, а скорее эскизом, проектом задачи. Такой проект недостаточен для решения задачи, но в то же время более понятен пользователю (легче убедиться в его корректности) и может использоваться для генерации тестов БЗ.

Спецификация. Пусть БЗ основана на правилах, т.е. представляет собой совокупность условных предложений. Каждое предложение задает зависимость некоторого утверждения от конъюнкции или дизъюнкции некоторых других утверждений, а также вес или вероятность выводимости утверждения. Мы будем делать три допущения относительно задачи.

I. Все утверждения системы можно сгруппировать, исходя из характера задачи.

2. Между группами утверждений можно установить связи, причем если одна группа влияет на другую, то не прямого, не косвенного (через промежуточные группы) обратного влияния нет.

3. Внутри каждой группы можно установить частичное упорядочение утверждений по некоторому признаку. Таких упорядочений может быть несколько, но в дальнейшем, принимаем, что существует лишь одно упорядочение.

Эти ограничения достаточно сильные, однако существует весьма много задач, для которых они удовлетворены. Исходя из этих ограничений, мы можем представить БЗ как ориентированный ациклический граф

$$\Gamma = (В, Д),$$

где В - множество вершин, а $Д \subseteq В \times В$ - множество дуг. Каждая вершина $в \in В$ представляет собой частично упорядочен-

ное множество (т.е. опять-таки ориентированный ациклический граф)

$$\mathfrak{b} = (Y, \pi),$$

где Y - множество утверждений, а $\pi \subseteq Y \times Y$ - частичное упорядочение на Y. Например, БЗ в системе ТЕСТСПЕЦ представляется в виде рис. I.

- система не оттестирована для данной задачи.

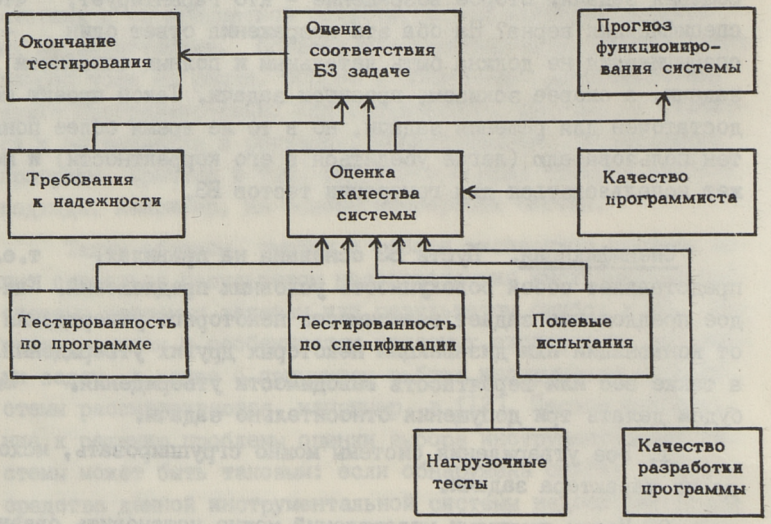


Рис. 1.

Каждая вершина на этом графе объединяет множество утверждений и правил, относящихся к одной категории, причем про некоторые утверждения можно сказать, что они "сильные" или "лучшие", чем другие. Например, вершина "оценка соответствия задаче" включает следующие утверждения:

- система очень хорошо оттестирована для данной задачи;
- система хорошо оттестирована для данной задачи;
- система оттестирована на уровне "достаточно для статьи";
- система удовлетворительно оттестирована для данной задачи;
- система не оттестирована для данной задачи.

Система правил в ТЕСТСПЕЦ содержит более 100 правил и ее трудно обозреть, схема же на рис. 1 вполне понятна для пользователя.

Генерация тестов. При тестировании БЗ, так же как и при тестировании программ, нельзя задать "исчерпывающего набора" тестов, так как этот набор практически бесконечен (при непрерывных значениях весов да/нет ответов, а тем более для ответов, принимающих значение из числового интервала). Это значит, что надо ограничивать набор тестов, но так, чтобы он остался репрезентативным, т.е. выявлял бы максимальное число ошибок в БЗ по сравнению со всеми возможными наборами того же объема. Основная возможность сокращения числа тестов - это выяснение классов эквивалентных тестов. Эквивалентные тесты могут выяснять одни и те же ошибки в БЗ. Далее, ошибки очень часто выявляются в граничных ситуациях - например, при максимальных значениях ответов или утверждений в группах утверждений. С граничными значениями сходны особые ситуации - например, когда ответов вообще не надо, или когда все ответы одинаковы. Наконец, необходимо оттестировать неправильные входные данные.

Алгоритмы генерации тестов будут приведены в отдельной статье. Изложенные в настоящей статье принципы тестирования БЗ приняты за основу построения консультационной системы по тестированию и оценке БЗ, являющейся расширением и модификацией экспертной системы тестирования программ ТЕСТСПЕЦ.

Литература

1. Липаев В.В. Тестирование программ. М.: Радио и связь, 1986. 296 с.

2. Майерс Г. Искусство тестирования программ. М.: Финансы и статистика, 1982. 174 с.

3. Фокс Дж. Программное обеспечение и его разработка. М.: Мир, 1985. 367 с.

4. Scarl E. et al. Knowledge-based fault monitoring and diagnosis in Space shuttle propellant loading // Proc. IEEE of National Aerospace and Electronics Conference. 1984. P. 768-774.

5. Schindler M. Artificial intelligence begins

to pay off with expert systems for engineering // Electronic Design. August, 1984. Vol. 32. P. 106-146.

6. M c C o y K. Misconception responses that change with context // Proc. of the 1st Conf. on Artificial Intelligence Applications. Denver, 5-7 Dec. 1984, Silver Spring, 1984. P. 530-536.

7. H á j e k P., H á j k o v a M. The consulting system EQUANT - brief description and user's manual // New Enhancements in GUHA Software. 1984. N 8. P. 21-36.

8. A g r e G. et al. An implementation of the expert system DIGS for diagnostics // Computers and Artificial Intelligence. 1985. Vol. 4, N 6. P. 495-502.

9. Т е п а н д и Я.Я. Экспертная система тестирования программ по спецификациям // Всесоюзная конференция "Автоматизация построения систем программирования", Таллин,1986: Тезисы докладов.

10. D a v i s R., L e n a t D.B. Knowledge-based systems in artificial intelligence. McGraw-Hill Inc., 1982.

11. H a y e s - R o t h F. The knowledge-based expert system: a tutorial // Computer. September, 1984. P. 11-28.

12. H a y e s - R o t h F., W a t e r m a n D.A., L e n a t D.B. Building expert systems // Addison-Wesley, Reading, Massachusetts, 1983.

J. Tepandi

Knowledge Base Testing

Abstract

A classification of errors in expert systems is given. Some possible reasons for those errors are discussed. A specification-based method for knowledge base testing is proposed.

J. Tepandi

Teadmusbaaside testimine

Kokkuvõte

Põhjendatakse ekspertsüsteemide teadmusbaaside testimise vajalikkust. Antakse vigade klassifikatsioon ekspertsüsteemides ja tuuakse nende põhjused. Pakutakse välja teadmusbaasi testimise põhimõtted.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.06

Я.Я. Тепанди

ИНСТРУМЕНТАЛЬНАЯ ЭКСПЕРТНАЯ СИСТЕМА ХЭЛИ

Инструментальные экспертные системы (ИЭС) - быстро развивающаяся отрасль технологии программирования [1-3]. Ниже предлагаются основные принципы реализации ИЭС ХЭЛИ и приводится структура базы знаний (БЗ) системы.

1. Общая характеристика системы. Система ХЭЛИ основана на продукциях. Поиск ведется от цели или от данных. Механизм обработки неопределенностей похож на аналогичные механизмы в системах PROSPECTOR, EQUANT [4]. Система включает следующие основные компоненты: подсистемы запуска, диалога, вывода заключений, объяснений, помощи; базу знаний (БЗ); доску.

Подсистема запуска загружает модули системы и базу правил, а также запускает систему. Подсистема диалога осуществляет весь диалог с пользователем. Отвечая на вопросы системы, пользователь может вводить численные ответы по определенной ранее шкале весов, а также прекращать диалог, возвращаться к началу диалога, обращаться за объяснениями, переходить на ввод новых фактов. Доска включает сведения, приобретенные или выведенные во время сеанса с пользователем.

Система поддерживает следующие виды объяснений:

- общая консультация по системе ХЭЛИ;
- консультация по вопросам, относящимся к данной ситуации общения с системой ХЭЛИ;
- консультация по текущей базе знаний;
- объяснения о ходе рассуждений системы по текущей базе знаний и по данным ответам пользователя типа "Как" (как получен этот факт?) и "Почему" (почему задан этот вопрос?).

Система ХЭЛИ получена путем извлечения ядра экспертной системы из ассистента по тестированию ТЕСТЕР [5]. Она реализована на ЭВМ ЕС-1840, на языке Микро-Пролог, в операционной системе MS-DOS.

2. База знаний. База знаний составляется администратором системы. Она включает следующие компоненты: комментарии, тему, шкалу, сокращения, цели, максимальные весы, априорные весы значений аргументов, дополнительные вопросы системы, дополнительные ответы на вопросы типа "Как" и "Почему", процедуры, продукции. Необходимыми из них являются цели и правила, остальные могут и отсутствовать - в этом случае используются их значения по умолчанию. Порядок следования компонентов БЗ важен лишь постольку, поскольку он влияет на понимание содержания БЗ.

Комментарии служат для улучшения чтения текста БЗ. Тема описывает назначение БЗ. Шкала определяет диапазон ответов пользователя на вопросы системы. Например, (шкала 3) задает диапазон ответов от -3 до 3, где -3 соответствует ответу "нет", 0 - ответу "не знаю", 3 - ответу "да". По умолчанию, значение шкалы принимается равным 5. Сокращения могут использоваться как для имен атрибутов, так и для имен их значений. Если введено сокращение, то в БЗ следует использовать только сокращение, а не само имя.

Цели выводятся на экран в начале сеанса, чтобы пользователь выбрал из них нужную. Максимальные весы могут быть установлены как для атрибутов, так и для пар (атрибут, значение). В обоих случаях они определяют вес, при достижении которого можно прекратить вычисления. В первом случае, этот вес одинаков для всех значений атрибута, во втором случае - для выделенного значения устанавливается свой собственный вес (например, когда ставится диагноз тяжелой болезни, то желательно проявлять большую осторожность). По умолчанию максимальные весы равны значению шкалы. Априорные весы значений атрибутов принимаются по умолчанию равными нулю (т.е. значение не известно). Вопросы системы генерируются либо автоматически (имеют вид: "атрибут значение?", например: "все-тесты-по-спецификациям проведены?"), либо задаются в БЗ произвольным образом. Ответы на вопросы "Как" и "Почему"

могут также генерироваться автоматически или задаваться в БЗ. С парой (атрибут, значение) может связываться процедура, которая запускается в момент окончательного установления веса этой пары.

Основное содержание БЗ заложено в продукциях. Продукция определяет посылку (условие), следствие (заключение) и вес. Пример продукции в слегка отредактированном формате: (Продукция I6 (тестирование по спецификациям очень хорошее, если все тесты по спецификациям завершены и спецификация детальная) 4). Часть продукции во внутренних скобках до связки "если" - заключение, остальная часть - посылка. Число 4 задает вес продукции.

3. Обработка весов. Вывод заключения в системе ХЭЛИ существенно зависит от механизма обработки неопределенностей, поэтому объясним этот механизм подробнее. С каждым утверждением и правилом системы ассоциируется вес - целое число между -Ш и Ш, где Ш - значение шкалы. Рассмотрим утверждение У вида "атрибут есть значение".

Вес утверждения У образуется в целом как комбинация (I) его априорного веса, (2) весов, вычисляемых из правил, (3) веса, сообщаемого пользователем. Представление о комбинировании этих весов дает процедура ВЫВОДИ в алгоритме А (см. ниже). Прокомментируем здесь два аспекта: образование веса утверждения из правила и комбинирования весов, полученных из разных источников.

Допустим, что имеется правило типа "Если посылка, то У с весом В", притом априорный вес утверждения У равен А. Тогда можно нарисовать график образования веса заключения в зависимости от веса посылки (толстая линия на рис. I)[4]. Этот график - не единственно возможный. Он имеет следующий смысл, довольно хорошо согласующийся с интуицией. Прежде всего, вес посылки вычисляется как минимум весов всех утверждений, входящих в посылку. Аналогично вычисляется априорный вес. Если вес меньше априорного, то информация о посылке имеет "отрицательный" характер. В частности, если априорный вес посылки равен нулю, то отрицательный вес посылки означает вероятность того, что условие посылки не выполнено. Но в таком случае данное правило неприменимо и

АЛГОРИТМ А. ВЫВОД ОДНОГО ЗАКЛЮЧЕНИЯ.

```
начало
    спроси цель Ц:
    определи (используя процедуры ЗНАЧЕНИЕ и ВЫВОДИ) первое значе-
      ние З цели Ц, вес которого превышает МАКС-ВЕС (Ц, З), или если
      такого значения нет - значение З цели Ц с максимальным весом:
    сообщи значение З и его вес
конец
процедура ВЫВОДИ (АТРИБУТ, ЗНАЧЕНИЕ, ВЕС):
начало
    коммент  искомый вес может быть уже вычислен заранее:
    если факт Ф относительно пары (АТРИБУТ,ЗНАЧЕНИЕ) существует
    то ВЕС := вес из факта Ф
    иначе
        коммент выводи и запомни новый факт:
        АПР := АПРИОРИ (АТРИБУТ, ЗНАЧЕНИЕ):
        ВЕС := АПР:
        коммент предполагается, что априорный вес меньше
                    максимального, поэтому вычисления продолжаются:
        повтори  для всех правил вида ((правило Н (АТРИБУТ есть
                    ЗНАЧЕНИЕ если ПОСЫЛКА) ВЕС-ПР))
            ВЕСА-ПОСЫЛКИ (ПОСЫЛКА, АПР-ВЕС-ПОС, ВЕС-ПОС):
            В := ВЕС-ИЗ-ПРАВИЛА (АПР, АПР-ВЕС-ПОС, ВЕС-ПОС, ВЕС-ПР):
            ВКЛАД-ПРАВИЛА := СУММА-ВЕСОВ (В, -АПР):
            ВЕС := СУММА-ВЕСОВ  (ВЕС, ВКЛАД-ПРАВИЛА):
            если ВЕС > МАКС-ВЕС (АТРИБУТ, ЗНАЧЕНИЕ) то выход
        конец повтори:
        коммент может быть,надо еще спросить ответ у пользователя:
        если ВЕС < МАКС-ВЕС (АТРИБУТ, ЗНАЧЕНИЕ)  и
            МОЖНО-СПРОСИТЬ (АТРИБУТ, ЗНАЧЕНИЕ)
        то СПРОСИ (АТРИБУТ, ЗНАЧЕНИЕ, ВЕС-ИЗ-ВОПРОСА):
            ВЕС := СУММА-ВЕСОВ (ВЕС, ВЕС-ИЗ-ВОПРОСА)
        конец если
        запомни факт (АТРИБУТ,ЗНАЧЕНИЕ,ВЕС)
    конец если
конец ВЫВОДИ.
```

мы заключаем, что правило не дает новой информации – вес заключения равен его априорному весу. В случае же, когда вес посылки равен максимальному (т.е. значению шкалы), целесообразно считать, что вес заключения тоже максимально допустимый для данного правила, т.е. равен весу правила. В промежутке между этими двумя значениями можно допустить, что зависимость между весом посылки и весом заключения является линейной (рис. I). Из этого графика получим формулу, используемую в функции ВЕС-ИЗ-ПРАВИЛА алгоритма А.

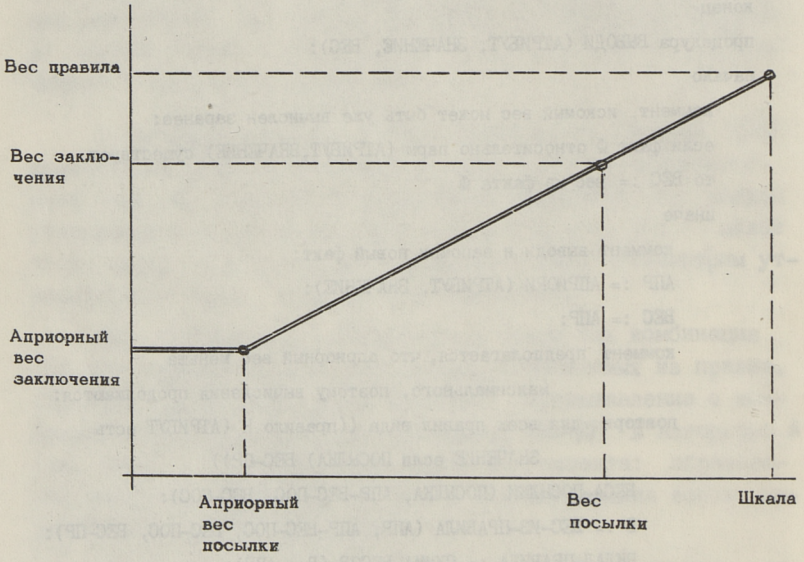


Рис. 1.

Для комбинирования двух весов В1 и В2, полученных из разных источников, используется операция <+>, определенная следующим образом (Ш – значение шкалы):

$$B1 \langle + \rangle B2 = Ш * (B1/Ш + B2/Ш) / (I + (B1/Ш) * (B2/Ш)).$$

Эта операция является модификацией операции, определенной как (В1 + В2) / (I + В1 * В2) для весов в промежутке от нуля до единицы. Операция <+> обладает следующими "хорошими" в данной ситуации свойствами: она ассоциативна, сохраняет шкалу (если оба операнда находятся в

промежутке от -Ш до Ш, то результат находится в том же промежутке). Далее, если один операнд равен Ш (абсолютная уверенность), а другой не равен -Ш, то результат равен Ш; если же один операнд равен 0 (ничего не известно), то результат равен другому операнду. Если один операнд равен -Ш ("нет"), а другой не равен Ш, то результат равен - Ш. Операция <+> не определена, когда один операнд равен Ш ("да"), а другой операнд равен -Ш ("нет").

4. Вывод заключения. При работе с системой, пользователь может требовать поиск одного или всех значений данного атрибута. В обоих случаях используется процедура ВЫВОДИ (АТРИБУТ, ЗНАЧЕНИЕ, ВЕС). Приведем алгоритм для первого случая, вместе с алгоритмом для процедуры ВЫВОДИ. Для записи алгоритма используется псевдо-алгол из [6].

В алгоритме вызываются следующие функции и процедуры:

- процедура ЗНАЧЕНИЕ (Ц, З, В) выдает очередное значение З и его вес В для данной цели Ц (при первом обращении выдается первое значение; если значений больше нет, выдается ПУСТО);

- функция АПРИОРИ(А,З) - выдает априорный вес утверждения "А есть З";

- функция МАКС-ВЕС(А) - выдает максимальный вес атрибута А, или - если этот вес не определен - значение шкалы;

- функция МАКС-ВЕС(А,З) - выдает максимальный вес утверждения "А есть З", или - если этот вес не определен - значение МАКС-ВЕС (А);

- функция ШКАЛА - выдает значение шкалы (по умолчанию число 5);

- процедура ВЕСА-ПОСЫЛКИ(ПОСЫЛКА,АПР,ВЕС) - выдает априорный вес АПР и вес ВЕС посылки ПОСЫЛКА. Предположим, что ПОСЫЛКА имеет вид "АI есть ЗI и ... и АК есть ЗК". Тогда АПР вычисляется как минимум значений АПРИОРИ (АI, ЗI), ..., АПРИОРИ (АК, ЗК). Вес ВЕС определяется как минимум из значений ВI, ... , ВК, вычисленных через ВЫВОДИ (АI, ЗI, ВI), ..., ВЫВОДИ (АК, ЗК, ВК);

- функция ВЕС-ИЗ-ПРАВИЛА(АПР-ВЕС, АПР-ВЕС-ПОСЫЛКИ, ВЕС-ПОСЫЛКИ, ВЕС-ПРАВИЛА) - выдает вес заключения "А есть З" по

его априорному весу АПР-ВЕС, по априорному весу посылки правила, по весу посылки правила и по весу правила (рис. I). Если ВЕС-ПОСЫЛКИ < АПР-ВЕС-ПОСЫЛКИ, то функция равняется АПР-ВЕС. В противном случае она вычисляется как АПР-ВЕС + + (ВЕС-ПРАВИЛА - АПР-ВЕС) (ВЕС-ПОСЫЛКИ - АПР-ВЕС-ПОСЫЛКИ)/(ШКАЛА - АПР-ВЕС-ПОСЫЛКИ);

- функция СУММА-ВЕСОВ(ВЕС1, ВЕС2) вычисляет сумму весов по формуле: СУММА-ВЕСОВ = ШКАЛА * (Р1 + Р2)/(I + Р1 * Р2), где Р1 = ВЕС1 / ШКАЛА, Р2 = ВЕС2 / ШКАЛА;

- функция МОЖНО СПРОСИТЬ(А, З) выдает значение "истина", если в БЗ нет правил с заключением "А есть З" или же имеется вопрос по паре (А, З)). В противном случае выдается "ложь";

- функция СПРОСИ (А, З, В) задает пользователю вопрос "А З" (если в базе знаний нет вопроса по паре (А, З)) или (в противном случае) вопрос, определенный парой (А, З). Потом считывается ответ В.

Для упрощения представления, в алгоритме не отражены некоторые аспекты обработки (использование сокращений, выдача объяснений). Отметим только, что во время поиска вывода, в основном, используются сокращения; длинные названия нужны лишь для общения с пользователем. Выдача объяснений типа "Как" основана на содержании доски. Выдача объяснений типа "Почему" является более сложной. Для ее осуществления в процедурах вывода заключения используется дополнительный параметр, содержащий информацию о цели, правилах и фактах, которые привели к появлению данного вопроса.

Литература

I. Представление знаний в человеко-машинных и робототехнических системах. Том С. М.: ВЦ АН СССР, ВИНИТИ, 1984. 379 с.

2. Buchanan B.G. Principles of rule-based expert systems // Advances in Computers. 1983. Vol. 22. P. 163-216.

3. Реальность и прогнозы искусственного интеллекта. М.: Мир, 1987. 245 с.

4. H á j e k P., H á j k o v a M. The consulting system EQUANT - brief description and user's manual // New Enchancements in GUHA Software. N 8. / Edited by P. Hájek. Matematickú Ústav CSAV. 1984. P. 21-36.

5. Т е п а н д и Я.Я. Экспертная система тестирования программ по спецификациям // Автоматизация производства систем программирования. Тезисы докладов. Таллин, 1986. С. 187-188.

6. А х о А., Х о п к р о ф т Дж., У л ь м а н Дж. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979. 535 с.

J. Tepandi

HELI - an Expert System Shell

Abstract

A rule-based expert system shell is proposed. The knowledge base structure is described. Inference algorithms are given. The system is implemented in Micro-Prolog.

J. Tepandi

Instrumentaalne ekspertsüsteem HELI

Kokkuvõte

Kirjeldatakse instrumentaalse ekspertsüsteemi HELI ehitusprintsiipe. Tuuakse ära teadmusbaasi struktuur ja järeldamise algoritm.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.06

А.О. Вооглайд, В.А. Йокк

ПРАКТИКА ПРИМЕНЕНИЯ ГРАММАТИЧЕСКИХ ФОРМАЛИЗМОВ КАК ПРЯМЫХ СРЕДСТВ ПРОГРАММИРОВАНИЯ

I

I. Введение

В последнее время наблюдается тенденция уменьшения традиционного программирования на языках Фортран, Паскал, Си и других при создании крупных систем программирования.

В ходе создания таких систем делается попытка максимального применения готовых программных систем и пакетов. Те компоненты, которые не удается заимствовать в готовом виде, описываются на сверхвысоком метауровне и соответствующие программы генерируются автоматически. При наличии современных инструментальных средств разработчик новой системы остается все время в роли проектировщика и конструктора.

В технологическом плане часто используются уже отработанные в производстве методы систем САД/САМ [I][x], как, например, в работе [2].

Самым распространенным средством описания на сверхвысоком уровне являются грамматические формализмы в виде атрибутных грамматик [3]. Для поддержки данного подхода в получении качественных инструментальных средств создана всеевропейская программа [4]. Кроме того, во Франции развертывается своя национальная программа [5] по поддержке такого подхода.

Целью настоящей статьи является попытка показать использование грамматических формализмов при создании реальных рабочих мест обработки данных для конечного пользо-

---

[x] См. литература, с. 40.

вателя. При этом учитываются конкретные требования конечного пользователя, рабочая обстановка и эффективность работы создаваемого продукта.

В первой части статьи рассматривается организация работ в виде иерархических рабочих мест, описываются разные возможности трансляции и принципы работы автоматически сгенерированного интерпретатора. Во второй части статьи приведен пример из конкретной реализации.

2. Организация работы при создании рабочих мест

На кафедре обработки информации ТПИ выработана стабильная технология создания систем обработки данных [6,7]. При этом из инструментальных средств используются инструментальная система ELMA и соответствующие метаязыки [8], поддерживающие технологию Джексона [9]. В последнее время, при выполнении заказа внешнего пользователя, установлено четкое распределение труда создателей рабочих мест конечного пользователя. Естественно, что оно влечет за собой и некоторую иерархию рабочих мест трех разных типов:

- метаконструктор,
- конструктор,
- конечный пользователь.

Рабочему месту конечного пользователя предъявляется особое требование: диалог пользователя с ЭВМ должен осуществляться с использованием его рабочих терминов.

На рабочем месте метаконструктора ELMA в роли грамматического формализма служит S-атрибутная грамматика [10]. Вычисление атрибутов осуществляется в линейном представлении абстрактного дерева [11]. Возможно разбиение вычисления атрибутов на несколько проходов аналогично работе [12]. Также применимы, наряду с атрибутами, глобальные переменные и структуры данных. На рабочем месте ELMA используется атрибутная грамматика для описания транслятора проблемно-ориентированного языка, генератора ввода-вывода и интерфейса между языковым процессором и базой данных. На рисунке I приведена схема, где показана роль атрибутной грамматики при описании интерфейса.

Поскольку описание транслятора с атрибутных грамматик является классическим, то основное внимание в дальнейшем

будет обращено описанию интерфейса на примерах практической работы.

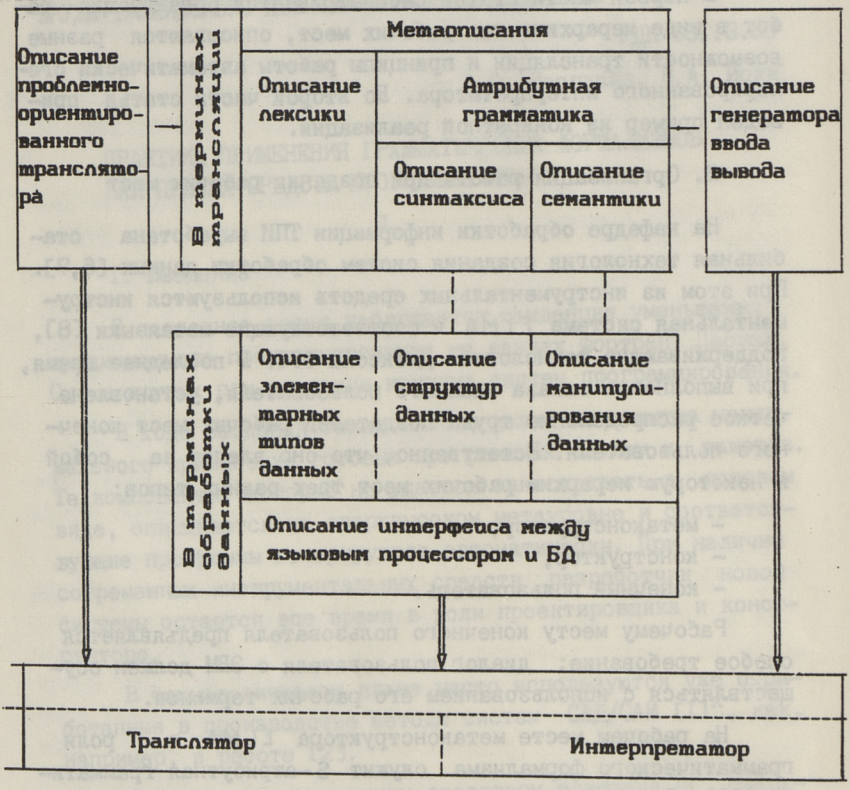


Рис. 1. Роль атрибутной грамматики при описании интерфейса.

На логическом уровне рабочие места можно охарактеризовать с помощью следующих компонентов [1]:

- класс задач,
- методы решения задач,
- инструментальные средства.

Приведем краткую характеристику трех вышеназванных рабочих мест.

Рабочее место метаконструктора ELMA создано по принципам программирования на уровне описаний и "ленивого программиста" [7]. Классом задач является создание инструмен-

тальных средств для генераторных систем рабочих мест обработки данных. Методом решения является развитой подход Джексона и атрибутные грамматики, а средствами - метаязыки ELMAMETA,ELMAGUIDE и диалоговая мониторная система.

Рабочее место конструктора (GENSI) [I3] создано по принципу разделения данных и описаний, а также разделения управления и действий [I4].

Классом задач является генерирование рабочих мест для оперативного управления, а также генерирование информационных систем, отчета, статистического анализа и т.д.

Методом решения являются проблемно-ориентированные методы и методы баз данных. Инструментальными средствами являются функциональные пакеты и специализированные языки.

3. Специфика создания программ обработки данных

При проектировании задач обработки данных опорным пунктом является правильное распределение работ (трансляция, интерпретация, решение) на две группы:

- однократная работа, например, трансляция входного текста и интерпретация описания переменных пользователя,
- повторная работа, например, интерпретация обработки запросов базы данных (БД) и интерпретация типов переменных БД.

В реализации однократных работ время трансляции и интерпретации не является существенным ввиду однократного выполнения. В этом случае работа происходит в интерактивном режиме и применение классических методов трансляции гарантирует необходимое качество результата.

При создании программ для многократных работ, если говорить вообще, то применять классическую технику компиляции не удается. Во-первых, как правило, во время трансляции большинство описаний данных не известно. Во-вторых, как показывает практика, проблемно-ориентированную программу обработки данных целесообразно применять для разных баз данных. Из этого вытекает серьезное требование к метасистемам, при помощи которых генерируют программы для конечного пользователя. Такие требования можно рассматривать в двух планах. Их, кстати, в последнее время называют программированием в "большом" и в "малом" [I5].

В системе ELMA средством первого плана является система управления ELMAGUIDE с собственным метаязыком [7, 8]. Эта система аналогична японской системе PAD [I6]. Преимуществом наших средств считаем то, что система ELMAGUIDE опирается на технологию Джексона и не связывает жестко структуру управления с условиями управления.

Метасистема, используемая во втором плане, должна содержать гибкие возможности применения разных схем трансляции и интерпретации. Эти возможности подробнее рассматриваются ниже, а в конце статьи дается пример реальной разработки.

Поскольку система управления со своим метаязыком ELMAGUIDE изложена в [7, 8], то в дальнейшем рассматриваются подробнее только разные возможности трансляции-интерпретации.

4. Разные возможности трансляции

В данном пункте описываются разные возможности организации процесса трансляции. Отдельно рассматриваются возможности описания процесса трансляций с помощью

- отдельных трансляций,
- функциональных частей транслятора,
- одно- или двухпроходного семантического анализатора.

Используя технику виртуальных трансляций [I7] и обозначения работы [I8], разные способы трансляции можно кратко представить следующим образом.

Обозначим трансляцию T с языка $\mathfrak{L}_a$ на язык $\mathfrak{L}_b$ следующим образом:

$$T=\{(x,y)\mid x\in\mathfrak{L}_a \& y\in\mathfrak{L}_b\},\quad Tx=y.$$

Данный перевод можно представить разными способами.

Слойная трансляция

$$T_2=\{(x,x_2)\mid x\in\mathfrak{L}_a \& x_2\in\mathfrak{L}_2\}$$

$$T_3=\{(x_2,x_3)\mid x_2\in\mathfrak{L}_2 \& x_3\in\mathfrak{L}_3\}$$

. . . . . .

$$T_{n-1}=\{(x_{n-1},y)\mid x_{n-1}\in\mathfrak{L}_{n-1} \& y\in\mathfrak{L}_b\}.$$

Здесь $\mathcal{L}_2, \ldots, \mathcal{L}_{n-1}$ – промежуточные языки.
Слойную трансляцию можно представить следующим образом:

$$Tx = T_{n-1} T_{n-2} \cdots T_3 T_2 x.$$

Последовательно-блочная трансляция

$$T = \{(x\, x_1 \ldots x_n, y) \mid x_1 \ldots x_n \in \mathcal{L}_a \,\&\, x_i \in \mathcal{L}_i \,\&\, y \in \mathcal{L}_b$$

$$Tx = T_1(x_1)\, T_2(x_2) \ldots T_n(x_n).$$

Вложенно-блочная трансляция

$$T = \{x_1 x_2 x_3, y) \mid x_1 x_2 x_3 \in \mathcal{L}_a \,\&\, x_1 \subset x_3 \in \mathcal{L}_1 \,\&\, x_2 \in \mathcal{L}_2 \,\&\, y \in \mathcal{L}_b\}.$$

В ходе одной трансляции имеется также возможность применения одного и того же анализатора (лексического, семантического, синтаксического) несколько раз подряд.

Например, типичный случай, когда при описании словарей с помощью грамматических формализмов, описание синтаксиса или описание лексики не однозначны. Простым решением является генерирование трех разных лексических анализаторов. Первый из них осуществляет т.н. грубый лексический анализ, т.е. статьи словаря. В результате полученной информации встает вопрос, какой анализатор использовать, второй или третий лексический. В аналогичных случаях иногда необходимо произвести следующую трансляцию:

Лексический анализ
↓
Синтаксический анализ
↓
Семантический анализ
↓
Синтаксический анализ

Параллельно с такими подходами можно использовать технику "прямого доступа". В этом случае в предыдущем проходе отмечаются места, которые требуют специальной обработки и которым обеспечен прямой доступ в рамках следующих проходов.

При создании инструментальных средств, процесс составления генератора интерпретаторов является наиболее трудоемким и плохо формализуется. Чтобы избежать подробного явления, в систему ELMA введен двухпроходной семантиче-

ский анализ. Надо отметить, что он кардинально отличается от варианта, в котором два раза подряд применяется семантический анализатор, описанный в технике атрибутной грамматики.

В системе ELMA можно описывать и автоматически генерировать интерпретатор, который работает как RISC вычислитель на магазинной памяти [19].

В следующем пункте рассматриваются описание и принципы работы интерпретатора.

5. Интерпретация

Для того, чтобы лучше объяснить работающий в настоящее время вариант интерпретатора, посмотрим прежние техники, которые можно было бы использовать в рамках системы ELMA для осуществления интерпретации.

В ходе практической работы использовались следующие варианты.

Первый вариант (1970–75 гг.).

Входом для интерпретатора служит дерево вывода. С каждой вершиной сопоставлена одна семантическая подпрограмма. Обход дерева зафиксирован.

Второй вариант (1975–82 гг.).

Этот вариант является дальнейшим развитием первого варианта. С каждой вершиной дерева вывода связано множество семантических подпрограмм, которые распределены на две части. Одна часть из них выполняется при обходе дерева сверху вниз, а другая – снизу вверх. Возможным является также гибкое управление обхода дерева по значениям данных.

Третий вариант (1982 – г.).

Использовали промежуточный язык, похожий на P-код и соответствующую абстрактную машину. В своих практических работах в роли промежуточного языка мы применяли язык FORTH [20].

При составлении интерпретаторов для проблемно-ориентированных языков в практике часто появлялись задачи, в которых всю информацию для интерпретации можно собрать во время

одного прохода в виде разреженных деревьев вывода [II], где с каждой вершиной связаны синтезированные атрибуты. Для этих случаев выработана специальная техника описания действия интерпретирующего автомата, которую мы рассмотрим в следующем пункте.

5.I. Описание интерпретатора на метауровне

В тех случаях, когда достаточно одного прохода для сбора информации для интерпретации, можно использовать следующую технику для описания интерпретатора.

Используется метаописание, в котором можно определить двухпроходной семантический анализатор. При этом описание семантики второго прохода служит описанием интерпретатора.

При таком описании из метаописания генерируется два разных конструктора разреженных деревьев, которые между собой связаны только при помощи инициализированных атрибутов. При этом второе разреженное дерево служит управляющей структурой составляемого интерпретатора.

Схема описания генерации интерпретатора приведена на рисунке 2.

