ISSN 0136-3549
0320-3409

TALLINNA
POLÜTEHNILISE INSTITUUDI
TOIMETISED

524

ТРУДЫ ТАЛЛИНСКОГО
ПОЛИТЕХНИЧЕСКОГО
ИНСТИТУТА

TPI '82

ОБРАБОТКА ДАННЫХ,
ПОСТРОЕНИЕ ТРАНСЛЯТОРОВ,
ВОПРОСЫ ПРОГРАММИРОВАНИЯ

524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

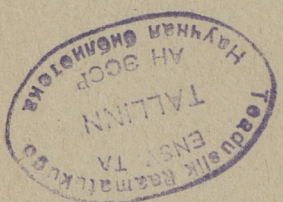
ТПИ '82

УДК 681.3

ОБРАБОТКА ДАННЫХ, ПОСТРОЕНИЕ ТРАНСЛЯТОРОВ, ВОПРОСЫ ПРОГРАММИРОВАНИЯ

Труды экономического факультета XLV

Таллин 1982



Таллинский политехнический институт
Труды ТПИ № 524
ОБРАБОТКА ДАННЫХ, ПОСТРОЕНИЕ ТРАНСЛЯТОРОВ, ВОПРОСЫ ПРОГРАММИРОВАНИЯ
Труды экономического факультета X LV
Отв. редактор И. Амитан. Технический редактор В. Ранник
Сборник утвержден коллегией Трудов ТПИ 07.01.82
Подписано к печати 14.05.82. Формат 60x90/16
Печ. л. 8,25 + 0,5 приложение. Уч.-изд. л. 7,43. Тираж 400
МВ-03789. Ротапринт ТПИ, Таллин, ул. Коскла, 2/9
Заказ № 307. Ц е н а 1 руб. 10 коп.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.06

Т.И. Микли, Я.Я. Тепанди

АЛГОРИТМ ОТБОРА ДАННЫХ ИЗ СЕТЕВОЙ БАЗЫ ДАННЫХ

Для разработки проблемно-ориентированных пакетов программ, основанных на системах управления базами данных (СУБД), необходимы средства простого доступа к базам данных (БД). Ниже приводится алгоритм отбора данных из сетевой БД [1] по запросу пользователя. В запросе задается некоторое множество S типов записей сужения поиска [2]. Каждому ReS ставится в соответствие сужение типа записей R, представляющее собой некоторое допустимое значение фиксированного элемента сужения поиска из R.

В общем случае, выполнение запроса состоит из следующих этапов.

1. Чтение запроса, проверка его синтаксической корректности.
2. Преобразование запроса во внутреннюю форму.
3. Поиск данных в БД и выдача их для обработки.
4. Обработка данных.

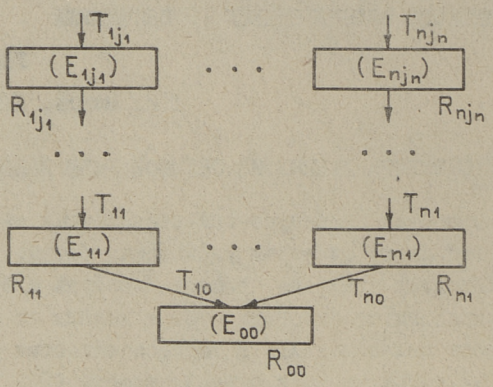
Приводимый алгоритм выполняет третий этап.

1. Данные и операции

Ниже приводится описание структур данных БД и запроса во внутренней форме, а также операций, допустимых на этих структурах.

1.1. Структура БД. Диаграмма Бахмана схемы БД, допустимой для приводимого алгоритма, представлена на фиг. 1. Предполагается, что способ локализации типов записей не объявлен как "прямой". Если он объявлен как "вычислимый", то элемент сужения поиска этого типа записей должен быть элементом, по которому вычисляется размещение записи. Остальных ограничений на характеристики типов записей и наборов не

вводится. Естественно, что операторы поиска данных должны соответствовать способу их локализации.



Фиг. 1. Допустимая схема БД для алгоритма А. R_{ij} – типы записей; R_{00} – искомый тип записей; E_{00}, E_{ij} – элементы сужения поиска; $T_{10}, \ldots, T_{no}, T_{ij}$ – типы наборов ($n \geq 1$; $i=1,\ldots,n$; $j_i \geq 1$; $j=1,\ldots,j_i$).

I.2. Операции над данными в БД. Язык манипулирования данными сетевой СУБД для включающего языка Кобол (ЯМД (К)) был описан в [I, 5]. Имеется версия для включающего языка PL/I [6]. Авторы имеют опыт для включающего языка Фортран в системе DDMF [7]. Чтобы не привязываться к какому-либо конкретному включающему языку, операции над БД в алгоритме не записываются в конкретном языковом оформлении. Ниже приводится представление основных операторов, используемых в алгоритме, через операторы ЯМД (К), используя русскую версию из [5 , с. 320-326].

I. Найти запись-владелец (В-запись) в наборе – форма 4 оператора НАЙТИ.

2. Найти первую (следующую) запись-член (Ч-запись) в наборе Т, элемент которой имеет значение С – могут использоваться: формы 6, 7 оператора НАЙТИ; форма 3 оператора НАЙТИ в цикле; форма 5 оператора НАЙТИ (в последнем случае Ч-тип записей набора Т должен быть объявлен как "вычислимый").

3. Извлечь запись или элемент – оператор ПОЛУЧИТЬ.

I.3. Запрос. Далее предполагается, что запрос пользователя задается в виде пар (имя типа записей, значение) и что с именем типа записей сопоставляется пара значений i, j соответственно фиг. I. Поэтому исходным множеством для описания внутренней формы запроса (далее просто: запроса) является множество M троек $\langle i, j, v \rangle$, где v – сужение типа записей R_{ij}, причем каждой паре значений i, j соответствует в точности одно (введенное последним) значение v. Выделяется неопределенное значение НО сужения типа записей. Если пользователем не задан ни один тип записей сужения поиска, то в M искусственно вводится одна тройка <I, I, НО>.

Введем обозначения:

$$I = \{ i \mid \langle i, j, v \rangle \in M \}; \quad d = |I|;$$

$$J_i = \{ j \mid \langle i, j, v \rangle \in M \}, \quad e_i = |J_i|,$$

$$m_i = \max_{j \in J_i} j, \quad i \in I.$$

Пусть $k_I, \ldots, k_d$ – перестановка чисел $i \in I$ такая, что $e_{k_1} \geqslant e_{k_2} \geqslant \ldots \geqslant e_{k_d}$.

Запрос задается в виде списка X, состоящего из d узлов в последовательности $X[k_1], \ldots, X[k_d]$. Узел $X[u]$ включает элементы:

– НАЧА – указатель на начало списка типов записей ветви u;

– СЛЕДХ – указатель на следующий узел списка X (для последнего узла СЛЕДХ равно ПУСТО).

Список типов записей ветви k_I состоит из последовательности узлов $A[k_1, j_{k_1}], \ldots, A[k_1, 0]$. Каждый список типов записей ветви k_b $(b = 2, \ldots, d)$ состоит из последовательности узлов $A[k_b, 1], \ldots, A[k_b, m_{k_b}]$. Каждый узел $A[u, w]$ включает элементы (P – указатель на $A[u, w]$:

– ПРЕДА – указатель на предыдущий узел списка A (у первого узла списка значения ПРЕДА равно ПУСТО);

– СЛЕДА – указатель на следующий узел списка A (у последнего узла списка значение СЛЕДА равно ПУСТО);

– ТН – имя типа набора (для $u = I$, $TH(P) = T_{uw}$; для $u > I$, $TH(P) = T_{u(w-1)}$;

– ЗЗ – имя элемента сужения поиска типа записей R_{uw}; (если $\omega = 0$, то типа записей R_{00});

– С – сужение типа записей $R_{u\omega}$ (если $\omega = 0$, то типа записей R_{00}). Если сужение типа записей не задано, то значение С равно НО;

– ПП – признак первой или следующей записи в наборе (исходное значение элемента ПП равно 0).

Для работы алгоритма нужны еще указатель РХКІ на начало списка Х и указатель РАКІО на узел А $[K_I, 0]$.

I.4. Операции над данными запроса. Операции над списками излагаются по [3].

2. Алгоритм отбора данных

Внешняя форма изложения алгоритма соответствует [4], переведены только ключевые слова. Если элемент сужения поиска записи r типа R принимает значение из сужения типа записей R, то будем говорить, что запись r удовлетворяет данному запросу.

Алгоритм А. Отбор данных из сетевой БД.

Вход. База данных со схемой фиг. I; запрос в виде списков Х, А и указателей РХКІ, РАКІО.

Выход. Множество записей типа R_{00}, связанных в БД с записями, удовлетворяющими данному запросу.

Метод. Используются переменные РХ – указатель на список Х и РА – указатель на список А. Выполняется следующий алгоритм.

```
      начало
I.    РХ ← РХКІ;
2.    РА ← НАЧА(РХ);
    ' коммент  движение вниз
3.МІ:пока РА ≠ ПУСТО делать
      начало
4.      если ПП(РА) = 0 то
          начало
5.          ПП(РА) ← I;
6.          найти первую Ч-запись ч в наборе ТН(РА), такую,
              что значение элемента ЭЗ(РА) записи ч при
              С(РА) ≠ НО равно С (РА)
```

```
            конец
7.          иначе  найти следующую Ч-запись ч в наборе ТН(РА),
            такую, что значение элемента ЭЗ(РА) записи ч  при
            С(РА) ≠ НО равно С(РА);
8.          если запись ч существует в БД, то РА ← СЛЕДА(РА)
            иначе
              начало
9.              ШП(РА) ← 0;
10.             РА ← ПРЕДА(РА);
                коммент конец ли работы алгоритма
11.             если РА=ПУСТО, то завершение
              конец
        конец
12.   РХ ← СЛЕД(РХ);
      коммент движение вверх
13.   пока РХ ≠ ПУСТО делать
        начало
14.       РА ← НАЧА(РХ);
15.       пока РА ≠ ПУСТО делать
            начало
16.           найти В-запись ч в наборе ТН(РА);
17.           если значение элемента ЭЗ(РА) записи ч  при
                С(РА) ≠ НО равно С(РА)
18.             то  РА ← СЛЕДА(РА)
19.           иначе идти к М2
            конец
20.       РХ ← СЛЕДХ(РХ)
        конец;
      коммент выдача на обработку
21.   извлекать текущую запись типа R00 из БД и выдать  на
      обработку;
22. М2:РХ ← РХК1;
23.    РА ← РАК10;
24.    идти к М1
      конец
```

Литература

1. Столяров Г.К. Обзор предложений Рабочей группы КОДАСИЛ по базам данных. – В кн.: Алгоритмы и организация решения экономических задач. М., Статистика, 1974, вып. 4, с. 48–77.