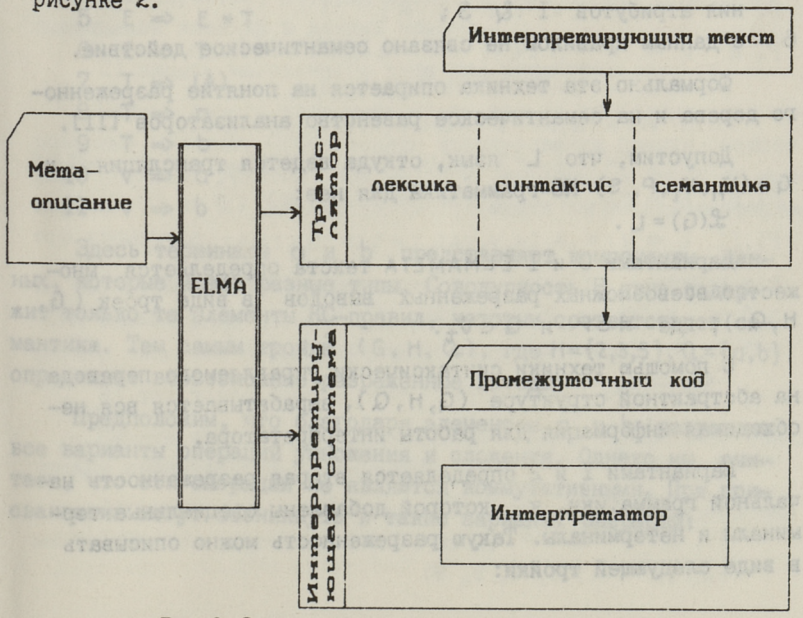
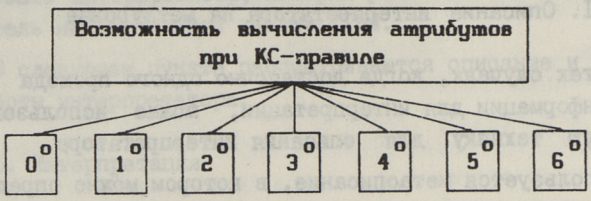


Рис. 2. Система описания генерации интерпретатора.

В этом случае интерпретирующая система может работать после трансляции самостоятельно.

Данное "двойное" описание реализовано в рамках метаязыка ELMAMETA [8]. К обычным правилам описания синтаксиса прибавлена возможность разбиения описания правил вычисления атрибутов по следующей схеме.



0 – действие выполняется **только** при первом проходе;
I – действие выполняется только при первом проходе и вычисленные атрибуты можно использовать во втором проходе;
2 – действие выполняется только при втором проходе;
3 – действие состоит из двух частей: первая часть выполняется на первом, а вторая при последующих проходах;
4 – действие выполняется как на первом, так и на втором проходе;
5 – действие выполняется так же, как и при правиле вычисления атрибутов I & 3;
6 – с данным правилом не связано семантическое действие.

Формально эта техника опирается на понятие разреженного дерева и на семантическое равенство анализаторов [II].

Допустим, что L язык, откуда ведется трансляция и $G = (V_n, V_t, P, S)$ КС-грамматика для нее:

$\mathcal{L}(G) = L$.

Вариантами 0 и I ELMAMETA текста определяется множество всевозможных разреженных выводов в виде троек (G, H, Q), где $H \subset P$ и $Q \subset V_t$.

С помощью техники синтаксически управляемого перевода на абстрактной структуре (G, H, Q), вырабатывается вся необходимая информация для работы интерпретатора.

Вариантами I и 2 определяется вторая разреженность начальной грамматики, к которой добавлены специальные терминалы и нетерминалы. Такую разреженность можно описывать в виде следующей тройки:

$$(G_1, H_1, Q_1),$$

где $G_1 = (V_n \cup V_n^1, V_t \cup V_t^1, P_1, S)$.

Абстрактная структура (G_1, H_1, Q_1) использована в интерпретаторе как управляющая структура. Семантические правила, связанные с элементами множества H_1, образуют проблемно-ориентированную часть команд интерпретатора.

Конкретный синтаксис языка ELMAMETA подробно приведен в работе [2I].

Приведем элементарный пример, в котором заданы тройки (G, H, Q) и (G_1, H_1, Q_1).

Пример

Допустим, что L = { оператор присваивания } и соответствующая грамматика

$$G = (\{S, V, A, E, T\}, \{(,), +, *, a, b, \}, P, S),$$

где P:

I S ⇒ (L)*
2 L ⇒ V := A
3 A ⇒ A + E
4 A ⇒ E
5 E ⇒ E * T
6 E ⇒ T
7 T ⇒ (A)
8 T ⇒ a
9 T ⇒ b
I0 V ⇒ a
II V ⇒ b

Здесь терминалы a и b представляют компоненты данных, которые имеют разные типы. Совокупность H явно содержит только те элементы КС-правил, которым соответствует семантика. Тем самым тройка (G, H, Q), где $H=\{2,3,5\}$, $Q=\{a,b\}$ определяет всевозможные разреженные выводы.

Предположим, что благодаря элементам a и b разрешены все варианты операций умножения и сложения. Однако мы считаем, что эти операции не являются коммутативными. При присваивании могут возникнуть и такие варианты операций:

a : = a

a : = b

b : = b

Запрещается операция присваивания следующего вида

b : = a.

Учитывая вышесказанное, можно представить (G_1, H_1, Q_1) следующим образом:

PI:

I	$A \Rightarrow A +_1 E$	8	$E \Rightarrow E *_4 T$
2	$A \Rightarrow A +_2 E$	9	$L \Rightarrow V :=_1 A$
3	$A \Rightarrow A +_3 E$	I0	$L \Rightarrow V :=_2 A$
4	$A \Rightarrow A +_4 E$	II	$L \Rightarrow V :=_3 A$
5	$E \Rightarrow E *_1 T$	I2	$L \Rightarrow V :=_4 A$
6	$E \Rightarrow E *_2 T$	I3	$A \Rightarrow a$
7	$E \Rightarrow E *_3 T$	I4	$A \Rightarrow b$

$H_1 = P_1$

$Q_1 = \{a, b\}$.

Далее приводим пример описания семантики для некоторых правил первого прохода (P.I0, P.II, P.2) и описания некоторых операций (* , : =) интерпретатора.

P. I0 V ⇒ a [* V·T = 1 *]

P. II V ⇒ b [* V·T = 2 *]

P. 2 L ⇒ V: = A [* IF V·T = Λ & A·T = 1 THEN D(*) = 9

IF V·T = 1 & A·T = 2 THEN D(*) = 10

IF V·T = 2 & A·T = 2 THEN D(*) = 11

IF V·T = 2 & A·T = 1 THEN D(*) = 12 ‹сообщение об ошибке›

PI.5 $E \Rightarrow E *_1 T$ [* E·V = UM1 (E.V, T.V)

PI.II $L \Rightarrow V :=_3 A$ [* V.VAL : = S (P (GETVAL)) *],

где действие GETVAL дает значение инициализированному атрибуту, действие P совершает преобразование типов и действие S сопоставляется со значением ссылки.

Допустим, что входной текст

a : = (a + a) * b b : = a,

тогда первый разреженный вывод $\mathfrak{D}_{H,Q}$ следующий

$\mathfrak{D}_{H,Q}$ =(3, 4, 3, 2, 2)

и $\mathfrak{D}_{H_1,Q_1}$ следующий

$\mathfrak{D}_{H_1,Q_1}$ = (I, 6, 2, 9, 9)

При этом интерпретатор выполняет действие на разреженном выводе $\Phi_{H_1 Q_1}$ до тех пор пока не изменятся типы элементов a и b используя конкретные операции интерпретатора: $+_1$, $*_2$, $+_2$, $:=_1$, $:=_1$.

5.2. Интерпретатор

На метауровне автоматически сгенерированный интерпретатор по сути является автоматом с магазинной памятью.

Имеются три разных типа команд:

- проблемно-ориентированные команды,
- команды инициализаций,
- команды управления.

При описании интерпретатора можно оперировать памятью шести типов. Ниже описываются типы памяти и ее команды.

Типы памяти

I) логический регистр.

Для каждой команды можно использовать "флажок" в логическом регистре, который является семафором и определяет выполнение команды в данный момент.

2) память магазинного типа для атрибутов.

Память такого типа используют только проблемно-ориентированные команды. Ограничением использования памяти такого рода является условие, при котором разреженный вывод из множества (G, H, Q) должен соответствовать их последовательности атрибутов.

3) общая память.

Все команды интерпретатора могут использовать память такого типа без дополнительного контроля.

4) текстовая память.

Память этого типа представлена в виде стрингов со счетчиками произвольной длины.

Вся обработка текстов и специальных операций с текстами базируется на памяти такого типа.

5) буфер стандартного ввода/вывода.

Этот буфер применяется в интерактивном режиме для ввода дополнительных запросов и для вывода коротких отчетов.

6) буфер интерфейса с базой данных.

Буфер интерфейса представлен в виде суперзаписи [I3].

Типы команд

Каждая команда интерпретатора является коротким целым числом с количеством адресов от 0 до 5. У каждой команды свой адрес. Программы промежуточного языка можно модифицировать во время выполнения.

Известно, что строго запрещается модификация программ "от руки" во время выполнения. В системе ELMA данная модификация осуществляется автоматически по описанию метаязыка и дает ощутимый выигрыш во времени при выполнении.

Ниже дается подробная характеристика разных типов команд:

I) команды управления.

Используются классические команды управления - команды безусловного перехода, команды перехода по "истине", команды перехода по "лжи".

2) команды инициализации.

Имеется два разных вида команд инициализации - команда инициализации атрибутов, которые связаны с элементами множества (G, H_1, Q_1) и команда инициализации атрибутов, которые вычисляются во время семантического анализа.

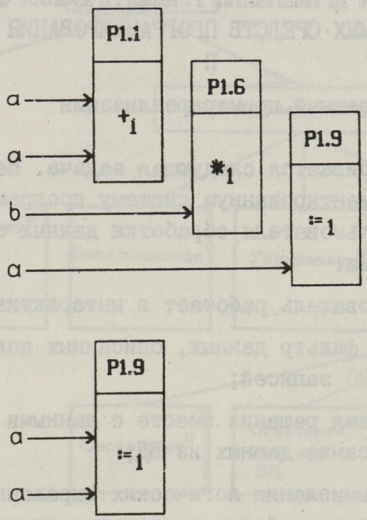
3) проблемно-ориентированные команды.

Это уникальные команды, которые описываются в атрибутной технике только для конкретного интерпретатора.

Из представленных команд видно, что среди них нет команд "старт", "стоп" и команд ввода-вывода. Команд остановки не существует, потому что управление интерпретатора описывается на один уровень выше -- при описании окружения создаваемой системы программирования. Команды ввода/вывода отсутствуют, потому что при каждой реализации они связаны со специфическими свойствами проблемной области. По этой

причине в атрибутной технике целесообразно каждый раз описывать специальные команды ввода/вывода, используя стандартные утилиты операционной системы. Во время инициализации создаваемой системы программирования уточняют утилиты конкретной ЭВМ.

Пример интерпретатора, описанный в пункте 5.I, можно графически представить в следующем виде



где в верхней части четырехугольника находится код команды, а в нижней части - операция.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.06

А.О. Вооглайд, В.А. Йокк

ПРАКТИКА ПРИМЕНЕНИЯ ГРАММАТИЧЕСКИХ ФОРМАЛИЗМОВ КАК ПРЯМЫХ СРЕДСТВ ПРОГРАММИРОВАНИЯ

П

6. Конкретный пример реализации

Рассматривается следующая задача. Необходимо создать проблемно-ориентированную систему программирования для конечного пользователя обработки данных со следующими характеристиками:

- пользователь работает в интерактивном режиме;

- через фильтр данных, описанных пользователем, проходит более 1000 записей;

- во время решения вместе с данными должно быть доступно и описание данных из БД;

- все вычисления логических выражений должны быть реализованы в технике "вычисление, управляемое данными" [22];

- данные из БД обрабатывают большими порциями, в пределах которых структура записи и типы данных не меняются.

Из этого вытекает важное требование к реализации: при обработке первой записи нужно сгенерировать новое промежуточное представление для интерпретатора, в котором уже нет проверки типа данных.

Для решения данной задачи нужно разработать язык программирования функции конечного пользователя, соответственно описать реализацию на метауровне и автоматически сгенерировать необходимые программы.

Структура описанного языка:

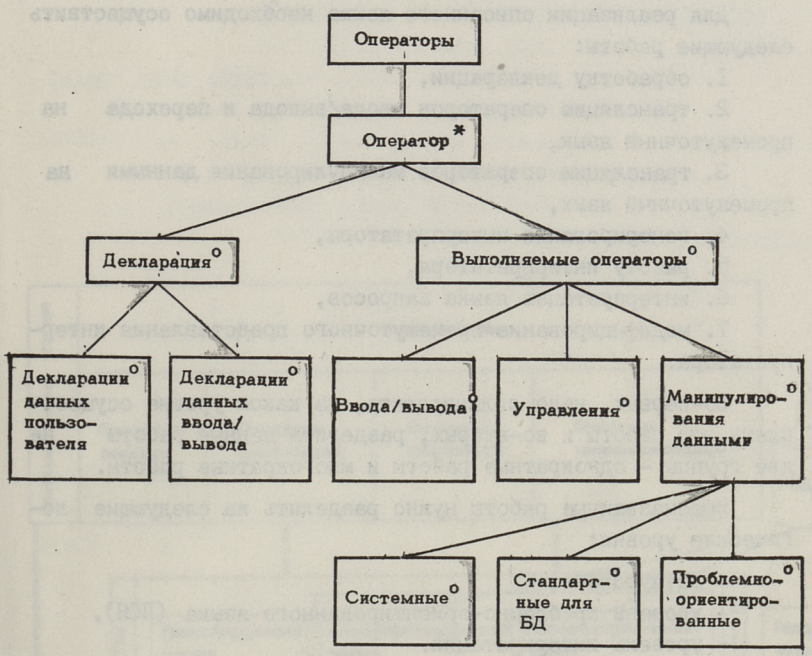


Рис. 3. Структура составляемого языка.

Ввод и вывод составляемой системы программирования следующий:

Ввод

- текст программы проблемно-ориентированного языка;
- ввод пользователя, во время выполнения программы является текстом дополнительного запроса;
- описание данных из БД;
- данные из БД.

Вывод

- вывод пользователя;

- промежуточное представление данных для генератора оформления отчетов.

Для реализации описанного языка необходимо осуществить следующие работы:

1. обработку декларации,
2. трансляцию операторов ввода/вывода и перехода на промежуточный язык,
3. трансляцию операторов манипулирования данными на промежуточный язык,
4. генерирование интерпретатора,
5. работу интерпретатора,
6. интерпретацию языка запросов,
7. модифицирование промежуточного представления интерпретатора.

Во-первых, надо спланировать, на каком уровне осуществлять эти работы и во-вторых, разделить данные работы на две группы - однократные работы и многократные работы.

Вышеназванные работы нужно разделить на следующие логические уровни:

- метауровень,
- уровень проблемно-ориентированного языка (ПОЯ),
- уровень интерпретации.

Распределение разных логических уровней и работ приведено на рис. 4.

При конкретной реализации все программы генерируются в виде текстов Фортрана. При этом возможны два варианта генерации - на ЭВМ ЕС или на ЭВМ СМ.

В приложении I приведена часть метаописания реализации, которая является третьим вариантом интерпретатора. Следующие варианты интерпретатора отличаются от предыдущих, прежде всего, скоростью работы. Последний реализованный интерпретатор работает в два раза быстрее, чем первый из них. В описании применяются обозначения метаописаний и используемых функций из работы [21] и функций, описанных в работе [22].

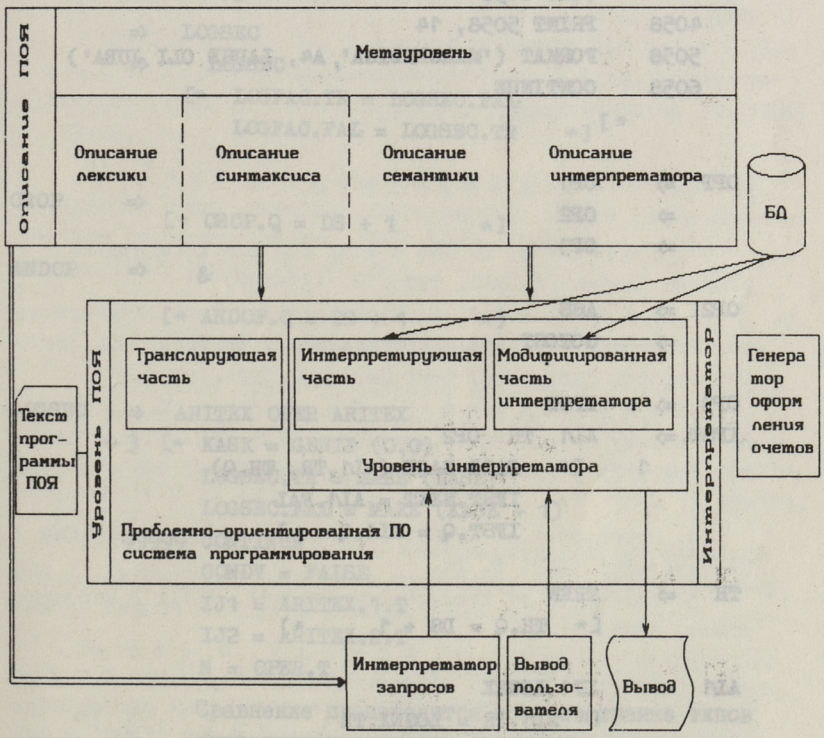


Рис. 4. Схема реализации и работы проблемно-ориентированной системы программирования.

```
OP   =>   OPP
     =>   CONST OPP
        [* CALL LHASH (CONST.VAL, 14, LAIG, LADR, LTAB)
           IF (AADR (LADR)    = 0) GOTO 4058
           AADR (LADR) = OPP.Q
           IF (LT1 (LADR) = 0) GOTO 6058
           CALL BACK (LT1 (LADR), AADR (LADR))
           GOTO 6058
           GOTO 6058
  4058     PRINT 5058, 14
  5058     FORMAT ('MARGENDIGA', A4, LAUSE OLI JUBA')
  6058     CONTINUE
         *]

OPP  =>   OP1
     =>   OP2
     =>   OP3

OP2  =>   ASS
     =>   GOTOST

OP3  =>   IFST
IFST =>   AL1  TH  OP2
       1    [*   CALL BACK (AL1.TR, TH.Q)
                 IFST.NEXT = AL1.FAL
                 IFST.Q = AL1.Q   *]

TH   =>   THEN
          [*  TH.Q = DS + 1      *]

AL1  =>   IF1 LOGEX
          [* AL1.TR = LOGEX.TR
             AL1.FAL = LOGEX.FAL   *]
IF1  =>   IF
          [*  IF1.Q = DS + 1     *]
```

```
LOGEX    ⇒  LOGEX OROP LOGFAC
            [*  CALL BACK  (LOGEX.FAL, OROP.Q)
                LOGEX.TR = MERGE  (LOGEX.TR, LOGFAC.TR)
                LOGEX.FAL = LOGFAC.FAL     *]
         ⇒  LOGFAC

LOGFAC   ⇒  LOGFAC ANDOP LOGSEC
            [*  CALL BACK (LOGFAC.TR, ANDOP.Q)
                LOGFAC.TR = LOGSEC.TR
                LOGFAC.FAL = MERGE (LOGFAC.FAL, LOGSEC.FAL)
              *]
         ⇒  LOGSEC
         ⇒    LOGSEC
            [*  LOGFAC.TR = LOGSEC.FAL
                LOGFAC.FAL = LOGSEC.TR     *]

OROP     ⇒
           [* OROP.Q = DS + 1        *]

ANDOP    ⇒    &
           [* ANDOP.Q = DS + 1       *]

LOGSEC   ⇒  ARITEX OPER ARITEX
         3 [*  KASK = GENIF (0,0)
               LOGSEC.TR = MAKE (KASK)
               LOGSEC.FAL = MAKE (KASK + 1)
        99999 CONTINUE
               CONDV = FALSE
               IJ1 = ARITEX.1.T
               IJ2 = ARITEX.2.T
               N = OPER.T
```

Сравнение производится соответственно типов операндов и операции сравнения.

```
        LL1 = ARITEX.1.IV
        LL2 = ARITEX.2.IV

        IF (LL1 = LL2) CONDV = TRUE
        GOTO 10350

    10350 CONTINUE

      => (LOGEX)

OPER  => =
        [* OPER.T = 1  *]
      => <=
        [* OPER.T = 2  *]
```

Литература

1. Schulz A. CAS a tool for the interactive program design // IEEE. 1985. P. 131-137.

2. Vooglaid A. Ein flexibles Fertigungssystem für Programme zur Datenkomprimierung / Bericht 140. Universität Karlsruhe, West Germany, 1984. 81 p.

3. Deransart P., Jourdan M., Lorho B. A survey on attribute grammars. Part III. Classified bibliography / INRIA, Rapports de Recherche. 1985. N 417. 61 p.

4. Prospektra an ESPRIT Projekt. Uni des Saarlandes / Teh. Report. 1987. P. 24.

5. Languages of Traducteurs Project. INRIA / Rocquencourt. 1986. P. 12.

6. Выханду Л.К., Йокк В.А., Нунапуу Э.Х.-Т. О технологизации построения прикладных систем обработки данных // Тр. Таллинск. политехн. ин-та. 1987. № 645. С. 33-38.

7. Вооглайд А.О., Лепп М.В. EIMA как набор инструментальных средств "ленивого программиста" // Тр. ВЦ СО АН СССР. Новосибирск, 1988. 19 с.

8. Вооглайд А.О., Лепп М.В. Входные языки системы ELMA // Тр. Таллинск. политехн. ин-та. 1982. № 524. С. 79-96.

9. Jackson M.A. Principles of program design. Academic Press, 1977. 371 p.

10. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. М.: Мир, 1979.

11. Вооглайд А.О. Семантическое равенство распознавателей, работающих на грамматике LR(k) и грамматике предшествования с (I/I) ограниченным каноническим контекстом // Тр. Таллинск. политехн. ин-та. 1976. № 411. С. 39-55.

12. Sassa M., Takuda J., Shinogi I., Inoue K. Design and implementation of a multipass-compiler generator // Journ. of Information Processing 32. 1980. P. 77-86.

13. Йокк В.А., Ыунапуу Э.Х.-Т. Разработка интерфейса между языковым процессором и СУБД в генераторной системе обработки данных ГЕНСИ // Тр. Таллинск. политехн. ин-та. 1987. № 645. С. 25-32.

14. Kowalski R. Algorithm = logic + control. // Comm. ACM. 1979. Vol. 22, N 7. P. 424.

15. Lecture notes in computer science / Edited by Goos and J. Hartmanis. N 224. Springer-Verlag. 1968. P. 1-196.

16. Futamura Y., Kawai T. Problem analysis diagram (PAD) // Jarect vol. 12. OHMSHA, LTD 1984. P. 97-115.

17. Вооглайд А.О. Система виртуальных трансляций // Автоматизация пакетов прикладных программ и трансляторов: Тезисы докладов. Таллин, 1983. С. 49-51.

18. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Т. I., М.: Мир, 1978. 613 с.

19. Wallach Y. Alternating sequential/parallel processing LN in CS. N 127. Springer-Verlag, 1982. 343 p.

20. Баранов С.Н., Ноздрунов Н.Р. Язык Форт и его реализация. Л.: Машиностроение. Ленингр. отд-ние, 1988. 157 с.

21. Henno J., Jokk V., Lepp M., Vooglaid A. Translaatorite koostamine. Tallinn: TPI, 1986.

22. Aho A.V., Ullmann J.D. Principles of compiler design. Addison-Wesley, Reading Maos., 1977.

A. Vooglaid, V. Jokk

Practical Use of Grammatical Formalisms as Direct Programming Tools I, II

Abstract

In this paper an effective technology to create real end-users work stations is presented. This technology is based on using grammatical formalisms as direct programming tools. It is shown that realization of work stations with the help of multipass compiling is very effective.

A. Vooglaid, V. Jokk

Grammatiliste formalismide kui otseste programmeerimisvahendite kasutamise praktika

Kokkuvõte

Artiklis näidatakse grammatiliste formalismide kasutamist otseste programmeerimisvahenditena reaalsete lõpptarbija andmetöötluse töökohtade loomisel. Seejuures arvestatakse lõpptarbija konkreetseid nõudmisi, töömiljööd ja genereeritava produkti töö efektiivsust.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

Л.К. Выханду, С.Р. Юргенсон

ГЕНЕРАТОР ТРИОДИК-ФОРТ

Одновременно с ростом объема и сложности информационных систем (ИС) возникла проблема их автоматизированного генерирования. Написание ИС вручную - работа сложная, требующая большого труда, много времени и довольно высокой квалификации. Автоматически сгенерированные ИС могут быть не так эффективны, как системы, написанные вручную, но они гораздо дешевле и это уже преимущество, которое в настоящее время становится все важнее.

Итак, нужно создать генераторную систему, которая генерировала бы сама программы создаваемых ИС, по их описанию.

Описание ИС должно происходить как можно более независимо от ЭВМ, т.е. на высоком логическом уровне.

Обычно верхние уровни оставляют неописанными и сразу переходят к программированию на уровне ЭВМ.

Наилучший подход тот, при котором система описывается на всех логических уровнях с помощью средств, удобных для пользователя (язык пользователя).

Так как работу системы на любом логическом уровне можно всегда разделить на управление и выполняемые действия, то в первую очередь необходимы средства для описания управления. Что же касается действий, то они являются обращением к следующему нижнему уровню. Такой подход и дает возможность описывать систему на всех уровнях одними и теми же средствами, что так важно при автоматизации.

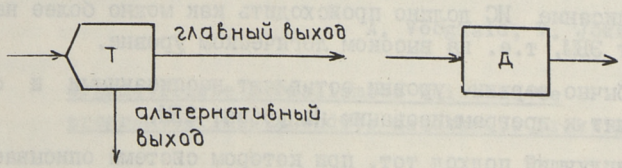
В настоящей статье рассматривается один из возможных языков пользователя для описания управления ИС (язык Триодик [1]) и даны примеры его возможностей. Целевым языком генерирования выбран язык Форт. Он требует малого объема памяти и кроме того, хорошо совмещается вертикально (через

разные типы ЭВМ) и горизонтально(через ЭВМ того же типа). Это дает возможность генерировать системы на больших ЭВМ и использовать их на маленьких персональных компьютерах.

I. Язык Триодик

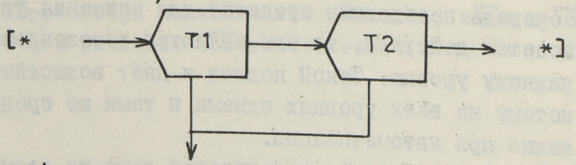
Язык Триодик является одним из перспективных средств для описания управления ИС на всех уровнях. Это обобщение схем структурного программирования, используемых на практике. Его можно рассматривать не только как язык программирования, но и как язык описания управляющих структур. Триодик является очень компактным и гибким языком. Автоматизированное генерирование систем, описанных на этом языке, осуществляется сравнительно просто.

Основные принципы языка Триодик лучше всего можно объяснить с помощью диаграмм. Базовыми блоками являются триоды (Т), т.е. программные блоки, имеющие один вход и два выхода - главный выход и альтернативный выход. Если последний отсутствует, получается частный случай триода - диод (Д). Диаграммы триода и диода следующие:



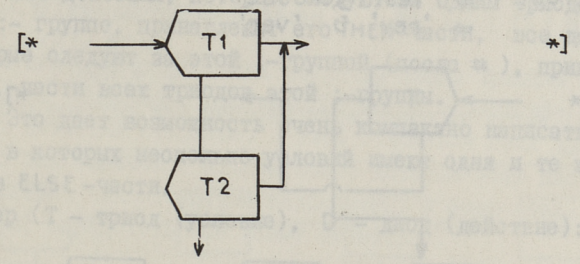
Описание языка Триодик в регулярной грамматике (семантика действий, определенных между триодами, пояснена с помощью диаграмм):

TRIODIC ⇒ D
D ⇒ D ; D1



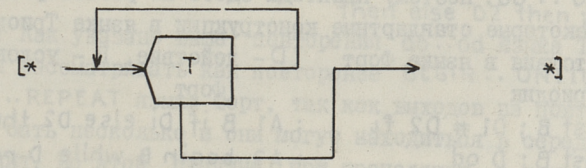
Аналогично логическому действию "И"
В языке ФОРТ: if

⇒ D # D1



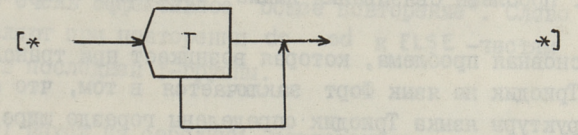
Аналогично логическому действию "или"
В языке ФОРТ: else

```
        ⇒  D1
D       ⇒  D1 ; C
        ⇒  D1 # C
        ⇒  C
C       ⇒  MONODE
        ⇒  DIODE
        ⇒  TRIODE
DIODE   ⇒  'do' D 'od'
```



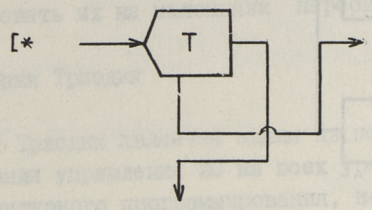
Повторение

⇒ 'if' D 'fi'



⇒ действие

```
TRIODE  ⇒  test условие
           ⇒ 'rev' D 'ver'
```




```
Отрицание
        ⇒ 'tri' D 'irt'
[*  Скобки                                  *]

MONODE ⇒ 'halt'
[* Прерывание                               *]
```

В языке Триодик описаны еще и квантоды, предназначенные для реализации циклов с заданным числом повторений, но на уровне управляющих структур они не так важны. И как сказал Голлер [I], они являются только желаемым дополнением к языку. Эти конструкции можно описать и при помощи повторения do .. od. Поэтому квантоды здесь не рассматриваются.

Некоторые стандартные конструкции в языке Триодик и их соответствия в языке Форт (D – действие, B – условие):

Триодик	Форт
if test B ; D1 # D2 fi	: A1 B if D1 else D2 then;
do test B ; D od	: A2 begin B while D repeat;
do D; test B od	: A3 begin D B until ;

2. Проблемы связывания языка Триодик с языком Форт

Основная проблема, которая возникает при трансляции с языка Триодик на язык Форт заключается в том, что основные структуры языка Триодик определены гораздо шире, чем соответствующие структуры в языке Форт. Стандартные структуры BEGIN .. UNTIL, WHILE .. REPEAT и др., которые даны в последнем разделе, являются только частными случаями языка Триодик.

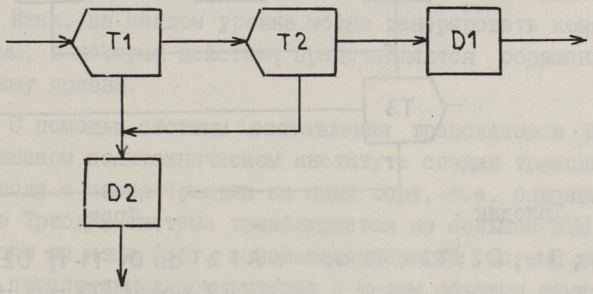
Надо было найти внутреннюю логику языка Триодик и приспособить ее к логике языка Форт.

* Внутренняя логика языка Триодик (;-группа – группа триодов, соединенных знаком ';'):

все действия, которые следуют за одним триодом в одной :- группе, принадлежат его THEN-части, все действия, которые следуют за этой ;-группой (после #), принадлежат ELSE-части всех триодов этой ;-группы.

Это дает возможность очень компактно написать выражения, в которых несколько условий имеют одни и те же действия в ELSE-части.

Пример (T - триод (условие), D - диод (действие):

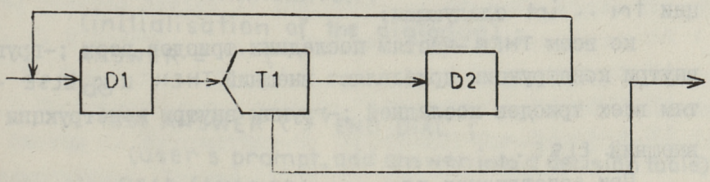


Триодик	Форт
T1 ; T2 ; D1 # D2	: A1 T1 if T2 if D1 else D2 then else D2 then ;

* Как указано выше, повторения do..od языка Триодик нельзя рассматривать как повторение BEGIN..UNTIL или WHILE..REPEAT языка Форт, так как выходов из повторения может быть несколько и они могут находиться в середине повторения. По этой причине, при трансляции, в языке Форт использовано повторение в виде 2 1 do..0 +loop, выход из которого происходит со словом LEAVE. Таким образом получено очень эффективное "общее повторение". Слово LEAVE прибавляют при повторении do..od к ELSE-частям всех триодов последней ;-группы.

Пример

а) выход из середины повторения:



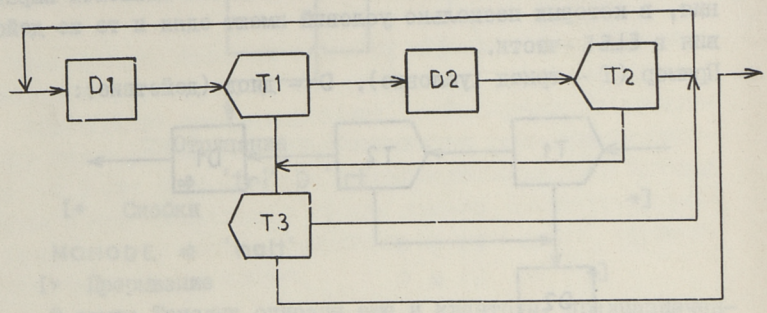
Триодик

```
do D1 ; T1 ; D2 od
```

Форт

```
: A1 2 1 do D1 T1 if D2
  else LEAVE then
  0 +loop ;
```

б) выход из повторения сложный или выходов много:



Триодик

```
do D1 ; T1 ; D2; T2 # T3 od
```

Форт

```
: A1 2 1 do D1 T1 if D2
  T2 not if T3 not if
  LEAVE then then else
  T3 not if LEAVE then
  then;
```

* У конструкции if..fi в Форте семантика отсутствует, она просто "закрывает" действие и превращает триод в диод.

* Трансляцию конструкции tri .. irt и rev..ver делает сложным то, что в языке форт они не "закрытые", т.е. THEN- и ELSE -части триодов, находящихся внутри конструкции, могут дополниться действиями вне конструкции.

Внутри этих конструкций действует та же логика, которая описана в начале раздела. Если назвать все действия, которые следуют за конструкцией tri .. irt или rev..ver в одной ;-группе внешний THEN и все действия, которые следуют за этой ;-группой внешний ELSE, то логика конструкции tri .. irt следующая:

ко всем THEN-частям последних триодов всех ;-групп внутри конструкции прибавляют внешний THEN и к ELSE -частям всех триодов последней ;-группы внутри конструкции внешний ELSE.

При конструкции rev..ver наоборот: к THEN-частям прибавляют внешний ELSE, к ELSE-частям - внешний THEN.

3. Генерирование систем

С помощью управляющих структур языка Триодик удобно описывать управление системы, оставляя содержание выполняемых действий открытым. В этом отношении язык Триодик хорошо совмещается с языком Форт, который тоже рассматривает действия как множество поддействий и позволяет, таким образом, определить все новые действия.

Итак, на каждом уровне можно генерировать конечные программы, в которых действия представляются обращениями к нижнему уровню.

С помощью системы составления трансляторов ELMA в Таллинском политехническом институте создан транслятор для перевода с языка Триодик на язык Форт, т.е. описанная на языке Триодик система транслируется на большой ЭВМ автоматически на язык Форт, использование этой системы возможно и на персональных компьютерах с малым объемом памяти.

4. Пример

Ниже демонстрируется применение управляющих структур языка Триодик в описании системного монитора, для создания интерактивной среды системы составления трансляторов ELMA [2]:

```
(initialisation of ELMA) ;
 READ (COM);
 do
  test COM <>'END';
  if
   test COM = 'HELP';
    (print information about commands);
   #
   test COM ='TRANSLATOR';
     (initialisation of the dialogue) ;
     ANSWER = ' ' ;
     do
      test ANSWER <> 'END DIAL' ;
        (user s prompt, add answer into a decision table);
       READ (ANSWER) ;
     od ;
```

```
    answer = ' ' ;
    do
      test ANSWER < > 'LEAVE' ;
        if
          test ANSWER = 'CONSTR' ;
              translator constructing)
          #
          test ANSWER = 'CORRECT';
            (correct decision)
        fi ;
      READ (ANSWER)
    od
   #
   test COM = 'EXEC';
     READ (NAME);
     ADD (RUN-ACT, NAME);
     (initialisation by NAME of a user's program)
  fi ;
  READ (COM)
od
```

Сгенерированная программа на языке Форт:

```
: A1 init_of_elma ;
: A2 read_com ;
: A3 COM_< >_END ;
: A4 COM_=_HELP ;
: A5 print_inf ;
: A6 COM_=_TRANSL ;
: A7 init_dial ;
: A8 answer_:=_' ' ;
: A9 ANSWER_< >_ENDDIAL ;
: A10 users_prompt ;
: A11 read_answer_add_dec_table ;
: A12 answer_:=_' ' ;
: A13 ANSWER_< >_LEAVE ;
: A14 ANSWER_=_CORRECT ;
: A15 transl_constr ;
```

```
: A16  ANSWER=CORRECT ;
: A17  corr_decision ;
: A18  read_answer ;
: A19  COM=EXEC ;
: A20  read_name ;
: A21  add_run_act_name ;
: A22  init_by_name ;
: A23  read_com ;
: A24  A1 a2
       2 1 do
        A3 if A if A5
                else A6 if A7 A8
                        2 1 do
                          A9 if A10 A11
                             else LEAVE then
                        0 +loop A12
                        2 1  do
                          A13 if A14 if A15
                                     else A16 if A17 then then
                                     A18
                              else LEAVE then
                        0 +loop
                else A19 if A20 A21 A22 then then then A23
           else LEAVE then
       0 +loop ;
```

Как показывает этот пример, Форт допускает реализацию языков очень высокого уровня.