2. Тепанди Я.Я. О методах трансляции языков с проблемной ориентацией. – В кн.: Всесоюзная конференция по методам трансляции. Тезисы докладов, Новосибирск, 1981, с. 75–77.

3. Кнут Д. Искусство программирования для ЭВМ. т.I. М., Мир, 1976, с. 302–329.

4. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. М., Мир, 1979, с. 47–53.

5. Информационные системы общего назначения. Перевод с англ. под ред. Е.Л. Ющенко. М., Статистика, 1975, с. 281–354.

6. Фридлендер Ф.Л., Савинков В.М. Пакет прикладных программ СУБД НАБОБ. – В кн.: Алгоритмы и организация решения экономических задач, вып. 12, М., Статистика, 1978, с. 25–40.

7. Выханду Л.К., Лучковский Т.Ф., Микли Т.И., Тепанди Я.Я. Система хранения и обработки дискретной информации. – УСиМ, 1981, № 1, с. 99–102.

T. Mikli, J. Tepandi

A Retrieval Algorithm for a Network Data Base

Summary

In this paper an algorithm for implementing a nonprocedural data retrieval query language is presented. The query involves a list of certain elements' values for given record types. A data base subschema consisting of hierarchic paths, which lead to the requested record type, is considered.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.06

Я.Я. Тепанди

БАЗА ДАННЫХ ДЛЯ ЯЗЫКОВ С ПРОБЛЕМНОЙ ОРИЕНТАЦИЕЙ

Работа с пакетом программ предполагает использование входных языков пакета. В [1] была выдвинута идея создания универсальной базы данных алгоритмических языков, которая содержит основную информацию для построения трансляторов с этих языков. Аналогичный подход целесообразен также при разработке систем автоматизации построения пакетов. К этому направлению можно отнести описание методики сопряжения системы управления базами данных (СУБД) и системы построения трансляторов [2]. Ниже делается попытка более подробной характеристики внешней формы и структуры данных, которые могут накапливаться и использоваться при описании входных языков пакетов программ.

1. Данные и их использование

Опыт проектирования баз данных (БД) показывает, что надо учитывать как данные, так и запросы пользователя [3]. Для характеристики данных полезно воспользоваться опытом разработки существующих систем построения пакетов программ и трансляторов. Анализ этих систем (см., например [4-8]) приводит к выводу, что в качестве основного содержания БД можно принять элементы описания языка (ЭОЯ) – стандартные блоки, из которых пользователь составляет описание своего входного языка. Возможны следующие типы ЭОЯ:

- элемент описания абстрактного синтаксиса;
- элемент описания лексики;
- элемент описания синтаксиса;
- элемент описания статической семантики;
- элемент описания динамической семантики.

С точки зрения внешней формы ЭОЯ может представляться в виде:

- фрагмента программы на метаязыке СПТ;

- вставки текстов, написанных на разных языках программирования в метаязык;

- самостоятельных семантических процедур на разных языках программирования;

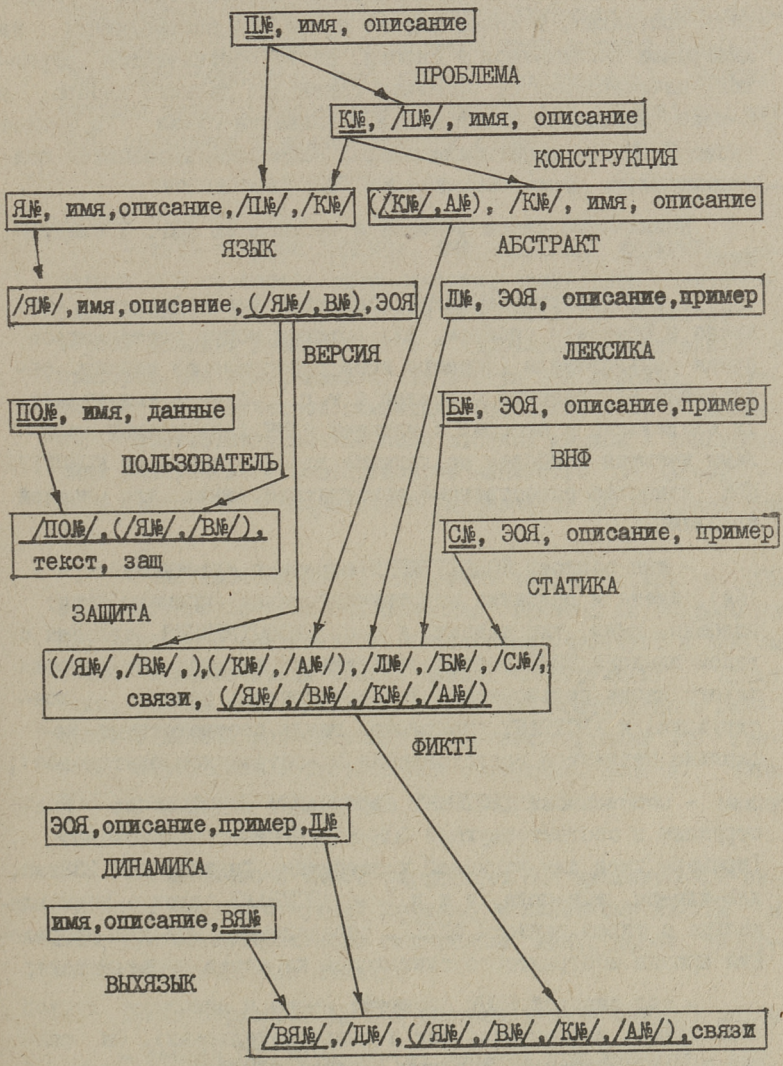
- таблиц внутреннего представления описания входного языка.

Относительно обработки в операционной системе ЭОЯ может иметь вид исходного текста программы, объектного или загрузочного модуля, макрорасширения или файла (таблиц) исходных данных.

Практика программирования на метаязыках показывает сложность их освоения пользователем [7]. С применением ЭОЯ эта сложность может уменьшаться лишь частично, если составление полного описания своего языка из ЭОЯ все же возложено на пользователя. Наличие БД ЭОЯ создает возможность применения нового способа определения входного языка по примерам конструкций языка пользователя, сходного в некотором смысле с языком манипулирования данными реляционной СУБД Q.BE [9]. Суть идеи состоит в том, что с одной стороны, для каждой языковой конструкции в БД предлагается иметь примеры, помогающие пользователю в выборе конструкции. С другой стороны, так как с помощью БД можно зафиксировать семантику конструкции и ее "скелет" в виде абстрактного синтаксиса, то пользователю предлагается возможность оформлять свой язык посредством ввода примеров конструкций на своем языке. Преобладающей формой диалога является "меню" - система предлагает альтернативы, из которых пользователь выбирает подходящую. Таким образом, ему доступны лишь имя и неформальное описание ЭОЯ.

2. Состав и структура базы данных

Структура БД представлена ниже на фиг. I в виде (расширенной) диаграммы Бахмана [IO]. На этой диаграмме имя типа записей пишется под соответствующим прямоугольником. В прямоугольник вписываются имена элементов записей. Элементы могут быть физическими или виртуальными (вычислимыми). Имя виртуального элемента расположено между наклонными черточками. Включение записей в набор определяется


ПЛ№, имя, описание
ПРОБЛЕМА
КЛ№, /ПЛ№/, имя, описание
КОНСТРУКЦИЯ
ЯЛ№, имя,описание,/ПЛ№/,/КЛ№/
(/КЛ№/,АЛ№), /КЛ№/, имя, описание
ЯЗЫК
АБСТРАКТ
/ЯЛ№/,имя,описание,(/ЯЛ№/,ВЛ№),ЭОЯ
ДЛ№, ЭОЯ, описание,пример
ВЕРСИЯ
ЛЕКСИКА
ПОЛ№, имя, данные
БЛ№, ЭОЯ, описание,пример
ПОЛЬЗОВАТЕЛЬ
ВНФ
/ПОЛ№/,(/ЯЛ№/,/ВЛ№/),
текст, защ
СЛ№, ЭОЯ, описание, пример
СТАТИКА
ЗАЩИТА
(/ЯЛ№/,/ВЛ№/,),(/КЛ№/,/АЛ№/),/ДЛ№/,/БЛ№/,/СЛ№/,
связи, (/ЯЛ№/,/ВЛ№/,/КЛ№/,/АЛ№/)
ФИКТ1
ЭОЯ,описание,пример,ДЛ№
ДИНАМИКА
имя,описание,ВЯЛ№
ВЫХЯЗЫК
/ВЯЛ№/,/ДЛ№/,(/ЯЛ№/,/ВЛ№/,/КЛ№/,/АЛ№/),связи
ФИКТ2

Фиг. 1.

элементами связи, значения которых в В- и Ч-записях одного экземпляра набора должны совпадать. Элемент связи может быть составным, в таком случае имена его составляющих на диаграмме заключаются в скобки. Набор обозначается стрелкой, начинающейся от элемента связи для данного набора в В-типе записей и кончающейся у элемента связи в Ч-типе записей. Имена идентификационных элементов выделяются подчеркиванием.

Назначение типов записей и наборов на фиг. I следующие:

- тип записей ПРОБЛЕМА содержит идентификаторы (П№), имена и описания проблем, для которых могут составляться языки пользователя (примеры имен : оформление таблиц, обновление БД, резание металлов и т.д.). Данный тип является владельцем в наборах с членами ЯЗЫК (для каждой проблемы имеются наиболее подходящие для нее языки) и КОНСТРУКЦИЯ (наиболее естественные конструкции языка для данной проблемы);

- тип записей КОНСТРУКЦИЯ содержит идентификаторы (К№), имена и описания конструкций языка (примеры имен: условие, цикл, тип данных, и т.д.). КОНСТРУКЦИЯ связана с типом записей ЯЗЫК (можно ответить на вопрос: в подобии какого языка пользователь захочет увидеть данную конструкцию) и АБСТРАКТ (например, циклу соответствует несколько вариантов представления в абстрактном синтаксисе);

- тип записей АБСТРАКТ определяет модификацию конструкции и соответствующий ЭОЯ абстрактного синтаксиса (примеры имен для языковой конструкции "цикл": для 238вып, для-клм-цк, вып-пока, и т.д.). АБСТРАКТ связи в роли владельца с типом записей ФИКТI, определяющим ЭОЯ для оформления правил абстрактного синтаксиса на языке пользователя;

- тип записей ЯЗЫК содержит имена и описания языков (как общего назначения, так и пользовательских). Он связан в роли владельца с типом записей ВЕРСИЯ;

- тип записей ВЕРСИЯ содержит неформальное описание версии языка и его описание в метаязыке, оформленное как ЭОЯ. ВЕРСИЯ представляет возможности хранения в БД ЭОЯ нескольких модификаций языка пользователя, которые тес-