Литература

1. Goller N.E. Triodic logic and its use in structured program design // The Computer Journal. 1982. Vol. 25, N 2. P. 218-226.

2. Вооглайд А.О., Выханду Л.К., Лепп М.В. Инструментальная система ELMA для создания программных систем // Информатика-85 (Материалы советско-французского симпозиума), Таллин, 1987, С. 126-136.

L. Võhandu, S. Jürgenson

Generator Triodic-Forth

Abstract

In this paper some aspects of the automatic generation of information systems are presented.

An example of the use of a high-level programming language Triodic as a user's language to describe the control structure of a system and Forth as a target language is given.

L. Võhandu, S. Jürgenson.

Genereerimine Triodic-Forth

Kokkuvõte

Artiklis käsitletakse mõningaid infosüsteemide tarkvara automaatse genereerimise probleeme. Kirjelduskeeleks on juhtimisstruktuuride kirjeldamise keel Triodic, sihtkeeleks programmeerimiskeel Forth.

Esitatakse keele Triodic süntaks ja semantika ning tuuakse näide tema abil kirjeldatud süsteemi transleerimisest keelde Forth. Transleerimiseks on kasutatud translaatorite koostamise süsteemi ELMA.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

Ю. Йоонсаар, Я. Хенно

ПРИМЕНЕНИЕ СЕТЕЙ ПЕТРИ ДЛЯ МОДЕЛИРОВАНИЯ УЧРЕЖДЕНЧЕСКОЙ ДЕЯТЕЛЬНОСТИ

I. Постановка проблемы

Автоматизирование учрежденческой деятельности стало особенно актуальным в связи с широким распространением микроЭВМ в 1980-ых годах. До этого времени в вычислительных центрах ЭВМ применялись для решения в пактеном режиме одиночных задач (счет зарплаты, учет основных средств и т.д.). При этом необходима дополнительная рабочая сила для подготовки данных и эксплуатации задач. Зачастую вычислительный центр находится где-то в другом месте, период с отдачи исходных данных до получения результатов измеряется в днях (а не в минутах), заказчик видит только конечные результаты и не имеет возможности работать с ЭВМ в диалоговом режиме.

Появление микроЭВМ сможет и должно изменить дело. Теперь уже возможно предлагать информационные системы, с которыми служащие могут работать у своего письменного стола в режиме, близком к реальному времени.

Использование учреждением информационной системы производится обычно на микроЭВМ, которые соединены в локальную сеть. С другой стороны, пользователь может, в зависимости от своих рабочих задач и специальности, играть в системе различные роли (планировщика, экономиста, бухгалтера и т.д.). Для каждой роли имеется в системе свое автоматизированное место (АРМ).

Применение АРМ поддерживает гуманизацию труда по следующим признакам:

- увеличение надежности;
- человеческая технология;

- улучшение требования к спецификациям должности (например, служащий не должен знать, от кого и каким образом обрабатываются поступающие к нему данные);

- повышение эффективности и качества труда.

С каждым АРМ связан список работ. Определенная работа выполняется, если в системе имеются соответствующие исходные данные. Результатом работы являются новые данные, которые, в свою очередь, могут быть исходными данными для других работ, выполняемых на этом же или ином АРМ. Другими словами, для выполнения работы нужны соответствующие исходные условия и результатом его будут исходные условия для других работ. Следовательно, работы связаны между собой через данные и в данный момент в системе можно выбрать только те работы, исходные условия для которых выполнены. При этом надо предусмотреть возможность выполнения исходных условий в результате разных работ.

В статье описывается модель сетей Петри для учрежденческой информационной системы, которая организует синхронизацию работ в системе и поддерживает выполнение основных функций АРМ для начальника.

Система должна дать пользователю ответы на следующие вопросы:

- какие работы можно выполнять в данный момент;
- если работа не выполнима, то по какой причине, т.е. какие работы должны быть предварительно выполнены.

Ясно, что ответы на эти вопросы связаны с исходными условиями, которые можно разделить на две группы:

1. Условия, которые связаны с существованием исходных данных и позволяющие выполнение соответствующих работ.

2. Условия, которые связаны с устранением исходных данных вследствие корректировки и требующие переделывания соответствующих работ.

Если система не может ответить на вышеприведенные вопросы, то отсутствие исходных данных все равно выясняется, но только позже по ходу работы при попытке их чтения, а устарение исходных данных остается скрытым.

2. Определение использованной модели систем Петри

Сеть Петри - это связной двухосновной ориентированный граф, т.е. граф, множество вершин которого разбито на множество мест $P=\{p1,\ldots,pn\}$ и множество переходов $T=\{t1,\ldots,tm\}$, где $P \cap T = \emptyset$. Переходы можно трактовать как отдельные акты обработки информации (действия), места - как условия, необходимые для срабатывания переходов (совершения отдельных актов обработки информации). Соответственно, дуги $I : P \rightarrow T$ характеризуют предусловия срабатывания переходов, т.е. условия, которые должны быть выполнены для срабатывания соответствующих переходов (наличие данных), а дуги $O : T \rightarrow P$ - постусловия, которые будут выполнены после срабатывания переходов (появление обработанных данных, которые в свою очередь необходимы, т.е. являются предусловиями для следующих переходов).

Обозначим через $I(t)$, $O(t)$ множества всех мест, в которые (соответственно из которых) входит (выходит) дуга в место $t \in T$.

Выполнены ли конкретные условия или нет, описывается маркировкой мест. Маркировка - это присвоение каждому месту какого-то числа $n >= 0$ фишек. Наличие фишки трактуется, как наличие каких-то ресурсов, необходимых выходным переходам (переходам, в которые идут дуги из данного места), т.е. для срабатывания выходных переходов выполнено предусловие или как ресурсы, которые вырабатывались в результате срабатывания входных переходов (переходов, из которых входит дуга в данное место), т.е. для них выполнено постусловие.

Функционирование сети Петри описывается последовательностью маркировок.

Обычно переход t может срабатывать, если во всех входных местах $I(t)$ есть по меньшей мере одна фишка. При срабатывании перехода t из всех входных мест удаляется одна фишка и в каждое выходное место $O(t)$ помещается по одной новой фишке, порождая таким образом новую маркировку.

Вышеописанная модель соответствует т.н. стандартным сетям Петри [1, 2]. При моделировании учрежденческой дея-

тельности, однако, требуются некоторые расширения стандартной модели.

I. Вышеописанные переходы требуют как необходимое условие для срабатывания выполнения всех предусловий (т.е. выполнение конъюнкции соответствующих предусловий). В конторской деятельности встречается и более общая ситуация - для срабатывания перехода достаточно выполнение хоть одного из некоторой группы предусловий (например, необходимые данные можно получить из нескольких мест), т.е. существование фишки хоть в одном из соответствующих входных мест (выполняется дизъюнкция соответствующих входов). В сети Петри такие переходы отмечаются дизъюнкцией.

2. В реальной работе учреждений производится в одно и то же время работа над данными нескольких периодов, которые между собой не связаны. Для описания этой особенности введена маркировка фишек - с каждой фишкой связан признак соответствующего периода. Это в некоторой степени изменяет и формальное описание работы сети Петри: для срабатывания некоторого периода необходимо наличие на всех входах фишек соответствующего периода и в результате перехода вырабатываются также фишки соответствующего периода.

3. После совершения обработки данных некоторого периода все соответствующие отчеты удаляются из системы. Это происходит срабатыванием специального терминального перехода, куда входят дуги из всех окончательных мест (мест, в которые попадут окончательные отчеты).

4. В работе учреждений существуют ситуации, когда входные данные изменяются и все работы, которые сделаны на основе старых данных, приходится переделывать. Такая ситуация характеризуется в сети Петри специальным переходом, срабатывание которой в отличие от обычных переходов не порождает новых фишек, а уничтожает фишки соответствующего периода на всех местах, куда идут дуги от этого перехода, (т.е. срабатывание данного перехода означает объявление недействительными всех работ, выполненных на основе старых данных). После срабатывания такого "уничтожающего" перехода работа сети продолжается обычным образом, т.е. снова

срабатывает соответствующий переход ввода новых данных вместо старых, недействительных данных и т.д.

3. Представление сетей Петри в ЭВМ

Сеть Петри можно описать как любой граф 0-1 матрицей смежности его вершин (см., например, [I, 4]). Однако, так как в двухосновном графе сети Петри все дуги идут либо от места к переходу, либо от перехода к месту, соответствующую $(n+m \times n+m)$-матрицу можно разбить на две матрицы [I]. Первая $(n \times m)$-матрица описывает функцию I. Строки матрицы (обозначим его также I) нумерованы местами $p_1,\ldots,p_n$, столбцы - переходами $t_1,\ldots,t_m$ и элемент матрицы a_{ij} равняется единице тогда и только тогда, если из места p_i идет дуга к переходу t_j (p_i является предусловием перехода t_j). Аналогично составляется вторая $(m \times n)$-матрица O, которая описывает функцию O.

Маркировка мест сети Петри описывается для каждого периода n-местным вектором M , компоненты которого указывают на число фишек в соответствующем месте. Легко видеть, что если переход t_j имеет только конъюнктивные входы, то этот переход может срабатывать при данной маркировке, если все компоненты столбца t_j матрицы I больше соответствующих компонент вектора M. В результате срабатывания сначала из вектора M (покомпонентно) вычитается столбец t_j матрицы I , а затем полученной разности добавляется строка t_j матрицы O, порождая таким образом новую маркировку M'. В случае наличия у перехода t_j дизъюнктивных входов для определения возможности срабатывания вместо покомпонентного сравнения применяется сравнение дизъюнкции соответствующих входов.

Раскраска переходов сети Петри определяется константным m-вектором.

4. Реализация

Описанный подход реализован в информационной системе планового сектора отдела оперативного управления главного правления пищевой промышленности Агропрома ЭССР. Система реализована на персональном компьютере с использованием

как инструментальной системы, так и интегрированной системы обработки таблиц.

Система позволяет планировщикам планировать и проводить анализ хозяйственных действий предприятий главного управления на основе экономических показателей. Система работает на одном персональном компьютере и его использование организовано через три АРМ. При составлении АРМ использованы концепции, излагаемые в работе [3].

Выбор работ на АРМ производится через иерархические меню. Как правило, все работы связаны с некоторой таблицей и поэтому в системе существенно использованы средства инструментальной системы обработки таблиц.

Состояние системы зависит от времени и определяется тем, какие работы каждого периода можно выполнять в данный момент. Следовательно, это сумма состояний исходных условий всех работ. Чтобы сделать эту информацию доступной для пользователя, перед началом сеанса каждого АРМ загружаются в оперативную память и вектор состояния условий (вектор состояния системы) и матрица исходных условий для работ соответствующего АРМ.

О работах, которые необходимо переделать по причинам корректировки исходных данных, система сообщает в начале сеанса АРМ и во время сеанса по требованию. В сообщении указывается также, какие данные изменились и в результате каких работ, на каких АРМ. Если работа не выполнима ввиду отсутствия исходных данных, то система сообщает об этом только после выбора этой работы через меню. В сообщении показывают, какие данные отсутствуют и какие работы для преодоления этого на каких АРМ необходимо выполнять.

5. Модель планового сектора

В плановом секторе занимаются девятью предприятиями главного управления пищевой промышленности АПК ЭССР, деятельность которых планируется и анализируется через экономические показатели.

Работа планового сектора делится на три области:

(I) учет продукции;

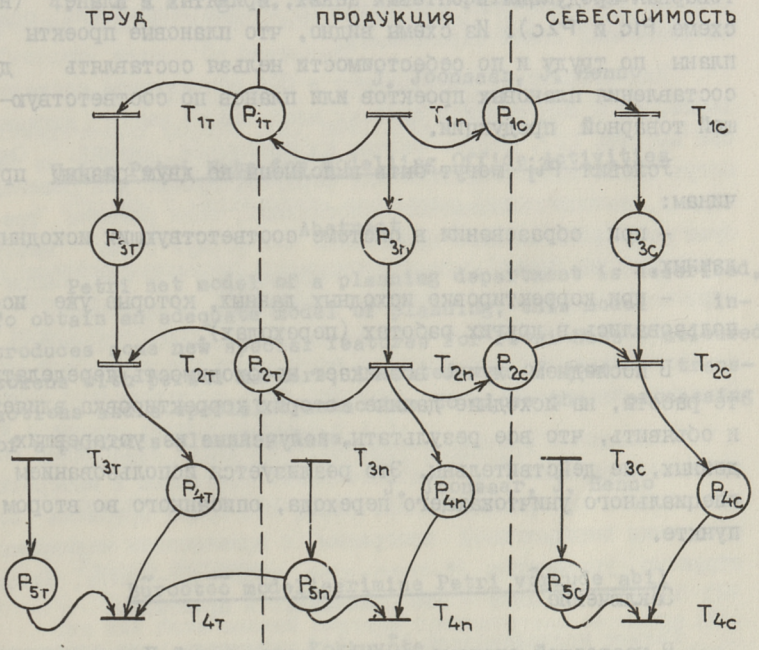
(2) учет труда, рабочей силы и зарплаты;

(3) учет себестоимости товарной продукции.

Каждой областью занимается отдельный планировщик, который выполняет следующие основные работы:

- составление годовых плановых проектов,
- хранение и корректировка годовых, квартальных и месячных планов,
- периодичный анализ плановых и фактических данных.

Исходные данные для анализа получаются из отчетов предприятий (форма № I-п, форма № 2-т и форма № I-с).



На схеме изображена модель планового сектора с использованием сети Петри, где работы представляются переходами и условия выполнения работ - местами (на схеме отсутствуют терминальные переходы; переходы, с которыми связаны уничтожающие действия, указаны жирными).

Здесь T_{1j} – составление годовых плановых проектов;

T_{2j} – составление годовых планов;

T_{3j} – ввод исходных данных для анализа и соответственно

$T_{3п}$ – из формы № I-п;

$T_{3т}$ – из формы № 2-т;

$T_{3с}$ – из формы № I-с;

T_{4j} – периодичный анализ плановых и фактических данных.

Основой для составления плановых проектов и планов по труду является товарная продукция в оптовых ценах на 1 января 1982 года (на схеме P1т и P2т), а по себестоимости товарная продукция в оптовых ценах, принятых в плане (на схеме P1с и P2с). Из схемы видно, что плановые проекты и планы по труду и по себестоимости нельзя составлять до составления плановых проектов или планов по соответствующей товарной продукции.

Условия P_{ij} могут быть выполнены по двум разным причинам:

– при образовании в системе соответствующих исходных данных,

– при корректировке исходных данных, которые уже использовались в других работах (переходах).

В последнем случае возникает необходимость переделать те работы, на исходные данные которых корректировка влияет и объявить, что все результаты, полученные из устаревших данных, не действительны. Это реализуется использованием специального уничтожающего перехода, описанного во втором пункте.

Заключение

В настоящей системе использование сетей Петри для моделирования синхронизации разных действий дает следующие преимущества:

– в каждый момент можно узнать, какие работы выполнимы и какие нет;

– если какая-то работа не выполнима, то можно узнать, по каким причинам и какие работы до этого необходимо завершить,

– для начальства полезно получить данные о положении всех выполняемых в системе работ (какие уже выполнили, какие нет, какие еще не начали и т.д.).

Литература

I. Котов В.Е. Сети Петри. М.: Наука, 1984.

2. Питерсон Дж. Теория сетей Петри и моделирование систем. М.: Мир, 1984.

3. Hansen B.S., Hansen M.R., Pletat U., Studer R. Abstract models of form handling systems // University of Stuttgart, Institut für Informatik, Technical Report, 10/83, 1983.

4. Reisig W. Petri Nets. Springer, 1982.

J. Joonsaar, J. Henno

Using Petri Nets for Modelling Office Activities

Abstract

Petri net model of a planning department is described. To obtain an adequate model of planning, this model introduces some new special features for Petri Nets – coloured tokens with period markers, iterated use of some transactions and a special transaction to close the processing of a period's planning data.

J. Joonsaar, J. Henno

Bürootöö modelleerimine Petri võrkude abil

Kokkuvõte

Artiklis on esitatud Petri võrgu mudel plaanisektori jaoks. Võrreldes standardse mudeliga, on sisse viidud järgmised täiendused: perioodi tunnused tingimuste markeritele, mõnede ülekannete iteratiivne täitmine ja spetsiaalne ülekanne perioodi andmete töötlemise lõpetamiseks.

№ 671
TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА
УДК 681.3

Я.А. Хенно, Р.З. Рюнне

ЯДРО ЭКСПЕРТНОЙ СИСТЕМЫ ИНТЕЛЛА

I. Мотивация системы и основные принципы ее реализаций

Владение навыками логического мышления, построение корректных умозаключений считается со времен Аристотеля одной из важнейших целей образования.

Как показывает проведенное в США официальное исследование, 40 % выпускников средних школ США не умеют логически мыслить [2], делать выводы из имеющихся знаний. Они могут знать все математические формулы, нужные для решения задачи, но она оказывается непосильной, если решение требует последовательного применения 4–7 формул (т.е. имеет глубину > 3). Такая же картина наблюдается и у нас. Например, анализ текстовых задач по математике, применяемых в последние годы в ТПИ на вступительных экзаменах, показывает, что они имеют небольшую глубину < = 5 (часто < = 3). Очевидно иначе эти задачи были бы непосильны.

Нахождение решения математических задач аналогично составлению линейной (последовательной) программы в императивных языках программирования. Ясно, что общая проблема составления императивной программы (с применением нелинейных структур управления if-then-else, цикл, do-while, do-until) намного сложнее – это показывает и практика.

Разные исследователи [3], [4] указывали, что основным методом обучения навыкам логического мышления является метод проб с последующим подтверждением. При этом очень существенными для успеха являются

– естественность обстановки обучения [3], т.е. естественность постановки задачи, использование общепринятых обозначений, естественные команды и т.д.

- быстрота ответной реакции обучателя [3], [4] (время реакции обучателя не должно превышать долей секунды).

- способность обучателя давать многосторонние, подробные объяснения и подсказки [5].

Система ИНТЕЛЛА спроектирована для обучения навыкам логического мышления с учетом вышеприведенных принципов (она сходна с системами ПРИЗ, МикроПриз и .т.д. [1], однако создана для других целей).

Система спроектирована модульная и многоуровневая с учетом возможности дальнейших усовершенствований и новых видов применения. Первый уровень системы предназначен для обучения навыкам логического рассуждения при решении математических задач. Он позволяет

- найти и исследовать логическую структуру отдельных математических формул (как вычисляемость одних величин следует из вычисляемости других);

- открыть и исследовать следствия (относительно вычислимости), вытекающие из отдельных формул, т.е. результат последовательного применения формул (можно ли искомую величину вычислить на основе заданных, какие вообще величины можно вычислить, исходя из заданных величин, какие величины требуются для вычисления искомой, является ли предлагаемый пользователем путь решения, т.е. нахождение искомой величины рациональным и т.д.).

- объяснить ответы и действия системы;

- конкретно (численно) решать задачи.

2. Работа системы

Так как база знаний системы сравнительно невелика 50-80 понятий, 150-200 следствий), система предназначена для изучения отдельных тем школьной математики - например, свойств отдельных геометрических фигур. Пользователь может либо сам создать базу знаний о данном объекте (треугольник, круг, конус и т.д.), либо использовать заранее подготовленные базы знаний, которые загружаются с диска (во время работы все знания о данном объекте находятся в основной памяти). Имеется возможность присоединения к данной базе знаний

заранее созданных баз знаний, например, при создании баз знаний о тетраэдре можно использовать знания о треугольнике.

Работу системы ИНТЕЛЛА можно разделить на два этапа. При создании новой базы знаний первым этапом является ввод известных фактов о данном объекте в базу знаний.

Обычно факты вводятся в систему в виде математических формул. Понятия могут быть представлены как произвольные строки символов (могут содержать также пробел, но не должны содержать математические символы и функции), например, при описании треугольника можно использовать понятия < a, < A, area ABC, SIDE A. После описания (перечисления) понятий вводятся формулы. Единственным ограничением является то, что формулы должны быть написаны как команды присваивания Бейсика. Например, формулу $c^2 = a^2 + b^2$ надо вводить в виде c = SQR (a^2 + b^2). Система сама выделит из математической формулы его логическую структуру, т.е. импликацию $(a, b \rightarrow c)$.

Для изучения "чистой" логики имеется и возможность введения только логических связей между понятиями, т.е. импликации. При надобности к ним можно потом присоединить и семантику, т.е. соответствующие математические формулы.

После ввода импликации (фактов, аксиом) происходит вычисление логического замыкания введенных фактов, т.е. нахождение всех (минимальных) следствий. При этом применяется правило вывода

$$\frac{A_1, \ldots, A_n \rightarrow B_i, \qquad B_1, \ldots, B_i, \ldots, B_k \rightarrow C}{B_1, \ldots, B_{i-1}, A_1, \ldots, A_n, B_{i+1}, \ldots, B_k \rightarrow C}$$

с последующей минимализацией выведенных импликаций (см. ниже).

Чтобы ускорить работу системы, производится вычисление всех логических связей только один раз, по специальной команде или после ввода первого запроса, т.е. следствия вычисляются методом прямой волны от аксиом к теоремам. Система запоминает все минимальные следствия, так что при (следующих) запросах почти никаких вычислений уже не происходит и результат на запрос получается очень быстро.

Минимальность вычисленных следствий означает, что

- не запоминаются импликации, где выведенная переменная содержится и среди предпосылок, т.е. импликация вида

$$A_1, \ldots, A_i, \ldots, A_k \rightarrow A_i$$

- не запоминаются импликации, где предпосылки сами уже содержат импликацию из базы знаний. Например, если в базе уже имеется импликация $A_1, A_2 \rightarrow B$ и в ходе вывода система найдет импликацию $A_1, A_2, B, A_3, \ldots \rightarrow C$, то последняя заменяется на $A_1, A_2, A_3, \ldots \rightarrow C$.

- если система находит импликацию $A_1, \ldots, A_k \rightarrow B$ и в системе уже имеется подобная импликация, где предпосылок меньше, т.е. импликация $A_1, \ldots, A_n \rightarrow B$, где $n < k$, то импликация $A_1, \ldots, A_k \rightarrow B$ не запоминается.

Эти условия гарантируют, что множество выводимых предложений является конечным.

Пользователь может задавать системе разные вопросы.

Запрос типа КАКОЙ используется для установления наличия конкретного факта (импликация) в базе знаний. Система также проверяет, содержит ли запрос несущественные предпосылки (в базе знаний системы сохраняются только следствия с минимальными предпосылками). В ответ система либо утверждает запрос, либо отрицает его.

При вводе запроса типа ЧТО ВЫЧИСЛИМО ИЗ в ответ выводятся все понятия, которые вычислимы на основе заданной совокупности понятий.

При запросе типа ИЗ ЧЕГО ВЫЧИСЛИМО в ответ выводятся те совокупности понятий, исходя из которых вычислимо заданное понятие.

Система способна обосновать свои ответы. Для этого используется запрос типа ПОЧЕМУ. Используя такой запрос, пользователь может исследовать ход логического вывода.

Систему можно также применять для проверки логического вывода пользователя. В таком режиме работы система сначала задает вопрос (например "КАК ИЗ A, < B, < C ВЫЧИСЛИТЬ S?"). В ответ пользователь начинает вводить шаги вычисления. Система проверяет, применим ли данный шаг (из-

вестны ли все предпосылки) и является ли предлагаемый шаг необходимым для достижения цели.

Систему можно использовать и для численного решения задач, т.е. для вычислений значений. В ходе вычислений система выдает весь логический путь решения (отдельные импликации), а также те формулы, которые система использует при вычислении.

Пользователь может также задавать вопросы о содержании и структуре базы знаний. Такими являются запросы:

ФАКТЫ? - выводятся все логические связи (факты), которые были введены пользователем (или получены в результате анализа введенных математических формул);

СВЯЗИ? - выводятся все логические связи базы знаний (т.е. введенные пользователем факты и их минимальные следствия);

ПОНЯТИЯ? - выводятся все понятия и их текущие численные значения;

ЗНАЧЕНИЯ? - можно ввести новые численные значения для некоторых понятий.

3. Пример работы системы

В качестве примера рассмотрим базу знаний о конусе. Ответы пользователя подчеркнуты.

В начале посмотрим, какие формулы имеются в базе знаний системы.

```
Ootan  :VALEMID?
Valem nr. 1   Sp1=3.14*r1^2
Valem nr. 2   Sp2=3.14*r2^2
Valem nr. 3   h  =SQR(m^2-(r1-r2)^2)
Valem nr. 4   H  =h+(h*r2)/(r1-r2)
Valem nr. 5   Sk =3.14*m*(r1+r2)
Valem nr. 6   S  =Sk+Sp1+Sp2
Valem nr. 7   V  =h/3*(Sp1+SQR(Sp1*Sp2)+Sp2)
```

Предполагаем, что нам надо вычислить объем конуса V.

```
Ootan  :Millest tuleneb V?
V tuleneb
r1,r2,m
r1,r2,h
```

```
Sp1,Sp2,h
r1,Sp2,h
Sp1,r2,h
```

Узнаем как из r1,r2,m можно вычислить объем V.

```
Ootan  :MIKS r1,r2,m → V?
Kuna r1,r2,m → h (sisestatud) ja r1,r2,h → V (leitud)
```

Введем начальные значения для переменных.

```
Ootan  :Algväärtused?
r1 = 5
r2 = 4
m  = 4
```

Приступим к решению.

```
Ootan  :LEIA V
r1,r2,m → h ja r1,r2,h → V
h=SQR (m^2-(r1-r2)^2)
m=4
r1=5
r2=4
h=3.873
r2 → Sp2 ja r1,Sp2,h → V
Sp2=3.14*r2^2
r2=4
Sp2=50.24
r1 → Sp1 ja Sp1,Sp2,h →V
Sp1=3.14*r1^2
r1=5
Sp1=78.5
V=h/3*(Sp1+SQR(Sp1*Sp2)+Sp2)
h=3.873
Sp1=78.5
Sp2=50.24
V=247.277
```

4. Реализация

В настоящее время реализованы две версии системы, одна на ЭВМ Sharp MZ -3500 и другая на ЕС-1840.

На ЭВМ Sharp ИНТЕЛЛА написана на языке ФДОС/Бейсик (интерпретатор) и работает под операционной системой FDOS. Более новая версия на ЕС-1840 написана на компилирующем Бейсике под операционной системой MS-DOS 3.10.

База знаний системы реализована при помощи пяти массивов (во время работы вся база находится в основной памяти). Из них самым важным является массив логических связей. Там закодированы все знания - введенные факты и следствия из них, а также путь вывода следствий. В остальных массивах хранятся коды понятий и т.д., а также соответствующие логическим связям математические формулы.

Литература

1. Кооов М.И., Минц Г.Е., Тыугу Э.Х. Система программирования МикроПриз. Таллин 1987. 95 с.

2. Castellan N.J. Jr. Computers and the shape of the future: Implications for teaching and learning // SIGCUE Bulletin, 1986. Vol. 18, N 2-4. P. 50-59.

3. Hudson K. Introducing CAL. Chapman and Hall, 1984.

4. Scloss P.J. et al. Efficacy of higher cognitive and factual questions // Journal of Computer-Based Instruction. Vol. 13, N 3. P. 75-79.

5. Anderson J., Skworecki E. The automated tutoring of introductory computer programming // Comm. ACM. 1986. Vol. 29, N 9. P. 842-849.

J. Henno, R. Rünne

The INTELLA Expert System Kernel

Abstract

A special-purpose expert system kernel INTELLA is described designed to create school tutoring CAL systems. INTELLA enables the user to create small knowledge bases covering e.g. some mathematical topics and to investigate logical consequences from this knowledge. The user can ask for explanations and solve numerical problems. The user interface is close to natural language.

J. Henno, R. Rünne

Ekspertsüsteemi tuum INTELLA

Kokkuvõte

Artiklis kirjeldatakse ekspertsüsteemi tuuma, mis võimaldab luua ekspertsüsteemi koolimatemaatika erinevate teemade õppimiseks. Süsteemi abil saab luua ja uurida väikesi teadmiste baase. Süsteem on võimeline oma tööd põhjendama. Artiklis on toodud näide süsteemi tööst.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

П.Л. Выханду

О НЕКОТОРЫХ АСПЕКТАХ СКОРОСТНЫХ СВОЙСТВ БИНАРНЫХ БАЗ ДАННЫХ

I. Введение. В последние годы в литературе очень часто встречаются статьи, рассматривающие бинарное представление данных. При таком представлении базу данных рассматривают как двухмерную матрицу, связывающую объекты со значениями атрибутов. Если строки матрицы представляют объекты и столбцы значений атрибутов, то элементы матрицы получают значение либо "I" либо "0", в зависимости от того, имеется ли у данного объекта данное значение атрибута или нет.

Такое представление удобно, так как значения "I" и "0" можно хранить только в битах и потребность к памяти относительно небольшая. Кроме того, такое битовое представление позволяет легко выполнять разные операции баз данных.

В разделе 2 данной статьи кратко рассматриваются некоторые техники и для поиска и для компрессии данных, используемые на бинарных базах данных. Дается анализ подходимости бинарных баз данных в сравнении с инвертированными файлами и определяют параметры оптимальной компрессии.

В разделе 3 дается анализ техник компрессии с точки зрения быстроты поиска по запросу. Показывается, какой является наилучшая структура поиска при методе простой компрессии. Еще показывается преимущество поисковой структуры, создание которой основывается на кластерном анализе.

2. Бинарные базы данных и поиск на них. В этом разделе введем понятие запроса, рассматриваем битовое представление данных и методы компрессии данных.

При поиске по запросу определяют, какими свойствами должны обладать искомые объекты. Допустим, что в запросе определяют q атрибутов, по которым проводится поиск, и каждый атрибут может иметь v_j значений. Так как в данной статье нас интересует процесс поиска, а не проблемы представления запросов, то для простоты предположим, что запрос имеет следующий схематический вид:

$$(X_{11} \vee X_{12} \vee \ldots \vee X_{1v_1}) \& \ldots \& (X_{q1} \vee X_{q2} \vee \ldots \vee X_{qv_q}).$$

Теперь рассмотрим битовое представление базы данных.

Пусть имеется n объектов с r признаками, где признак j имеет v_j различных значений. Тогда каждому объекту можно поставить в соответствие битвектор длиной m, где

$$m = \sum_{j=1}^{r} v_j .$$

Из этих векторов образуется битматрица и общий вид элемента матрицы будет следующий

$$a_{ij} = \begin{cases} 1 , & \text{если объект } i \text{ имеет значение } j, \\ 0 , & \text{если объект } i \text{ не имеет значения.} \end{cases}$$

В статье [I] предлагается следующий способ поиска по запросу.

Каждый столбец образует т.н. битовое изображение, где каждой позиции соответствует один объект. При поиске в запросе определяют значения, по которым берут битовые изображения и выполняются между ними логические операции. Логические операции производятся по известным правилам, как указано ниже:

x	y	x∨y	x&y
0	0	0	0
0	I	I	0
I	0	I	0
I	I	I	I

В результате получим новое битовое изображение, которое соответствует данному запросу. Каждой позиции, где находится I, соответствует объект, удовлетворяющий запросу.

Поиск по такой технике можно сравнивать с поиском в инвертированных файлах, который очень быстрый. Рассмотрим, насколько битовые изображения требуют памяти в сравнении с инвертированными файлами. Допустим, что у нас имеется n записей, p – процент записей, имеющих заданное значение и l – число битов для сохранения заданного значения в инвертированном списке. Тогда для хранения инвертированного списка понадобится $n \cdot p \cdot l$ битов. Расход памяти при обоих методах равен, если

$$n \cdot p \cdot l = n$$

и отсюда вытекает, что бинарное представление лучше, если

$$p > 1/l.$$

Так как обычная длина ключа имеет 32 бита, то оказывается, что эффективнее хранить данные в битовых изображениях при $p > 3\ \%$.

С другой стороны, предположим, что файл имеет m значений. В этом случае метод битовых изображений потребует $m \cdot n$ битов, а инвертированный список – $l \cdot n$ битов. Отсюда следует, что использование битовых изображений оправдано при $m < 32$. Кроме того, с ростом числа объектов растет и длина малозаполненной матрицы и хранение битматрицы в оперативной памяти невозможно. В статье [1] отмечают, что такую технику целесообразно использовать в том случае, когда число объектов в пределах от 10 000 до 100 000.

Из этого следует, что возникает потребность в компрессии битовых изображений. Для этого авторы статьи [3] предлагают следующий подход.

Для заданного числа последовательных битов в одном столбце битматрицы, которую назовем матрицей порядка ноль, хранится один бит. Значение его получается дизъюнкцией битов, которые этот бит представляют. Из этих битов составляется новая матрица порядка один. Число битов, между которыми проводится дизъюнкция, называется фактором компрессии и обозначается через букву c.

Если размеры начальной матрицы

$$T_0 = n \cdot m,$$

где n – число записей и
m – число значений,

тогда

$$T_1 = m \cdot \lceil n/c \rceil$$

является размером матрицы порядка один.

В общем случае, компрессия выражается в следующем.

Каждому c битов в матрице порядка f соответствует один бит в матрице порядка $f+1$, значение которого получено дизъюнкцией этих c битов. Тогда размеры матрицы порядка

$$T_f = m \cdot \lceil n/c^f \rceil .$$

Поиск записей, удовлетворяющих запрос, начинается в матрице высшего порядка. Каждый бит в матрице порядка f представляет c^f битов в матрице порядка ноль. Если в строке матрицы порядка f найдем бит со значением "0", то в матрице порядка ноль в c^f строках в соответствующих битах тоже значение "0", и их соответствие запросу уже не надо проверять. Но такое преимущество ограничено - чем выше порядок матрицы, тем больше в его битах значений "1".

Это значит, что где-то есть граница, где дальнейшая компрессия себя не оправдывает, так как число значений "1" настолько велико, что матрица дает слишком малое количество информации для эффективного поиска.

В статье [3] проведен анализ, в результате которого указаны параметры, при каких компрессия себя оправдывает. В анализе учитывается ситуация, где биты в матрице порядка ноль располагаются на максимальном расстоянии друг от друга.

Используются следующие обозначения:

n - число записей;
k_j - число записей, имеющих значение j;
a_j - частота значения ($a_j = k_j/n$);
c - фактор компрессии;
f - порядок компрессии.

В результате анализа выясняется [3], что компрессия порядка f оправдывает себя, если

$$a_j = \left[\frac{f}{f+1}\right]^{f(f+1)}$$

и оптимальным фактором компрессии является

$$c = \left[\frac{f+1}{f}\right]^{f}.$$

Вероятность появления значения "0" возрастает, если дизъюнкция проводилась между теми строками матрицы, которые между собой наиболее схожи. Для нахождения таких строк возникает потребность в кластеризации. В статье [2] предлагается метод, где для этой цели используются монотонные системы. При использовании монотонных систем, должна существовать функция π, которая измеряет значимость элементов монотонной системы. Точки экстремума функции π определяют ядра монотонной системы, с помощью которых выделяют группы похожих записей. Дизъюнкцией соответствующих битовых изображений, принадлежащих к одной группе, образуется одно суперизображение, называемое адресом. После того, как будут созданы адреса всех групп, из них образуется новая монотонная система и снова начинается выделение групп схожих адресов, с помощью дизъюнкции которых получают новые адреса, на уровень выше. Монотонные системы являются очень подходящим средством для кластеризации в сравнении с классическими методами, так как они требуют лишь $o(n^2)$ операций.

Ясно, что процесс поиска при двух последних методах тот же самый, но надо рассматривать скоростные свойства поиска в обоих случаях.

3. Анализ скоростных свойств поиска. В этом разделе будем показывать, какой является наилучшая структура поиска для метода описанного в статье [3] и что метод, где группы образуются по схожести записей, никогда не бывает хуже.

Как уже отмечено, поиск записей, удовлетворяющих запросу, проводится тем быстрее, чем больше на высших уровнях можно исключить группу записей, т.е. чем больше значений "0" в представителях. Для получения такой ситуации будем проводить перестановку строк битматрицы. Для нахождения наилучшего порядка строк матрицы используем следующую процедуру.