тируются для выбора наилучшего. Тексты тестов и данные о разрешении различных видов доступа (чтение, модификация) пользователя к данной версии языка хранятся в типе записей ЗАЩИТА. Последняя подчиняется типу записей ПОЛЬЗОВАТЕЛЬ (идентификатор, имя, данные пользователя). ВЕРСИЯ связана также с типом записей ФИКТ1, чем определены все ЭОЯ, связанные с данной версией языка;

- тип записей ВЫХЯЗЫК содержит имена и описания выходных языков;

- типы записей ЛЕКСИКА, БНФ, СТАТИКА, ДИНАМИКА содержат ЭОЯ соответственно лексики, синтаксиса, статической семантики и динамической семантики для различных версий входных языков вместе с неформальным описанием и примером для пользователя. Множества ЭОЯ лексики, синтаксиса и статической семантики разбиваются на подмножества, соответствующие версиям входных языков и конструкциям абстрактного синтаксиса, а множество ЭОЯ динамической семантики дополнительно разбивается по выходным языкам;

- тип записей ФИКТ1 связывает версии языков и ЭОЯ абстрактного синтаксиса, позволяя отвечать на вопросы типа: какие ЭОЯ абстрактного синтаксиса участвуют в данном языке? В какие языки входят данные конструкции? Кроме того, записи типа ФИКТ1 связывают ЭОЯ абстрактного синтаксиса данной версии языка с соответствующими ЭОЯ лексики, синтаксиса и статической семантики. Записи типа ФИКТ1 содержат информацию о согласующих связях между ЭОЯ в данной версии языка;

- тип записей ФИКТ2 определяет ЭОЯ динамической семантики, соответствующий входному и выходному языкам и конструкции абстрактного синтаксиса. ФИКТ2 несет информацию о согласующих связях.

Таким образом, в БД ЭОЯ имеются данные четырех видов:

- тексты ЭОЯ в элементах записей под именами ЭОЯ, входящих в типы АБСТРАКТ, ЛЕКСИКА, БНФ, СТАТИКА, ДИНАМИКА, ВЕРСИЯ;

- информация, необходимая для идентификации записей и создания структуры во множестве ЭОЯ. Ее несут элементы записей, имена которых оканчиваются на знак №;

– информация для объяснения пользователю назначения, возможностей и представления ЭОЯ в языке программирования. Она заключена в элементах "описание" и "пример";

– служебная информация системы и пользователя. Она представлена на фиг. I типом записей ЗАЩИТА, содержащим информацию о защите БД и тестовые программы пользователей для своих языков.

Виртуальные элементы на фиг. I служат следующим целям:

– выражению семантики реальных наборов и процессов обновления в сетевой БД. Их значения равняются значениям соответствующих элементов В-записей в наборах. Например, значение виртуального элемента К№ в записи типа АБСТРАКТ равняется значению элемента К№ в В-записи типа КОНСТРУКЦИЯ, связанной с данной Ч-записью типа АБСТРАКТ. Если запись не включена в набор, то значением виртуального элемента является "не определено";

– идентификации записей и их элементов. Например, именем записи типа АБСТРАКТ (а значит, и соответствующего ЭОЯ абстрактного синтаксиса) может служить совокупность поименованных значений элементов К№ и А№.

Литература

I. Ершов А.П., Грушецкий В.В. Метод описания алгоритмических языков, ориентированный на реализацию. – Препринт ВЦ СО АН СССР, Новосибирск, 1977.

2. Тепанди Я.Я. О проблемах применения системы управления базами данных в СПТ. – В сб.: Автоматизация производства пакетов прикладных программ (автоматизация производства трансляторов). Тезисы докладов. Таллин, 1980, с. 183–187.

3. Михновский С.Д., Стогний А.А. Вопросы автоматизации проектирования баз данных. – УСиМ, 1979, № 6, с. 29–35.

4. Ершов А.П. Проектные характеристики многоязыковой системы программирования. – Кибернетика, № 4, 1975, с. 11–27.

5. Мяннисалу М.А., Тыугу Э.Х., Унт М.И., Фуксман А.Л. Язык УТОПИСТ. - В сб.: Алгоритмы и организация решения экономических задач, вып. 10. М., Статистика, 1977, с. 80-118.

6. Вооглайд А.О., Томбак М.О. Система построения эффективных многопроходных трансляторов с LR(k)-семантикой. - Программирование, 1976, № 5, с. 28-38.

7. Report on the Definition and Implementation of Specialized Languages/Edited by N.J. Lehmann. Technische Universität Dresden, 1980. 296 pp.

8. Räihä K.-J., Saarinen M., Soisalon-Soininen E., Tienari M. The Compiler Writing System HLP (Helsinki Language Processor). - Department of Computer Science, University of Helsinki, Finland, Report A-1978-2, pp. 1-20.

9. Senko M.E. Data structures and data accessing in data base systems, past, present, future. - IBM Syst. J., 1977, N 3, pp. 208-257.

10. Bradley J. An extended owner - coupled set data model and predicate calculus for database management. - ACM Trans. on Database Systems, December 1978, vol. 3, N 4, pp. 385-416.

J. Tepandi

A Data Base for Problem-Oriented Languages

Summary

In this paper possible contents and organization of a data base to describe problem-oriented languages are introduced. An argument for the "language by example" design method is given. An extended Bachman diagram for a corresponding data base schema is presented.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

Т.Ф. Лучковский

ИСПОЛЬЗОВАНИЕ НИСХОДЯЩЕГО МЕТОДА ПРИ ПРОЕКТИРОВАНИИ ИНФОРМАЦИОННЫХ СИСТЕМ

Возрастающие возможности ЭВМ влекут за собой создание более мощных и сложных информационных систем (ИС). Особенности больших систем исключают возможность использования той же методики, что при малых системах. Уже нельзя полагаться только на свою интуицию или на огромную работоспособность некоторых программистов, или как писал Дейкстра, "любые два объекта, которые отличаются в чем-то по меньшей мере в сто или более раз, становятся совершенно несопоставимыми".[1]

Данная статья посвящена некоторым аспектам проектирования информационных систем. Характеристика таких систем приведена в статье [1].

I. Этапы создания информационных систем

Основные этапы создания ИС следующие:
- планирование;
- проектирование;
- реализация;
- опытное внедрение.

В ходе планирования разработчики ИС знакомятся с объектной системой решаемой проблемы и с требованиями пользователей к создаваемой системе, а также выбирают общую стратегию реализации системы.

Этап проектирования начинается с разбиения системы на подсистемы. Решается, каким образом будут передаваться дан-

[1] У. Дал, Э. Дейкстра, К. Хоор. Структурное программирование. М., Мир, 1975, с. 8.

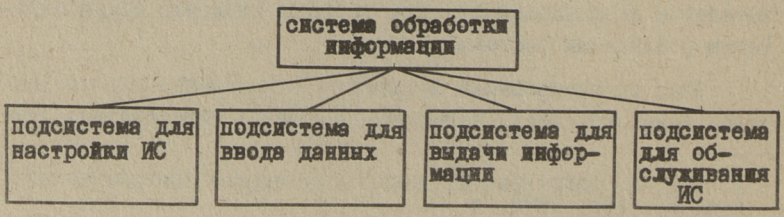
ные от одной подсистемы к другой, какие уже существующие пакеты прикладных программ можно использовать при реализации данной системы. Общая стратегия должна базироваться на нисходящем методе проектирования [3]. Эта методика требует итеративного разбиения задачи на подзадачи, пока не достигнут некоторого элементарного уровня.

Нисходящий метод используется и при реализации системы, хотя более распространенным является восходящий метод программирования. В интересах успешного создания ИС не следует реализацию откладывать до завершения этапа проектирования, а некоторые работы можно и нужно делать как можно раньше – на начальных стадиях проектирования. Более того, в ходе проектирования принимаются решения, правильность которых целесообразно проверить сразу же, чтобы в основу дальнейшего проектирования не легли неверные предпосылки.

При создании ИС зачастую приходится возвращаться к предыдущему этапу, исправлять неправильные решения, и по-новому подходить к решаемой проблеме. Чем меньше таких возвращений, тем успешнее создается данная ИС.

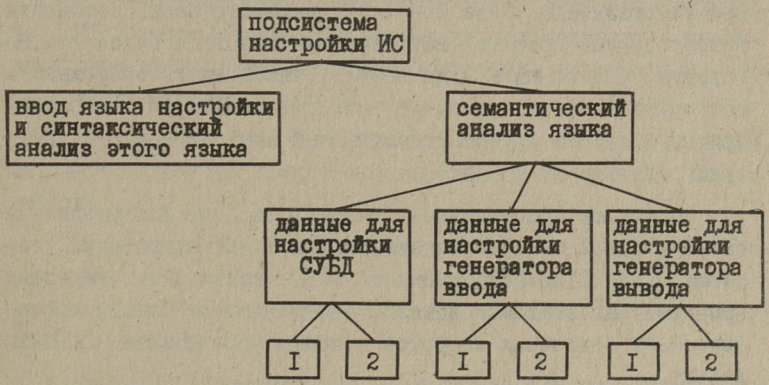
2. Проектирование информационной системы

Типичная система обработки информации состоит из средств ввода и выдачи информации. Но так как система создается не для конкретной задачи, а для определенного класса задач, то пользователю нужны и соответствующие средства настройки данной системы. Работу пользователя с системой облегчают программы обслуживания. Система обработки информации по нисходящему методу строится поэтапно. Первый этап разбиения ИС на относительно независимые части приведен на фигуре I.



Фиг. 1. Разбиение системы на подсистемы.

Далее определим главные функции подсистемы для настройки ИС. Общая стратегия реализации ИС определит область настраиваемости ИС. Если в создаваемой системе будут использоваться разные пакеты прикладных программ (ППП), например, конкретная СУБД, генератор ввода одной и генератор вывода другой системы, тогда системному аналисту придется иметь дело с несколькими языками настройки ИС. Поэтому иногда целесообразно создать новый язык настройки, покрывающий возможности других языков и являющийся наиболее удобным для системного аналиста. Разработчики должны реализовать такие средства, которые из создаваемого языка выделяют нужные для конкретного ППП данные и преобразуют их в нужный формат. На фигуре 2 приведен один из возможных вариантов разбиения подсистемы на функции.



I – выбор нужных данных и преобразование их в нужный формат,

2 – анализ выбранных данных с помощью средств конкретного ППП.

Фиг. 2. Подсистема настройки ИС.

Некоторые части подсистемы можно реализовать с помощью ППП. Так для автоматического создания транслятора языка настройки ИС используется система построения трансляторов. При реализации других частей подсистемы можно использовать, например, пакет списков или пакет метода хэширования. Как будет выглядеть конкретная реализация, зависит, во-первых, от формата, в который транслятор переводит язык настройки ИС и, во-вторых, какого формата данных требуют программы

ППП. Решение этих вопросов является целью третьего шага разбиения проблемы на отдельные функциональные части.

Подсистема ввода данных состоит из следующих функций:

1) ввод (сканирование) данных;
2) логический контроль данных;
3) локализация места хранения информации;
4) выдача диагностических сообщений.

Перечисленные функции реализованы в большой мере с помощью ППП. Может оказаться, что локализацию, которая зависит от конкретной схемы базы данных, нужно реализовывать самому разработчику. В этом случае при проектировании следует учитывать, что с той же проблемой мы сталкиваемся при выдаче данных и в программах обслуживания (например, при реорганизации базы данных). Диагностические сообщения целесообразно проектировать едиными для всех подсистем. Необходимо свести их в один модуль, чтобы иметь возможность без труда вносить изменения в тексты сообщений, особенно при переходе на другой естественный язык (эстонский, русский, английский и т.д.).

Примером использования одного средства для различных целей в одной системе служит язык логических условий системы СХОДИ [1]. Пользователь СХОДИ использует этот язык при проверке вводимых данных, при выдаче отчетов, при формировании одиночных запросов и при реорганизации базы данных.

С другой стороны, нельзя надеяться при проектировании, что все функции можно реализовать с помощью ППП, особенно тогда, когда число пакетов в создаваемой ИС будет слишком велико. Неотлаженные до конца или плохо документированные, слишком мощные или сложно настраиваемые пакеты в итоге могут увеличить затраты на реализацию ИС, вместо того, чтобы их уменьшить.

3. Реализация информационных систем

Хотя этап реализации и следует за этапом проектирования, это не означает, что некоторые его работы можно сделать с помощью ЭВМ еще до завершения проектирования. Существуют работы двух типов, которые можно сделать на ранних стадиях:

I) проверка возможности реализации принимаемых решений;

2) вспомогательные работы для ускорения последующей реализации ИС.

Нисходящее программирование и тестирование [3] позволяют проверить ключевые моменты проектирования, такие как интерфейс данных и подключение ППП сразу же после выбора метода реализации. Создание наборов данных и библиотек, создание процедур и комплектация пакетов на ЯУЗ - все эти работы можно выполнять, не ожидая всех результатов проектирования. Модули выдачи сообщений, в том числе и сообщения об ошибках, нужно разработать как можно быстрее по двум причинам:

I) это способствует взаимопониманию будущего пользователя ИС и разработчика ИС; позволяет пользователю высказать свои предложения, которые могут быть полезными на этапе проектирования;

2) в конце этапа реализации остается очень мало времени для обеспечения корректной справочной информации пользователю.

Применение ППП требует выполнения таких работ, как копирование библиотек, ознакомление с документацией, апробирование пакета. Ясно, что и эти действия нельзя откладывать в конец этапа реализации.

Части ИС, которые надо реализовать самому разработчику, после этапа проектирования, должны быть разработаны до уровня модулей. В пределах модуля нужно использовать метод структурного программирования [4].

При использовании ППП возникает проблема, которая связана с возможными ошибками в программах этого пакета. Чтобы уменьшить время отладки, разумно воспользоваться методом нисходящего тестирования. На первом шаге тестирования программа ППП заменяется фиктивной программой, которая генерирует и передает другим программам данные в нужном формате. Замена фиктивной программы реальной программой ППП на другом шаге тестирования показывает правильность использования и работы данного ППП.

Успешное и своевременное создание ИС предполагает знание и опыт разработчика по применению разных ПШП, модульного и структурного программирования и нисходящего метода.

Литература

I. Выханду Л.К., Лучковский Т.Ф., Микли Т.И., Тепанди Я.Я. Система хранения и обработки дискретной информации. – УСиМ, № I, 1981, с. 99–102.

2. Дал У., Дейкстра Э., Хоор К. Структурное программирование. М., Мир, 1975.

3. Йодан Э. Структурное программирование и проектирование программ. М., Мир, 1979.

4. Хьюс Дж., Мичтом Дж. Структурный подход к программированию. М., Мир, 1980.

T. Luchkovsky

Top-Down Approach to Information Systems Design

Summary

Information systems design consists of four stages. In this paper we discuss the project and development stage and show how top-down approach helps to solve some problems arising in those stages.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.016

Т.Ф. Лучковский, Т.И. Микли, А.В. Рензер

ЭКОНОМИЧЕСКИЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ КОЛЛЕКТИВНОГО ПОЛЬЗОВАНИЯ

В данной статье рассматриваются проблемы создания больших информационных систем (ИС), находящих применение в анализе экономической деятельности различных звеньев народного хозяйства. Разработка таких систем требует больших материальных затрат и много времени, поэтому ИС создаются не для одной конкретной задачи, а для целого класса схожих между собой задач. Подготовка ИС для выполнения конкретных задач заключается в настройке системы, которая в основном подразумевает уточнение семантики данных и функций системы. Настройка выполняется администратором ИС, который должен сопровождать систему и при необходимости производить перенастройки ИС.

ИС создается в общем итоге для группы конечных пользователей, т.н. коллективного пользователя. Конечному пользователю не обязательно иметь квалификацию программиста, свои требования к нужной ему информации он выражает на относительно простом языке запросов.

Авторы данной статьи имеют некоторый опыт разработки настраиваемых ИС [2, 3, 4]. Имеющаяся технология требует расширения и обобщения, к чему авторы будут стремиться в дальнейшей своей работе. Ниже приведены только наиболее глобальные принципы, из которых следует исходить при проектировании ИС коллективного пользователя.

Далее рассмотрим взгляд разработчика, администратора системы и конечного пользователя на конкретную ИС.

I. Разработчик исследует объектную систему, составляет информационную модель и определяет функциональные свойства создаваемой ИС. Тем же он определяет класс задач, для

выполнения которых пригодна данная ИС. Для удобства и простоты внедрения ИС разработчик выбирает самое типичное применение системы и производит соответствующую конкретную настройку ИС. Чтобы администратор системы имел возможность менять предлагаемую разработчиком настройку системы, должны быть предусмотрены средства настройки.

Реализация ИС коллективного пользования происходит, как правило, с использованием пакетов прикладных программ и таким образом, ИС представляет собой организованный через интерфейсы набор разных пакетов.

2. Администратор системы является специальным пользователем ИС, задача которого состоит в надстройке, обслуживании и сопровождении ИС. Более детальные задачи администратора системы следующие:

- определение точной семантики для информационных объектов, используемых в системе;
- уточнение шифраторов, справочников и т.д.;
- ввод справочной информации;
- описание вводимых данных;
- описание выходной информации;
- реорганизация базы данных в интересах более эффективного использования системы;
- создание необходимых наборов данных и библиотек;
- создание копии наборов данных для восстановления работоспособности ИС после сбоя ЭВМ;
- организация защиты данных, указание кодов защиты каждому пользователю ИС;
- оповещение разработчика об ошибках в ИС.

Роль администратора системы в функционировании ИС очень велика, так как он является промежуточным звеном между разработчиком и пользователями ИС. Он должен в равной мере знать объектную систему пользователей, основы реализации данной ИС и операционную систему используемой ЭВМ.

3. Конечный пользователь нуждается в информации, которая вводится, хранится в базе данных и выдается в нужной форме с помощью ИС. Так как ИС – это система коллективного пользования, то каждый конкретный пользователь имеет свой отличающийся от других взгляд на хранимые данные. ИС должна обеспечить хранение и выдачу информации таким образом,

чтобы удовлетворять запросы каждого пользователя. При выдаче информации могут происходить разные структурные преобразования данных, о существовании которых пользователю знать не обязательно.

Основой запросов конечного пользователя является модель объектной системы. Информационная модель выражает функционирование объектной системы. При создании этой модели определяют объекты, их атрибуты (показатели, признаки) и связи между объектами. Одним возможным вариантом описания информационной модели является схема базы данных типа CODASYL [I]. Объектам и атрибутам присваиваются имена, которые пользователь использует при составлении запросов.

Для выражения своих желаний пользователь системы имеет язык запросов, с помощью которого он управляет манипуляцией данными. Существуют два типа показателей:

I) простые показатели, хранимые в базе данных;

2) вычисляемые показатели, компонентами которых могут быть и простые и вычисляемые показатели.

При формировании запроса пользователь не должен знать, какого типа показатели он запрашивает. Нужные алгоритмы для вычисления показателей находятся в информационной базе и каждому вычисляемому показателю соответствует функция-подпрограмма. Вычисляемые показатели характерны при экономических анализах и необходимо иметь возможность добавления в ИС новых правил для вычислений.

Язык запросов конечному пользователю ИС должен предоставлять следующие возможности:

- выбирать данные, необходимые для получения нужных результатов;
- описывать ограничения при поиске информации в виде логических условий;
- указывать действия над данными;
- описывать форму выдачи информации;
- выбирать внешние устройства, с которыми пользователь будет работать;
- идентифицировать пользователя, т.е. организовывать защиту данных от неуполномоченного использования.

Самым удобным средством для описания запросов является дисплей или специальный бланк, где указаны соответствующие ключевые слова, так что пользователю нужно вводить только значения.

Хотя каждая ИС имеет свою специфику, тем не менее можно выделить функциональные свойства, характерные для всех ИС. Так, например, каждая ИС имеет по меньшей мере два языка - один для настройки ИС и другой для запросов - следовательно, нужны и средства трансляции этих языков. Хранимые в информационной базе данных и необходимые для ввода-вывода средства должны быть защищены от неуполномоченного использования. Самые характерные функциональные свойства для любой ИС следующие:

- трансляция языка описания данных и языка запросов;
- ввод и контроль данных;
- реорганизация хранимых данных;
- обработка запросов;
- выдача информации в нужной форме;
- защита данных;
- статистика об использовании данных.

Литература

I. Martin J. Computer Data-Base Organization. Prentice-Hall, Englewood Cliffs, N.J., 1975. 560 p.

2. Лучковский Т. Использование нисходящего метода при проектировании информационных систем. См. наст. сб., с. I7.

3. Рензер А. Об автоматическом интерфейсе между базами данных типа CODASYL и системами ввода-вывода. См. наст. сб., с. 29.

4. Система хранения и обработки дискретной информации СХОДИ. Материалы для выставки "НТИ-80". Таллин, I980. 40 с.

5. Шеребин В.М., Мальцев В.Н., Соваловв М.С. Экономические информационные системы: Рационализация проектирования. М., Наука, I978. II0 с.

T. Luchkovsky, T. Mikli, A. Renzer

Corporated Business Information Systems

Summary

In this article the main principles of designing business information systems are discussed. Designer's, system administrator's and end user's views are presented as well as the main functions of such systems.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.016

А.В. Рензер

ОБ АВТОМАТИЧЕСКОМ ИНТЕРФЕЙСЕ МЕЖДУ БАЗАМИ ДАННЫХ ТИПА CODASYL И СИСТЕМАМИ ВВОДА-ВЫВОДА

В статье рассматривается проблема интерфейса на модели автоматического отбора данных. Исследуются условия выбора путей локализации в базе и предлагается вариант реализации на специальных списках.