Допустим, что задана битматрица и число единиц в каждом столбце κ_j, $j=1,\ldots,m$. Для каждой записи найдем ее значение конформизма S_i, которое определяется

$$S_i = \sum_{j=1}^{m} \sigma_{ij}, \quad \sigma_{ij} = \begin{cases} \kappa_j, & \text{если } a_{ij} = 1 \\ n - \kappa_j, & \text{если } a_{ij} = 0 \end{cases}$$

Найдем $\min_i S_i$ и отметим соответствующую запись как первую в новом порядке. После того найдем среди остальных записей запись с минимальным расстоянием от отмеченной записи. Если найдется несколько равных расстояний, то выберем запись с минимальным значением конформизма. Для измерения расстояния используем расстояние Хемминга

$$d(a_q, a_p) = \sum_{j=1}^{m} (a_{qj} \oplus a_{pj}),$$

где операция $\oplus$ определяется следующим образом

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Повторив рекурсивно эту процедуру, получим такое упорядочение строк битматрицы, где все записи располагаются на минимальном расстоянии от своих соседей.

Скорость поиска зависит с одной стороны, от числа значений "I" в представителе и с другой стороны, от числа записей в группе, которую надо проверить, если представитель отвечает запросу. Это значит, что самый быстрый поиск осуществляется, если минимизируется мера.

$$M = \sum_{i=1}^{\lceil n/c \rceil} c\omega_i,$$

где ω_i – количество значений "I" в представителе i-ой группы,

c – фактор компрессии.

Эту меру назовем мерой эффективности поиска.

Рассмотрим подробнее, как выражается величина ω_i при описанном виде битового представления данных.

Допустим, что в группе имеется q записей с r признаками, где признак j имеет v_j различных значений. Тогда расстояние каждой записи в группе от своего представителя будет равным и число значений "I" в представителе i-ой группы вычисляется

$$\omega_i = \sum_{j=1}^{n} d_{ij} + 1,$$

где d_{ij} - расстояние записей i-й группы от своего представителя при j-м признаке.

Так как строки битматрицы упорядочены таким образом, что они расположены на минимальном расстоянии друг от друга, то величины ω_i минимизированы. Фактор компрессии является константой и оптимальным для заданной битматрицы, поэтому можно утверждать, что не существует лучшей структуры для поиска при методе простой компрессии.

Очевидно, что абсолютный минимум меры M достигается в том случае, когда все строки битматрицы одинаковые, тогда

$$\min M = \min \sum_{i=1}^{\lceil n/c \rceil} c\,\omega_i = \lceil n/c \rceil\, cr.$$

Абсолютный максимум меры M достигается в том случае, если во всех группах имеются такие строки, когда все биты их представителей заполнены значением "I", тогда

$$\max M = \max \sum_{i=1}^{\lceil n/c \rceil} c\omega_i = \lceil n/c \rceil\, cm.$$

Такие ситуации в реальных базах данных, естественно, не встречаются. Для исследования распределения меры эффективности поиска M проведена симуляция, в ходе которой строки матрицы упорядочены в случайном порядке и каждый раз найдена мера эффективности поиска. Результаты этого процесса показаны на рисунке I.

Далее рассмотрим случай, где группы записей создаются по их схожести. Для этого битматрицу надо рассматривать как монотонную систему и выделить ее ядра. Выделение ядер происходит удалением всех строк битматрицы в порядке их расположения. В качестве значений функции π будем использовать значения конформизма s_i и найдем точки экстремума. Тогда можно делить битматрицу на T групп и каждой группе принадлежит t_i, $(i = 1,\ldots,T)$ записей. По такой группировке мера эффективности выражается

$$M' = \sum_{i=1}^{T} t_i \omega_i .$$

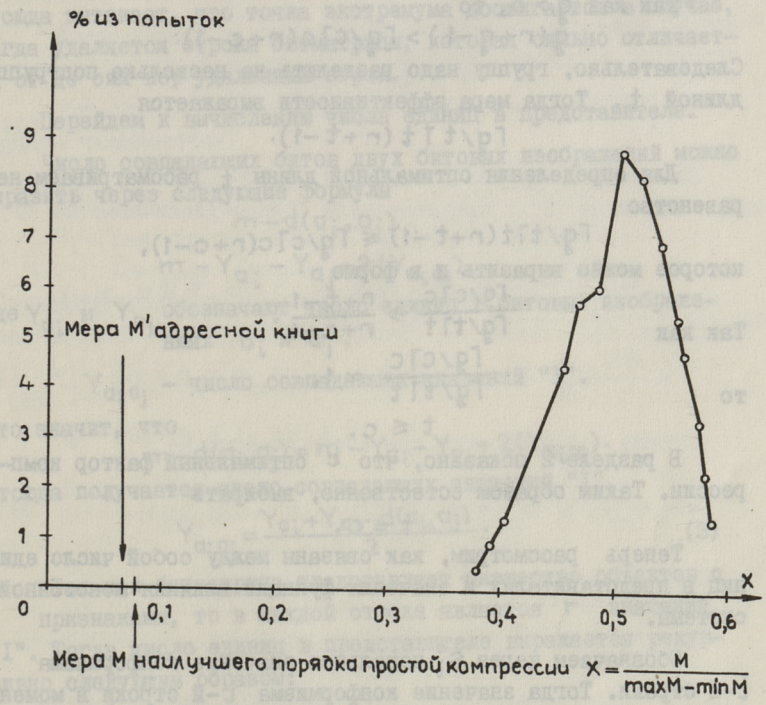


Рис. 1.

Отдельно должен быть рассмотрен случай, когда в одну группу входят такие строки битматрицы, которые различаются друг от друга таким образом, что при дизъюнкции каждой строки с представителем, в представитель добавляется один бит со значением "I".

Предположим, что в группу входит g записей и $g > c$, где c – фактор компрессии при методе простой компрессии.

Мера эффективности для такой группы равняется

$$g(r+g-1),$$

где r – число атрибутов в записи.

Мера эффективности, если эта группа разделена на подгруппы длиной c, равняется

$$\lceil g/c \rceil c(r+c-1).$$

Так как $g > c$, то

$$g(r+g-1) > \lceil g/c \rceil c(r+c-1).$$

Следовательно, группу надо разделить на несколько подгрупп длиной t. Тогда мера эффективности выражается

$$\lceil g/t \rceil t(r+t-1).$$

Для определения оптимальной длины t рассматриваем неравенство

$$\lceil g/t \rceil t(r+t-1) \leq \lceil g/c \rceil c(r+c-1),$$

которое можно выразить и в форме

$$\frac{\lceil g/c \rceil c}{\lceil g/t \rceil t} \geq \frac{r+t-1}{r+c-1}.$$

Так как

$$\frac{\lceil g/c \rceil c}{\lceil g/t \rceil t} \approx 1,$$

то

$$t \leq c.$$

В разделе 2 показано, что c оптимальный фактор компрессии. Таким образом естественно, выбирать

$$t = c.$$

Теперь рассмотрим, как связаны между собой число единиц в представителях и значения функции влияния монотонной системы.

Обозначаем через S_i^0 начальное значение конформизма i-й строки. Тогда значение конформизма i-й строки в момент ее удаления равно

$$S_1 = S_1^0, \quad S_i = S_i^0 - \sum_{j=1}^{i-1}(m - d(a_i, a_j)), \quad i = 2, \ldots, n,$$

где m – количество значений признаков

$d(a_i, a_j)$ – расстояние между i-й и j-й строками.

Точка экстремума найдется тогда, когда

$$S_i \leq S_{i+1}$$

или

$$S_i^\circ - \sum_{j=1}^{i-1} m + \sum_{j=1}^{i-1} d(a_i, a_j) \le S_{i+1}^\circ - \sum_{j=1}^{i} m + \sum_{j=1}^{i} d(a_{i+1}, a_j) \Rightarrow$$

$$S_i^\circ - S_{i+1}^\circ + m \le \sum_{j=1}^{i} d(a_{i+1}, a_j) - \sum_{j=1}^{i-1} d(a_i, a_j). \qquad (2)$$

Отсюда вытекает, что точка экстремума достигается в случае, когда удаляется строка битматрицы, которая сильно отличается от до сих пор удаленных строк.

Перейдем к вычислению числа единиц в представителе.

Число совпадающих битов двух битовых изображений можно выразить через следующие формулы

$$m - d(a_i, a_j),$$

и

$$m - Y_{a_i} - Y_{a_j} + 2(Y_{a_i a_j}),$$

где Y_{a_i} и Y_{a_j} обозначают число единиц в битовых изображениях a_i и a_j,

$Y_{a_i a_j}$ - число совпадающих значений "I".

Это значит, что

$$m - d(a_i, a_j) = m - Y_{a_i} - Y_{a_j} + 2(Y_{a_i a_j}).$$

Отсюда получается число совпадающих значений "I".

$$Y_{a_i a_j} = \frac{Y_{a_i} + Y_{a_j} - d(a_i, a_j)}{2}. \qquad (3)$$

Так как битматрица представляет множество объектов с признаками, то в каждой строке является r значений "I". Тогда число единиц в представителе выражается рекурсивно следующим образом:

$$\omega_1 = r,$$

$$\omega_2 = Y_{D_1 D_2} + d(D_1, D_2) = \frac{Y_{D_1} + r + d(D_1, D_2)}{2},$$

$$\vdots$$

$$\omega_{i+1} = Y_{D_i D_{i+1}} + d(D_i, D_{i+1}) = \frac{Y_{D_i} + r + d(D_i, D_{i+1})}{2},$$

где

$$D_i = \bigvee_{j=1}^{i} a_j.$$

После несложных упрощений количество единиц ω_k, в представителе D_k выражается через расстояния соседей следующим образом

$$\omega_k = \frac{\omega_{k-1} + r + \sum_{j=1}^{k-1} d(a_j, a_{j+1}) - \omega_{k-1} + \omega_1}{2}.$$

Так как $\omega_1 = r$, то

$$\omega_k = \frac{2r + \sum_{j=1}^{k-1} d(a_j, a_{j+1})}{2}$$

и

$$\omega_{k+1} = \frac{2r + \sum_{j=1}^{k} d(a_j, a_{j+1})}{2}. \qquad (4)$$

Следовательно, при добавлении одной строки к группе, число значений "I" в представителе группы увеличивается на

$$\omega_{k+1} - \omega_k = \frac{2r + \sum_{j=1}^{k-1} d(a_j, a_{j+1}) + d(a_k, a_{k+1}) - 2r - \sum_{j=1}^{k-1} d(a_j, a_{j+1})}{2} =$$

$$= \frac{d(a_k, a_{k+1})}{2}. \qquad (5)$$

Отсюда получаем, что при точке экстремума

$$d(a_i, a_{i+1}) \geq S_i^o - S_{i+1}^o + m - \sum_{j=1}^{i-1} (d(a_{i+1}, a_j) - d(a_i, a_j)), \qquad (6)$$

а в противоположном случае

$$d(a_i, a_{i+1}) < S_i^o - S_{i+1}^o + m - \sum_{j=1}^{i-1} (d(a_{i+1}, a_j) - d(a_i, a_j)).$$

При заданном порядке строк битматрицы разность начальных конформизмом $S_i^o - S_{i+1}^o$ для i-й строки будет постоянной и выражается следующим образом:

$$S_i^o - S_{i+1}^o = \sum_{j=1}^{n} (m - d(a_i, a_j)) - \sum_{j=1}^{n} (m - d(a_{i+1}, a_j)) =$$

$$= \sum_{j=1}^{n} (d(a_{i+1}, a_j) - d(a_i, a_j)). \qquad (7)$$

Отсюда условие (5) преобразуется на вид

$$d(a_i, a_{i+1}) \geq m + \sum_{j=i}^{n} d(a_{i+1}, a_j) - \sum_{j=i}^{n} d(a_i, a_j). \qquad (8)$$

Таким образом, получено простое правило, связывающее число увеличения количества единиц в представителе с выделением ядра по теории монотонных систем.

Полученный результат дает возможность провести сравнение между простой компрессией и группировками по схожести.

Для сравнения мер эффективности M и M' введем следующие обозначения:

G_i – i-я группа по схожести длиной t_i, $i = 1, \ldots, T$,

F_i – i-я группа простой компрессии длиной c, $i = 1, \ldots, \lceil n/c \rceil$.

Порядок строк битматрицы при обоих способах группировки одинаковый. При группировке можно различить две ситуации:

1) $F_k = \{a_i \mid a_i \in G_e\}$,

2) $G_e = \{a_i \mid a_i \in F_k \vee a_i \in F_{k+1}\}$.

Если $T \geq \lceil n/c \rceil$, то из выражения (8) очевидно, что

$$\sum_{i=1}^{T} t_i \omega_i' = M' \leq M = \sum_{i=1}^{\lceil n/c \rceil} c\omega_i,$$

так как группы образуются перед тем, когда число единиц в представителе существенно увеличивается.

Если же $T < \lceil n/c \rceil$, то существует несколько групп F_j, которые соответствуют ситуации 1). Так как порядок строк битматрицы определен, то исключение этих групп не влияет на число единиц представителей ω_i'. Исключение тех групп приводит нас поэтому к ситуации, где все группы соответствуют ситуации 2). В этом случае $T \geq \lceil n/c \rceil$, а тогда $M' \leq M$.

4. Заключение. Бинарные базы данных являются хорошим средством для хранения и поиска данных. Обработка сложных запросов проводится легче, чем при традиционных методах. Компрессия битматриц ускоряет поиск. Особенно эффективной является компрессия, проведенная по схожести данных. Как показывают испытания, поиск на адресных книгах проводится в 1,5–3 раза быстрее, чем при методе простой компрессии.

Литература

I. Выханду Л.К., Выханду П.Л. Быстрый поиск на битматрицах / Тр. Таллинск. политехн. ин-та. 1983. № 554. С. 49-60.

2. Vyhandu P. New ideas in data base segmentation // Proceedings of Tallinn Technical University. 1986. N 614. P. 93-105.

3. Spiegler I., Maayan R. Storage and retrieval considerations of binary databases // Information Processing and Management. 1985. Vo. 21, N 3. P. 234-254.

P. Võhandu

Some Aspects of Quick Search in Binary Data Bases

Abstract

In this paper binary data bases are introduced. For more effective search special structures are created without and with preliminary data analysis. The comparative analysis of these structures is given.

P. Võhandu

Binaarsete andmebaaside mõnedest kiirusomadustest

Kokkuvõte

Artiklis kirjeldatakse andmebaaside binaarset esitust. Päringute järgi andmete otsimiseks luuakse otsistruktuurid nn. lihtkompressiooni meetodil ja meetodil, mis arvestab ka andmete sarnasust. Seejärel esitatakse nende struktuuride võrdlev analüüs.

№ 671
TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

Т.Э. Вашпер, Ю.Г. Лааст-Лаас,
П.У. Распель

ПРОБЛЕМЫ СТРАТЕГИЧЕСКОГО ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

I. Введение

Опыт показал, что проектирование информационных систем (ИС) без использования специального математического обеспечения процесс слишком трудоемкий, медленный и результаты его не всегда достаточны для реализации системы.

В данной статье представим коротко основные шаги методики проектирования с точки зрения создания инструментального средства стратегического проектирования информационных систем.

Под информационной системой понимается автоматизированное средство проведения информационной работы (ИР) на предприятии (в организации) [1].

Понятие "стратегическое проектирование" используем для обозначения по существу однородных процедур стратегического планирования и детального проектирования информационных систем. Разница между стратегическим планированием и концептуальным проектированием ИС заключается в аспектах описания и анализа, а также в степени детализации (см. п. 5). Стратегическое планирование связано в большей степени с производительностью создаваемой ИС, т.е. реорганизацией ИР предприятия в целях повышения эффективности и минимизации расходов на ИР. Проектирование информационной системы связано со стабильностью ИС, т.е. с проектированием третьей нормальной формы, гибкостью и быстротой разработок будущих приложений и т.д., у которого основой является заранее разработанный стратегический план [2].

Далее рассмотрим в статье, каким информационным потребностям и приложениям должна отвечать создаваемая автоматизированная система проектирования информационных систем (СПИС), и как смоделировать нижеописанную объект-систему. С этой целью составим соответствующую концептуальную схему и представим требования к реализации системы. Если проследить последовательность жизненного цикла ИС [4], то автоматизированная система проектирования информационных систем (СПИС) должна иметь следующие основные возможности:

- описание реальной системы;
- анализ реальной системы;
- проектирование логических структур ИС.

2. Описание реальной системы

Для описания реальной системы представим ИР предприятия в форме "Кубика" (рис. I).

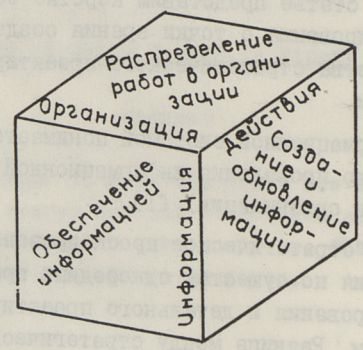


Рис. 1. Модель ("кубик") ИР предприятия.

"Кубик" включает в себя следующие компоненты ИР: организация, работники, занимающиеся действиями ИР и данными (субъекты ИР);
данные, информация о явлениях, событиях, оборудовании, о материалах и т.д. (объекты ИР);
действия ИР (рабочие процедуры, процессы).

Все вышеуказанные компоненты ИР тесно взаимосвязаны. Работники предприятия занимаются определенными действиями, которые требуют той или иной информации. В результате этой

деятельности создают новую информацию, которая, в свою очередь, необходима другим работникам и действиям.

Данные представляют в виде каких-либо носителей данных, обычно это документы (отчеты, таблицы, акты, справки и т.д.). Кроме официальной информации (документы) существует и неофициальная информация (по телефону, устно). И та и другая информация должна быть описана с помощью СПИС. На последующих этапах жизненного цикла ИС нужна информация о конкретных данных и о соответствующих носителях данных (см. п. 5).

Организационную структуру необходимо описывать как на уровне отделов, так и на уровне конкретных лиц, занятых в одной или разных областях деятельности предприятия.

Действия связывают субъекты и объекты ИР.

В качестве действий, информации и организации можно описывать и ранее использовавшиеся файлы и базы данных, а также программные системы и автоматизированные процедуры.

При сборе информации о компонентах ИР, к проектировщикам ИС поступают разного рода предложения о реорганизации ИР от работников на соответствующих участках предприятия, однако они требуют анализа. Таким образом, и у проектировщиков в свою очередь возникают некоторые проблемы, требующие решений на дальнейших этапах проектирования. Информация о предложениях и проблемах должна быть описана с помощью СПИС.

СПИС должен отвечать на следующие вопросы:

- кто какими действиями занимается;
- кто какие документы использует;
- кто какие данные использует;
- кто какие документы создает;
- кто какие данные создает;
- какие данные на каких документах (реквизиты документов);
- какие действия используют те или иные документы;
- какие действия создают те или иные документы;
- какие действия используют те или иные данные;
- какие действия создают те или иные данные;

- какие проблемы и предложения требуют анализа.

СПИС соединяет в себе все традиционные ("ручные") методы описания реальной системы. Для анализа, тестирования и наглядного представления описания реальной системы потребуются специальные формы. Поэтому СПИС должен выдавать следующие приложения:

- схемы организационной структуры предприятия,
- функциональные схемы действий ИР,
- матрицы процессов (действий) и организационной структуры,
- матрицы организационной структуры и программных систем,
- матрицы программных систем и баз данных (файлов),
- матрицы информационных потоков [2, 3].

3. Анализ реальной системы

Главными задачами анализа реальной системы являются:

- определение "уязвимых мест" ИР,
- разработка проанализированных предложений о реорганизации ИР предприятия [4].

На базе описания реальной системы анализируют, насколько существующая ИС удовлетворяет потребности работников ИР в информации. Выясняются недостатки обеспечения информацией, т.н. "уязвимых мест", требующих особого внимания во время проектирования ИС. Часто незначительное количество "уязвимых мест" служит причиной неудовлетворенности существующей ИС (порядка ИР) и крайне негативно влияет на эффективность всей работы предприятия в целом.

При ручной обработке данных их часто приходится дублировать. У разных работников одинаковые потребности в информации, но преследующих разные цели. Это порой приводит к дублированию документов и созданию новых, в которых избыток данных, полностью или частично. Со дня существования предприятия количество документов быстро растет, что ведет и к дублированию действий. Работники разных отделов, выполняющие разные функции, занимаются идентичными действиями.

Объем ИР растет и требуется дополнительный состав трудовой

силы. Чтобы избежать подобных аномалий, надо с помощью СПИС определить:

- избыточность данных (документов);
- избыточные действия ИР.

Для каждого действия ИР необходима информация. На основе входной информации создают те или иные приложения (отчеты, новые документы и т.д.), являющиеся входной информацией для других работников и действий. Образуются потоки информации, которые описываются матрицей информационных потоков.

Реорганизация ИР (ликвидация документа, действия, изменение структуры документа и т.д.) на одном участке ИР вызывает изменения на другом участке. ИР можно реорганизовать только комплексно, анализируя все возможные аномалии, которые возникли и были обусловлены какими-то изменениями.

Обновление данных (в документах, в базе данных) вызывает необходимость обновления и других данных. Проведение одного действия ИР вызывает проведение другого действия (процессы ИР).

Изменения действий и данных влияют на организационную структуру предприятия. Ликвидация избытка данных уменьшает функции субъектов ИР, позволяет сократить количество рабочей силы и дает возможность перестроить организационную структуру предприятия.

СПИС проводит анализ последствий изменений:

- по потокам данных;
- по процессам ИР;
- по субъектам ИР;
- по потокам данных и по процессам;
- по потокам данных, по процессам и по субъектам ИР.

СПИС способствует разработке и представлению лучших вариантов проведения ИР.

4. Проектирование логических структур ИС

В результате описания и анализа реальной системы определяется информационная структура, которая представляется в виде концептуальной схемы. Составление концептуальной

схемы, определение объектов и отношений между ними не подчиняется формальным алгоритмам и остается творческой работой проектировщиков ИС. С помощью СПИС дается необходимая информация, описывается созданная схема и тестируется ее адекватность.

На этапе проектирования логических структур СПИС проводит:

- декомпозицию концептуальной схемы на логические схемы;
- определяет число и последовательность действий ИР в соответствии с логической схемой (определение подсистемы ИС);
- определяет потоки данных (создание, удаление, обновление данных) в подсистеме;
- определяет потоки данных между подсистемами;
- определяет процессы (последовательность действий) между подсистемами;
- определяет потребности и доступ к данным субъекта ИС.

Составленная на основе потребностей информации описания и анализа реальной системы, а также проектирования логических структур ИС, концептуальная схема СПИС представлена на рис. 2.

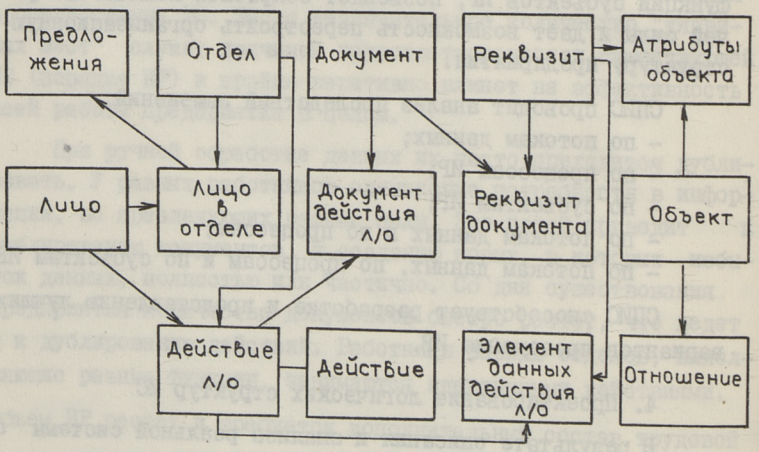


Рис. 2. Концептуальная схема СПИС.

5. Логические уровни и степени детализации

Концептуальная схема СПИС (см. рис. 2) создана для проведения работ проектирования на разных логических уровнях. СПИС поддерживает стратегическое планирование (информационных ресурсов), концептуальное проектирование ИС, проектирование реализации ИС (логическое проектирование ИС), проектирование информационных подсистем, проектирование автоматизированных рабочих мест, создание информационной архитектуры предприятия, проектирование и реализацию математического обеспечения на разных логических уровнях.

Проектирование системы на определенном логическом уровне предполагает выбор использования понятий в соответствии со степенью детализации. Например, действия можно описать следующими степенями детализации: элементарные действия, действия, процессы, функции (комплексные действия). Если элементарное действие реализуется как соответствующая автоматизированная процедура, то функция может описать несколько тысяч элементарных действий, а служит для описания и анализа, но не для реализации системы. Данные можно описать такими степенями детализации: документы, реквизиты документов, элементы данных (атрибуты объектов).

В зависимости от выбранного логического уровня и субъектов ИР делается описание и анализ отдельно по лицам, по рабочим группам, по отделам и т.д.

Например: одним из результатов стратегического проектирования является карта объектов, которой на логическом уровне детального проектирования соответствует концептуальная схема. Информационной базе подсистемы, при создании информационной архитектуры предприятия, на другом логическом уровне соответствует база данных автоматизированного рабочего места.

Разные логические уровни и степени детализации компонентов ИР отличает потребность в точности.

Если описание существующей системы не требует дифференциации степеней детализации, то анализ можно проводить только для конкретной цели на соответствующем логическом уровне, определенном степенью детализации.

6. Проектирование реализации

Независимо от незначительного объема нами исследуемого объекта - в определенный момент его описание будет обладать такими измерениями, которые превысят охват проектировщиков. К тому же еще, чем больше исследуемый нами объект, тем с большей вероятностью мы можем упустить из вида то или иное очень существенное отношение между объектами, которое уже ранее было просмотрено. Большой проблемой является и оптимизация собранных данных, требующая одновременного наблюдения всех отношений и групп отношений.

Большинства указанных трудностей можно избежать, если создать инфосистему (СПИС), которая бы поддерживала методику проектирования. Эта система должна охватывать все основные этапы и действия проектировочных работ. Так как подобная система выполняет в основном операции сохранения, сортировки и корректировки данных, а также операции определения отношений между данными, то для ее реализации можно использовать любую СУБД, в которой имеются функции сортировки и прямого доступа к данным.

Ядром такой инфосистемы является база данных, хранящих все данные, которые нужны для проектировочной работы. Таким образом, эффективность системы в основном зависит от того, насколько адекватно можно отражать отношения описываемой объект-системы в базе данных. Взяв за основу инфологическую схему структуры предприятия и информационных работ, приведенную на рис. 2., можно построить схему базы данных, которая с довольно большой точностью удовлетворит наши потребности (см. рис. 3).

Разница между этими двумя схемами состоит в том, что изменен центральный объект. Если на первой схеме центральным объектом является "действие л/о", то на второй схеме таковым является "действие". Такое изменение центрального объекта уменьшает степень интеграции в схеме, и тем самым свободнее определяются отношения между разными объектами

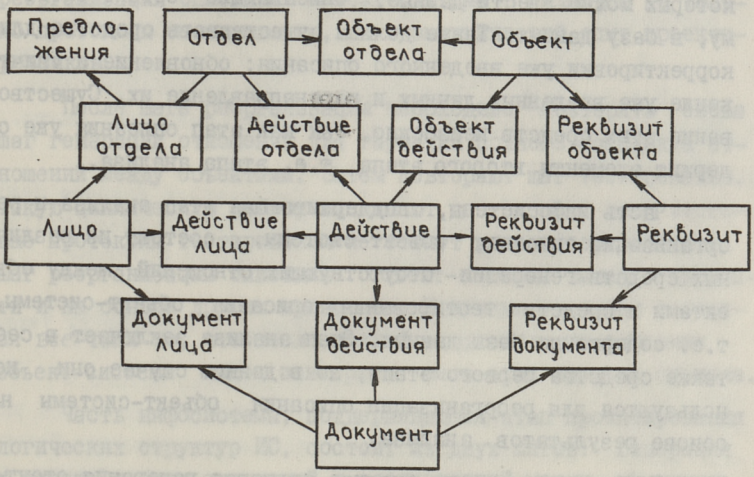


Рис. 3. Схема БД СПИС.

описываемой системы. Оно сопровождается и возможностью генерации отношений между объектами, которую рассмотрим ниже.

Кроме того, изменение центрального объекта дает нам возможность отказаться от "трех- и четырехмерных" отношений: "документ действия л/о", "элемент данных действия л/о" и "действие л/о" (см. рис. 2). Анализ этих отношений по причине их "трех и четырехмерности" очень трудно осуществить и к тому же результаты этого анализа не дают ясного представления об этих отношениях. Вместо них между парами объектов использованы обычные отношения типа "много-много". Это дает возможность значительно упростить анализ отношений между объектами и сделать результаты анализа более четкими.

Как уже сказано выше, создание концептуального проекта предприятия состоит в основном из трех этапов: описание объектной системы, анализ описания объект-системы, проектирование логических структур ИС. Все эти этапы должны быть отражены и в инфосистеме, поддерживающей процесс проектирования.

Часть инфосистемы, поддерживающая этап описания объект-системы, состоит в основном из средств, с помощью

которых можно ввести данные, описывающие объект-систему, в базу данных. Также должны существовать средства для корректировки уже введенного описания: обновление и уничтожение уже введенных данных и перенаправление их. Существование этих средств неизбежно, так как этап описания уже содержит элементы второго этапа, т.е. этапа анализа.

Часть инфосистемы, поддерживающая этап анализа и реорганизации описания объект-системы, состоит из различных средств генерации отсутствующих отношений между объектами и средств тестирования описания объект-системы, т.е. содержания базы данных. Этап анализа заключает в себе также средства первого этапа, но в данном случае они используются для реорганизации описания объект-системы на основе результатов анализа.

Первым шагом этапа анализа является генерация отсутствующих отношений между объектами. Если посмотреть схему базы данных на рисунке 3, то можно увидеть, что схема состоит из симметрических треугольников. Во всех этих треугольниках можно найти одну такую связь между объектами, которые располагаются в углах треугольника, где из двух отношений между ними вытекает и третье отношение. Например, если в треугольнике "отдел" - "действие" - "объект" существуют отношения "действие отдела" и "объект действия", то существует и отношение "объект отдела". Это значит, что если какой-то отдел занимается действием и действие занимается объектом, то и этот отдел занимается этим объектом. Другие подобные связи в этом треугольнике найти невозможно - связь некоммутативная. В то же время эту связь можно перевернуть, т.е. возможна ситуация, где прерывание одного отношения между двумя объектами вызывает исчезновение еще двух отношений.

После шага генерации в описании объект-системы имеется полный комплект отношений между объектами, и можно начинать тестирование оптимальности описания объект-системы. В ходе этого шага проверяют однократность всех действий, совпадение действий рабочих, целесообразность реквизитов на документах и т.д. (см. п. 3), итоги шага тестирования - протоколы тестирования, на основе которых будет проведена реорганизация описания объектной системы. Основой

реорганизации описания объект-системы являются не только протоколы тестирования, но и эмпирический опыт проектировщиков.

После шага реорганизации необходимо повторять снова шаг генерации отношений для гарантии полного комплекта отношений между объектами. Затем повторяют шаг тестирования. Такую циклическую работу выполняют до тех пор, пока итоговые протоколы тестирования не станут удовлетворительными. Шаг реорганизации описания объект-системы можно провести и на основе других принципов. Конечным условием остается все-таки то, что после каждой реорганизации описания объект-системы надо повторять весь этап анализа сначала.

Часть инфосистемы, поддерживающая этап проектирования логических структур ИС, состоит из двух шагов: генерация и печать таблиц отношений и соответствий между объектами (см. п. 3); генерация и печать итоговых описаний логических структур ИС. На основе первого шага вновь производится реорганизация описания объект-системы, если это необходимо. После этого возвращаются к этапу анализа. Второй шаг заканчивает процедуру проектировочных работ. В ходе этого шага делают генерацию и распечатку всех итоговых документов проектирования, которыми являются, например, распределение рабочих операций по рабочим местам, полные перечни нужных документов, рабочих мест, отделов, информационных потоков и т.д. (см. п. 2, 3, 4, 5).

Описываемая инфосистема не является средством автоматного проектирования ИС, а только дает в руки проектировщиков мощные средства для поддержки их рабочих операций. Это значит, что для проектирования ИС недостаточно опыта работы с данной системой, а требуются еще и высокие профессиональные знания и опыт по проектированию ИС.

Литература

I. Ваппер Т., Касемаа Р., Лааст-Лаас Ю., Тепанди Я. О создании информационных систем в Эстонии // Вычислительная техника и обработка данных - 1988. № I. ЭНИ. Таллин. С. 6-I?. (на эст. языке).

2. Мартин Дж. Развитие планирования автоматизированных систем. М.: Финансы и статистика, 1984.

3. Ваппер Т., Лаасг-Лаас Ю., Урвак А., Эльмик Л. Практические аспекты проектирования информационных систем // Тр. Таллинск. политехн. ин-та. 1987. № 645. С. 118-125.

4. Lockemann P.C. Information systems design: techniques and software support // IFIP 86. North-Holland Publ. Co., 1986. P. 617-631.

J. Laast-Laas, P. Raspel, T. Vapper

Problems of Information Systems Strategic Design

Abstract

Various aspects of creating business information systems are discussed in this paper. Some experience in this field has shown that without special software the efficiency of the design process is very low. Requirements and conceptional design process of the mentioned tool are introduced here.

J. Laast-Laas, P. Raspel, T. Vapper

Infosüsteemide strateegilise projekteerimise probleemidest

Kokkuvõte

Infosüsteemide strateegiline planeerimine ja projekteerimine ilma spetsiaalse tarkvara kasutamiseta on osutunud väga töömahukaks ja aeganõudvaks tööks, kusjuures resultaadid ei pruugi olla piisavad, vastamaks süsteemi realiseerimise nõudmistele. Käesolevas artiklis näidatakse, missuguseid infovajadusi peab projekteerimissüsteem rahuldama, luuakse vastav kontseptuaalskeem ja kirjeldatakse realisatsiooni projekteerimist.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

А.В. Рензер

СЖАТИЕ ДАННЫХ ПРИ АВТОМАТИЗИРОВАННОМ СОЗДАНИИ СЛОВАРЕЙ ЕСТЕСТВЕННЫХ ЯЗЫКОВ

Введение

В статье рассматриваются проблемы сжатия текстов для их более компактного хранения во внешней памяти ЭВМ. Основой создания соответствующей надстройки для системы автоматизированного создания прикладных систем ГЕНСИ является метод кодирования данных, предложенный Хаффмэном. Описывается метод кодирования, представляются структуры данных для декодирования и соответствующие алгоритмы, дается краткая характеристика конкретной реализации на FORTRAN-е.

I. Сжатие и восстановление текстов с использованием кодов Хаффмэна

Метод создания кодов с минимальной избыточностью был предложен Хаффмэном уже в 50-ые годы. Тем не менее этот метод является основой создания различных систем сжатия данных и в настоящее время (см., например, развитие этого подхода Кнутом в [3, 4]).

Сущность метода Хаффмэна заключается в следующем. При известных вероятностях $(w_1 \ldots w_n)$ появления символа i в тексте, можно создать двоичное несбалансированное дерево из n висящих и $n-1$ промежуточных вершин, где висящие вершины (листья) обозначены вероятностными весами $(w_1 \ldots w_n)$ в определенной последовательности.

Путь по дереву от корня до висящей вершины образует цепочку из "0" и "1", называемую кодом листа. Нули обозначают левую ветвь прохождения дерева, 1 - правую. Висящая вершина на уровне m достигается по пути, длина которого m битов, т.е. листу соответствует двоичный код из m битов.

Можно доказать, что дерево кодов Хаффмэна дает уникальные коды минимальной длины для всех вероятностей: значение суммы $w_1m_1+\ldots+w_nm_n$ является минимальным для всех двоичных деревьев (m_k - уровень появления w_k в дереве).

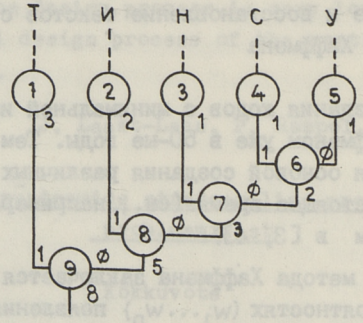
Построение двоичного дерева кодов покажем на небольшом примере.

Для кодирования текста "ИНСТИТУТ" сначала определим используемый алфавит, частоту появления каждой буквы, нарисуем кодовое двоичное дерево и покажем таблицу полученных кодов Хаффмэна.

Текст содержит следующие символы:

символ	частота попадания
И	2/8
Н	I/8
С	I/8
Т	3/8
У	I/8

Дерево удобнее всего строить в возрастающем порядке вероятностей w_1:



Ребра дерева определяются соединением двух наименьших весов (вероятностей) в промежуточную вершину. Соединением вершин 4 и 5, имеющих весы I, получим вершину 6 весом 2 и т.д. При нескольких равных вероятностях выбор соединяемых вершин может быть произвольным. Начиная с корня, всем ребрам присваивается значение 1 или 0 таким образом, что ребро в левом поддереве всегда обозначается нулем, а ребро в правом поддереве - единицей.