При работе с базами данных (БД) одной из основных проблем представляется ввод и вывод данных, организация надстройки для занесения документов в базу и получения ответов на запросы. Между системой управления базой данных и системами ввода-вывода нужен интерфейс, с помощью которого данные стали бы доступными надстройке и который управлял бы манипулированием данными.

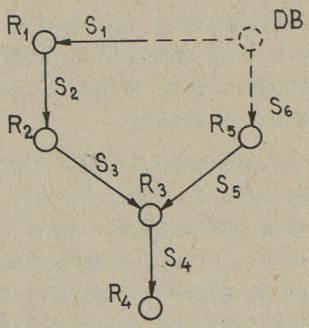
Организация такого интерфейса предполагает автоматическую локализацию записей в базе, т.е. любые действия с базой (обновление данных, выполнение запросов) должны происходить без участия программиста. Описание документа обновления [4] или запрос на языке высокого уровня [1] должны давать достаточную информацию для создания рабочей программы на языке манипулирования данными (ЯМД) [1, 2], используя, конечно, и информацию о структуре базы из описания данных [1, 2] базы.

Ключевым вопросом автоматической локализации является организация доступа к искомым типам записей, т.н. вопрос выбора путей поиска в базе [3, 5]. Нельзя не согласиться с С. Озборн [3], по мнению которой при семантически независимых данных существует возможность выявления всех путей поиска в этой базе еще до начала работы, например, при инициализации базы.

Для конкретной задачи на базе из множества всех путей по некоторым признакам можно выбрать один подходящий путь и по нему составить рабочую программу из операторов ЯМД. Ниже будет показано, как в некоторых случаях можно избавиться и от выбора.

Составление рабочей программы можно свести к последовательному активизированию готовых программ – операторов ЯМД.

Приведем для примера схему базы типа CODASYL в виде графа:



Фиг. 1.

Граф состоит из множества вершин $R=\{R_1,R_2,R_3,R_4,R_5\}$ и множества дуг $S=\{S_1,S_2,S_3,S_4,S_5,S_6\}$. R представляет типы записей БД, S – отношения между этими записями. Точками входа в базу являются отношения S_1 и S_6; их владельцами будем считать фиктивного типа записи DB.

Путем до вершины R_i на графе называется множество дуг, которые необходимо проходить для достижения вершины R_i. Аналогично путем поиска для типа записей R_k является множество отношений БД, которые необходимо проходить для локализации этого типа записей. Поскольку R_k в разных отношениях соответствуют разные типы владельцев, в путь поиска придется включить все отношения, определенные этими владельцами. Обозначим один путь поиска через W_i. Одному запросу удовлетворяет подмножество путей, которое обозначим через $\overline{W}=\{W_i\}$, где $\overline{W}\subset W$. W множество всех W_i.

Для выбора подходящих членов из отношений нужна определенная информация. Ею могут служить значения каких-либо

полей данных в записях или ключей прямого доступа к записям, которые будем называть признаками поиска. Информацией поиска назовем предлагаемое в одном запросе множество признаков и обозначим это через $I=\{I_1,\ldots,I_N\}$, где $I_i=\langle i_1,\ldots,i_K\rangle$, K - количество признаков поиска для одного типа записей, N - количество типов записей в базе. Так как все фактические типы записей являются членами в каких-то отношениях, в одном запросе может оказаться информация поиска обо всех типах записей.

Вернемся к проблеме автоматической локализации. Здесь можно различать два класса действий:

I. Обработка базы для обновления данных по зафиксированным документам и/или выполнения зафиксированных запросов.

2. Обработка базы для обновления отдельных данных и/или выполнения любого запроса, возможного в базе.

В первом случае необходимо обеспечить максимальную скорость поиска, так как большинство задач обработки базы именно из этого класса действий. Второму классу необходимо обеспечить широкие возможности доступа к любым данным в любых сечениях. Это несколько снизит оперативность локализации, так как средства должны быть универсальными для всех баз с сетевой структурой.

Для зафиксированных документов и запросов множество $\bar{W}$ селектируется до начала обработки и присоединяется к синтаксическому и семантическому описаниям [4] документа или запроса. Таким образом, необходимость выбора путей поиска во время обработки для задач первого класса вообще отпадает.

Нахождение $\bar{W}$ для зафиксированных задач и выбор путей поиска для задач второго класса выполняются по трем условиям:

I. Необходимо выяснить, для каких W_i хватает информации I.

2. Необходимо достичь максимальной скорости поиска.

3. Необходимо учесть активность отношений к типам записей.

Условие активности отношений состоит в том, что пользователь базы не обязан привлекать все определенные на базе отношения. Например, при добавлении данных нет необходимости включения экземпляра во все отношения, имеющиеся на этом типе записей.

При оценке скорости поиска учитывают типы и длины отношений, входящих в путь. Предпочтительнее короткие отношения, т.е. с меньшим числом членов. Для запросов удобнее отношения прямого доступа, для добавления записей – с произвольной расстановкой членов. Из разных путей предпочитают более длинные в силу того, что они перекрывают одно и то же количество данных с большим количеством отношений, которые, следовательно, должны быть короче.

Первоначальный выбор путей поиска производится по количеству информации поиска I. При анализе текста запроса или обновляемых данных выясняют типы R_i, для которых представлена информация $I=\langle i_1,\ldots,i_K\rangle$, и соответственно определяют, для какого W_i достаточно признаков поиска. Далее предположим, что $K=1$, т.е. для каждого типа записей предъявлен максимально один признак поиска. В этом случае кортеж $\langle i_1,\ldots,i_K\rangle$ состоит из одного члена, и мы не обязаны анализировать структуру множеств $I_1 \ldots I_N$.

В базе со схемой, представленной на фиг. I, для локализации экземпляра R_3 должны существовать признаки поиска для выбора экземпляра R_3 (информация I_3) и его владельцев R_2 и R_5 (соответственно I_2 и I_5). Далее задача поиска сводится к рекурсивному поиску владельцев до точки входа, где владельцем является тип DB.

Определим операции локализации.

I. FMK (отн., ключ-прямого-доступа) – найти из отношения прямого доступа первый член с требуемым значением ключа.

2. FM (отн.) – найти следующий член отношения.

3. SO (отн.-I, отн.-2) – установить текущий член отношения-I текущим владельцем отношения-2.

4. SM (отн.-I, отн.-2) – установить член отн.-I членом отн.-2.

5 CFM (отн., член-поле) – проверить, имеет ли член отношения в требуемом поле искомое значение.

6. CFO (отн., владелец-поле) – проверить, имеет ли владелец отношения в требуемом поле искомое значение.

7. FNK (отн., ключ-прямого-доступа) – найти из отношения прямого доступа следующий член с требуемым значением ключа.

Будем считать, что отношения точек входа типа прямого доступа. Если при операциях FM, FMK или FNK встретится конец отношения или подходящего члена не находится, поиск прерывается.

В зависимости от количество подмножеств в I поиск можно осуществлять по нескольким путям. Достаточная информация для поиска экземпляра R_3 $I^1=\{I_3,I_2,I_5\}$ и $I^2=\{I_3,I_2,I_1,I_5\}$. По I^1 экземпляр R_3 с признаком I_3 локализируется так:

```
P1: FMK(S6,I5)          вход
    SO(S6,S5)           переход в другое отношение
P2: FM(S5)              следующий член
    if CFM(S5,I3) = true then P3; else P2
P3: SM(S5,S3)           переход в отношение владельца
    if CFO(S3,I2) = true then end; else P2
```

Поиск по I^2 ведется начиная с отношения S_1. Соответствующие пути поиска будут $W_1=\{S_6, S_5, S_3\}$ и $W_2=\{S_1,S_2,S_3,S_5\}$. По длине предпочтительнее путь W_2.

Далее приведем один вариант реализации нахождения и выбора путей поиска и автоматической локализации на специальных списках.

Элементарной операцией выполнения запроса или обновления данных всегда является локализация того или иного типа записей. Поэтому при инициализации базы каждому типу записей генерируют списки локализации, представляющие все возможные пути к нему. При прохождении списка активируют операции локализации и найденный экземпляр устанавливается текущим владельцем или членом во всех отношениях, в которые он входит и которые признаны активными.

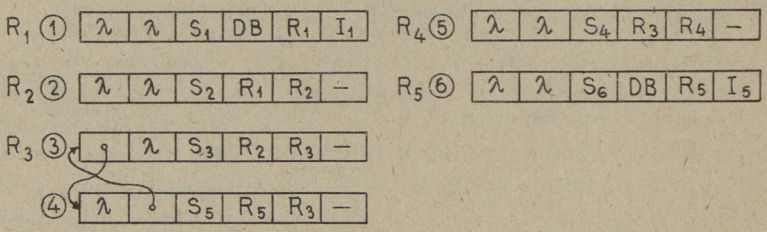
Источником информации для создания списков служит описание данных (ОД) базы, где описываются атрибуты всех типов записей и отношений между ними.

Каждому типу записей составляют список владельцев из элементов со следующей структурой:

ссылка на следующий	ссылка на предыдущий	отношение	владелец	член	ключ прямого доступа

Фиг. 2.

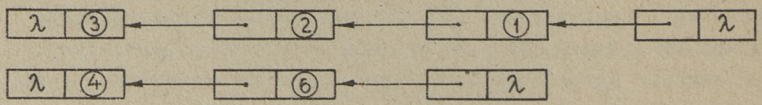
Каждый элемент этого списка составляют на одного члена отношения из описаний отношений в ОД. Соответствующие одному типу записей элементы списка связываются ссылками. Пустую ссылку обозначим через λ и составим списки владельцев по примеру фиг. I:



Фиг. 3.

Выяснение пути поиска одному типу записей состоит в обработке списков владельцев. Определяется последний в пути тип записей и начинают искать его владельцев, после этого их владельцев и т.д. до тех пор, пока не выходят в какую-то точку входа. Так, для каждого типа записей образуются списки путей поиска, где один список представляет один путь. Элементом списка окажется ссылка на подходящий элемент списка владельцев, который описывает отношение с нужным владельцем и членом.

Найдем списки путей поиска для R_3 с фиг. I.



Фиг. 4.

Множество всех возможных путей W образуется как совокупность переупорядоченных некоторым образом элементов спи[illegible] владельцев. Для выполнения локализации из списков

путей поиска составим списки локализации, которые будут активизировать определенные выше операции локализации. Один список управляет локализацией на одном пути, и элементом списка являются ссылки на одну операцию и его операнды. Структура элемента списка локализации в зависимости от выполняемой операции следующая:

код	операнд I	операнд 2	ссылка
FMK I	отношение	ключ	
FM 2	отношение		
SO 3	отношение-I	отношение-2	
SM 4	отношение-I	отношение-2	
CFO 5	отношение	владелец	
CFM 6	отношение	член	

Фиг. 5.