Структура дерева задает пути от корня до каждой из висящих вершин, определяя таким образом уникальный код для каждого символа, связанного с висящей вершиной:

символ	код	длина	вероятность попадания	вероятностная длина в битах
Т	I	I	3/8	(I * 3)/8=0.375
И	0I	2	2/8	(2 * 2)/8=0.500
Н	00I	3	I/8	(3 * I)/8=0.375
С	000I	4	I/8	(4 * I)/8=0.500
У	0000	4	I/8	(4 * I)/8=0.500

Выясняется, что самые короткие коды присваиваются наиболее часто встречающимся символам. Следовательно, большая часть закодированного текста будет состоять из коротких кодов. Средняя длина кода символа в тексте "ИНСТИТУТ" составляет 3 * 0.500+2 * 0.375=2.25 бита.

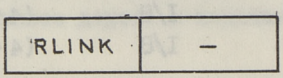
Применение такой технологии кодирования текстов на естественном языке потребует изучения обрабатываемых текстов на вероятность попадания в них различных символов. Например, для словаря диалектов эстонского языка определены следующие процентные частоты символов:

Пробел - I0.35, A - 6.96, разделитель ключевого слова - 6.86, I - 5.70, S - 4.86, T - 4.0I, L - 3.92, U - 3.6I, K - 3.44, V - 3.0I, M - 2.96, N - 2.8I, R - 2.70, запятая - 2.I0, O - 2.30, P - I.79, D - I.69, точка - I.45, N - I.36, X - I.26, 7 - I.22, Q - I.I7, G - I.22, B - I.09, J - 0.95, апостроф - 0.88, 4 - 0.74, минус - 0.60, плюс - 0.54, 2 - 0.52, < - 0.49, 0 - 0.45, B - 0.39, кавычки - 0.35, > - 0.33, 1 - 0,3I, C - 0.29, (- 0.28,) - 0.28, точка с запятой - 0.27, F - 0.I7, двоеточие - 0.06, 9 - 0.04, 5 - 0.04, 3 - 0.04, звездочка - 0.04, 8 - 0.02, 4 - 0.02, Z - 0.0I, W - 0.00I.

Остальная часть 65-символьного алфавита словарей диалектов содержит специальные символы, предназначенные для управления форматированием при распечатке текста словаря. Средняя длина кодов Хаффмэна для алфавита словарей составляет 5.03 битов, что в сравнении с обычным 8-битовым представлением дает 37,7 % экономии. При более коротких алфавитах экономия будет, конечно же, расти.

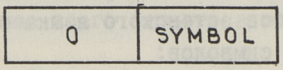
Для обеспечения автоматического декодирования кодовое дерево представляется в виде списка. Разработанная в ТПИ система сжатия текстов использует одну из наиболее эффективных структур данных для подобных задач – упорядоченный список с одной ссылкой.

Все элементы списка имеют одинаковую структуру, но семантика полей зависит от типа вершины дерева. Для промежуточных вершин структура следующая:



Поле RLINK содержит указатель на правое поддерево. Указатель вниз не требуется, так как дерево представляется в виде упорядоченного списка. А именно, элемент списка, описывающий левое поддерево вершины i, должен всегда следовать именно за этим элементом списка, который описывает эту вершину.

Висящая вершина дерева описывается следующей структурой:



Значение 0 первого поля в элементе списка является признаком висящей вершины (ссылка со значением 0 невозможна). В этом случае второе поле элемента содержит код ASCII символа, связанного с этой вершиной. Если элемент списка, соответствующий промежуточной вершине, имел всегда за собой определенного последователя (реализация ссылки вниз), то последователь элемента "висящей вершины" в списке остается непредсказуемым.

Прохождение такого списка будет очень простым: движение направо осуществляется заменой значения указателя прохождения POINTER на ссылку RLINK (POINTER). Движение вниз по списку осуществляется операцией POINTER=POINTER+1, т.е. увеличением значения указателя на единицу. Программа, реализующая алгоритм декодирования на таком дереве, будет относительно несложной. В следующем разделе статьи опишем алгоритмы кодирования и декодирования.

2. Алгоритмы сжатия и восстановления

Принцип кодирования изложен в разделе I. Соответствующий алгоритм использует таблицу кодов Хаффмэна HUFTAB, состоящую в общем случае из 256 элементов (один на каждый возможный код символа), а также таблицу HUFLEN, в которой значением элемента является длина в битах для соответствующего кода в HUFTAB-е. Кодируемые данные (исходный текст) находятся в поле DATA, побайтное продвижение вперед в котором осуществляется с помощью указателя DPOINT. Упакованные данные (результатный код) выводятся в поле BUF, в котором на первый свободный бит указывает значение BPOINT. В алгоритме используется макрокоманда MOVE(ARG1,ARG2,ARG3), которая переносит ARG3 битов с адреса ARG2 на адрес ARG1.

```
PACKER:
      BPOINT=1
; указатель на первый свободный бит в результатном коде
      DPOINT=0
; количество обработанных байтов исходного текста
      NR-SPACE=0
; текущее количество прочитанных пробелов
      PACK: if DPOINT>=LEN then goto END
; LEN - длина исходного текста
      FLD=DATA(DPOINT+1)
        if FLD=' 'then goto SPACE
    NON-SPACE:
            if NR-SPACE=0 then goto SET-CODE
          WRITE-SPACES:
            move (BUF(BPOINT),HUFTAB(32),HUFLEN(32))
            BPOINT=BPOINT+HUFLEN(32)
; записали код пробела
            move (BUF(BPOINT),NR-SPACES,8)
; записывается количество последовательных пробелов
            BPOINT=BPOINT+8
            NR-SPACES=0
        SET-CODE
            if HUFLEN(FLD)=0 then FLD=95
; неизвестные в данном алфавите символы заменяются на код
; 95 - это подчеркивание по ASCII
```

```
                move(BUF(BPOINT,HUFTAB(FLD),HUFLEN(FLD)
                BPOINT = BPOINT+HUFLEN(FLD)
                goto OK
        SPACE:
            NR-SPACES=NR- SPACES+1
    OK:
        DPOINT = DPOINT+1
        goto PACK
END: end PACKER
```

Восстановление текста выполняется на дереве HUFTRE, заранее созданном по методике, описанной в разделе I. Соответствующие висящим вершинам символы находятся в таблице HUFSYM. Движение по дереву осуществляется использованием указателя LINK, поле LEN должно содержать длину искомого текста. Другим способом обнаружения конца кода (код находится в поле CODE) является использование специального символа-завершителя.

Значением BPOINT является количество обнаруженных символов, DPOINT - указатель на изучаемый бит кода.

```
UNPACKER:
      BPOINT=0
; количество восстановленных символов
      DPOINT=1
; указатель на изучаемый бит в коде
      START:
           LINK=1
; начнем всегда с корня дерева
           LOOK:
                if CODE(DPOINT)=0 then LINK=HUFTRE(LINK)
                                  else LINK=LINK+1
; изучаем один бит. Если он 0 , то движемся направо, если
  I - вниз
                DPOINT=DPOINT+1
                if HUFTRE(LINK)=0 then BPOINT=BPOINT+1
                                  else goto LOOK
; нашли ли символ? Если нет, продолжим поиск на дереве.
; Если нашли, запишем найденный символ в буфер
              BUF(BPOINT)=HUFSYM(LINK)
          if BPOINT <= LEN then goto START
end UNPACKER
```

3. Реализация

Вышеописанные методы сжатия текстовых данных использованы при реализации части пакета автоматизированного генерирования систем ГЕНСИ. Последняя, имея базу данных реляционного типа, пригодна для решения широкого круга задач различного характера, в том числе и хранения большого количества многопризнаковых записей.

Однако описание базы данных имеет некоторые недостатки, связанные с фиксированной длиной обрабатываемых записей. Во многих случаях это приводит к нежелаемому хранению текстов, состоящих в основном из пробелов.

Для обеспечения высокоэффективной занятости объема внешней памяти, используемой ЭВМ типа СМ, была разработана специальная подсистема сжатия и восстановления записей базы данных. Она основывается на общей технологии Хаффмэна, изложенной в первом разделе данной статьи. Дополнительно организован сбор пробелов с вводом в сжатый код их количества вместо повторения кодов пробела. Декодирование (восстановление текста) выполняется на специальной древовидной структуре.

Тестированием быстродействия конкретной системы управления базой данных ГЕНСИ автор разработки пришел к выводу, что приемлемо создание подсистемы сжатия и восстановления текстов на языке FORTRAN-77 – потери в скорости обработки данных будут незначительными. Для удобства их реализации операции на битовом уровне выполняются подпрограммами, написанными на языке Ассемблер. Таблицы для кодирования и дерево декодирования хранят в виде программ BLOCK DATA на FORTRAN-77. Необходимая иногда замена обрабатываемого алфавита (см. раздел I) осуществляется перезагрузкой используемых BLOCK DATA. Передача данных между подпрограммами выполняется через общие области памяти типа COMMON, там же хранится описание структуры обрабатываемой записи по элементам данных. Для каждой записи имеется количество элементов, для всех элементов даны длина, сдвиг начала данных элемента с начала записи и тип данных, т.е. информация, необходимая для правильного восстановления текста.

Подпрограмма BL1 является блоком данных для сжатия и включает следующие общие области.

HUFTAB – алфавит обрабатываемых символов в виде массива кодов Хаффмэна. Код находится в десятичном представлении именно в том элементе массива, индекс которого соответствует коду символа в таблице ASCII.

HUFLEN – аналогичный HUFSYM-у массив, где значения элементов представляют длины кодов Хаффмэна в битах. Нулевое значение элемента является признаком отсутствия в алфавите символа, соответствующего в ASCII индексу данного элемента массива.

Длина массива кодирования определяется наибольшим кодом ASCII в используемом алфавите.

Подпрограмма BL2 является также блоком данных и включает информацию для восстановления закодированного текста:

HUFTRE – массив вершин дерева декодирования, упорядоченный по номерам вершин.

HUFSYM – упорядоченный по номерам вершин дерева массив символов, соответствующих листьям дерева декодирования, т.е. символы, выводимые в восстановленный текст по выявленным кодам Хаффмэна.

Для предотвращения неправильного прохождения дерева при возникновении ошибок в данных, вместе с деревом хранится и его длина в виде количества вершин.

Подпрограмма HUPACK предназначена для кодирования исходного текста по таблице кодирования из BL1.

Выполняется цикл по всем элементам записи и на каждом шагу прохождения выясняется тип данных в элементе. Все данные, кроме текстовых, записываются в выходной буфер без изменений. Для каждого байта текстовых данных из таблицы длины кодов HUFLEN узнают, имеется ли для данного символа i код с ненулевой длиной. Неизвестный в данном алфавите символ заменяется утановленным при настраивании подсистемы знаком (подчеркивание, звездочка или др.). Затем в выходной буфер заносится HUFLEN(i) битов из таблицы кодирования HUFTAB. Последняя операция выполняется специальной подпрограммой HUFMOV, написанной на Ассемблере.

Таким образом текст обрабатывается до количества байтов, указанных в описании этого элемента данных. После вывода в буфер всех кодов символов текстового элемента указатель в выходном буфере ставится на границу целого байта. При совершении сжатия всей записи в специальном поле передается длина полученного кода.

Для восстановления сжатых записей предназначена подпрограмма HUFUNP, написанная также на FORTRAN-77. Декодирование производится аналогично процессу кодирования: по описанию данных записи выявляют типы элементов. Числовые элементы любого типа переводятся в выходной текст без изменений, тексты декодируются отдельно. При обнаружении в сжатой записи начала текстового элемента, специальной подпрограммой GETBIT, написанной на Ассемблере, поочередно выдаются в программу HUFUNP значения всех битов кода. По ним осуществляется поиск на дереве декодирования, в котором всем возможным комбинациям последовательности битов 0 и 1 соответствует какой-то символ. Так как коды чаще встречающихся символов короче, они декодируются быстрее. При обнаружении кода пробела, следующие за ним 8 битов интерпретируются как количество последовательных пробелов. Соответствующие коды символов и строки пробелов выводятся в выходной текст.

Объем памяти, занимаемый программами системы сжатия данных, составляет около 7 Кбайт.

Литература

1. Cormack G.V. Data compression on a database system // Communications of the ACM. 1985. Vol. 28, N 12. P. 1336-1340.

2. Huffman D.A. A method for the construction of minimum redundancy codes // Proc. IRE. 1951. Vol. 40, N 9. P. 1098-1101.

3. Knuth D.E. Dynamic Huffman coding // Journal of Algorithms. 1985. N 6. P. 163-180.

4. Knuth D.E. Huffman's algorithm via algebra // Journal of Combinatorial Theory. Series A. 1982. Vol. 32, N 2. P. 216-224.

A. Renzer

Data Compression for Natural Language Dictionaries

Abstract

The article deals with data compression for the effective use of disc storage capacity. The Huffman's coding method is reviewed, original decoding data structures are presented and a specific system for natural language text compression is described.

A. Renzer

Andmetihendus loomuliku keele sõnastike automatiseeritud koostamisel

Kokkuvõte

Artiklis käsitletakse andmetihendust arvuti välismälu paremaks ärakasutamiseks. Esitatakse Huffmani kodeerimismeetod ja kirjeldatakse originaalseid andmestruktuure loomulikus keeles antud teksti efektiivseks säilitamiseks.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

Л.Н. Эльмик, Т.А. Лумберг

РАЗРАБОТКА ИНТЕРФЕЙСА МЕЖДУ ОКНАМИ ДОКУМЕНТА И АТРИБУТАМИ БАЗЫ ДАННЫХ

Введение

Для согласования внешних технологических действий с внутренними действиями в информационной системе используются разные формы для представления информации. Самыми распространенными среди них являются базы данных и окна дисплеев [9]. Эти элементы являются основными во всех, пользующихся наибольшим спросом, пакетах конторских информационных систем: Framework [6], Lotus Symphony [10]. Пакеты конторских информационных систем нового поколения имеют следующие преимущества: образуют инструментальную среду для создания (настройки) конкретной информационной системы, а затем поддерживают ее дальнейшее функционирование. Тем самым они отвечают требованиям Вассермана [3] о том, что пакеты должны охватывать всю деятельность жизненного цикла информационной системы. Вассерман представляет жизненный цикл инфосистем как состоящий из двух основных этапов:

- создание системы,
- развитие функционирующей системы.

С этой точки зрения пользователи пакета, создающие и поддерживающие информационную систему, подразделяются на две совершенно различные группы:

- конечные пользователи функционирующей системы,
- создатели и настройщики конкретной информационной системы.

Рассмотрим одну из основных проблем, возникающую при использовании информационной системы в течение всего жизненного цикла.

Простоты и удобства работы с конкретной информационной системой можно добиться, организовав ее таким образом, чтобы конечный пользователь работал только бы с окнами, ничего не зная о базе данных. Окна служат для обмена данными между конечными пользователями и базой данных. Общепринято, что окно - это поле или область представления данных на экране дисплея, с которым можно работать независимо от остальных областей представления данных [8]. В автоматизированных конторских информационных системах растет значение документов, состоящих из окон. В настоящее время оно вполне сравнимо со значением базы данных.

Необходимо добиться, чтобы настройщик при управлении процессом манипулирования данными оперировал лишь именами окон дисплея и атрибутов (элементов) базы данных.

В данной статье представлен разработанный нами метод решения проблемы связывания элементов базы данных и окон дисплеев конторской информационной системы. Использование этого метода освобождает настройщика систем от забот о согласовании видов (типов, длин) данных, с тем чтобы не возлагать и не перекладывать эту работу на конечного пользователя.

Метод основывается на автоматическом образовании буферов фиксированной структуры и на обмене данных с использованием этих буферов. Обмен данными происходит между базой данных и документами, которые представляют собой множество упорядоченных окон с данными различного типа. Для образования буферов используются описания базы данных и документов.

I. Конфликт между структурами хранения и использования данных

Документы в конторе имеют огромное значение. В конторских информационных системах документ, составленный из окон, является удобным средством макетного обмена, так как легко монтируется и разбирается.

Макет образуется из нескольких упорядоченных необходимым образом окон дисплея, содержащих данные различных типов.

Например, как правило, одним из окон является заголовок документа. Тип данного окна - текст. Это окно, в свою оче-

редь, заключает в себе окна, данные которых уточняют время и место использования документа. Остальная часть документа состоит из соответствующим образом объединенных окон, содержащих разные типы данных.

Основная особенность использования данных в инфосистемах конторского типа состоит в том что те же исходные данные используются многими конечными пользователями. Конечный пользователь использует данные для выполнения своей определенной функции. При применении контекста, как логической связи, можно сказать, что каждая такая функция создает конкретный контекст для использования данных. Контекст определяет количество, состав и вид данных, необходимых для данной функции. Функция может нуждаться в осуществлении нескольких интерактивных операций с данными. Поэтому упорядочение окон зависит от операций внешних действий.

До сих пор мы рассматривали использование данных в конкретном контексте и выявили, что вид данных, как и другие компоненты, образующие контекст, зависят от внешних действий за пределами информационной системы. Теперь перейдем к вопросу их хранения.

Чтобы хранение данных не зависело от их использования, они освобождаются от контекста, т.е. представляются в виде концептуальной модели базы данных [3]. На физическом уровне база данных состоит из файлов, которые, в свою очередь, состоят из записей, а те - из элементов данных. Физическая и логическая структура базы данных согласована следующим образом: каждому концепту соответствует файл, каждому конкретному экземпляру этого концепта соответствует запись и каждому атрибуту конкретного экземпляра соответствует элемент записи. Наиболее удобным является такое согласование физической и логической структуры базы данных, в котором запись определяется ключом, оставляя элементы записи независимыми между собой. Таким образом, согласование физической и логической структуры при хранении данных обеспечивает независимость от контекста.

Рассмотрим заполнение окон документа данными. Окна документа заполняются данными по мере развития технологического процесса, см. [4]. Каждый документ отражает некоторую определенную технологию производства информации, т.е. его

данные направлены на выполнение конкретной функции. Тем самым данные отражают соответствующий контекст, который обеспечит конечного пользователя информацией в удобном окончательном виде.

Таким образом, в документе вид данных зависит от контекста, а в базе данных - свободен от него. В этом и заключается конфликт между использованием и хранением данных, так как тип, состав и структура данных различны при использовании и хранении.

Для преодоления конфликта необходимо использовать преобразование данных как при движении их от базы данных к документу, так и в противоположном направлении. Другими словами, необходимо соединить структуру каждого элемента зоны хранения со структурой соответствующего элемента (или элементов) зоны использования данных.

Во всех конторских системах нового поколения такое преобразование возможно, но программируется вручную и входит либо в обязанность настройщика (который обычно по собственному усмотрению выбирает количество буферов и определяет их структуру), либо в обязанность конечного пользователя.

Для полного устранения конфликта необходимо освободиться от программирования преобразований данных. При этом задание буферов должно быть определено фиксированным способом, т.е. надо создать интерфейс между зоной хранения и зоной использования данных.

2. Требования, предъявляемые к окружающей среде интерфейса

Для того, чтобы использовать автоматически генерирующий интерфейс между базой данных и документом, как между зонами хранения и использования данных, необходимо знать структуры базы данных и документа. В каждом конкретном случае концептуальная структура базы данных отвечает конкретной объект-системе [I], а структура документа определяется последовательностью соединения окон разных типов. Существуют всевозможные способы описания этих структур [2], но результат - описание базы данных и описание документа - является одной из главных предпосылок для создания автоматического интерфейса.

Поскольку концептуальная база данных физически реализуется как файлы и их элементы, то в описании необходимо представить имена файлов и элементов. В описании документа необходимо представить имена документа и образующих его окон.

Для определения обеих структур для каждого элемента необходимо представить:

- название (имя),
- тип,
- длину.

Определение этих трех компонентов для каждого элемента обеспечивает определение структур зоны хранения и зоны использования данных.

Зная структуры этих зон, для преодоления конфликта необходимо найти принципы преобразования структур каждого элемента. Рассмотрим передачу одного элемента данных из одной зоны в другую. При вводе элемент движется из зоны использования в зону хранения. Таким образом, источник передачи - документ, а принимает передаваемый элемент - база данных. В документе структура элемента имеет вполне определенное имя, тип и длину, которые могут отличаться от соответствующих значений этих компонент элемента, хранящихся в базе данных. В соответствии с направлением движения структура элемента должна быть преобразована к его виду в базе данных.

При выводе элемент движется из зоны хранения в зону использования и следовательно, выполняется преобразование структуры элемента к его виду в документе. Таким образом, принцип преобразования структуры элемента для преодоления конфликта между его описаниями в зоне хранения и в зоне использования состоит в приведении его структуры к виду принимающей зоны.

3. Разработка интерфейса

Для автоматического преодоления конфликта между представлением данных в зоне хранения и в зоне их использования нами разработан интерфейс между документом и базой данных.

Рассмотрим состав, создание, функционирование и управление такого интерфейса.

3.I. Состав интерфейса

Интерфейс состоит из трех буферов, их описаний и программы преобразования данных. А именно, для этого используются:

- макетный буфер,
- буфер базы данных или так называемая суперзапись [5],
- буфер обработки.

Структура макетного буфера соответствует структуре документа. В буфере последовательно размещены все окна, образующие документ. Аналогично суперзаписи буфер такого типа можно условно назвать суперокном.

Суперзапись образуется из всех файлов базы банных, связанных с данным конкретным документом. В состав суперзаписи входит по одной записи из каждого файла, связанного с данным документом. Буфер обработки предназначен для промежуточной обработки данных, его структура зависит от технологии обработки данных.

3.2. Создание компонентов интерфейса

При активизации документа все части интерфейса создаются автоматически на основе описания базы данных, описания документа и описания технологии обработки данных.

При активизации любой операции, требующей обмена данными между документом и базой данных, заполнение и обновление данных в буферах происходит в соответствии с логикой операции.

3.3. Функционирование интерфейса

Как уже было показано, при осуществлении передачи данных между зонами хранения и использования данных может возникнуть конфликт между их описаниями в каждой зоне. Для преодоления этого конфликта создан интерфейс между документом и базой данных для преобразования структуры каждого элемента данных. При этом структура элемента приводится к структуре в принимающей зоне. Делается это очень простым способом:

Если $t \neq t'$, то

n преобразуется в n', где

t - начальная структура передаваемого элемента;

t' - структура элемента в принимающей зоне;

n - начальное значение элемента;

n' - значение элемента в принимающей зоне.

Как мы уже видели, описание документа состоит из описаний всех содержащихся в нем окон, а описание файлов базы данных состоит из описаний всех атрибутов. Мы можем рассматривать эти описания как способ косвенной адресации, если в описания добавить местонахождения каждого элемента в обоих структурах, т.е. местонахождения атрибутов в базе данных и местонахождения окон в документе.

И тогда имена структурного элемента в его описаниях определяют его адрес как в документе, так и в базе данных. Использование имен элемента в соответствующей зоне в качестве адреса дает возможность однозначно определять адрес каждого элемента как в базе данных, так и в документе.

На основе этого мы можем в языке управления интерфейсом использовать имена переменных базы данных и окон документа.

При этом имена переменных состоят из трех компонентов A, B, C,

где A - определяет описание зоны данного;

B - определяет имя либо файла, либо документа в зависимости от описания зоны;

C - определяет имя элемента, который является либо окном, либо атрибутом в зависимости от A и B.

4. Преимущества интегрированности потоков данных

В системе "документ - база данных" входные и выходные потоки данных можно рассматривать интегрированно. Процесс интегрирования входных и выходных потоков данных полезен как с точки зрения использования документа, так и с точки зрения использования базы данных. При этом возможны следующие четыре варианта:

- ввод через окно,
- вывод данных через окно документа,

- ввод в базу,
- вывод из базы данных.

Для каждого из них разработанный интерфейс обеспечивает согласование структур данных с остальными тремя. Перечислим теперь основные преимущества бесконфликтного интегрирования потоков для этих четырех случаев.

4.I. Ввод данных через окно документа

При этом вводе самым серьезным вопросом является правильность и безошибочность вводимых данных. Имея вышеописанный интерфейс, мы можем описывать контрольные условия, используя файлы (каталоги) базы данных. Если мы имеем целостную и полную концептуальную базу данных (см. [7, I2]). то у нас имеются все каталоги о ресурсах, в которые вводятся данные. Мы можем образовать над этими каталогами выходные потоки из базы. При сравнении вводимых через окно данных с данными, уже содержащимися в каталогах базы данных, мы можем, практически, в момент ввода выявить ошибочные данные, ошибочными будут считаться данные, которые отсутствуют в каталогах.

4.2. Вывод данных через окно документа

Теперь, когда у нас имеется возможность бесконфликтного образования связей между элементами документа и элементами базы данных, мы получаем два основных преимущества при выводе данных через окно документа:

I) можно интегрированно представить условия для вывода данных через окно,

2) можно образовывать сам элемент данных, выводящийся через окно из обоих потоков, т.е. соединить в один элемент некоторые элементы данных и из базы, и из ранее заполненных окон документа, используя при необходимости арифметические и текстовые операции.

4.3. Ввод данных в базу данных

Ввод данных в базу - это, другими словами, обновление базы данных. Если при обновлении базы данных используются интегрированные потоки входных и выходных данных, то появляются следующие преимущества:

I) можно использовать интерфейс для образования основного ключа записи, что, в свою очередь, определяет запись для прямого доступа; таким способом можно организовать прямую зависимость между вводимыми данными и определением обновленной записи;

2) можно интегрированно представлять условие для определения обновленных записей.

4.4. Вывод данных из базы

Вывод из базы - это запрос прежних информационных систем, т.е. систем базы данных. Здесь самым используемым преимуществом, которое мы уже неоднократно упоминали, является интегрирование входных и выходных данных для представления условия вывода, используя при этом либо арифметические связи между всеми элементами базы и документа, либо - логические связи между ними, либо текстовые, либо и те и другие, и третьи.

5. Реализация

Интерфейс для преодоления конфликта между контекстом хранения и использования данных реализован в информационной системе "SERVICE", которая внедряется в объединении АВТОВАЗ.

Заключение

Таким образом, в данной статье:

I) представлен принцип и метод преодоления конфликта между контекстом хранения и использования данных,

2) выработан интерфейс, при помощи которого автоматически сопоставляются структуры элементов данных в зоне хранения и в зоне их использования,

3) выявлены преимущества интегрирования данных входных-выходных потоков, которые существенно облегчают работу настройщика инфосистем,

4) разработано трехуровневое задание переменных в языке настройки.

Литература

1. Ваппер Т.Э., Лааст-Лаас Ю.Г., Урвак А., Эльмик Л.Н. Практические аспекты проектирования информационных систем // Тр. Таллинск. политехн. ин-та. 1987. № 645. С. 118-125.

2. Лумберг Т.А., Распель П.У. О разработке технологических структур для систем реального времени // Тр. Таллинск. политехн. ин-та. 1987. № 645. С. 100-113.

3. Мартин Дж. Организация баз данных в вычислительных системах. М.: Мир, 1980. С. 662.

4. Микли Т.И. Проблемы создания информационных систем реального времени // Тр. Таллинск. политехн. ин-та. 1987. № 645. С. 81-89.

5. Эльмик Л.Н. Проектирование систем обработки анкет. Дипломный проект. Таллин, 1981. С. 64.

6. Advanced topics framework II. Ashton-Tate. 1985.

7. Allen T.J. Organizational structure, information technology and R&D productivity // IEEE Transactions on Engineering Management. Nov. 1986. Vol. Em-33, N 4.

8. Arvutustehnika ja andmetöötluse põhisõnavara // Infoseeria XVIII. Arvutustehnika ja andmetöötlus. 1986. Lk.30.

9. Gaines B.R., Shaw M.L. Foundations of dialog engineering: the development of human-computer interaction. Part II // Int. J. Man-Machine Studies. 1986. 24. P. 101-123.

10. Lotus Sumphony. Satra/PSI-Data. Tallinn Symposium 23-th-30th October 1987.

11. Wasserman A.I. Modern software development methodologies and their environments // Computer Physics Communications. 1985. 38. P. 119-134.

12. Staff writer. E-project-plan for industrial production of computer programs // TECHNO JAPAN. May 1986. Vol. 19, N 5.

L. Elmik, T. Lumberg

Mapping the Data Base Attributes on Document Windows

Abstract

In this paper the problems, arising with the need to represent data differently from the way they are stored, are considered.

To overcome this situation an approach, where the data are flexibly restructured to meet the operational context, is presented. It makes it possible to view I/O flows in their integrity.

L. Elmik, T. Lumberg

Dokumendi akende ning andmebaasi atribuutide vahelise liidese väljatöötamine

Kokkuvõte

Artiklis käsitletakse vahendeid, mis tagavad sisend- ja väljundandmevoogude integreerimise infotöötlussüsteemis. Need vahendid kergendavad oluliselt infotöötlussüsteemi haalestaja tööd.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

М.Х. Роост

СРЕДА ДАННЫХ ПРИ РАЗРАБОТКЕ ИНТЕРФЕЙСА В СИСТЕМАХ СВОЕВРЕМЕННОЙ РЕАКЦИИ

I. Введение

В статье [2] мы рассмотрели систему своевременной реакции с точки зрения интерфейса между системой и пользователем, ввели понятие рабочей среды интерфейса пользователя (ИП) и описали ее основные функции. Настоящая статья предлагает подход к рабочей среде, в котором она рассматривается как среда данных, и затрагиваются проблемы ее структуры и организации.

В настоящее время в связи с развитием персональной вычислительной техники возникнут предпосылки промышленного производства матобеспечения для систем своевременной реакции [1]. Но пока отсутствует полностью подходящее для этой цели инструментальное матобеспечение. По нашему мнению, причины этого связаны с недостатками в моделировании систем своевременной реакции. Поэтому в широком масштабе наша цель – это разработка подхода к системам своевременной реакции. На его основе было бы возможно создать инструментальную среду для дальнейшего развития систем своевременной реакции на базе персональной вычислительной техники. Данной теме посвящена статья [2], где система своевременной реакции и ее матобеспечение рассмотрены, со стороны ИП, с анализом основных свойств и функций последнего. Естественным продолжением темы является более глубокое рассмотрение окружающей среды ИП.

Поводом появления этой статьи являются и проблемы единой трактовки данных, которые возникают при создании ИП (организация данных разных типов по единой модели). Проб-

лемы связаны с отсутствием таких моделей. Мы столкнулись с этим при разработке инструментального матобеспечения для создания рабочих мест. В литературе на эту тему мы не нашли достаточно гибких и эффективных решений, подходящих для создания систем своевременной реакции на базе микроЭВМ. Доступные нам пакеты для микроЭВМ тоже не предлагают решений нужного качества. Поэтому в данной статье окружающая среда ИП рассматривается в основном с аспекта ее структур данных и их отношений между собой. Среда рассматривается как среда данных, которая предполагает единый, целостный, а в то же время структурный и разделенный подход. Затрагиваются проблемы структуры и организации среды, представлены ее основные свойства и функции.

2. Интерфейс пользователя и его среда данных

Первичное абстрактное деление системы своевременной реакции (в качестве инфосистемы) – это деление на часть, которая находится вне ЭВМ, и на часть, неотделимую от ЭВМ (система обработки данных). Последнюю мы делили бы так: интерфейс пользователя и среда данных. Такое деление вытекает из подхода, ориентированного на конечного пользователя [2]. Это позволит вводить параллели с психикой человека, полезные для первичного представления будущей модели. Мы имеем в виду деление психики на сознание и подсознание. Представим сущность работы системы как представление актуальных компонентов среды данных пользователю через ИП. Здесь интерпретируется среда данных в выводе микроЭВМ по требованиям пользователя, и она воздействует на пользователя анологично тому, как подсознание интерпретируется в сознании и как оно воздействует на последнего.

Для определения понятия среды данных ИП в системах своевременной реакции уточняем понятие окружающей среды ИП.

Окружающая среда – это все, что воздействует на функционирование ИП. Мы рассматриваем только ту часть окружающей среды, которая вовлечена в систему обработки информации, и различаем два ее аспекта: функциональный аспект и аспект данных. При первом аспекте мы говорим об операционной среде, при втором – о среде данных.

Под средой данных ИП мы подразумеваем все данные, которые непосредственно или косвенно воздействуют на его функционирование и которые вовлечены в систему своевременной реакции.

При этом среда данных имеет более глубокое значение. Мы можем и не подчеркивать посредническую роль ИП, так как нас интересуют также закономерности, которые свойственны данным и проявляются в среде данных. Это поможет иметь о среде целостную картину, которая в дальнейшем послужит основанием для разработки модели данных при формализации трактовки данных. Ниже мы будем говорить о среде данных в системах своевременной реакции или просто о среде данных.

Среда данных охватывает данные, ориентированные на разные функциональные подсистемы (звенья) матобеспечения, но в конечном счете все эти данные ориентированы на конечного пользователя. Например, в системе своевременной реакции данные для общего пользования непосредственно связаны с системой управления базами данных (СУБД), а данные конкретного документа – системой обработки документов. Следовательно, мы не можем обойтись без обсуждения проблем интерпретирования, воздействия и поддержания среды данных. Поэтому, мы связываем структуры данных в среде действиями для манипулирования этими структурами. Данные окружающей среды ИП связываются теорией абстрактных типов данных [7].

Данные и действия мы рассматриваем вместе как взаимодействие, когда действия воздействуют на данные и наоборот. Если в дальнейшем нет прямой необходимости различать функциональный аспект окружающей среды ИП и аспект данных, то мы говорим о воздействиях и о структурах воздействий. Разные структуры воздействий влияют друг на друга, а также на действия пользователя, через ИП. И наоборот, внешняя технология инфообработки и действия пользователя оказывают влияние через ИП на структуры воздействий.

Данные могут воздействовать на действия только при переносе между разными структурами данных (потоки данных). Действия влияют на данные путем таких переносов или потоков данных. Далее, говоря о среде данных, мы имеем в виду как структуры данных, так и потоки данных между ними. Это позволяет нам рассматривать среду данных как динамичную и активную среду, которая позволяет с достаточной точ-

ностью [3] отражать внешнюю технологию и реагировать на события, чтобы активно воздействовать на действия пользователя и на ход внешней технологии. Следовательно, это позволит в дальнейшем и управлять ими.

3. Структура среды данных

Чтобы иметь представление о структуре среды данных, коротко ознакомимся с принципами архитектуры и развития матобеспечения, которые дают основу нашей модели.

3.I. Слои в архитектуре матобеспечения

С аспекта разработки системы своевременной реакции рассмотрим систему матобеспечения как многослойную. Предположим, что каждый слой (кроме нижнего) базируется на нижнем, а сам является базой для реализации следующего слоя (рис.I). Внешний слой(ИП) может также являться базой для дальнейшего автоматизирования действий пользователя посредством матобеспечения.

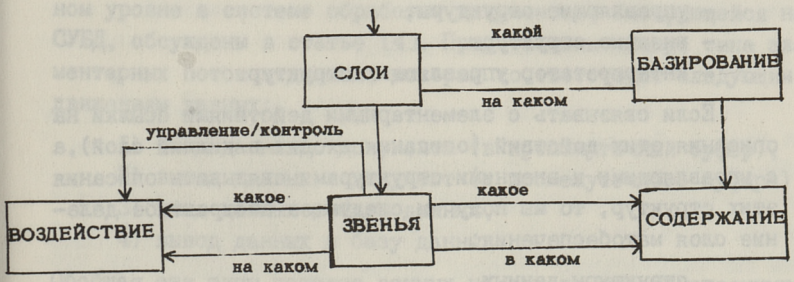


Рис. 1. Логическая структура среды матобеспечения в системе своевременной реакции (представлены посредством инфологической модели).

Слой матобеспечения состоит из функциональных звеньев (подсистем) определенного вида. Например, подсистема для обработки таблиц и система для обработки текстов как конторские пакеты.

Каждое звено реализует определенную абстракцию данных или модель данных для верхнего слоя. С этой моделью связывается множество возможных элементарных действий для манипулирования данными, включая и действия для настройки звена (подсистемы). Конкретные структуры (данные + действия) таких моделей данных в определенном слое мы называем внешними структурами этого слоя. Через эти структуры реализуются взгляды верхних слоев на этот слой, а также на базовые слои. Если мы не рассматриваем действия настройки звеньев, то верхним слоям не следует ничего "знать" о внутренних структурах, которые связаны с внешними структурами данного слоя, а также о их деталях функционирования. Функциональные звенья слоя и их внешние структуры реализуются путем интеграции внешних взглядов на нижний слой и элементарных действий, связанных с этими взглядами через управляющие структуры (последним посвящена статья [5]).

На основании вышесказанного мы представим основные элементы слоя матобеспечения:

- элементарные действия,
- управляющие структуры,
- внешние структуры,
- интерпретатор управляющих структур.

Если связывать с элементарными действиями ссылки на описания этих действий (описания входят в нижний слой), а с управляющими и внешними структурами связывать описания этих структур, то мы получим следующее абстрактное деление слоя матобеспечения:

- структуры данных,
- интерпретатор.

3.2. Слои данных

Рассмотрим далее слои матобеспечения с аспекта данных. Последний охватывает структуры данных и их динамику - потоки данных между этими структурами.

Структуры данных слоя делятся логически на функциональная части, которые назовем в дальнейшем звеньями данных.

Потоки данных внутри звена организуются между элементами его структур данных. Структуры данных звена охватывают внешние структуры звена, промежуточные структуры для обработки данных и внешние структуры нижнего слоя, которые связаны со звеном (дальше назовем их базовыми структурами). Потоки данных между звеньями организуются в верхнем слое. Физически они реализуются в системе с открытой архитектурой (см. следующие разделы) через нижние слои (см. рис. 2).

Внутри звена мы различаем потоки данных разного типа. Тип потока данных определяется типом вводной структуры и типом выводной структуры данных. Под типом структуры имеется в виду подмножество структур данных, например: документ, файл БД и т.д. Подмножество структур данных представляет в звене некоторый вид структуры, либо внешние структуры, либо промежуточные или базовые структуры (т.е. внешние структуры нижнего слоя).

Типы потоков данных обобщенно рассмотрены в статье [2], где из функций этих типов образованы и функции ИП (как внешнего слоя матобеспечения). Потоки данных на элементарном уровне в системе обработки документов, базирующейся на СУБД, обсуждены в статье [4]. Представлены четыре типа элементарных потоков данных, которые соответствуют следующим движениям данных:

1) ввод данных с документа (в промежуточный буфер),
2) вывод данных в документ (с промежуточного буфера),
3) ввод данных с базы данных,
4) вывод данных в базу данных.

Обобщая эти типы потоков данных на основе вышеприведенных видов структур, мы получаем следующие основные виды элементарных потоков данных:

1) внешняя структура -> промежуточная структура,
2) промежуточная структура -> внешняя структура,
3) базовая структура -> промежуточная структура,
4) промежуточная структура -> базовая структура.

Проиллюстрируем виды этих потоков данных на рис. 2.

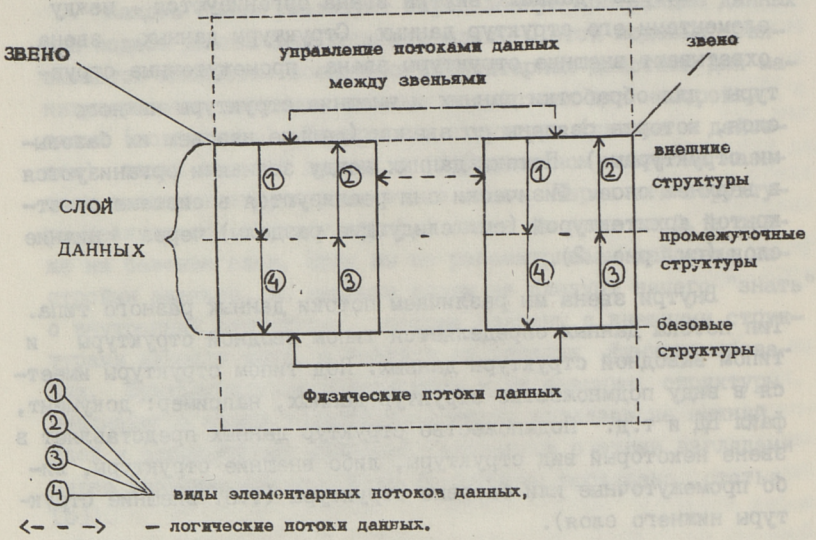


Рис. 2. Организация потоков данных в слое данных системы своевременной реакции.

3.3. Управление данными в слое

Под управлением данными мы понимаем управление потоками данных. Средствами управления потоками в слое являются элементарные действия и управляющие структуры как средства согласования, интегрирования и структуризации этих элементарных действий. Каждое такое действие реализует определенный тип элементарных потоков данных. Для связывания разных потоков данных в слое создаются соответствующие промежуточные структуры данных (см. рис. 2).

Посредством этих промежуточных структур можно соединять данные из разных потоков данных в один поток и управлять элементарными действиями, а также потоками данных на основе управляющих структур. Разные потоки данных можно связывать между собой, используя при этом арифметические, текстовые, логические операции и операции сравнения [4]. Это позволяет на уровне ИП соединять потоки данных из раз-

ных слоев в потоки информации, необходимые пользователю. Вышеупомянутые операции уточняют контекст данных на каждом уровне. Можно осуществить и обратный ход действий: потоки информации, поступающие от пользователя, расслаиваются в потоки данных, которые содержат и единицы хранения, необходимые в данном слое.

Таким образом, основной целью управления данными в слое является согласование данных внешних и базовых структур слоя. Такое согласование удается, если:

- базовые структуры слоя (т.е. соответствующая модель данных) обеспечивают все нужные компоненты для реализации внешних структур, а следовательно, и для реализации управляющих и промежуточных структур. Например, если СУБД, реализующий базовые структуры, поддерживает и трактовку форм и сценарий документов [8],

- все эти компоненты можно настраивать и управлять ими через внешние структуры,

- нужные промежуточные структуры создаются на основе управляющих структур автоматически,

- функции управления данными слоя удается точно распределить между разными звеньями слоя.

3.4. Свойства слоя

Цель расслаивания матобеспечения - это отделение его компонентов (звеньев) и их аспектов, которые логически независимы от остальных компонентов и их аспектов [6, 7]. Это имеет силу и в среде данных, которая представляет аспект данных соответствующей системы своевременной реакции.

Под логической независимостью слоя данных мы понимаем его логическую целостность в структурном и функциональном смысле. В слое имеются самостоятельные механизмы управления и обработки данных, которые соответствуют функциям слоя и являются независимыми от реализации внутренних механизмов в других слоях. Логическая независимость слоя достигается его ориентацией на реализацию внешних структур, которые являются посредником доступа к слою. При этом пользователь имеет возможность формировать внешние структуры независимо от базовых структур, и наоборот. Внешние

структуры ориентированы на конкретного пользователя, имеющего свои определенные требования. В роли пользователя может выступать и звено верхнего слоя. Внутренние детали от него "спрятаны".

В связи с логической независимостью слоев возникают структурные конфликты между внешними и базовыми структурами слоя, а вслед за этим и между внешними структурами разных слоев. Такие конфликты можно преодолеть с помощью управления данными по тем принципам, которые представлены в предыдущем разделе. Подробнее эта проблема рассматривается в статье [4].

Логически независимые слои являются результатом структуризации системы матобеспечения и ее среды данных. Это позволяет добиться и прозрачности структуры матобеспечения, сводя до минимума необходимый объем матобеспечения [7]. Мы можем говорить о нормализации структуры матобеспечения и среды данных. Это необходимо при больших сложных системах.

Кроме логической независимости, слой матобеспечения или данных должен иметь и свойство открытости. Это дает возможность комплектовать слой из звеньев разными способами, добавлять, удалять и модифицировать звенья без существенного изменения остальных частей слоя. Логическая независимость слоя является предпосылкой его открытости.

3.5. Характеристика открытой среды данных

Послойная архитектура матобеспечения, логическая независимость и открытость слоев позволяет говорить об открытой архитектуре в системе матобеспечения [6] и об открытой среде данных. Только система с такими свойствами может удовлетворять требованиям гибкости и эффективности, которые предъявляются матобеспечению системы своевременной реакции (см. [2]).

Учитывая предыдущее, среду данных системы своевременной реакции целесообразно моделировать в двух измерениях. Одно из них (см. вертикальную ось на рис. 3) разделяет среду данных на слои, т.е. на уровни абстракции [7], а другое разделяет слой на характерные ему звенья (см. горизонтальную ось на рис. 3). В частности, целесообразнее отде-

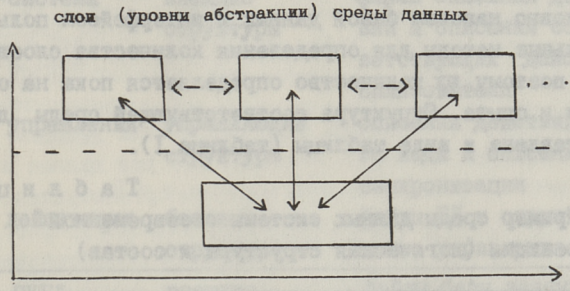


Рис. 3. Соединение звеньев в открытой среде данных.

лять в среде данных только два слоя - какой-либо основной слой вместе с нижним, соседним слоем. Основной слой базируется на нижнем слое, который охватывает и остальные нижние слои. Нижние слои рассматриваются как базовая среда, которая обслуживает звенья основного слоя.

Разные звенья и слои среды данных связываются между собой через потоки данных. Логические потоки данных между звеньями одного слоя реализуются физически через звено соседнего слоя или через отдельные звенья разных слоев. Промежуточными структурами в контексте такого соединения звеньев являются единые элементы структуры звеньев соседних слоев. Таким образом, уменьшается взаимозависимость звеньев внутри слоя, так как количество типов связей (потоков данных) станет на разряд меньше.

Пример

Проиллюстрируем вышесказанное примером. Мы рассматриваем систему своевременной реакции, матобеспечение которой состоит из пяти подсистем:

- система управления базами данных,
- система управления диалогами,
- система макетного обмена,
- система коммуникации,

– система управления действиями.

Разделим матобеспечение на два основных слоя, которые мы условно назовем базой данных и интерфейсом пользователя. Формальные методы для определения количества слоев отсутствуют, поэтому их количество определяется пока на основе интуиции и опыта. Структура соответствующей среды данных представлена в виде таблицы (таблица I).

Таблица I

Пример среды данных системы своевременной реакции (логическая структура и состав)

Слои	Звенья	Общие названия структур данных	Конкретные названия структур данных
I	2	3	4
Интерфейс пользователя	Система управления диалогами	Внешние структуры	Формы диалогов и описания допустимых пользовательских реакций
		управляющие структуры	сценарии диалогов
		базовые структуры	файлы БД и структуры языка управления БД
	система макетного обмена	внешние структуры	формы макетов и описания допустимых пользовательских действий над ними
		управляющие структуры	сценарии макетного обмена
		базовые структуры	файлы БД и структуры языка управления БД
	система коммуникации	внешние структуры	формы сообщений и описания допустимых реакций пользователя
		управляющие структуры	описания маршрутов движения сообщений и описания протоколов
		базовые структуры	файлы БД и структуры языка управления БД

I	2	3	4
	система	внешние структуры	формы описания действий и описания соответствующих действий пользователя
	управления	управляющие структуры	описания действий разного вида и описания их синхронизации
	действиями	базовые структуры	файлы БД и структуры языка управления БД
База данных	СУБД	внешние структуры	файлы базы данных и структуры языка управления базы данных

4. Создание среды данных

Среда данных создается на основе системы управления базами данных (СУБД). Идеальной мы считаем такую ситуацию, где все данные системы своевременной реакции были бы организованы средствами существующей СУБД. Мы получаем такую возможность тогда, когда СУБД реализует модель данных, которая позволяет единым способом трактовать все данные разного типа [8] в системе своевременной реакции. К сожалению, нам более известные СУБД такой возможности не имеют (dBase II, III и др.).

Чтобы достигнуть таких свойств, как гибкость, настраиваемость, эффективность, о которых мы уже упоминали в статье, к существующей СУБД необходимо органично добавить средство, способное трактовать данные тех типов, которые не рассмотрены в СУБД.

При организации открытой среды данных применяемая СУБД должна быть открытой, чтобы имелась возможность добавлять и модифицировать структуры данных разного типа (а также базы данных при использовании разделенной СУБД [10]) и связи между ними без существенных изменений.

Для достижения необходимой синхронизации в системе своевременной реакции, СУБД должна позволять описание разных состояний базы данных, чтобы потом на их основе узнавать эти состояния и реагировать на них сигнализацией соответствующему функциональному звену (через систему управления действиями). Это и есть синхронизация согласно

данным. Для этого применяются специальные механизмы, так называемые "сторожи" и триггеры [9]. Они усиливают непосредственное воздействие данных на поведение системы и повышают активность самих данных. В практике СУБД вышеописанных возможностей пока нет.

Рассматривая уровень СУБД как нижний уровень (слой) матобеспечения, мы можем сказать, что все данные в системе своевременной реакции внутри системы организуются, в конечном счете, на одном, т.е. на физическом уровне. Такой подход позволяет упростить архитектуру матобеспечения, поскольку определяется понятие состояния системы на основе понятия состояния базы данных.

5. Интерпретирование среды данных и воздействие на нее

Пользователь системы интерпретирует среду данных и воздействует на нее через ИП, который является внешним слоем матобеспечения. Мы предполагаем в системе своевременной реакции двух пользователей: администратора системы и конечного пользователя.

ИП должен обеспечивать администратору системы прямой доступ ко всем внутренним слоям матобеспечения для составления или модификации необходимых звеньев использования путем представления или модифицирования структур данных настройки системы. Конечный пользователь системы не должен иметь прямой доступ к внутренним слоям матобеспечения (т.е. и среды данных). Доступ пользователя к ним обеспечивается через звенья конечного использования, которые формирует администратор системы. Вершина таких звеньев находится в верхнем слое. При этом мы не исключаем и некоторой возможности настройки системы конечным пользователем.

Выводы

Подведем следующие итоги нашей работы:

- дана формулировка понятия среды данных системы своевременной реакции и ее логическая структура,

- проанализированы некоторые проблемы организации и реализации этой среды,

- понятие среды данных служит основой для единой трактовки всех данных с учетом динамики данных в системах своевременной реакции, тем самым дается основа для моделирования и проектирования этих систем, а также основа для выработки инструментальной среды при создании систем своевременной реакции.

Литература

I. Громов Г.Р. Национальные информационные ресурсы: проблемы промышленной эксплуатации. М.: Наука, 1984.

2. Лумберг Т.А., Роост М.Х., Эльмик Л.Н. О подходе к интерфейсу между пользователем и системой своевременной реакции // Тр. Таллинск. политехн. ин-та. См. наст. сб., с. 131.

3. Микли Т.И. Проблемы создания информационных систем реального времени // Тр. Таллинск. политехн. ин-та. 1987. № 645. С. 81-89.

4. Эльмик Л.Н., Лумберг Т.А. Разработка интерфейса между окнами документа и атрибутами базы данных // Тр. Таллинск. политехн. ин-та. См. наст. сб., с. 105.

5. Лумберг Т.А., Распел П.У. О разработке технологических структур для систем реального времени // Тр. Таллинск. политехн. ин-та. 1987. № 645. С. 100-113.

6. Ларионов А.М., Майоров С.А., Новиков Г.И. Вычислительные комплексы, сети и системы. Л.: Энергоатомиздат, 1987.

7. Richmond A. Software design by object-oriented functional layering // Computer Physics Communications. 1986. 41. P. 377-384.

8. Mark L., Roussopoulos N. The new database architecture framework - a progress report// Information systems: Theoretical and Formal Aspects, IFIP 1985.

9. Chang S.-K., Chan W.-L. Transformation and verification of office procedures // Transactions on Software Engineering. 1985. Vol. SE-11, N 8.

10. Deen S.M., Amin R.R., Ofori-Dwumbuo G.O., Taylor M.C. The architecture of a genera-

lised distributed database system - PRECI* // The Computer Journal. 1985. Vol. 28, N 3.

M. Roost

Data Environment of User's Interface for Just-In-Time Systems

Abstract

The article deals with data environment of user's interface for the just-in-time system and describes the structure, qualities and functions of the latter.

The approach to the data environment is a base for the creation of data model when constructing user's interface in just-in-time systems.

M. Roost

Ajastussüsteemi kasutajaliidese andmekeskkond

Kokkuvõte

Artiklis käsitletakse ajastussüsteemi kasutajaliidese andmekeskkonnaga seotud probleeme. Kirjeldatakse andmekeskkonna struktuuri, omadusi ja funktsioone. Iseloomustatakse lähenemist andmekeskkonnale, mis on aluseks andmemudeli väljatöötamisele nõutavate omadustega kasutajaliidese loomiseks ajastussüsteemides.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

Т.А Лумберг, М.Х. Роост,
Л.Н. Эльмик

О ПОДХОДЕ К ИНТЕРФЕЙСУ МЕЖДУ ПОЛЬЗОВАТЕЛЕМ И СИСТЕМОЙ СВОЕВРЕМЕННОЙ РЕАКЦИИ

I. Введение

В настоящее время у микроЭВМ есть все преимущества стать универсальным средством. Это позволяют их свойства (небольшие размеры, низкая стоимость, мощность, эргономичность), но применение микроЭВМ в качестве универсального средства возможно лишь с помощью качественного матобеспечения.

Так как человек и ЭВМ "говорят" на разных языках, то для осуществления возможности их общения необходимо создать "переводчик" как часть матобеспечения, назовем ее интерфейсом пользователя. С этой позиции рассмотрим средства коммуникации между пользователем и остальной системой матобеспечения.

В начале развития матобеспечения, интерфейса в современном понятии не существовало, функции "переводчика" выполнял при этом специально подготовленный человек. Позднее, когда произошло изменение функции ЭВМ, значительное расширение круга общения с ЭВМ, эту роль стало осуществлять матобеспечение. Можно сказать, что интерфейс стал самостоятельной функциональной частью системы матобеспечения.

По мере расширения универсальности ЭВМ значительно увеличилась роль интерфейса пользователя. В настоящее время разработка его относится к одной из трех основных проблем, которой необходимо заниматься создателям интегрированной системы матобеспечения помимо осуществления выполнения и управления [I].

В статье дается наша оценка об интерфейсе пользователя как части системы матобеспечения и его внешнего слоя: анализируются свойства интерфейса и рассматриваются основные функции, которые базируются на этих свойствах.

2. Интерфейс пользователя

2.I. Общий обзор

В качестве системы матобеспечения мы рассматриваем интерактивные, со своевременной реакцией, инфосистемы для микроЭВМ, делая упор на их применение в сфере конторской деятельности. Инфосистема со своевременной реакцией является системой обработки информации с рабочими местами, которая следует за внешней технологией как во времени, так и в пространстве.

Система со своевременной реакцией образует автоматизированное звено и является промежуточной ступенью к полной автоматизации деятельности предприятия.

Соответствующие системы обладают многими хорошими техническими свойствами, но что касается интерфейса пользователя, здесь дело обстоит не столь идеально. Имеются системы, где общение не приспособлено к пользователю - интерфейс фактически отсутствует. Есть системы, которые применяются на одном и том же рабочем месте, но общение с ними неодинаково. Подобные недостатки не позволяют достичь эффективной универсальности использования микроЭВМ. В целом, матобеспечение развивается в направлении создания сред, а это значит и в направлении интегрирования различных функциональных целостностей [2]. С другой стороны, имеется необходимость осуществить настройку интегрированного матобеспечения (среды) на конкретную технологию инфообработки. Мы рассматриваем такую настройку как "программирование без программистов" [3] (т.е. деятельность, позволяющую получить программный продукт, не прибегая к составлению программы как определенной совокупности связанных между собой операторов). С точки зрения пользователя интерфейс должен дать в результате настройки возможность представить среду в едином и исчерпывающем виде.

Такая позиция интерфейса пользователя несомненно связана с возросшими требованиями по отношению к нему и, соответственно, формирует подходы для создания интерфейса пользователя. Чтобы создать приемлемые части, которые соответственно формировали бы интерфейс пользователя, создателю системы нужно глубокое понимание процесса интеракции, т.е. как выбранная форма диалога общается с пользователем и инфосистемой и на какие функции распадается ее реализация.

2.2. Цели

Принимая во внимание вышеупомянутые прикладные особенности инфосистемы, необходимо выбрать комбинированную форму диалога, которая использует методы коммуникации с различной степенью сложности (вопрос/ответ, меню, макетный обмен). Из них два первых метода широко известны, что касается последнего, то он рассмотрен в статье [4]. Для замысла реализации сформулируем сначала цели, которых мы хотим достигнуть:

1) эффективная работа интерфейса согласно требованиям своевременной реакции в процессе инфообработки;

2) дружелюбие интерфейса к пользователю.

Под работой интерфейса согласно требованиям имеем в виду, что общение с ЭВМ (обмен данными) происходит с той скоростью, которая вполне достаточна для осуществления действия инфообработки (нормальная работа пользователя не должна нарушаться). Мы имеем в виду требование, по которому плохая реализация матобеспечения не помешала бы эффективности технического обеспечения.

Относительно дружелюбия к пользователю нет точных правил. По существу, эта личная точка зрения конечного пользователя и ее можно лишь описать. К понятию дружелюбия относятся следующие факторы: время, затраченное на обучение пользования системой, удобство в пользовании, продуктивность, утомляемость и количество ошибок. Значение этих факторов зависит от применения, от желаний пользователя, привычек и обстоятельств.

В настоящее время мерой качества интерфейса является то, что общение для пользователя естественно, насколько это возможно - он не познает существования хорошего интерфейса.

Например, пользователю необязательно знать об образе данных как при вводе, так и при выводе, т.е. данные представлены в образе, необходимом ему, и не в системе. Например, код какой-либо детали может сохраниться в памяти ЭВМ в другой форме, непривычной для пользователя. Система сама произведет вычисления согласно единице измерения, независимо от того, в каком виде в измерения введены данные. Преобразования для пользователя и для системы осуществляет интерфейс.

2.3. Свойства

При анализе свойств направленность такова, чтобы пользователь общался с ЭВМ по возможности непосредственно, употребляя здравый смысл и знания, полученные по специальности. Интерфейс с такой степенью дружелюбия к пользователю хорош везде, и особенно на рабочих местах предприятий, у которых терминал ЭВМ расположен за пределами конторы (производство, склады).

Отсюда следует, что в такой мере специализированный интерфейс пользователя должен быть широко настраиваемым. Это имеет смысл и в том случае, если настройка требует большого объема работ. Работа ЭВМ, включенная в процесс производства, окупит эти затраты. Было бы идеально, если бы и настройка проходила легко, не требуя от работника специальных знаний. Другими словами, если бы можно было распространить свойства интерфейса пользователя и на настройку! Для этого нужен единый интерфейс.

Часто возникает необходимость изменить или дополнить матобеспечение. Для этого хорошо подходят открытые системы. Под открытой системой мы подразумеваем систему, в которую можно включить новые компоненты (подсистемы), не изменяя самой системы. Если система открытая, то и интерфейс должен быть таковым. Интерфейс должен покрывать всю систему, а при добавлении к ней новой подсистемы также и ее.

С другой стороны, покрывающая способность имеет значение и для пользователя. Пользователь должен иметь доступ ко всем возможностям и действиям системы посредством единого интерфейса.

Из предыдущих рассуждений следуют четыре свойства интерфейса пользователя: дружелюбие к пользователю, настраиваемость, единость, покрывающая способность.

Исходя из свойств интерфейса пользователя, можно вывести функции, которые поддерживают реализацию этих свойств. На основе этих функций можно определить и действия, которые выполняют эти функции.

Поскольку функции конкретнее свойств, мы должны определить прикладную область инфосистемы, для которой нами рассматривается интерфейс пользователя. Определение проиллюстрируем сравнением.

Имеется в виду интерактивная инфосистема со своевременной реакцией с рабочими местами для микроЭВМ, которая охватывает:

1) управление деятельностью;
2) обработку документов;
3) управление базой данных;
4) коммуникацию между рабочими местами.

Например, система Framework, созданная для микроЭВМ, является системой для поддерживания интеллектуальной работы, т.е. работник не только знает, но и создает технологию своей работы, пользуется информацией и осмысливает ее [5]. Следовательно, Framework является рабочим средством для анализирующего руководителя вне производства. Мы занимаемся областью, в которой ЭВМ является частью технологического процесса инфообработки в производственном звене. При этом не имеет значения, происходит это в конторе или вне ее - имеется в виду рабочее место работника, квалификация которого ограничивается только выполнением действий, не касаясь организации процесса. Таким образом, учитывая особенности такой прикладной области, общение инфосистемы с пользователем должно охватывать следование технологии инфообработки.

Системы dBaseII и dBaseII имеют те же возможности, которые охватывает вышеописанная инфосистема. Вопрос в том, как может пользователь их реализовать. В системах dBase и Framework для этого необходимо их запрограммировать. Там создана операционная среда, основное назначение которой - интегрирование автономных программ, составлен-

ных пользователем. Мы имеем в виду наиболее развитую форму настройки - описание, при этом графическое описание. В сравниваемых здесь системах дружелюбие к пользователю на настройку не распространяется. К тому же эти системы не открытые. Функциональные возможности определены уже при разработке - язык FRED в системе Framework и макропрограммы в других системах дают ограниченные добавочные возможности.

3. Функции и рабочая среда интерфейса пользователя

Матобеспечение ЭВМ образует несколько слоев. Самый внешний слой - это интерфейс пользователя. Он является здесь средством коммуникации между частями инфосистемы (внутренних слоев матобеспечения) и пользователя (представителя внешнего мира).

У функции интерфейса есть два аспекта - коммуникация и обработка данных. Это проявляется в следующем.

Интерфейс пользователя образует совместно с соответствующим функциональным обеспечением т.н. пользуемое звено во внутренних слоях матобеспечения. Название этого звена находится в соответствии со взглядом пользователя. Например, база данных и система макетного обмена вместе с интерфейсом образуют средство для обработки документов. Это уже новое качество в сравнении с его составными частями. Здесь и уточняется функциональный подход. На уровне используемого звена создается рабочая среда интерфейса с помощью частей инфосистемы, которые ограничиваются интерфейсом, и мы говорим, что он поддерживает коммуникацию между ними.

Такая точка зрения дает нам возможность сосредоточить работу с данными, относящуюся к области интерфейса на том, где она особенно целесообразна и в добавление к этому дает возможность объединить части инфосистемы в отдельные единицы пользования.

Разделим функции интерфейса на две части:

1) работа интерфейса с данными;

2) прямое общение с пользователем (обмен информации через дисплей, сообщения об ошибках, реакции на действия пользователя).

Работу интерфейса в качестве посредника можно охарактеризовать следующей основной схемой:

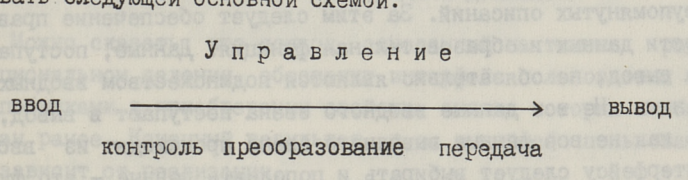


Рис. 1.

В зависимости от направления передачи данных одна и та же часть инфосистемы может выступать в роли ввода или вывода.

Анализируя основную схему с точки зрения рассматриваемой системы матобеспечения, дополним ее следующим.

Управление определяет на основании предписания, когда, что, откуда, куда надо передать. Управление - это действие, которое имеет функцию с покрывающей способностью - синхронизация с внешней технологией обработки информации. Для управления работой интерфейса пользователя будет достаточно, если представлена элементарная ведущая информация. В предписаниях не должна дублироваться информация, которая содержится в описаниях других частей инфосистемы. Отсюда следует необходимость использования описаний интерфейсом. Это позволяет сэкономить память, а настройка удобна и более гибка. Пользователь должен иметь возможность, но не обязанность определить желаемый образ данных, контрольное предписание и различные операции.

Исполнительная часть охватывает работу с данными. Если исходить из свойств рассмотренного интерфейса пользователя, то работа с данными - это их перенос из контекста ввода в контекст вывода, сопровождаемый контролем и преобразованием данных.

В работе с данными у интерфейса должна быть связь как с вводом, так и с выводом, назовем ее связующим звеном. Звено с вводом по существу является отпечатком вводных данных, определенного предписанием. Звено с выводом по существу является отпечатком данных, полученных из вводного звена, которые проверены, приведены в соответствующий вид для размещения в контекст вывода.

Интерфейс, таким образом, обладает функцией синхронизирования данных - генерирование звеньев, использование вышеупомянутых описаний. За этим следует обеспечение правильности данных и образовательная функция. Данные, поступающие на вывод, не обязательно являются подмножеством вводных данных. Не все данные вводного звена поступают в вывод, так же как не все данные выводного звена происходят из ввода. Интерфейсу следует выбирать и пополнять данные - это функция комплектации данных.

На основании предыдущего, представим дополненную схему:

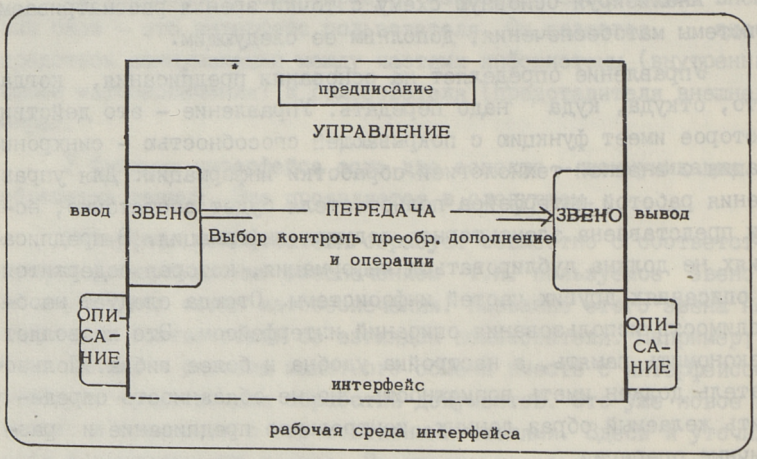


Рис. 2.

На основании вышесказанного выведем функции, определим действия:

Синхронизация с внешней технологией инфообработки	- управление;
Прямое общение с пользователем	- обмен информации через дисплей;
Синхронизация данных	- генерирование звеньев, использование описаний частей инфосистемы;
Обеспечение правильности данных	- контроль данных;
Формирование данных	- операции и преобразования;

Комплектация данных - выбор и пополнение данных;
Передача данных - передача данных.

Можно сказать, что подход, основанный на такого рода функциональном делении, обеспечит интерфейс пользователя предпосылками в приобретении свойств, анализ которых был сделан ранее. Конечный результат, т.е. интерфейс пользователя, зависит от реализации.

4. Выводы

Приведена авторская точка зрения об интерфейсе пользователя, проанализированы свойства и сформулированы функции. Проблемы структуры и реализации интерфейса являются предметом отдельного обсуждения.

Наш подход конечно же, не является единственным, но по нашему мнению, он бы позволил приблизиться к целям качественного обслуживания пользователя и в то же время добиться эффективности работы всей системы матобеспечения.

Литература

1. Nagl M. An incremental programming support environment // Computer Physics Communications. 1985. Vol. 38. P. 245-276.

2. Wasserman A.J. Modern software development methodologies and their environments // Computer Physics Communications. 1985. Vol. 38. P. 119-234.

3. Martin J. With research by Murch R. Application development without programmers // Savant Inst. Seminar Documentation by J. Martin. Carnforth: Savant Res. Studies, 1981.

4. Лумберг Т.А., Распель П.У. О разработке технологических структур для систем реального времени // Тр. Таллинск. политехн. ин-та. 1987. № 645. С. 100-112.

5. Aaltonen M. ym. Sovelluskehittimet ja tukipalvelu. Tietotekniikkan kehittamiskeskys ry. Tutkimusraportti. Helsinki. 1986. P. 35-40.

T. Lumberg, M. Roost, L. Elmik

An Approach to the Creation of the User's Interface for Just-In-Time System

Abstract

Organization of just-in-time data processing is a necessary pre-condition for the automation of current information processes.

This paper deals with some problems of the creation of the user's interface for just-in-time system. For creating such an interface qualities, which the interface must have, are taken into consideration. At the end the functional approach to the creation of the user's interface is described.

T. Lumberg, M. Roost, L. Elmik

Ajastussüsteemi kasutajaliides

Kokkuvõte

Artiklis käsitletakse ajastussüsteemi kasutajaliidese väljatöötamise probleeme. Analüüsitakse kasutajaliidese omadusi ning kirjeldatakse funktsionaalset lähenemist liidese väljatöötamiseks, arvestades liidese töökeskkonda.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.06

П. Пармаксон

ГЕНЕРАЦИЯ ОТЧЕТОВ В СИСТЕМЕ ТААТ

I. Постановка проблемы и исходные идеи

Составление отчетов статистического типа – одна из основных функций проблемно-ориентированных систем обработки данных. Отчет статистического типа – это таблица, для которой имеются логические условия отбора данных для каждой строки и каждого столбца. В каждую клетку отчета суммируется некоторая функция от значений записей, удовлетворяющих одновременно условиям соответствующей строки и соответствующего столбца. Довольно часто требуется в клетку таблицы отчета суммировать число записей, удовлетворяющих условию выбора. Отчеты формируются генераторами отчетов (ГО), которые имеют непроцедурные входные языки описания отчетов (ЯОО).

В течение длительного времени на кафедре обработки информации ТПИ разработано несколько инструментальных систем обработки данных экономического типа, главным мотивом их был принцип генерации прикладных программ исходя из описания задачи на непроцедурном проблемно-ориентированном языке (ПОЯ). Рассматриваемая система обработки данных ТААТ является развитием системы СХОДИ [1] в области генерирования матричных отчетов.

В статье [2] представлено понятие иерархического статистического отчета, в котором как строки, так и столбцы организованы во многоуровневую структуру. Прикладные системы, построенные с помощью системы СХОДИ, показали целесообразность введения следующих возможностей, кроме имеющихся [3]:

– возможность определения числа строк (столбцов) исходя из содержимого базы данных;

- возможность определения иерархических отчетов;
- возможность проведения арифметических действий над группами строк и столбцов (агрегирование арифметических действий над строками и столбцами);
- возможность определения перед составлением отчета переменных, которые входят как в оформление (тексты шапки и боковины), так и в условия таблицы.

Типичный запрос на отчет состоит из следующих компонентов: отбор данных средствами СУБД и их преобразование; описание содержания таблицы; преобразование таблицы; оформление таблицы; вывод таблицы в оформленном или неоформленном виде.

В системе СХОДИ перечисленные операции были реализованы отдельностоящими частями системы (подсистемы запросов в базу данных, описания и преобразования таблицы, оформления и печати), которые имели собственные входные языки. В статье [3] исследованы возможности совмещения языка манипулирования данными СУБД с Я00; предложен также проект языка описания матричных форм в среде СУБД.

В настоящей статье описывается система составления иерархических отчетов статистического типа ТААТ.

2. Общее описание системы ТААТ

Так называемые языки 4-го поколения могут условно быть разделены на две подгруппы: генераторы приложений для пользования профессиональными программистами и информационные генераторы для конечных пользователей. Для каких пользователей предназначена система ТААТ? ГО работает в интерактивном режиме, но предполагается, что доля одноразовых запросов будет довольно малой, так что описания отчетов оформляются программными файлами, которые вызываются по необходимости.

Из приведенных в разделе I компонентов запроса на отчет система ТААТ поддерживает все кроме отбора данных из БД и их первоначального преобразования. Так как вычислительная система реализации - ЭВМ СМ-4, операционная система ОС-РВ - налагает строгие ограничения на используемую системой память, то интегрирование ГО с СУБД не является целесообразным. В системе ТААТ связь со СУБД осуществляет-

ся с помощью последовательного файла. Запись файла выступает в роли суперзаписи, сформулированной запросом извлечения данных из БД. Система ТААТ располагает достаточно развитым ЯОО, который можно разделять на следующие подъязыки или группы конструкций: описание файла данных; описание матричной таблицы отчета; управление заполнением отчета; использование кодовых таблиц; преобразование таблицы отчета; вывод таблицы отчета; хранение полусоставленного отчета.

Так как разработка ПОЯ связана с конкретной вычислительной средой, то является целесообразным различать "ядро" ЯОО и зависимые от вычислительной среды конструкции вспомогательного характера (форматы спецификаций файлов, интерфейс с обслуживающими программами операционной системы). В дальнейшем будем рассматривать только основные языковые конструкции ГО, определяющие ядро. Полное описание синтаксиса ЯОО занимало бы слишком много места, поэтому приведем только самые существенные фрагменты.

Принципиально новыми отличительными чертами системы ТААТ по сравнению с предыдущими системами являются:

- иерархичность отчетов по строкам и по столбцам;
- возможность составлять отчеты переменной длины (отчеты, управляемые данными);
- логические условия выбора унифицированы по образцу паскалевских логических выражений, в них можно обращаться к предопределенным переменным;
- суммируемое в клетку выражение можно выбирать логическими условиями;
- заголовки строк, столбцов и всей таблицы специфицированы в описании структуры отчета, а не отдельно;
- в составе ЯОО имеется развитый подязык преобразования таблицы, содержащий стандартные типы данных и ряд общераспространенных и специализированных управляющих конструкций.

3. Основные функциональные возможности системы

При описании синтаксиса ЯОО в дальнейшем будем использовать расширенную форму записи Бэйкуса-Наура [4].

Описание файла данных. Входные файлы могут быть составлены по данным СУБД или иным способом. Описание файла данных - это определение логической структуры записи. Предусмотрены типы полей записи INTEGER, REAL, CHAR, причем числовые данные могут быть в двоичной или символьной формах.

Следующие примеры предполагают наличие файла с описанием

```
RECORD
    ЦЕХ  CHAR 6
    ИЗДЕЛИЕ  INTEGER SYMBOL4
    СТОИМОСТЬ  REAL SYMBOL9
    КОЛ-ВО INTEGER SYMBOL5;
```

Часть SYMBOL показывает длину символьно представленного числового поля. Семантика полей следующая: ЦЕХ - имя производственного отдела, ИЗДЕЛИЕ - имя изделия, СТОИМОСТЬ - стоимость одного изделия, КОЛ-ВО - выработка за день.