Списки локализации составляются совершенно общими, не зависимыми от данных в базе. Например, при операции FMK нужно значение ключа прямого доступа. В элементе списка дается не это значение, а ссылка на поле, где хранится такое текущее значение. Информация поиска делается доступной спискам локализации с помощью списков признаков поиска для каждого типа записей.

Алгоритм составления списков локализации обрабатывает списки путей поиска и формирует элементы в четыре логические группы.

I. Если отношение пути прямого доступа (в списке владельцев шестое поле не равно нулю), то формируется элемент FMK.

2. Если у члена очередного отношения существуют еще какие-то владельцы (в его списке владельцев первые два поля элемента не имеют значения λ), то формируется соответствующее количество элементов SM и CFO. Эти операции всегда встречаются попарно: с помощью первого доходят до отношения, где имеется владелец, и с помощью второго проверяют, подходящий ли он.

3. Если отношение не прямого доступа, то формируется группа из элементов FM и CFM.

4. После элемента FM или FMK необходимо проверить, имеются ли в пути еще отношения. Если имеются, составляют элемент SO.

С помощью таких групп окажется возможным определить необходимые операции для локализации экземпляра любого типа записей.

Если не все владельцы экземпляра, найденного операцией FMK, оказались подходящими, то автоматически выполняются операции FNK с этими же операндами до достижения положительного или отрицательного результата. Аналогично повторяется группа FM и CFM.

Если удалось пройти весь список локализации и нашелся подходящий экземпляр, он устанавливается текущим владельцем во всех отношениях, куда он входит. Для быстрого выполнения этого по ОД генерируют списки членов для всех типов записей. Такой список проходят, и на каждом элементе выполняется операция SO. В конце списка членов локализация совершена и алгоритм закончит работу.

Литература

1. Date C.J. An Introduction to Database Systems. California, IBM Laboratories, 1975. 360 p.

2. Martin J. Computer Data-Base Organization. Prentice-Hall, inc., Englewood Cliffs, N.J., 1975. 560 p.

3. Osborn S.L. Towards universal relational interface. - Very Large Data Bases. 1979, N 2, Rio de Janeiro, pp. 52-60.

4. Лийб Д.Б., Рензер А.В. Синтаксически ориентированный генератор ввода. - Тр. Таллинск. политехн. ин-та, 1981, № 511, с. 71.

5. Тепанди Я.Я. О методах трансляции языков с проблемной ориентацией. - В кн.: Всесоюзная конференция по методам трансляции. Тезисы докладов, Новосибирск, 1981, с. 75-77.

A. Renzer

About Automatic Interface Between CODASYL Data Bases and Input-Output Systems

Summary

In this article the interface problem is viewed upon an automatic data selection model. Access path selection and automatic locating are described and an implementation model on special linked lists is presented.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

Е.Б. Бернштейн

СРЕДСТВА ПЕЧАТИ ТАБЛИЦ В СИСТЕМЕ СХОДИ

I. Введение

В большинстве систем обработки данных требуется выдавать разного рода машинные документы - отчеты, сводки, табуляграммы. Не является исключением и Система хранения и обработки дискретной информации (СХОДИ), разработанная в ТПИ. Система СХОДИ позволяет извлекать из базы данных документы (или их части), удовлетворяющие некоторым условиям [1]. Результатом этой работы является массив чисел (или символов), который в дальнейшем следует распечатать в виде таблицы с заголовком, шапкой и т.д. Предлагаемые средства печати служат для оформления массивов чисел (или символов) в виде машинных документов. При выборе общего вида печатаемых таблиц разработчики ориентировались на требования, предъявляемые к отчетной документации в МВД СССР. При этом ставилась задача облегчить пользователю описание таблицы.

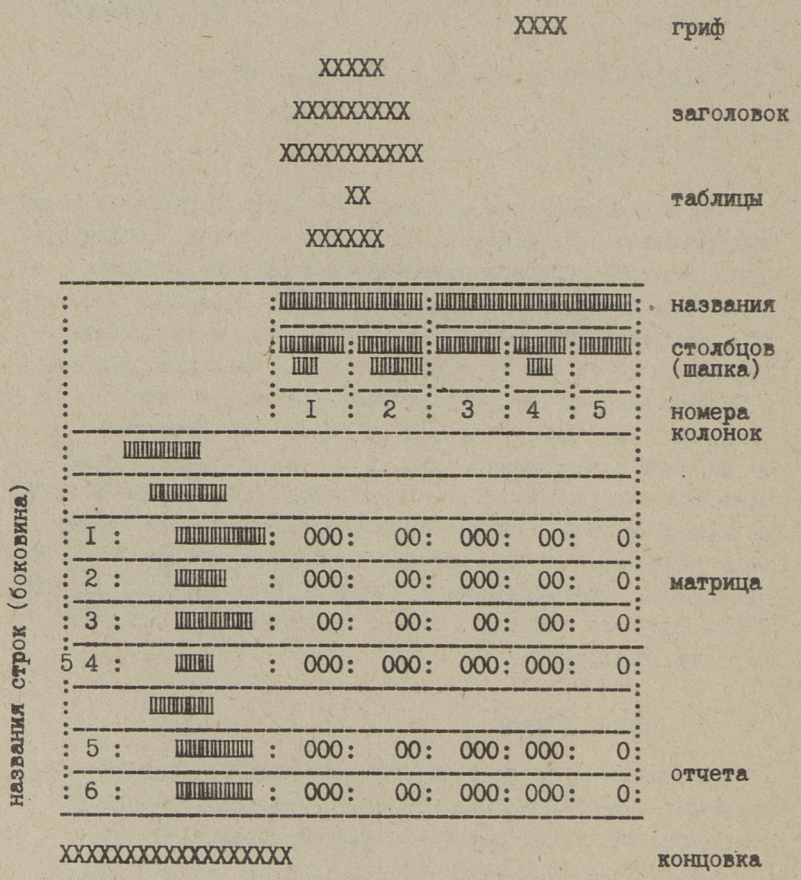
2. Общие понятия

Печатаемая таблица состоит из следующих частей: заголовка, тела и концовки таблицы. Заголовок и концовка представляют собой тексты, задаваемые пользователем.

Тело таблицы состоит из названий столбцов (шапка таблицы), названий строк (боковина таблицы) и собственно числовой (или символьно-числовой) матрицы отчета. Матрица отчета получается в результате работы других программ системы СХОДИ и представляет собой входной набор данных для программ печати. Этот набор данных является единственным связующим звеном программ печати таблиц и системы

СХОДИ, что позволяет пользоваться программами печати независимо от системы СХОДИ.

Заголовок таблицы состоит максимально из шести строк текста. Пять из них расположены в центре листа АЦПУ, а одна – первая, т.н. гриф, отстоит на I/3 от правого края



Пояснения:

XXXX – тексты заголовка, грифа и концовки таблицы;

▒▒▒▒ – названия столбцов и строк;

0000 – матрица отчета.

Фиг. 1. Общий вид таблицы отчета.

листа. Заголовок таблицы может занимать отдельный титульный лист или располагаться на первой странице тела отчета.

Тело таблицы может занимать произвольное число страниц. Распределение по страницам происходит автоматически во время печати. Число страниц определяется размерами печатаемых данных. При этом предлагаемые средства печати обеспечивают возможность разбивать на несколько страниц "широкие таблицы", т.е. такие, которые не помещаются на один лист АЦПУ в ширину. Каждая страница снабжается номером и некоторой дополнительной информацией.

Общий вид таблицы приведен на фиг. I.

Настоящие программы написаны на языке ПЛ/I и реализованы на машинах серии ЕС ЭВМ под управлением операционной системы ОС ЕС.

3. Описание таблицы пользователем

Процесс печати таблиц разбит на два этапа:

I) создание файла названий данной формы отчета, при этом происходит считывание названий строк и названий столбцов, задаваемых пользователем, и запись их в библиотеку;

2) непосредственно печать таблицы, в ходе которой используются названия строк и столбцов, записанные в библиотеке. На этом же этапе пользователь может задавать параметры, управляющие как видом и форматом отчета, так и текстами заголовка и концовки таблицы. Например, можно задать число строк на странице отчета или число символов в строке, можно решить, будет ли заголовок таблицы печататься на одной странице с телом таблицы или же на отдельной. В целях лаконичности всем параметрам присвоены значения по умолчанию.

Как видно из фиг. I, названия столбцов и названия строк могут иметь подпункты, которые, в свою очередь, разбиваются на подпункты и т.д. Пользуясь терминологией языка ПЛ/I, можно сказать, что названия столбцов и названия строк образуют структуру. Поэтому пользователь, задавая текст названия, должен указать номер уровня этого названия, как элемента структуры.

Дадим определение уровня.

Название строки или столбца имеет номер уровня I, если оно не является подпунктом никакого другого названия. Название, являющееся подпунктом названия уровня I, является названием уровня 2. Название является названием уровня n, если оно есть подпункт названия уровня n-1.

Далее определим последовательность задания названий.

Первыми задаются названия столбцов уровня I до тех пор, пока среди них не встретится название, имеющее подпункты уровня 2; затем следуют подпункты уровня 2, пока у какого-либо из них не встретятся подпункты уровня 3 и т.д. Затем переходят к названиям строк, последовательность задания которых аналогична правилу для столбцов. Кроме того, названия столбцов нижнего уровня, т.е. тех, которые не имеют подпунктов, и являются тем самым собственно названиями колонок, требуют дополнительной информации. А именно, пользователь должен указать формат печати и тип данных этой колонки. Тип описывает, какие данные - целые, действительные или символьные - будут печататься в колонке. В зависимости от этого он будет принимать значения I, 2 или 3 соответственно.

Что касается шапки таблицы, то, как видно из фигуры I, способ размещения подзаголовков традиционен - они печатаются непосредственно под соответствующим заголовком. Боковина оформлена следующим образом. Название строки, имеющее подпункты, не будет, как принято, располагаться левее этих подпунктов в вертикальном положении, а будет занимать отдельную строку в боковине. Подпункты же этого названия печатаются в следующих строках, причем текст названия подпункта смещен вправо относительно текста названия пункта - печать "лесенкой". Строки нижнего уровня снабжаются номерами, и только им в самой матрице отчета соответствует непустая строка.

Таким образом, на первом этапе пользователь должен задать названия столбцов, снабженные номерами уровней и форматами печати колонок таблицы, и названия строк, снабженные номерами уровней.