Описание матричной таблицы отчета. Структуры строк и столбцов определяются вложенными конструкциями разложений.

```
описание отчета = заглавие "ROW" разложение
                          "COL" разложение ":".
```

Разложение - это упорядоченная совокупность компонентов, которыми могут быть нижестоящие в иерархии разложения или в случае терминального (т.е. конечного) разложения - отдельные строки (столбцы). Разложения могут быть

а) эксплицитные, где структура полностью определяется описанием разложения;

б) имплицитные, где описываются только условия выбора, а состав компонентов определяется действительными входными данными.

В тексте программы-описания разложения ставят в квадратные скобки.

```
разложение = "[" эксплицитное_разложение !
                 имплицитное_разложение "]".
```

Структура разложений - иерархическая, т.е. каждый компонент разложения может содержать подструктуру. Так образуются деревья разложения для строк и столбцов. Компонентам конечно-

го разложения (которые не имеют подструктуры) соответствуют строки и столбцы таблицы.

Пример I. Таблица

Продукция за день

	Количество продукции		Стоимость	
	Изделие А	Изделие Б	Изделие А	Изделие Б
Цех № I				
Цех № 2				
Итого				

описывается текстом

```
REPORT 'Продукция за день'
    ROW [ TEXT 'Цех но.1' WITH ЦЕХ = 'но.1'
          TEXT 'Цех но.2' LINE AT END
                          WITH ЦЕХ = 'но.2'
          TEXT Итого ]
    COL [ TEXT 'Количество/продукции'
                        SUM КОЛ-ВО
            [ TEXT 'Изделие/А' WITH ИЗДЕЛИЕ = 1
              TEXT 'Изделие/Б' WITH ИЗДЕЛИЕ = 2 ]
          TEXT 'Стоимость'
                SUM СТОИМОСТЬ * КОЛ-ВО
            [ TEXT 'Изделие/А' WITH ИЗДЕЛИЕ = 1
              TEXT 'Изделие/Б' WITH ИЗДЕЛИЕ = 2 ]
        ] :
```

Здесь мы имеем дело только с эксплицитными разложениями. Приведем фрагмент из синтаксиса Я00, относящийся к определению эксплицитных разложений:

```
эксплицитное_разложение =   компонент_разложения .
компонент_разложения = "TEXT" симв_строка
                       [ "NAME" симв_строка ]
                       [  параметр_разлинования]
                       [  параметр_формата_чисел ]
                [ параметры_выбора_и_действий_заполнения ]
                [ разложение ] .
```

Кратко объясняем смысл параметров компонентов.

Выражение "TEXT" симв_строка указывает на заголовок строки (столбца), используемый при оформлении отчета, его можно также использовать в роли идентификатора при обращении к строкам (столбцам).

"NAME" симв_строка определяет идентификатор строки (столбца), используемый при преобразовании составленной таблицы. Фраза NAME функционально равносильна фразе TEXT, но в первом случае составные идентификаторы будут короче.

Параметры разлинования и формата чисел позволяют более гибко управлять оформлением отчета. В предыдущем примере с помощью фразы LINE AT END проводится линия под соответствующей строкой.

Параметры выбора имеют форму

параметр_выбора = "WITH" выражение.

Выражение должно быть логического типа, его значение вычисляют для каждой записи входного файла. Если условие выбора выполнено, то при нетерминальном компоненте приступят к исследованию подструктуры. При терминальном компоненте возможны два случая: 1) если пройдена строковая структура, то начинают прорабатывать столбцовую структуру; 2) если пройдена столбцовая структура, то исполняется действие заполнения отчета. При опускании параметра выбора по умолчанию предполагается значение TRUE.

Действием заполнения может быть или суммирование какого-нибудь выражения или подсчет количества записей, удовлетворяющих условиям.

параметр действия заполнения =
("SUM" выражение) ! "COUNT".

По умолчанию предполагается действие подсчета записей. В описании компонента можно совместно с параметрами выбора указать несколько параметров действий, например

```
TEXT A WITH ЦЕНА < 100,0 SUM ЦЕНА * КОЛ * 1.1
       WITH ЦЕНА > = 100,0 SUM ЦЕНА * КОЛ * 1.3
```

где при суммировании используются разные коэффициенты в зависимости от значения поля ЦЕНА.

Условие выбора для каждой клетки образуется конъюнкцией условий строки и столбца. При определении более низких

уровней иерархии мы можем сузить условия, с которыми имеем дело, но не расширить их : представитель более низкой ступени иерархии унаследует все свойства высшей ступени, и приобретает вдобавок новые свойства. Унаследованными параметрами являются параметры формата чисел, выбора и действий. Переопределение параметров формата чисел и действий отменяют соответствующие унаследованные значения.

Пример 2. Предположим, что в момент составления описания отчета нет сведений о конкретных именах цехов и изделий или нужно сделать сводку для всех цехов и изделий. К таким ситуациям эксплицитные разложения не приспособлены: добавление нового цеха, например, требует введения новой строки, и соответственно изменения запроса на таблицу. С помощью имплицитных разложений таблица

Продукция за день

	Количество продукции		Стоимость	
	<изделие>	...	<изделие>	...
Цеха . . .				
Итого				

описывается текстом

```
REPORT 'Продукция за день'
    ROW [ TEXT 'Цеха:'
            [BY ЦЕХ SORTED ]
          TEXT Итого ]
     COL [ TEXT 'Количество/продукции'
            [ BY ИЗДЕЛИЕ SUM КОЛ-ВО
                                  LINE AT EACH ]
          TEXT 'Стоимость'
            [ BY ИЗДЕЛИЕ SUM  КОЛ-ВО * СТОИМОСТЬ
            LINE AT EACH ]
        ] :
```

Синтаксис имплицитного разложения следующий:

```
имплицитное_разложение = "BY"  имя_поля_записи
                         [ параметр_сортировки ]
                         [ параметр_разлинования ]
```

[параметр_формата_чисел]
[имя_кодовой_таблицы]
[параметры_действий_заполнения]
[разложение] .

Фразой BY имя_поля_записи указывается поле записи, по значениям которой производится образование имплицитного разложения. В процессе прохождения файла появление каждого нового, ранее не наблюдаемого значения данного поля порождает новую компоненту с теми же параметрами, что и у остальных компонентов разложения. Таблица расширяется динамически во время заполнения отчета. Обычно появившиеся компоненты размещают в порядке появления, но с параметром сортировки можно отсортировать их в возрастающем или убывающем порядке. Параметры разлиновання, формата чисел и действий заполнения совпадают с определенными ранее. Отметим, что подструктуру, если это указывается в описании имплицитного разложения, приписывают каждому создаваемому компоненту. Имя кодовой таблицы указывается, если значения поля данных закодированы, тогда при оформлении заголовок используются тексты из кодовой таблицы, а не коды из файла данных.

Заполнение отчета. По отношению к заполнению различаем простые и групповые отчеты. Простой отчет заполняют командой

заполнение_простого_отчета = "EXEC"
"FILE" спецификация_файла
[условие_выбора_записи]
[общее_действие] ";" .

Наличие условия выбора записи и общего действия увеличивает гибкость настраивания описания отчета. В отчет входят только записи, удовлетворяющие условиям выбора. Общее действие исполняется между считыванием очередной записи и прохождением строковых и столбцовых структур, это предоставляет возможность вычислять промежуточные переменные, которые применяются при заполнении таблицы.

При составлении группового отчета предполагается, что одно или несколько полей данных во файле отсортированы. Групповой отчет - это отчет, составляемый для каждого различного значения указанного управляющего поля (или нескольких уп-

равляющих полей). Типичный пример – в исходном файле имеются данные о производимых в цехах деталях. Если отсортировать файл по названиям цехов, то можно описывать отчет "Производство деталей в цехе" и одной командой составить его для каждого цеха отдельно.

Преобразование таблицы отчета. Ясно, что изложенные средства описания отчетов подходят для хотя довольно часто встречаемого, но все-таки ограниченного класса отчетов. Иногда некоторые клетки вычисляются только после окончательного заполнения иных клеток, приходится как-то корректировать составленную таблицу или выдавать некоторые результаты на основе содержания таблицы, но вне таблицы. Следовательно, необходимы средства преобразования таблицы. Преобразования по характеру бывают самые разнообразные и выявить общую типологию трудно. Поэтому вместо выделения отдельных проблемно-зависимых операций (например, арифметические действия над строками или столбцами) выбран другой путь. Средства преобразования таблицы ЯОО представляют собой универсальный алгоритмический язык со следующими предопределенными типами данных: INTEGER, REAL, CHAR, индекс строки (столбца), клетка таблицы, строка, столбец. В управляющих конструкциях мы по возможности следовали стилю языка Паскаль. Рассмотрим особенности средств обращения к таблице. К клеткам таблицы обращаются с помощью составных имен вида

```
имя_клетки = "[" составное_имя_строки "."
                 составное_имя_столбца "]" .
```

Например, в примере I клетке на перекрестке второй строки и третьего столбца соответствует имя

```
[ 'Цех но.2'. Стоимость: 'Изделие/А' ]
```

Путь в дереве строковой (столбцовой) структуры определяется именами компонентов разложения или при их отсутствии текстами заголовков компонентов.

Управляющие конструкции EACH ROW и EACH COL суть групповые операции, определенные соответственно над всеми строками или столбцами.

Пример 3. Присваивание всем элементам таблицы значения 0 производится вложением конструкции EACH ROW и EACH COL. Звездочкой отмечена текущая строка или столбец.

```
EACH ROW DO
    EACH COL DO
        [*,*] := 0 ;
```

Оператор FOR определяет повторяющееся выполнение тела FOR-цикла для всех или некоторых компонентов разложения.

Пример 4. Конструкция

```
VAR X: INDEX;
VAR Y: INDEX;
VAR Z: INDEX;
FOR X UNDER 'Цехи:' INDEX DO
    FOR Y BETWEEN 'Количество/продукции'
                  AND Стоимость DO
        FOR Z UNDER Y IN ИЗДЕЛИЕ DO
            [ Итого.* ] := [ Итого.* ] +
                           [ X,Z ];
```

производит вычисление строки Итого, хотя и не рациональнейшим образом (см. пример 2).

Вывод таблицы отчета. Размеры таблицы и определяемые пользователем размеры страницы определяют размещение таблицы на странице. Конечных компонентов разложения пополам не делят. Пользователь может потребовать повторения заглавий столбцов на каждой странице. Символы / , использованные в описании заголовков компонентов разложений, определяют размещение заголовка на нескольких строках страницы.

Составляемые средствами ГО системы ТААТ отчеты могут быть объектом дальнейшей автоматизированной обработки. Для этого предусмотрена возможность выдачи матрицы отчета без строковых и столбцовых структур во внутреннем представлении числовых значений.

Хранение отчета. Исходные данные для отчета могут накапливаться в течение длительного времени. Также, бывают случаи, когда объем данных настолько большой, что их одновременное хранение в вычислительной системе не представля-

ется возможным или эффективным. Для таких ситуаций предназначены средства записывания частично заполненной таблицы во внешнюю память и обратного считывания. Порядок работы будет тогда следующим:

- отчет заполняется на основе доступных данных и записывается на диск;

- по прибывании дополнительных порций данных отчет загружается в основную память, дополняется новыми данными и записывается снова.

Например, если параллельно с квартальными отчетами надо составлять и годовой отчет, то кумулятивное заполнение годового отчета позволяет по принципу отказаться от сохранения данных каждого квартала в продолжении всего года.

4. Заметки по реализации

С развитием средств инструментальной системы, усложнением их структуры все важнее становятся элементы сопряжения системы с пользователем. Нагромождение параметров разного рода, вообще выразительных возможностей ПОЯ отмечено как "смертельный недуг" генераторных систем, так как пользователям будет трудно создать себе ясное представление о системе, а разработчикам - обеспечить цельность и согласованность составных частей ПОЯ. Особой поддержки со стороны инструментальной системы требует процесс разработки программ на ПОЯ, от этого зависит успех у пользователей. ГО позволяет чередовать редактирование описания отчета с отладкой. Перспективным кажется интегрирование в систему синтаксически-ориентированного редактора, который может повысить интерактивность диалога и поддерживать пользователя при выборе подходящих языковых конструкций.

ГО реализован как интерпретатор ЯОО. Испытания показали, что преобразование входного текста в атрибутное дерево абстрактного синтаксиса отнимает ничтожное время по сравнению с прохождением файла данных и действительным составлением отчета. ЯОО имеет LL-грамматику. Обработка вводимых конструкций состоит из двух шагов: I) преобразование запроса во внутреннее описание (в атрибутное дерево абстрактного синтаксиса). При этом использована методика синтакси-

ческого анализа Н. Вирта [5]. Мы думаем, что целесообразность использования систем построения трансляторов для реализации интерпретируемых ПОЯ остается открытой, так как это существенно зависит от квалификации разработчика и качества документации СПТ; 2) прохождение созданного атрибутного дерева с вычислением атрибутов и преобразованием дерева. Отметим, что внутреннее дерево описания отчета является одновременно и управляющей структурой и структурой данных для хранения элементов составляемого отчета.

Литература

I. Выханду Л.К., Лучковский Т.Ф., Микли Т.И., Тепанди Я.Я. Система хранения и обработки дискретной информации // Управляющие системы и машины. 1981. № 1. С. 99-102.

2. Тепанди Я.Я. Генерация отчетов статистического типа в среде СУБД // Тр. Таллинск. политехн. ин-та. 1984. № 568. С. 23-24.

3. Tepandi J. Data-driven matrix forms // Trans. of Tallinn Tech. Univ. 1986. N 614. P. 31-42.

4. Wirth N. What can we do about the unnecessary divergence of notations for syntactic definitions? // CACM. Nov. 1977. Vol. 20, N 11.

5. Вирт Н. Алгоритмы + структуры данных = программы // М. 1985. 406 с.

P. Parmakson

Report Generating System TAAT

Abstract

A hierarchical statistical report is a table containing some data from the data base. It is determined by two description structures: one for rows and one for columns. In this article the main designing and implementing principles of hierarchical report generating system TAAT are presented.

P. Parmakson

Aruannete genereerimise süsteem TAAT

Kokkuvõte

Hierarhiline statistiline aruanne on andmebaasi andmete põhjal koostatud tabel, milles nii read kui veerud moodustavad mitmetasemelise struktuuri. Artiklis käsitletakse hierarhiliste aruannete kirjeldamise, koostamise ning teisendamise probleeme. Kirjeldatakse aruannete genereerimise süsteemi TAAT.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 519.4

П. Пуусемп

СВОЙСТВА ЕСТЕСТВЕННОЙ ПРОЕКЦИИ СТАНДАРТНОГО СПЛЕТЕНИЯ ГРУПП

I. Введение

Для заданных групп A и B можно рассматривать множество A^B всех функций $f: B \to A$. Рассмотрим множество A^B как группу, где произведение fg элементов $f, g \in A^B$ определено равенством

$$(fg)(b) = (f(b))(g(b)), \quad b \in B.$$

Образуем полупрямое произведение $G = A^B \lambda B$ групп A^B и B с правилом умножения

$$(bf)(cg) = bc f^c g, \tag{I.1}$$

где $b, c \in B$; $f, g \in A^B$

и

$$f^c(d) = f(d c^{-1}), \quad d \in B. \tag{I.2}$$

Полученная группа G называется (стандартным) сплетением групп A и B и обозначается $G = A \operatorname{Wr} B$. Если в определении сплетения заменить группу A^B ее подгруппой $A^{(B)}$, где $A^{(B)}$ состоит из таких $f \in A^B$, для которых носитель $\{b \in B \mid f(b) \neq 1\}$ конечен, то полученная группа $A^B \lambda B$ с правилом умножения (I.1) называется ограниченным сплетением групп A и B и обозначается $A \operatorname{wr} B$.

Многие важные свойства групп $A \operatorname{Wr} B$ и $A \operatorname{wr} B$ описаны П. Нейманом [1]. В работе Хактона [2] изучено строение группы автоморфизмов сплетения. Дополнительную информацию об абелевых нормальных делителях и автоморфизмах сплетений задает Я.Г. Беркович [3].

Мы занимались исследованием определяемости сплетения $G = A \operatorname{wr} B$ его полугруппой эндоморфизмов $\operatorname{End} G$ в классе всех групп. Оказалось, что решение этого вопроса сущест-

венно зависит от свойств одного естественно возникающего идемпотента x полугруппы $\mathrm{End}\,G$ и связанных с ним подполугрупп $\rceil_G(x)$ и $P_G(x)$ полугруппы $\mathrm{End}\,G$, где

$$\rceil_G(x) = \{ y \in \mathrm{End}\,G \mid yx = xy = 0 \},$$
$$P_G(x) = \{ y \in \mathrm{End}\,G \mid yx = xy = x \}.$$

Этим важным идемпотентом x является проекция группы $G = A \wr_r B = A^{(B)} \lambda B$ на ее подгруппу B. В настоящей работе дается описание указанных совокупностей $\rceil_G(x)$ и $P_G(x)$ эндоморфизмов группы G (теоремы 3.1 и 4.4, следствия 3.2, 4.5 и 4.6). Отметим, что некоторое описание элементов полугруппы $P_G(x)$ для мономиальных групп дал О. Оре [4].

Будем придерживаться следующих обозначений: $\mathrm{End}\,G$ – полугруппа всех эндоморфизмов группы G; A' – коммутант группы A; $\mathrm{Hom}(A,B)$ – множество всех гомоморфизмов из группы A в группу B; $[a,b] = a^{-1}b^{-1}ab$; $\langle a,b,\ldots\rangle$ – подгруппа, порожденная элементами $a,b,\ldots$; $a^b = b^{-1}ab$.

2. Некоторые свойства ограниченного сплетения

В настоящем пункте опишем некоторые свойства ограниченного сплетения

$$G = A \wr_r B = A^{(B)} \lambda B \qquad (2.1)$$

групп A и B.

Для любого элемента $b \in B$ обозначим

$$A_b = \{ f \in A^{(B)} \mid f(d) = 1 \text{ при } d \neq b \}.$$

Множество A_b является подгруппой в группе G. Между группами A_b и A существует естественный изоморфизм φ_b, определенный правилом $f\varphi_b = f(b)$, где $f \in A_b$. Подгруппа $A^{(B)}$ сплетения $A \wr_r B$ является (внутренним) прямым произведением групп A_b, $b \in B$. Ввиду равенства (1.2) имеем

$$(A_b)^c = c^{-1}A_b c = A_{bc}. \qquad (2.2)$$

Следовательно, при каждом $b \in B$ имеем

$$A^{(B)} = \Pi_{c \in B} A_c = \Pi_{c \in B} (A_b)^c. \qquad (2.3)$$

Поэтому каждый элемент подгруппы $A^{(B)}$ группы G выражается в виде

$$f_1^{c_1} \cdots f_n^{c_n}, \qquad (2.4)$$

где $c_1,\ldots,c_n$ – разные элементы подгруппы B и $f_1,\ldots,f_n \in A_b$.

Через $[B, A^{(B)}]$ обозначим взаимный коммутант подгрупп B и $A^{(B)}$ группы G:

$$[B, A^{(B)}] = \langle [b, f] \mid b \in B, f \in A^{(B)} \rangle .$$

Лемма 2.I (Нейман [I], теорема 4.I). Если группа B неединична, то

$$[B, A^{(B)}] = \{ f \in A^{(B)} \mid \pi(f) \in A' \},$$

где $\pi(f)$ означает произведение всех значений функции f (в любом порядке).

Отсюда вытекает сразу

Лемма 2.2. Если группа B неединична, то $A'_b = A_b \cap [B, A^{(B)}]$.

Лемма 2.3. Подгруппа $[B, A^{(B)}]$ является нормальным делителем в группе $A \wr B$.

Доказательство. Если $d, c \in B$ и $f, g \in A^{(B)}$, то в силу нормальности подгруппы $A^{(B)}$ в $A \wr B$ имеем

$$d^{-1}[c, f]\, d = [c^d, f^d] \in [B, A^{(B)}],$$

$$g^{-1}[c, f]\, g = [c^{-1}, g^c] \cdot [c^{-1}, (fg)^c]^{-1} \in [B, A^{(B)}],$$

откуда следует нормальность подгруппы $[B, A^{(B)}]$ группы $A \wr B$. Лемма доказана.

Лемма 2.4. $A^{(B)} = A_b \cdot [B, A^{(B)}]$.

Доказательство. Обозначим $K = A_b \cdot [B, A^{(B)}]$. Ввиду леммы 2.3 K является подгруппой группы G. Покажем равенство $A^{(B)} = K$. Для этого достаточно доказать включение $A^{(B)} \subset K$, ибо включение $K \subset A^{(B)}$ очевидно. Так как группа $A^{(B)}$ порождается подгруппами A_c, $c \in B$, и $A_b \subset K$, то надо показать включение $A_c \subset K$ для каждого $c \in B$, $c \neq b$.

Пусть $c \in B$, $c \neq b$ и $g \in A_c$, т.е. $g(d) = 1$ для каждого $d \neq c$. Тогда существует единственный $f \in A_b$, так что $f(b) = g(c)$. Такой f удовлетворяет равенству $f \cdot [b^{-1}c, f^{-1}] = g$. Поэтому $g \in K$. Следовательно, $A_c \subset K$ и равенство $A^{(B)} = K$ показано. Лемма доказана.

Лемма 2.5. Если группа A коммутативна, то $A^{(B)} = A_b \times [B, A^{(B)}]$.

Доказательство. Пусть группа A коммутативна. Тогда группа $A^{(B)} = A_b \cdot [B, A^{(B)}]$ также коммутативна. Ввиду $A_b' = \langle 1 \rangle$ и леммы 2.2 имеем $A_b \cap [B, A^{(B)}] = \langle 1 \rangle$. Поэтому $A^{(B)} = A_b \times [B, A^{(B)}]$. Лемма доказана.

Ввиду $B \cap [B, A^{(B)}] = \langle 1 \rangle$ и леммы 2.3 имеем $B \cdot [B, A^{(B)}] = [B, A^{(B)}] \lambda B$. Так как $[B, A^{(B)}]$ нормальна в G и согласно разложению (2.I) и лемме 2.4 группа G порождается подгруппами $A_b, B, [B, A^{(B)}]$, то подгруппа $B \cdot [B, A^{(B)}]$ нормальна в G. Если группа A коммутативна, то в силу леммы 2.5 и разложения (2.I) будет $A_b \cap (B \cdot [B, A^{(B)}]) = \langle 1 \rangle$. Поэтому имеет место:

Лемма 2.6. Если группа A коммутативна, то

$$A \wr_r B = (B \cdot [B, A^{(B)}]) \lambda A_b = ([B, A^{(B)}] \lambda B) \lambda A_b = ([B, A^{(B)}] \times A_b) \lambda B.$$

3. Описание множества $\rbrack_G(x)$

Сохраняем обозначения предыдущего пункта. Обозначим через x идемпотент полугруппы $End\, G$, соответствующий полупрямому разложению (2.I), т.е. $Im x = B$ и $Ker x = A^{(B)}$. В настоящем пункте определим элементы подполугруппы

$$\rbrack_G(x) = \{ y \in End\, G \mid xy = yx = 0 \}$$

полугруппы $End\, G$.

Множество $\rbrack_G(x)$ состоит из таких эндоморфизмов y группы G, для которых $Im x = B \subset Ker\, y$ и $Im\, y \subset Ker\, x = A^{(B)}$. В силу $B \subset Ker\, y$ также $[B, A^{(B)}] \subset Ker\, y$. Следовательно,

$$\rbrack_G(x) = \{ y \in End\, G \mid B \cdot [B, A^{(B)}] \subset Ker\, y,\ Im\, y \subset A^{(B)} \}. \quad (3.I)$$

Из последнего равенства следует, что каждый $y \in \rbrack_G(x)$ является произведением естественного гомоморфизма $G \to G / (B \cdot [B, A^{(B)}])$ и гомоморфизма $y^* : G/(B \cdot [B, A^{(B)}]) \to A^{(B)}$, где $(g(b \cdot [B, A^{(B)}]))y^* = gy$. Соответствие $y \to y^*$ является взаимно однозначным соответствием между множествами $\rbrack_G(x)$ и $Hom(G/(B \cdot [B, A^{(B)}]), A^{(B)})$. Но при $B \neq \langle 1 \rangle$ имеем $A_b' = A_b \cap [B, A^{(B)}]$ и

$$G/(B \cdot [B, A^{(B)}]) = (A_b \cdot B \cdot [B, A^{(B)}])/(B \cdot [B, A^{(B)}]) \cong$$

$$\cong A_b/(A_b \cap (B \cdot [B, A^{(B)}])) =$$

$$= A_b/(A_b \cap [B, A^{(B)}]) = A_b/A_b' \cong A/A'.$$

Здесь изоморфизм $G/(B \cdot [B, A^{(B)}]) \cong A/A'$ достигается соответствием $fu \to f(b)A'$, где $f \in A_b$ и $u \in B \cdot [B, A^{(B)}]$. Поэтому имеет место следующая теорема.

Т е о р е м а 3.I. Если группа B не единична, то между множествами $\mathcal{J}_G(x)$ и $\mathrm{Hom}(A/A', A^B)$ существует естественное взаимно однозначное соответствие $y \to y^*$, определенное правилами:

I) если $y \in \mathcal{J}_G(x)$ и $aA' \in A/A'$, то $(aA')y^* = fy$, где $f \in A_b$ и $f(b) = a$;

2) если $y^* \in \mathrm{Hom}(A/A', A^{(B)})$ и $c \in B$, $gf \in A^{(B)} = [B, A^{(B)}] \cdot A_b$ $(g \in [b, A^{(B)}]$; $f \in A_b)$, то $(cgf)y = (f(b)A')y^*$.

С л е д с т в и е 3.2. Если группа B не единична и группа A коммутативна, то между множествами $\mathcal{J}_G(x)$ и $\mathrm{Hom}(A, A^{(B)})$ существует естественное взаимно однозначное соответствие.

4. Строение полугруппы $P_G(x)$

Пусть снова x идемпотент, соответствующий полупрямому разложению (2.I). Для него можно рассматривать множество эндоморфизмов

$$P_G(x) = \{y \in \mathrm{End}\, G \mid yx = xy = x\},$$

которое является подполугруппой полугруппы $\mathrm{End}\, G$. Наша цель в настоящем пункте описать строение полугруппы $P_G(x)$. Непосредственно из определения полугруппы $P_G(x)$ следует:

Л е м м а 4.I. Эндоморфизм y группы G принадлежит множеству $P_G(x)$ тогда и только тогда, когда $cy = c$ для каждого $c \in b$ и $A^{(B)}y \in A^{(B)}$.

Ввиду разложения (2.I) и представления элементов группы $A^{(B)}$ в виде (2.4) (взять $b = 1$) каждый элемент группы G выражается в виде $cf_1^{c_1} \dots f_n^{c_n}$, где $c \in b$; $f_1, \dots, f_n \in A_1$ и $c_1, \dots, c_n$ разные элементы из B. Если $y \in P_G(x)$, то в силу леммы 4.I

$$(cf_1^{c_1} \dots f_n^{c_n})y = c(f_1 y)^{c_1} \dots (f_n y)^{c_n}. \qquad (4.I)$$

Поэтому эндоморфизм y полностью определен его действием на подгруппе A_1 группы $A^{(B)}$. При этом $A_1 y \in A^{(B)}$. Это является ключевым свойством для описания полугруппы $P_G(x)$.

Предположим, что $y \in P_G(x)$. Обозначим через y_c гомоморфизм $A_1 \to A_c$, являющийся произведением гомоморфизма $y \mid A_1$: $A_1 \to A^{(B)}$ и проекции $A^{(B)} \to A_c$ ($y \mid A_1$ — ограничение эндоморфизма y на подгруппу A_1). Тогда

$$fy = \Pi_{c \in B} fy_c \text{ для каждого } f \in A_1. \qquad (4.2)$$

Теперь возникает эндоморфизм γ_c группы A, определенный равенством $\gamma_c = \varphi_1^{-1} \gamma_c \varphi_c$, где φ_c – естественный изоморфизм $A_c \to A$. Следовательно, каждому $\gamma \in P_G(x)$ соответствует семейство $\{\gamma_c\}_{c \in B}$ эндоморфизмов группы A. Действие эндоморфизма γ при этом полностью восстанавливается этим семейством (см. формулы (4.1) и (4.2)):

$$(a f_1^{c_1} \dots f_n^{c_n})\gamma = a \cdot ((\Pi_{d \in B} f_1(\varphi_1 \gamma_d \varphi_d^{-1}))^{c_1} \cdot \dots \cdot (\Pi_{d \in B} f_n(\varphi_1 \gamma_d \varphi_d^{-1}))^{c_n}), \tag{4.3}$$

где $a \in B$; $c_1, \dots, c_n$ – разные элементы из B и $f_1, \dots, f_n \in A_1$.

Остается установить условия, которым должны удовлетворять эндоморфизмы γ_c группы A.

Лемма 4.2. Если $\gamma \in P_G(x)$, то соответствующее ему семейство $\{\gamma_c\}_{c \in B}$ эндоморфизмов группы A удовлетворяет условию:

$$(a\gamma_c)(a_1\gamma_d) = (a_1\gamma_d)(a\gamma_c) \tag{4.4}$$

для каждых $a, a_1 \in A$ и $c, d \in B$, $c \neq d$.

Доказательство. Пусть $a, a_1 \in A$ и $c, d \in B$, $c \neq d$. Покажем равенство (4.4). Обозначим $f = a\varphi_1^{-1}$ и $g = a_1\varphi_1^{-1}$. Тогда $f, g \in A_1$ и ввиду равенства (2.2) $f^{c^{-1}} \in A_{c^{-1}}$, $g^{d^{-1}} \in A_{d^{-1}}$. В силу разложения (2.3) и $c \neq d$ элементы $f^{c^{-1}}$ и $g^{d^{-1}}$ коммутируют. Поэтому коммутируют также их образы $f^{c^{-1}}\gamma$ и $g^{d^{-1}}\gamma$ при эндоморфизме γ. Но по равенствам (4.1) и (4.2)

$$f^{c^{-1}}\gamma = (f\gamma)^{c^{-1}} = \Pi_{\lambda \in B}(f\gamma_\lambda)^{c^{-1}}, \tag{4.5}$$

$$g^{d^{-1}}\gamma = (g\gamma)^{d^{-1}} = \Pi_{t \in B}(g\gamma_t)^{d^{-1}}, \tag{4.6}$$

где $(f\gamma_\lambda)^{c^{-1}} \in A_{\lambda c^{-1}}$ и $(g\gamma_t)^{d^{-1}} \in A_{td^{-1}}$.

Из коммутирования элементов $f^{c^{-1}}\gamma$ и $g^{d^{-1}}\gamma$ следует, что коммутируют также их компоненты в разложениях (4.5) и (4.6), принадлежащие к A_1. Следовательно,

$$(f\gamma_c)^{c^{-1}} \cdot (g\gamma_d)^{d^{-1}} = (g\gamma_d)^{d^{-1}} \cdot (f\gamma_c)^{c^{-1}}. \tag{4.7}$$

Применяя обе стороны равенства (4.7) к единице группы B, в силу равенства (1.2) получим

$$((f\gamma_c)(c))((g\gamma_d)(d)) = ((g\gamma_d)(d))\cdot((f\gamma_c)(c)),$$

откуда ввиду $(f\gamma_c)(c) = (f\gamma_c)\varphi_c$ и $(g\gamma_d)(d) = (g\gamma_d)\varphi_d$ вытекает

$$(f(\gamma_c\varphi_c))\cdot(g(\gamma_d\varphi_d)) = (g(\gamma_d\varphi_d))\cdot(f(\gamma_c\varphi_c)). \tag{4.8}$$

Равенство (4.8) равносильно равенству (4.4), если учитывать определения элементов f,g и эндоморфизмов γ_c, γ_d. Лемма доказана.

Л е м м а 4.3. Если группа B конечна и некоторое семейство $\{\gamma_c\}_{c\in B}$ эндоморфизмов группы A удовлетворяет при каждых $c,d\in B, c\neq d$ и $a,a_1\in A$ равенству (4.4), то отображение $\gamma: G \to G$, определенное равенством (4.3), принадлежит полугруппе $P_G(x)$.

Д о к а з а т е л ь с т в о. Пусть выполнены предположения леммы. Каждый элемент группы G выражается в виде $cf_1^{c_1}\dots f_n^{c_n}$, где $c\in B; f_1,\dots,f_n\in A_1$ и $c_1,\dots,c_n$ – разные элементы из B. Определим отображение $\gamma: G\to G$ равенством (4.3) и покажем, что $\gamma\in P_G(x)$. Отметим, что конечность группы B гарантирует существование всех произведений в равенстве (4.3). Поэтому отображение γ определено корректно.

Обозначим $\varphi_1\gamma_d\varphi_d^{-1}=\gamma_d$, $d\in B$. Отображение γ_d является гомоморфизмом из группы A_1 в группу A_d. Тогда равенство (4.3) равносильно равенствам

$$(cf_1^{c_1}\dots f_n^{c_n})\gamma = c(f_1\gamma)^{c_1}\dots(f_n\gamma)^{c_n}, \tag{4.9}$$

$$f_i\gamma = \Pi_{d\in B} f_i\gamma_d,\quad i=1,2,\dots,n, \tag{4.10}$$

и равенство (4.4) равносильно равенству (4.8), где $f, g\in A_b$; $d,c\in B$, $d\neq c$.

Докажем сначала, что при $s,t\in B, s\neq t$ и $f,g\in A_1$ имеет место равенство

$$(f\gamma)^s\cdot(g\gamma)^t=(g\gamma)^t\cdot(f\gamma)^s. \tag{4.11}$$

Ввиду равенства (4.10) надо показать равенство

$$(\Pi_{d\in B}(f\gamma_d)^s)\cdot(\Pi_{d\in B}(g\gamma_d)^t)=(\Pi_{d\in B}(g\gamma_d)^t)(\Pi_{d\in B}(f\gamma_d)^s). \tag{4.12}$$

Для этого надо убедиться, что компоненты обеих сторон равенства (4.12), принадлежащие в разложении (2.3) одному и тому же прямому сомножителю A_c, совпадают. Так как $(f\gamma_d)^s\in A_{ds}$ и $(g\gamma_d)^t\in A_{dt}$, то компоненты, принадлежащие к A_c, в левой и правой сторонах равенства (4.12) соответственно равны $(f\gamma_{cs^{-1}})^s\cdot(g\gamma_{ct^{-1}})^t$ и $(g\gamma_{ct^{-1}})^t\cdot(f\gamma_{cs^{-1}})^s$. Поэтому надо показать, что

$$(f\gamma_{cs^{-1}})^s\cdot(g\gamma_{ct^{-1}})^t=(g\gamma_{ct^{-1}})^t\cdot(f\gamma_{cs^{-1}})^s. \tag{4.13}$$

Но равенство (4.I3) выполняется, ибо в силу равенства (4.8) его обе стороны действуют одинаково на элементах группы B. Следовательно, справедливость равенства (4.II) доказана.

Установим, что отображение γ является эндоморфизмом группы G. Так как $\gamma_d \in \mathrm{Hom}(A_1, A_d)$ для каждого $d \in B$, то ввиду разложения (2.3) и равенства (4.I0) $\gamma | A_1 \in \mathrm{Hom}(A_1, A^{(B)})$. Тогда по формуле (4.9) $\gamma | A_d \in \mathrm{Hom}(A_d, A^{(B)})$ при любом $d \in B$. Ввиду равенства (4.II) и определения отображения γ ясно, что $\gamma | A^{(B)} \in \mathcal{E}nd\, A^{(B)}$. Поэтому для произвольных элементов cf и dg группы $G (c, d \in B; f, g \in A^{(B)})$ имеем

$$((cf)(dg))\gamma = (cdf^d g)\gamma = cd((f^d g)\gamma) =$$
$$= cd(f^d \gamma)(g\gamma) = cd(f\gamma)^d (g\gamma),$$
$$((cf)\gamma)((dg)\gamma) = c(f\gamma)\, d(g\gamma) = cd\,(f\gamma)^d (g\gamma),$$

ибо из равенства (4.9) следует равенство $f^d \gamma = (f\gamma)^d$. Следовательно, γ является эндоморфизмом группы G.

Остается еще доказать, что $\gamma \in P_G(x)$. Но это следует сразу из леммы 4.I и определения эндоморфизма γ. Лемма доказана.