На втором этапе печати управляющие параметры позволяют регулировать следующие моменты:

I) ширину отчета (параметр COL);

2) количество строк на странице (параметр POW);

3) ширину боковины (параметр W|D);

4) количество экземпляров таблицы (параметр NEX);

5) отделять ли строки в таблице друг от друга линией (параметр LIN);

6) печатать ли заголовок таблицы на одной странице с телом таблицы или выносить его на середину отдельной страницы (параметр SEP);

7) нумеровать ли столбцы и строки (параметры NUM и LNB);

8) определять текст пяти строк заголовка таблицы (параметры REP, NAM, REG, MON и FOR);

9) определять текст концовки таблицы (параметр FIN);

I0) снабжать ли таблицу грифом - секретно (параметр SEC).

Необходимость разбивки процесса печати на два этапа была вызвана тем, что отчет приходится печатать десятки раз по одной и той же форме, т.е. с теми же названиями столбцов и строк. При предложенном способе каждая следующая печать таблицы сводится лишь к указанию имени формы отчета - по нему отыскивается файл названий и, в случае необходимости, к заданию управляющих параметров.

4. Пример

Предположим, что надо составить отчет об успеваемости и напечатать его по форме 2а. См. фиг. 2.

Данные для первого этапа печати таблицы надо подготовить в такой форме:

```
I успеваемость по группам¤
2 группа 20I¤
3 всего студентов!4¤
3 из них сдали на положит. оценку!4¤
3 из них не сдали!4¤
3 % сдавших от общего числа студ!5,2,2¤
2 группа I02¤
```

```
3 всего студентов!4¤
3 из них сдали на положит. оценку 4 ¤
3 из них не сдали!4¤
3 % сдавших от общего числа студ.!5,2,2¤
I общая успеваемость¤
2 всего студентов!4¤
2 из них сдали на положит. оценку!4¤
2 из них не сдали!4¤
2 % сдавших от общего числа студ.!5,2,2¤
¤
I спец. дисциплина¤
2 дифф. уравнения ¤
2 мат. анализ ¤
2 теор. вер.¤
I обществ. дисциплины ¤
2 диалект. материализм ¤
¤
```

Пояснения

Номера, идущие перед текстом названия, означают номер уровня соответствующего названия. Текст названия строки или столбца, если это не столбец нижнего уровня, заканчивается символом-разделителем "¤". После названия столбца нижнего уровня ставится символ " ! " и затем следуют три позиционных параметра, означающие:

1. ширину колонки;
2. число знаков, печатаемых после запятой в числах этой колонки;
3. тип данных, печатаемых в колонке – I целые числа,
   2 – нецелые числа с фиксированной точкой,
   3 – символьные данные

Символ "¤" кроме того означает конец списка названий столбцов и конец списка названий строк.

Данные для второго этапа таблицы можно подготовить таким образом:

В ДЕКАНАТ

СВОДКА УСПЕВАЕМОСТИ СТУДЕНТОВ ВТОРОГО КУРСА МЕХАНИКО-МАТЕМАТИЧЕСКОГО ФАКУЛЬТЕТА
ПО РЕЗУЛЬТАТАМ ВЕСЕННЕЙ СЕССИИ (ФОРМА 2А)

		Успеваемость по группам								Общая успеваемость			
		Группа 201				Группа 202							
		всего студентов	из них сдали на полож. оценку	из них не сдали	% сдавших от общего числа студентов	всего студентов	из них сдали на полож. оценку	из них не сдали	% сдавших от общего числа студентов	всего студентов	из них сдали на полож. оценку	из них не сдали	% сдавших от общего числа студентов
спец дисциплины	дифф.уравн.												
	мат.анализ												
	теор.вер.												
общ. дисц.	анал. мат.												

СТАРОСТА КУРСА ПЕТРОВ

Фиг. 2.

```
REP = 'сводка'
NAM = 'успеваемости студентов второго курса'
REG = механико-математического факультета
MON = по результатам весенней сессии
FOR = форма 2а
FIN = староста курса Петров
SEC = в деканат
```

5. Возможности печати широких таблиц

В предложенной системе существует возможность печатать "широкие" таблицы, т.е. такие, которые в ширину не помещаются на одном листе бумаги АЦПУ. При этом будет осуществляться автоматическая разбивка каждого листа таблицы на необходимое число страниц. На первом листе будет напечатано столько первых колонок, сколько помещается на одном листе, остальные колонки будут перенесены на вторую страницу. Если на второй странице все колонки не поместятся, то будет сформирована следующая страница и т.д. В шапке страницы продолжения будет указан номер страницы и продолжением какой страницы она является. Шапка таблицы разобьется на столько частей, сколько страниц АЦПУ занимает отчет в ширину. Первая часть содержит названия тех столбцов, которые появляются на первой странице. Вторая часть содержит названия тех столбцов, которые появятся на второй странице и т.д. Если какое-то название столбца имеет подпункты, находящиеся на разных страницах, то само это название появится во всех соответствующих этим страницам частях шапки таблицы. Каждая часть шапок таблицы печатается на каждой основной странице и на каждой странице продолжения соответственно.

Что же касается названий строк, то существует две возможности, регулируемые задаваемым параметром BOK. Если BOK = Ф, то они печатаются только на основной странице, а на страницах продолжения фигурирует лишь номер строк. При BOK=1 боковина печатается на каждой странице продолжения.

В системе предусмотрено соответствие между расположением строк таблицы на основной странице и на странице про-

должения, что позволяет склеить страницы и получить "широкий" отчет в развернутом виде.

Разумеется, пользователь всегда имеет возможность разбить большой "широкий" отчет на несколько маленьких и не возиться со страницами продолжения.

6. Сравнение с другими системами печати таблиц

Разработка систем печати таблиц была вызвана необходимостью автоматизировать процесс изготовления документов при помощи ЭВМ. В литературе по теме принято следующее разбиение на два основных класса [2].

Программный способ предполагает написание программы, в результате работы которой печатается таблица. При этом способе задается не только способ изображения объектов, но и формируются данные для вывода. Например, они позволяют управлять процессом печати в зависимости от выполнения условий над исходными данными и промежуточными результатами. Средства печати языков программирования – (ПЛ/I, Фортран)– можно отнести к этому классу. При этом задание таблицы есть детализация таблицы вплоть до печатаемых строк и полное описание вида каждой строки. Средства Кобола позволяют описывать и печатать более крупные логические единицы печатаемых данных. В [2] предложен сравнительно простой язык, в терминах которого легко описываются логические единицы печатаемых таблиц – полосы. В этом языке средства печати реализованы в виде некоторого набора операторов печати.

Сюда же можно отнести и [6].

Программные способы хотя и обладают универсальностью, непригодны для пользователей-непрограммистов, так как требуют составления алгоритма и программы печати.

Описательный способ задания таблицы, к которому относятся и рассматриваемые средства печати, состоит в описании вида таблицы. Обычно это описание составляется на некотором специально предназначенном для этого языке. Исходные данные при этом должны быть подготовлены заранее в соответствии с требованиями системы печати. Примеры таких способов приведены в [3] и [4].

В [3] документ рассматривается, как совокупность абзацев. Построенный Специальный алгоритмический язык документирования (САЯД) позволяет описывать типовой абзац, состоящий из набора форматов печатных строк, и формулу документа, т.е. программу построения документов из заданного набора типовых абзацев.

Аналогично, в [4], документ рассматривается как совокупность полос, занимающих одну или несколько строк. Каждая полоса состоит из прямоугольных элементов, в которые могут быть помещены текст или число. Описываются элементы, их содержимое и их положение в полосе.

Системы, предложенные в [3] и [4], обладают известной универсальностью, т.е. разрешают печатать таблицы любого вида и сложности, но это достигается дорогим путем детализации вплоть до элементов строк.

В настоящей работе вид таблицы зафиксирован, т.е. таблица не конструируется из набора описанных элементов. (Такой же подход использован в [5]). Это позволило укоротить и упростить описание таблицы по сравнению с вышеупомянутыми системами. Вместе с тем, варьируемость нефиксированных частей и частота употребления таблиц такого вида делают такую фиксированность оправданной.

Литература

I. Система хранения и обработки дискретной информации (СХОДИ). Материалы для выставки "НТИ-80". ТПИ, кафедра обработки информации. Таллин - 1980.

2. Собельман В.И., Старчеус Т.М. Средства печати таблиц в системе обработки данных. - Сб.: Алгоритмы и организация решений экономических задач, вып. 12, М., Статистика, 1978.

3. Коченов А. Специализированный алгоритмический язык документирования (САЯД) - Сб.: Алгоритмы и организация решений экономических задач, вып. I, М., Статистика, 1973.

4. Кострюков А.С., Собельман В.И. Язык для описания таблиц. - Сб.: Цифровая вычислительная техника и программирование, вып. 5, М., Сов.радио, 1969.

5. М и х а й л о в В. Система автоматического редактирования таблиц (САРТ). - Сб.: Электронно-вычислительная техника и программирование, вып. 3, М., Статистика, 1969.

6. Пакет прикладных программ "Генератор табуляграмм ОС". Научно-производственное объединение "Центрпрограммсистем", Калинин, 1980.

E. Bernstein

Output of Tables in SHODI System

Summary

Output tables formating system of data management system SHODI is described. A comparison with other analogous systems is given.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 330.115

Э.Х.-Т. Цунапуу

ОЦЕНКА ИЗМЕНЯЮЩИХСЯ ВО ВРЕМЕНИ РЕГРЕССИОННЫХ ЗАВИСИМОСТЕЙ

I. Введение

При построении регрессионных моделей обычно принимается гипотеза о постоянстве ее параметров во времени. Существует множество хорошо разработанных методов оценки моделей с постоянными параметрами, нашедших широкое применение на практике. Меньше уделено внимания оценке и использованию моделей с переменными параметрами. В то же время в реальной жизни сила взаимодействия факторов не остается постоянной, поскольку изменяется внешняя среда, в которой происходит наблюдаемый процесс. Использование же уравнений с постоянными параметрами позволяет дать процессу лишь усредненную характеристику и поэтому при использовании таких уравнений для прогноза трудно ожидать хороших результатов.

В настоящей статье дается метод оценки линейных регрессионных моделей с изменяющимися во времени параметрами. Характерной чертой приводимого метода является то, что изменение параметров характеризуется нелинейной функцией, аргументом которого является время.

В конце статьи приводятся результаты оценки параметров динамической производственной функции по данным Эстонской ССР.

2. Постановка задачи

Рассмотрим модель линейной регрессии

$$Y_t = B_t X_t + \varepsilon_t, \qquad (I)$$

где Y_t – зависимая переменная в момент t;

$X_t = (x_{1t}, x_{2t}, \ldots, x_{mt})$ – вектор независимых переменных в момент t;

ε_t – ошибка измерения.