Определим еще правило умножения в полугруппе $P_G(x)$. Пусть $\gamma, \zeta \in P_G(x)$ и

$$\gamma = \{\gamma_c\}_{c \in B}; \quad \zeta = \{\zeta_c\}_{c \in B}; \quad \gamma\zeta = \{U_c\}_{c \in B}$$

(здесь мы отождествили элементы полугруппы $P_G(x)$ с соответствующими им семействами эндоморфизмов группы A). Предположим, что $f \in A_1$. Согласно равенству (4.I0) имеем

$$f\gamma = \Pi_{d \in B} f(\varphi_1 \gamma_d \varphi_d^{-1}) \quad \Pi_{d \in B} f_d .$$

Здесь $f_d = f(\varphi_1 \gamma_d \varphi_d^{-1}) \in A_d$ и

$$f_d(c) = (f(\varphi_1 \gamma_d \varphi_d^{-1}))(c) = \begin{cases} f(1)\gamma_d, & \text{если } c = d, \\ 1, & \text{если } c \neq d. \end{cases}$$

Поэтому для любого $b \in B$ имеем

$$(f\gamma)(b) = f(1)\gamma_b .$$

Аналогичные формулы справедливы также для эндоморфизмов ζ и $\gamma\zeta$ (с соответствующими изменениями в обозначениях). В силу этих замечаний и равенств

$$f(\gamma\zeta) = (\Pi_{d \in B} f_d)\zeta = \Pi_{d \in B}(((f_d)^{d^{-1}})^d \zeta) = \Pi_{d \in B}((f_d)^{d^{-1}} \zeta)^d,$$

где $(f_d)^{d^{-1}} \in A_1$, имеем

$$(f(\gamma\zeta))(b) = \Pi_{d \in B}(((f_d)^{d^{-1}} \zeta)^d (b)) = \Pi_{d \in B}(((f_d)^{d^{-1}} \zeta)(bd^{-1})) =$$

$$= \Pi_{d\in B}((f_d)^{d^{-1}}(1))z_{bd^{-1}} = \Pi_{d\in B}(f_d(d))\,z_{bd^{-1}} =$$

$$= \Pi_{d\in B} f(1)\,y_d\,z_{bd^{-1}} = (f(1))\,\Sigma_{d\in B} y_d z_{bd^{-1}}\,. \qquad (4.14)$$

С другой стороны, учитывая равенство $yz=\{u_c\}_{c\in B}$, получим

$$(f(yz))(b) = f(1)\,u_b. \qquad (4.15)$$

Так как каждый элемент группы A имеет при подходящем $f\in A_1$ вид $f(1)$, то из равенств (4.14) и (4.15) следует равенство

$$u_b = \Sigma_{d\in B} y_d z_{bd^{-1}}, \qquad (4.16)$$

которое справедливо для каждого $b\in B$. Равенство (4.16) задает правило умнскения в полугруппе $P_G(x)$.

Объединяем полученные результаты в теорему:

Т е о р е м а 4.4. Каждому элементу $y\in P_G(x)$ однозначно соответствует семейство $\{y_c\}_{c\in B}$ эндоморфизмов группы A, удовлетворяющих условию

$$(a\,y_c)(a_1 y_d) = (a_1 y_d)(a\,y_c) \qquad (4.17)$$

для каждых $a, a_1\in A$ и $c, d\in B$, $c\neq d$. Если группа B конечна, то заданное соответствие между элементами полугруппы $P_G(x)$ и семействами, удовлетворяющими указанному условию (4.17), является взаимно однозначным. Если отождествить элементы полугруппы $P_G(x)$ с соответствующими им семействами, то произведение элементов $\{y_c\}_{c\in B}$ и $\{z_c\}_{c\in B}$ полугруппы $P_G(x)$ задается правилом:

$$\{y_c\}_{c\in B}\cdot\{z_c\}_{c\in B} = \{u_c\}_{c\in B},$$
$$u_c = \Sigma_{d\in B} y_d z_{cd^{-1}}. \qquad (4.18)$$

Из правила умножения (4.18) в полугруппе $P_G(x)$ вытекают сразу следующие два следствия:

С л е д с т в и е 4.5. Если группа B конечна и полугруппа $\mathrm{End}\,A$ коммутативна, то полугруппа $P_G(x)$ изоморфна мультипликативной полугруппе группового кольца группы B^* над кольцом $\mathrm{End}\,A$, где B^* – группа, антиизоморфная группе B.

С л е д с т в и е 4.6. Если группа B и полугруппа $\mathrm{End}\,A$ коммутативны, то полугруппа $P_G(x)$ коммутативна.

Л и т е р а т у р а

1. N e u m a n n P.M. On the structure of standart wreath products of groups // Math. Z., 1964. 84. P. 343–373.

2. H o u g h t o n C.H. On the automorphism groups of certain wreath products // Publ. Math. 1962. 9, N 3-4.. P. 307-313.

3. Б е р к о в и ч Я.Г. Абелевы нормальные делители и автоморфизмы некоторых стандартных сплетений // Изв. высш. уч. заведений. Математика. 1973. № II. С. 15-20.

4. O r e O. Theory of monomial groups // Trans. Amer. Math. Soc. 1942. 51. P. 15-64.

P. Puusemp

Properties of the Natural Projection of Standard Wreath Product of Groups

Abstract

Let G = AWrB be the standard wreath product of groups A and B and x the natural projection of G onto B. In this paper we characterize the subsemigroups

$$J_G = \{ y \in \mathrm{End}G \mid yx = xy = 0 \}$$

and

$$P_G = \{ y \in \mathrm{End}G \mid yx = xy = x \}$$

of the semigroup EndG (Theorems 3.1 and 4.4) for arbitrary groups A and B.

P. Puusemp

Rühmade standardse põimiku loomuliku projektsiooni omadused

Kokkuvõte

Vaadeldakse rühmade standardse põimiku loomulikku projektsiooni, mis on idempotent tema endomorfismipoolrühmas. Selle projektsiooni jaoks kirjeldatakse tema kahepoolsete nullide ja kahepoolsete ühikute poolrühmad.

№ 671

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 519.4

П. Пуусемп

О ПОЛУГРУППЕ ЭНДОМОРФИЗМОВ СТАНДАРТНОГО СПЛЕТЕНИЯ ГРУПП

I. Введение

В работе [1] мы описали две подполугруппы полугруппы всех эндоморфизмов стандартного сплетения $G = A \mathrm{Wr} B$ групп A и B. В настоящей работе продолжаем исследование связи между сплетением $G = A \mathrm{Wr} B$ и его полугруппой всех эндоморфизмов $\mathrm{End}\, G$. При этом мы предполагаем, что группа B конечна и A – конечная циклическая группа порядка p^n (p – произвольное простое число, n – натуральное число). Основным результатом настоящей работы является следующая теорема.

Т е о р е м а. Пусть B – неединичная конечная группа, A – циклическая группа порядка p^n (p – простое число) и $\varkappa$ – проекция сплетения $G = A \mathrm{Wr} B = A^B \lambda B$ на подгруппу B. Если группа $D_G(\varkappa) = \{\gamma \in \mathrm{Aut}\, G \mid \gamma\varkappa = \varkappa\gamma = \varkappa\}$ разрешима и полугруппа всех эндоморфизмов сплетения $A \mathrm{Wr} B$ изоморфна полугруппе всех эндоморфизмов некоторой другой группы G^*, то группа G^* разлагается в полупрямое произведение $G^* = \mathrm{Ker}\,\varkappa^* \lambda \mathrm{Im}\,\varkappa^*$, где $\mathrm{Ker}\,\varkappa^* \cong A^B$, $\mathrm{End}(\mathrm{Im}\,\varkappa^*) \cong \mathrm{End}\, B$ (изоморфизм полугрупп) и $\varkappa^*$ – образ проекции $\varkappa$ при изоморфизме $\mathrm{End}\, G \cong \mathrm{End}\, G^*$.

При доказательстве теоремы будем пользоваться результатами работы [1]. Кроме того, будем придерживаться следующих обозначений: $\mathrm{End}\, G$ – полугруппа всех эндоморфизмов группы G; $\mathrm{Aut}\, G$ – группа всех автоморфизмов группы G; $J(G)$ – множество всех идемпотентов полугруппы $\mathrm{End}\, G$; C_κ – циклическая группа порядка κ, $\mathrm{Hom}(A,B)$ – множество всех гомоморфизмов из группы A в группу B; $g^\wedge$ – внутренний автоморфизм, порожденный элементом g; $K^\wedge = \{g^\wedge \mid g \in K\}$; $\langle g \rangle$ – подгруппа, порожденная элементом g; $J_G(\varkappa) = \{\gamma \in \mathrm{End}\, G \mid \gamma\varkappa = \varkappa\gamma = 0\}$; $K_G(\varkappa) = \{\gamma \in \mathrm{End}\, G \mid \gamma\varkappa = \varkappa\gamma = \gamma\}$; $P_G(\varkappa) = \{\gamma \in \mathrm{End}\, G \mid$

$yx = xy = x\}$; $D_G(x) = \{y \in AutG \mid yx = xy = x\}$; $V_G(x) = \{y \in AutG \mid yx = x\}$.

2. Свойства подугруппы $End(C_{p^n} WrB)$

Всюду в этом пункте предполагаем, что $A = C_{p^n}$, B – неединичная конечная группа и G – сплетение групп A и B (p – фиксированное простое число):

$$G = AWrB = A^B \lambda B. \qquad (2.1)$$

Обозначим через $\varkappa$ проекцию группы G на подгруппу B, т.е. $Im\varkappa = B$ и $Ker\varkappa = A^B$.

Согласно лемме 2.6 из [1] (взять там $b = 1$)

$$G = ([B, A^B] \lambda B) \lambda A_1, \qquad (2.2)$$

где $A_1 = \{f \in A^B \mid f(b) = 1$ при $b \neq 1\}$ и $[B, A^B]$ – взаимный коммутант групп B и A^B. Обозначим через γ идемпотент полугруппы $EndG$, соответствующий полупрямому разложению (2.2), т.е. $Im\gamma = A_1$ и $Ker\gamma = [B, A^B] \lambda B$.

При сделанных предположениях справедливы следующие свойства.

Свойство 1. $K_G(\gamma) \cong End C_{p^n}$.

По лемме 1.6 из [2] $K_G(\gamma) \cong End(Im\gamma)$. Но $Im\gamma = A_1 \cong A = C_{p^n}$. Поэтому $K_G(\gamma) \cong End C_{p^n}$.

Свойство 2. $x\gamma = \gamma x = 0$.

Действительно, по построению $Im\gamma = A_1 \subset Ker\varkappa$ и $Im\varkappa = B \subset Ker\gamma$, т.е. $x\gamma = \gamma x = 0$.

Свойство 3. $|\rfloor_G(x)| = p^{n \cdot |B|}$.

Доказательство. Группа A^B изоморфна прямому произведению $|B|$ экземпляров группы C_{p^n}. Поэтому $|Hom(A, A^B)| = |A^B| = p^{n \cdot |B|}$. В силу следствия 3.2 из [1] $|\rfloor_G(x)| = p^{n \cdot |B|}$. Свойство доказано.

Свойство 4. Если $z \in \rfloor_G(x)$, то $\gamma z = z$.

Доказательство. Пусть $z \in \rfloor_G(x)$ и g – произвольный элемент группы G. По равенству (3.1) из [1]

$\mathrm{Ker}\gamma \subset \mathrm{Ker}\varepsilon$. В силу идемпотентности эндоморфизма γ имеем $g^{-1}\cdot(g\gamma) \in \mathrm{Ker}\gamma \in \mathrm{Ker}\varepsilon$ и $(g^{-1}\cdot(g\gamma))\varepsilon = 1$, т.е. $g(\gamma\varepsilon) = g\varepsilon$. Следовательно, $\gamma\varepsilon = \varepsilon$. Свойство доказано.

С в о й с т в о 5. Если $u, v \in \mathrm{End}G$ и $xu = xv$, $\gamma u = \gamma v$, то $u = v$.

Действительно, в таком случае u и v действуют одинаково на подгруппах $B = \langle x \rangle$ и $A_1 = \langle \gamma \rangle$, которые порождают всю группу G.

С в о й с т в о 6. Если $\varepsilon \in \mathrm{End}G$ и $\gamma\varepsilon = 0$, то $\varepsilon = x\varepsilon$.

Д о к а з а т е л ь с т в о. Пусть $\varepsilon \in \mathrm{End}G$ и $\gamma\varepsilon = 0$. Тогда $A_1 = \langle \gamma \rangle \subset \mathrm{Ker}\varepsilon \cap \mathrm{Ker}x$ и $g\varepsilon = g(x\varepsilon) = 1$ для каждого $g \in A_1$. С другой стороны, если $g \in B = \langle x \rangle$, то $gx = g$ и $g\varepsilon = g(x\varepsilon)$. Поэтому ε и $x\varepsilon$ действуют одинаково на подгруппах A_1 и B группы G. Следовательно, $\varepsilon = x\varepsilon$. Свойство доказано.

Хактоном [3] найдена группа всех автоморфизмов сплетения $A \mathrm{Wr} B$, причем случай $B = C_2$, $A = C_2$ или A – диэдральная группа порядка $4m+2$, исключается. Чтобы воспользоваться результатами Хактона, при доказательстве свойств 7–12 предположим, что не имеет места случай $A = B = C_2$. Но непосредственные вычисления показывают, что свойства 7–12 верны также при $A = B = C_2$.

Следуя [2], имеем:

$$\mathrm{Aut}G = K \cdot I \cdot B^*, \tag{2.3}$$

где

$$I = (A^B)^{\wedge} = \{ f^{\wedge} \mid f \in A^B \},$$

$$K = \{ \varepsilon \in \mathrm{Aut}\, G \mid b\varepsilon = b \text{ для каждого } b \in B \},$$

$$B^* \cong \mathrm{Aut}\, B. \tag{2.4}$$

При этом, если элементу $\varepsilon \in \mathrm{Aut}B$ соответствует при изоморфизме (2.4) элемент $\varepsilon^* \in B^*$, то

$$(bf)\varepsilon^* = (b\varepsilon)(f\varepsilon^*), \tag{2.5}$$

где $b \in B$, $f \in A^B$. Там же отмечено, что I – нормальный делитель группы $\mathrm{Aut}\, G$.

Исходя из этих замечаний, определим строение подгруппы

$$V_G(x) = \{ \varepsilon \in \mathrm{Aut}\, G \mid \varepsilon x = x \}$$

группы $\mathrm{Aut}G$. Ясно, что группа $V_G(x)$ состоит из таких

$z \in Aut\,G$, для которых $g^{-1}\cdot(gz) \in Ker\,x$ при каждом $g \in G$. Так как подгруппа A^B характеристична в G (см. [4], теорема 9.12) и $Ker\,x = A^B$, то при $g \in Ker\,x$ всегда $g^{-1}\cdot(gz) \in Ker\,x$ и, следовательно,

$$V_G(x) = \{z \in Aut\,G \mid b^{-1}\cdot(bz) \in A^B \text{ для каждого } b \in B\}. \qquad (2.6$$

Поэтому ясно, что $I, K \subset V_G(x)$ и

$$KI \subset V_G(x). \qquad (2.7)$$

С другой стороны, в силу равенств (2.5) и (2.6)

$$V_G(x) \cap B^* = \langle 1 \rangle, \qquad (2.8)$$

ибо подгруппа A^B характеристична в G. Теперь из формул (2.3) (2.7) и (2.8) следует $V_G(x) = KI$. Очевидно, что $K \cap I = \langle 1 \rangle$. Следовательно, $V_G(x) = I \lambda K$. Поскольку подгруппа

$$D_G(x) = \{z \in Aut\,G \mid zx = xz = x\} = \{z \in V_G(x) \mid xz = x\}$$

группы $V_G(x)$ состоит из таких элементов $z \in V_G(x)$, которые действуют тождественно на $Im\,x = B$, то $K = D_G(x)$ и

$$V_G(x) = I \lambda K = (A^B)^{\wedge} \lambda D_G(x). \qquad (2.9)$$

Свойство 7. $V_G(x) \cap D_G(y) \cong C_{p^n} \times \ldots \times C_{p^n}$ ($|B| - 1$ раз).

Доказательство. Убедимся сначала, что

$$V_G(x) \cap D_G(y) = (A^B)^{\wedge}. \qquad (2.10)$$

Для этого достаточно доказать включение

$$(A^B)^{\wedge} \subset D_G(y) \qquad (2.11)$$

и равенство

$$D_G(x) \cap D_G(y) = \langle 1 \rangle \qquad (2.12)$$

(см. равенство (2.9)). Равенство (2.12) вытекает сразу из свойства 5. Для доказательства включения (2.11) предположим, что $f^{\wedge} \in (A^B)^{\wedge}$. Тогда $yf^{\wedge} = y$, ибо $A_1 = Im\,y \subset A^B$ и группа A^B коммутативна. С другой стороны,

$$b(f^{\wedge}y) = (b[b,f])y = 1 = by, \quad b \in B;$$

$$g(f^{\wedge}y) = (gf^{\wedge})y = gy, \quad g \in A^B,$$

т.е. $f^{\wedge}y = y$. Поэтому $f^{\wedge} \in D_G(y)$, т.е. справедливы включение (2.11) и равенство (2.10).

Остается определить строение группы $(A^B)^{\wedge}$. По следствию 3.4 из [4] центр $Z(G)$ группы G имеет вид $Z(G) = \langle f \rangle \subset A^B$, где $f(b) = a$ для каждого $b \in B$ и a – образующий элемент группы

$A = C_{p^n}$. Поэтому $Z(G) \cong C_{p^n}$. Так как группа A^B изоморфна прямому произведению $|B|$ экземпляров группы C_{p^n} и $Z(G)$ выделяется прямым сомножителем в группе A^B (см. [5], лемма I5.I), то

$$(A^B)^\wedge \cong A^B \cdot \langle f \rangle / \langle f \rangle \cong C_{p^n} \times \ldots \times C_{p^n} (|B| - 1 \text{ раз}).$$

Свойство доказано.

Из равенств (2.9) и (2.I0) следует

С в о й с т в о 8. $V_G(x) = (V_G(x) \cap D_G(y)) \lambda D_G(x)$.

С в о й с т в о 9. Если $u \in \mathfrak{J}_G(x)$, то $uz = u$ для каждого $z \in V_G(x) \cap D_G(y)$.

Д о к а з а т е л ь с т в о. Предположим, что $u \in \mathfrak{J}_G(x)$ и $z \in V_G(x) \cap D_G(u)$. Согласно равенству (3.I) из [I] $\mathrm{Im} u \subset A^B$. Ввиду равенства (2.I0) $z = f^\wedge$ для некоторого $f \in A^B$. Поэтому z действует тождественно на $\mathrm{Im} u$, т.е. $uz = u$. Свойство доказано.

С в о й с т в о I0. $x \cdot V_G(x) = \{z \in \mathfrak{J}(G) \mid xz = z, zx = x\}$.

Д о к а з а т е л ь с т в о. Обозначим ради краткости

$$[x] = \{z \in \mathfrak{J}(G) \mid xz = z, \ zx = x\}.$$

Из равенства (2.9) вытекает равенство $x \cdot V_G(x) = x \cdot (A^B)^\wedge$. Нам надо доказать равенство

$$x \cdot (A^B)^\wedge = [x]. \qquad (2.I3)$$

Из включения $A^B \subset \mathrm{Ker} x$ вытекает включение $x \cdot (A^B)^\wedge \subset [x]$. Для получения обратного включения предположим, что $z \in [x]$. Тогда $G = \mathrm{Ker} z \lambda \mathrm{Im} z$ (см. [2], лемма I.I) и $\mathrm{Ker} x = \mathrm{Ker} z = A^B$ (см. [6], лемма 5). Но каждые два полупрямого дополнения подгруппы $\mathrm{Ker} x = A^B$ сопряжены в группе G (см. [4], теорема I0.I). Поэтому существует такой $f \in A^B$, что $\mathrm{Im} z = (\mathrm{Im} x) f^\wedge$. Отсюда вытекает равенство $z = x f^\wedge$, т.е. $z \in x \cdot (A^B)^\wedge$. Следовательно, $[x] \subset x \cdot (A^B)^\wedge$ и справедливо равенство (2.I3). Свойство доказано.

С в о й с т в о II. В полугруппе $P_G(x)$ существует ровно p^n элементов u, удовлетворяющих равенству

$$\{z \in \mathrm{End}\, G \mid zx = zy = 0\} \cdot u = \{0\}. \qquad (2.I4)$$

Д о к а з а т е л ь с т в о. Равенство (2.I4) равносильно условию: если $\mathrm{Im} z \subset \mathrm{Ker} x \cap \mathrm{Ker} y$, то $(\mathrm{Im} z) u = \langle 1 \rangle$. Но $\mathrm{Ker} x \cap \mathrm{Ker} y = [B, A^B]$ и для каждого элемента $g \in [B, A^B] \subset A^B$ существует такой $z \in \mathrm{End}\, G$, что $g \in \mathrm{Im} z \subset [B, A^B]$. Действительно, если $g \in [B, A^B]$, то порядок p-элемента g не превышает чис-

ло p^n и в качестве нужного эндоморфизма z можно взять произведение $\gamma\nu$, где ν сопоставляет образующему группы $\mathrm{Im}\gamma = A_1 \cong C_{p^n}$ элемент g. Поэтому условие (2.I4) равносильно равенству

$$[B, A^B]u = [Bu, A^B u] = \langle 1 \rangle. \qquad (2.I5)$$

Определим число таких $u \in P_G(x)$, которые удовлетворяют равенству (2.I5). По лемме 4.I из [I] каждый $u \in P_G(x)$ действует тождественно на B и $A^B u \subset A^B$. Поэтому надо определить число таких $u \in P_G(x)$, для которых $A^B u$ содержится в подгруппе

$$A^B \cap C_G(B) = \{f \in A^B \mid f(b) = f(1) \text{ при каждом } b \in B\}.$$

(см. [4], лемма 3.2). Если $u = \{U_c\}_{c \in B} \in P_G(x)$, то $A^B u \subset A^B \cap C_G(B)$ тогда и только тогда, когда все эндоморфизмы из семейства $\{U_c\}_{c \in B}$ совпадают. Следовательно, число эндоморфизмов $u \in P_G(x)$, удовлетворяющих включению $A^B u \subset A^B \cap C_G(B)$ (а также равенству (2.I4)), равняется $|\mathrm{End}\, A| = |\mathrm{End}\, C_{p^n}| = p^n$. Свойство доказано.

С в о й с т в о I2. Существует такой $u \in P_G(x)$, который удовлетворяет равенству (2.I4) и для которого всегда из $\nu \in K_G(\gamma)$, $\nu u = 0$ следует $\nu = 0$.

Д о к а з а т е л ь с т в о. Пусть a – образующий группы $A = C_{p^n}$ и $f \in A^B$, причем $f(b) = a$ для каждого $b \in B$. Тогда $a\varphi_i^{-1} = g$ является образующим группы A_1 и $f \in A^B \cap C_G(B)$ (см. [4], лемма 3.2). Порядки элементов f и g равны числу p^n. В силу равенства (2.2) ясно, что отображение u, определенное равенством

$$(g^i b h)u = f^i b, \text{ где } b \in B,\ h \in [B, A^B],$$

является эндоморфизмом группы G. Так как u действует на B тождественно и $A^B u \subset A^B$, то по лемме 4.I из [I] $u \in P_G(x)$. При этом $\mathrm{Ker}\, u = [B, A^B]$, т.е. u удовлетворяет равенству (2.I5) и равносильному с (2.I5) равенству (2.I4).

Предположим теперь, что $\nu \in K_G(\gamma)$ и $\nu u = 0$. Тогда $\mathrm{Im}\nu \subset \mathrm{Ker}\, u = [B, A^B]$, $\mathrm{Im}\nu \subset \mathrm{Im}\gamma = A_1$ (см. [2], лемма I.6) и $\mathrm{Im}\nu \subset [B, A^B] \cap A_1 = \langle 1 \rangle$, т.е. $\nu = 0$. Свойство доказано.

3. Группа, удовлетворяющая свойствам I–I2

В настоящем пункте выясним строение группы, удовлетворяющей свойствам I–I2 предыдущего пункта.

Итак, фиксируем конечную группу G и идемпотенты x и y ее подугруппы эндоморфизмов. Обозначим $B=Jmx$ и предположим, что для x,y и B справедливы свойства I-I2 предыдущего пункта. Тогда

$$G = Kerx \lambda Jmx = Kerx \lambda B = Kery \lambda Jmy \qquad (3.1)$$

(см. [2], лемма I.I). Согласно свойству 2 $yx=xy=0$. Поэтому

$$G = (N \lambda Jmx) \lambda Jmy = (N \lambda Jmy) \lambda Jmx , \qquad (3.2)$$

где $N = Kerx \cap Kery$,

$$Kerx = N \lambda Jmy, \quad Kery = N \lambda Jmx \qquad (3.3)$$

(см. [5], стр. 78). В силу свойства I группа Jmy является циклической группой порядка p^n:

$$Jmy = \langle a \rangle \cong C_{p^n} \qquad (3.4)$$

Обозначим

$$T = \{ g \in Kerx \mid g - p\text{-элемент порядка} \leq p^n \}.$$

Ясно, что $Jmy \subset T$. В этих предположениях справедливы леммы 3.I-3.6.

Л е м м а 3.I. Подмножество T является коммутативной подгруппой группы $Kerx$.

Д о к а з а т е л ь с т в о. Пусть g,f - произвольные элементы из T. Тогда существуют эндоморфизмы u и v группы G, образами которых являются соответственно $\langle g \rangle$ и $\langle f \rangle$:

$$(a^i h)u = g^i, (a^i h)v = f^i; \; i \in Z_{p^n}, \; h \in Kery.$$

Для таких u и v имеем $ux=xu=0$ и $vx=xv=0$, т.е. $u, v \in \lambda_G(x)$. Так как $a^\wedge \in V_G(x) \cap D_G(y)$ (см. [7], лемма 6), то по свойству 9 $ua^\wedge = u$, т.е. $a(ua^\wedge) = au$ и $a^{-1}ga = g$, $ga = ag$. Поэтому $g^\wedge \in V_G(x) \cap D_G(y)$ (см. [7]. лемма 6) и снова ввиду свойства 9 $vg^\wedge = v$, т.е. $a(vg^\wedge) = av$ и $fg^\wedge = f$, $fg = gf$. Следовательно, любые два элемента множества T коммутируют между собой. Отсюда сразу следует утверждение леммы. Лемма доказана.

Попутно мы доказали также включение

$$T^\wedge \subset V_G(x) \cap D_G(y). \qquad (3.5)$$

Л е м м а 3.2 $|T| = p^{n \cdot |B|}$.

Доказательство. Пусть $z \in \rfloor_G(x)$, т.е. $zx = xz = 0$. Тогда $\mathrm{Im}\, z \subset \mathrm{Ker}\, x$, $(\mathrm{Im}\, y)z \subset \mathrm{Ker}\, x$ и ввиду свойства 4 $\mathrm{Ker}\, y \subset \mathrm{Ker}\, z$. Поэтому в силу определения множества T и условия (3.4) имеем

$$(\mathrm{Ker}\, y)z = \langle 1 \rangle, (\mathrm{Im}\, y)z \subset T. \tag{3.6}$$

Наоборот, каждый эндоморфизм z группы G, удовлетворяющий условиям (3.6), принадлежит множеству $\rfloor_G(x)$. Из условий (3.6) и разложений (3.I) следует теперь, что между множествами $\rfloor_G(x)$ и $\mathrm{Hom}(\mathrm{Im}\, x, T)$ существует взаимно однозначное соответствие. В силу условия (3.4) и определения множества T будет $|\mathrm{Hom}(\mathrm{Im}\, x, T)| = |T|$. Следовательно, $|\rfloor_G(x)| = |T|$ и по свойству 3 $|T| = p^{n \cdot |B|}$. Лемма доказана.

Лемма 3.3. $V_G(x) \cap D_G(y) = T^{\wedge}$.

Доказательство. По свойству 7 $|V_G(x) \cap D_G(y)| = p^{n(|B|-1)}$. Поэтому из леммы 3.2 и включения (3.5) следует неравенство

$$|T \cap Z(G)| \geq p^n. \tag{3.7}$$

Покажем справедливость противоположного неравенства. Для этого возьмем произвольный элемент $g \in T \cap Z(G)$. Тогда

$$\langle g, \mathrm{Im}\, x \rangle = \langle g \rangle \times \mathrm{Im}\, x \tag{3.8}$$

и можно построить эндоморфизм u_g группы G следующим образом. Согласно равенствам (3.2) имеем

$$G/N = (\mathrm{Im}\, x N/N) \times (\mathrm{Im}\, y \cdot N/N). \tag{3.9}$$

причем прямые множители в разложении (3.9) изоморфны соответственно группам $\mathrm{Im}\, x$ и $\mathrm{Im}\, y = \langle a \rangle$. Так как $|g| \leq |a|$, то в силу (3.8) и (3.9) существует такой гомоморфизм $\nu_g : G/N \to \langle g \rangle \times \mathrm{Im}\, x$, что

$$(bN)\nu_g = b, \quad b \in \mathrm{Im}\, x; \quad (aN)\nu_g = g.$$

Определим $u_g \in \mathrm{End}\, G$ как произведение естественного гомоморфизма $G \to G/N$ на гомоморфизм ν_g. Ясно, что соответствие $g \to u_g$ является инъективным. По построению имеем $u_g \in P_G(x)$ и $N \subset \mathrm{Ker}\, u_g$. Поэтому

$$\{z \in \mathrm{End}\, G \mid zx = zy = 0\} \cdot u_g = \{0\}$$

и в силу свойства II

$$|T \cap Z(G)| \leq p^n. \tag{3.I0}$$

Из неравенств (3,7) и (3.I0) вытекает равенство

$$|T \cap Z(G)| = p^n. \tag{3.II}$$

Согласно лемме 3.2 это означает, что $|T^{\wedge}| = p^{n(|B|-1)} = |V_G(x) \cap D_G(y)|$. Отсюда следует, ввиду включения (3.5), утверждение леммы. Лемма доказана.

Лемма 3.4. $T \cong C_{p^n} \times \ldots \times C_{p^n}$ ($|B|$ раз).

Доказательство. Сохраняем обозначения леммы 3.3. Ввиду равенства (3.II) эндоморфизмы $u \in P_G(x)$, удовлетворяющие равенству (2.I4), исчерпываются эндоморфизмами $u = u_g$, $g \in T \cap Z(G)$. По свойству I2 существует такой $g \in T \cap Z(G)$, что всегда из равенства $vu_g = 0$, где $v \in K_G(y)$, следует $v = 0$. Фиксируем элемент g таким образом.

Порядок элемента g равен p^n, ибо в противном случае существовал бы ненулевой $v \in K_G(y)$, для которого $vu_g = 0$:

$$(\mathrm{Ker}_y)v = \langle 1 \rangle, \quad av = a^{p^{n-1}}.$$

Следовательно, $T \cap Z(G) = \langle g \rangle = C_{p^n}$. По лемме I5.I из [5] группа $\langle g \rangle$ выделяется в группе T в качестве прямого множителя: $T = T_0 \times \langle g \rangle$. Поэтому $T^{\wedge} \cong T/(T \cap Z(G)) = T/\langle g \rangle \cong T_0$. По свойству 7 и лемме 3.3 $T_0 \cong C_{p^n} \times \ldots \times C_{p^n}$ ($|B|-1$ раз). Следовательно, $T \cong C_{p^n} \times \ldots \times C_{p^n}$ ($|B|$ раз). Лемма доказана.

Лемма 3.5. Подгруппа $T\lambda \mathrm{Im}\, x$ является нормальным делителем в группе G.

Доказательство. По определению группы T ясно, что T – нормальный делитель группы G. Поэтому $\langle T, \mathrm{Im}\, x \rangle = T\lambda \mathrm{Im}\, x$. Для доказательства леммы надо показать еще включение $(\mathrm{Im}\, x)g^{\wedge} \subset T\lambda \mathrm{Im}\, x$ для каждого $g \in \mathrm{Ker}\, x$.

Пусть $g \in \mathrm{Ker}\, x$. Тогда $(xg^{\wedge})^2 = xg^{\wedge}$ и $(xg^{\wedge})x = x, x(xg^{\wedge}) = xg^{\wedge}$. По свойству I0 $xg^{\wedge} = xz$ для некоторого $z \in V_G(x)$. В силу свойства 8 $xg^{\wedge} = xz = xu$ при некотором $u \in V_G(x) \cap D_G(y)$. Ввиду леммы 3.3 $u = h^{\wedge}$ для подходящего $h \in T$. Следовательно,

$$(\mathrm{Im}\, x)g^{\wedge} = G(xg^{\wedge}) = G(xu) = G(xh^{\wedge}) = (\mathrm{Im}\, x)h^{\wedge} \subset T\lambda \mathrm{Im}\, x.$$

Лемма доказана.

Лемма 3.6. Если группа $D_G(x)$ разрешима, то $G = T\lambda B$, $T = \mathrm{Ker}\, x$, $B = \mathrm{Im}\, x$.

Доказательство. Предположим, что группа $D_G(x)$ разрешима. По свойству 8 группа $V_G(x)$ также разрешима, ибо ранее была показана коммутативность группы $V_G(x) \cap D_G(y)$. Так как $(\mathrm{Ker}\, x)^{\wedge} \subset V_G(x)$, то группа $\mathrm{Ker}\, x$ разрешима. Поэтому будет разрешимой группа $G/(T\lambda \mathrm{Im}\, x)$.

Предположим, что $T\lambda]mx \neq G$. В силу разрешимости группы $G/(T\lambda]mx)$ существует такой нормальный делитель M группы G, что $T\lambda]mx \subset M$ и $G/M \cong C_q$, где q – некоторое простое число. Тогда в группе G существует элемент g порядка q и можно рассматривать ненулевой эндоморфизм z группы G, являющийся произведением естественного гомоморфизма $G \to G/M$ и изоморфизма $G/M \cong \langle g \rangle$. Такой z удовлетворяет равенствам $xz = 0 = x0$ и $yz = 0 = y0$. По свойству 5 $z = 0$. Получено противоречие с тем, что $z \neq 0$. Следовательно, $G = T\lambda]mx$ и $T = Kerx$. Лемма доказана.

4. Доказательство теоремы

Докажем теперь теорему, сформулированную во введении. Пусть выполнены предположения теоремы. Так как группа G конечна, то группа G^* конечна (см. [8], теорема 2). Обозначим через z^* образ элемента $z \in End\,G$ при изоморфизме $End\,G \cong End\,G^*$. Группа G удовлетворяет свойствам I–I2 пункта 2. Свойства I–I2 сформулированы так, что они сохраняются при изоморфизме полугрупп эндоморфизмов групп. Поэтому идемпотенты x^* и y^* полугруппы $End\,G^*$ удовлетворяют свойствам, аналогичным свойствам I–I2 (G надо заменить на G^*, а каждый $z \in End\,G$ на $z^* \in End\,G^*$) и к группе G^* применимы леммы 3.I–3.6. Следовательно, $G^* = Kerx^* \lambda]mx^*$, где $Kerx^* \cong C_{p^n} \times \ldots \times C_{p^n}$ ($|B|$ раз). Так как $A^B \cong C_{p^n} \times \ldots \times C_{p^n}$ ($|B|$ раз), то $Kerx^* \cong A^B$. С другой стороны, в силу леммы I.6 из [2] имеем

$$End\,(Jmx^*) \cong K_{G^*}(x^*) \cong K_G(x) \cong End\,(Jmx) = End\,B.$$

Теорема доказана.

Литература

I. Пуусемп П. Свойства естественной проекции стандартного сплетения групп. См. наст. сб., с I54.

2. Пуусемп П. Идемпотенты полугрупп эндоморфизмов групп // Уч. зап. Тартуск. ун-та. I975. 366. С. 76–I04.

3. Houghton C.H. On the automorphism groups of certain wreath products // Publ. Math. 1962. 9, N 3-4. P. 307-313.

4. Neumann P.M. On the structure of standard wreath products of groups // Math. Z. 1964. 84. P. 343–373.

5. Ф у к с Л. Бесконечные абелевы группы. Том I. М.: Мир, 1974.

6. П у у с е м п П. Подгруппа эндоморфизмов полупрямого произведения двух циклических р-групп // Уч. зап. Тартуск. ун-та. 1976. 390. С. 104-133.

7. П у у с е м п П. Подгруппы эндоморфизмов обобщенных групп кватернионов // Уч. зап. Тартуск. ун-та. 1976. 390. С. 84-103.

8. A l p e r i n J.L. Groups with finitely many automorphisms // Pacif. J. Math. 1962. 12, N 1. P. 1-5.

P. Puusemp

On Semigroup of Endomorphisms of Standard Wreath Product

Abstract

Let AWrB be the standard wreath product of groups A and B. The following theorem is proved.

T h e o r e m. Let B be a nontrivial finite group, A a cyclic group of order p^n (p is a prime) and x the natural projection of the $G = AWrB = A^B \lambda B$ onto B. Suppose that the group $D_G = \{ y \in AutG \mid yx = xy = x \}$ is solvable and G* is such a group that the semigroups EndG and EndG* are isomorphic. Then G* is a semidirect product of A* by B* where the group A* is isomorphic to A^B and the semigroups EndB and EndB* are isomorphic.

P. Puusemp

Rühmade standardse põimiku endomorfismipoolrühmast

Kokkuvõte

Käsitletakse kahe lõpliku rühma A ja B standardset põimikut G = AWrB. Uuritakse sellise rühma ehitust, mille kõigi endomorfismide poolrühm on isomorfne rühma G kõigi endomorfismide poolrühmaga.

Содержание

Цена 1 руб. 80 коп.