При стандартной линейной регрессии предполагают, что параметры уравнения являются постоянными во времени ($B_t = B$ для всех t). В постановке, принимаемой в настоящей статье, предполагают, что параметры изменяются во времени и тенденцию их изменения можно охарактеризовать нелинейной монотонной функцией от времени.

$$B_{ti} = f_i(t) \quad i = 1, \ldots, m. \tag{2}$$

Такая постановка проблемы сходна с той, которая принимается в адаптивных моделях множественной линейной регрессии [1, 6, 7], но отличается тем, что вводится функция изменения параметров.

3. Решение проблемы

Для решения приведенной задачи сначала определяют значения параметров на каждый момент времени $t = 1, \ldots, T$, а затем выявляют функцию изменения этих параметров во времени.

Определение значений параметров на каждый момент времени производится в два этапа. На первом определяется линейная тенденция изменения параметров, а на втором этапе полученные параметры корректируются методом адаптивной корректировки [1, 4].

Выявление линейной тенденции изменения параметров производится линейным регрессионным методом, т.е. оценивается модель

$$Y_t = (B_{(0)} + B_{(1)} t) X_t + \varepsilon_t, \tag{3}$$

где $B_{(1)}$ и показывает линейную тенденцию изменения параметров.

Адаптивная корректировка параметров проводится методом наискорейшего спуска. Для этого сравним оценку $\hat{Y}_t$, полученную по уравнению (3), с фактической точкой ряда Y_t и вычислим ошибку

$$\varepsilon_t = Y_t - \hat{Y}_t = Y_t - (B_{(0)} + B_{(1)} t) X_t, \tag{4}$$

и на основе полученного результата произведем корректировку коэффициентов $B_{(1)}$. Корректировку коэффициентов осуществляем по следующему правилу:

$$B_н = B_с - к\,\mathrm{grad}(\varepsilon_t^2), \tag{5}$$

где $B_с$ – вектор старых коэффициентов;
$B_н$ – вектор новых коэффициентов;
$к$ – коэффициент ($к > 0$);

$\mathrm{grad}(\varepsilon_t^2)$ – вектор, градиент ε_t^2.

Элементы градиента вычисляем, используя выражения для ε_t (4), откуда получаем

$$\frac{\partial \varepsilon_t^2}{\partial b_{i,t}} = 2\varepsilon_t \frac{\partial \varepsilon_t}{\partial b_{i,t}} = -2\varepsilon_t t X_{it}.$$

В целом градиент равняется

$$\mathrm{grad}(\varepsilon_t^2) = -2\varepsilon_t t X_t. \qquad (6)$$

Подставляя (6) в (5), получаем выражения для корректировки коэффициентов:

$$B_н = B_c + 2\kappa t \varepsilon_t X_t. \qquad (7)$$

В полученном выражении неизвестным осталось только значение коэффициента κ, определяющее реакцию модели на полученную текущую ошибку. Чтобы придать названному коэффициенту независимость от шкал измерения, нормируем остатки и наблюдения следующим образом:

$$x_{tj}^* = \frac{x_{tj}}{\left(\sum_{j=1}^{m} x_{tj}^2\right)^{1/2}} \qquad 1 < j \leq m,$$

$$\varepsilon_t^* = \frac{\varepsilon_t}{\left(\sum_{j=1}^{m} x_{tj}^2\right)^{1/2}}.$$

Соответственно изменится и выражение для корректировки коэффициентов

$$B_н = B_c + 2t\kappa^* \varepsilon_t^* X_t. \qquad (8)$$

Нормирование остатков и наблюдений делает коэффициент κ^* постоянным для данной модели. Это значит, что его не надо определять заново на каждый момент времени.

После определения линейной тенденции изменения параметров и их адаптивной корректировки мы получаем векторы значений параметров на каждый момент времени. Далее выявляются функции изменения этих параметров во времени (2). Для этого используется методика автоматического восстановления монотонных зависимостей, приведенная в настоящем сборнике [2].

4. Оценка параметров производственной функции по данным Эстонской ССР

Производственные функции нашли широкое применение в экономическом анализе (см. [3], [4]). В качестве примера оценки функции с изменяющимися параметрами рассмотрим оценку динамической производственной функции Кобба–Дугласа [4]:

Таблица I

Динамическая производственная функция
экономики Эстонской ССР (1965–1977)

Год	Исходные данные			Результаты	
	Индекс основных производственных фондов	Индекс численности занятых	Индекс произведенного национ. дохода	Параметр производственной функции	Параметр производственной функции по (2)
1965	1	1	1	0.472	0.437
1966	1.075	1.019	1.039	0.484	0.434
1967	1.143	1.040	1.169	0.396	0.432
1968	1.210	1.061	1.274	0.413	0.429
1969	1.317	1.082	1.340	0.409	0.427
1970	1.415	1.101	1.454	0.380	0.424
1971	1.526	1.113	1.530	0.399	0.422
1972	1.671	1.123	1.522	0.443	0.419
1973	1.812	1.145	1.589	0.423	0.416
1974	1.951	1.156	1.709	0.415	0.414
1975	2.084	1.167	1.758	0.431	0.411
1976	2.199	1.178	1.875	0.397	0.408
1977	2.316	1.190	1.966	0.412	0.406

$$Y = AL^{\alpha}F^{1-\alpha}, \qquad (9)$$

где A, α – параметры;

Y – производственный национальный доход;

L – численность занятых в сфере материального производства;

F – основные производственные фонды.

При этом будем считать, что параметр α функции (9) изменяется во времени. Оценка параметра α производится по приведенной в настоящей статье методике. Для этого уравнение (9) необходимо сперва линеаризовать. Разделим обе части исходного уравнения (9) на F и логарифмируем:

$$\ln(Y/F) = \alpha \ln(L/F) + \ln A. \qquad (10)$$

Используем метод оценки линейных моделей с изменяющимися параметрами. Сперва оценивается следующее уравнение

$$\ln(Y/F) = (\alpha + \Delta\alpha t)\ln(L/F) + \ln A. \qquad (11)$$

Получаем следующие значения параметров:

$$\alpha = 0{,}504$$
$$\Delta\alpha = -0{,}005$$

Затем произведем адаптивную корректировку параметров и получим картину эволюции параметра α (см. таблицу I). Далее выявим функцию изменения этого параметра во времени

$$\alpha(t) = \sqrt{0{,}1933 - 0{,}0022\,t}\,.$$

После статистической оценки всех параметров получим производственную функцию следующего вида:

$$Y = L^{\sqrt{0{,}1933-0{,}0022t}}F^{1-\sqrt{0{,}1933-0{,}0022t}}.$$

Литература

I. Лукашин Ю.П. Адаптивные методы краткосрочного прогнозирования. М., Статистика, 1979. 254 с.

2. Нунапуу Э.Х.-Т. Автоматическое восстановление монотонных зависимостей по эмпирическим данным. См. наст. сб., с. 57.

3. Дадаян В.С. Экономические законы социализма и оптимальные решения. М., Мысль, 1970, с. 328.

4. Vensel V. Tootmis- ja kasvufunktsioonid. Tln., Valgus, 1979. 198 lk.

5. Brown R.L., Durbin J., Evans J.M.

Techiques for testing the constancy of regression relationships over time. - J. of the Royal Statistical Soc., ser. B., 1975, vol. 37, N 2, pp. 149-192.

6. N a u R.F., O l i v e r R.M. Adaptive filtering revisited. - J. Oper. Res. Soc., vol. 30, N 9, pp. 825-831.

7. M a k r i d a k i s S., W h e e l w r i g h t S.C. An examination of the use of adaptive filtering in forecasting. - Oper. Res. Quaterly, vol. 24, pp. 55-64.

E. Õunapuu

An Estimation of Time-Varying Parameter Models

Summary

In this paper the unadequateness of constant parameter models of some time series is exposed. A method for estimating time-varying parameter models is presented. An example of dynamic production function estimation is given.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 330.115

Э.Х.-Т. Кунацуу

АВТОМАТИЧЕСКОЕ ВОССТАНОВЛЕНИЕ МОНОТОННЫХ ЗАВИСИМОСТЕЙ ПО ЭМПИРИЧЕСКИМ ДАННЫМ

1. Введение

На практике социально-экономического анализа нередко возникает необходимость выявления аналитической зависимости между парами переменных $y = f(x)$. При этом часто отсутствуют теоретические соображения о том, каким должен быть конкретный вид формулы. Целью настоящего исследования является разработка методики и алгоритма для выявления конкретного вида зависимости между парами переменных и выработка устойчивой оценки параметров выявленной зависимости. Предлагаемая методика основывается на преобразованиях зависимой или (и) независимой переменной, в результате которых функциональная связь между ними становится линейной.

2. Обзор проблемы

Рассмотрим линейную модель регрессии

$$Y = XB + \varepsilon,$$

где Y – вектор наблюдений;
X – матрица независимых переменных;
B – вектор оцениваемых параметров;
ε – вектор ошибок измерений (остатков).

При правильном выборе модели, остатки должны быть независимыми и подчиняться нормальному распределению с нулевым средним и постоянной дисперсией. Однако на практике это далеко не всегда так. В этих случаях делу может помочь нелинейное преобразование данных.

Идея использования преобразований предложена в работах Тьюки [1], Бокса и Кокса [2]. Тьюки предложил следующее семейство преобразований зависимой переменной

$$y^{(\lambda)} = \begin{cases} y^{\lambda} & \lambda \neq 0, \\ \log y & \lambda = 0. \end{cases} \qquad (1)$$

Он изучил топологию этих преобразований для случая $|\lambda| \leq 1$. Позднее в работе [3] он рассмотрел использование преобразований на обширном конкретном материале. В работах Тьюки основой для выбора преобразований является изучение остатков.

Чтобы избежать разрыва при $\lambda = 0$, Бокс и Кокс [2] рассмотрели модифицированное семейство

$$y^{(\lambda)} = \begin{cases} \dfrac{y^{\lambda}-1}{\lambda} & \lambda \neq 0, \\ \log y & \lambda = 0. \end{cases} \qquad (2)$$

Как показал Шлесселман [4], семейство (1) по существу идентично (2), если модель регрессии содержит постоянную составляющую B_0.

Бокс и Кокс [2] ввели функцию правдоподобия, которая в случае линейной регрессии имеет следующий вид

$$(2\pi\sigma^2)^{-(\frac{1}{2})n} \cdot \exp\left\{-\frac{1}{2\sigma^2}(y^{(\lambda)}-XB)'(y^{(\lambda)}-XB)\right\}J, \qquad (3)$$

где $$J = \prod_{i=1}^{n} \frac{dy_i^{(\lambda)}}{dy_i} = \prod_{i=1}^{n} y_i^{\lambda-1},$$

и является абсолютной величиной якобиана преобразований. Как известно из [12], максимальное значение этой функции правдоподобия равно $(2\pi\hat{\sigma}^2)^{-(\frac{1}{2})n} e^{-(\frac{1}{2})n} J$,

где $$n\hat{\sigma}^2 = y^{(\lambda)}(I_n - X(X'X)^{-1}X')\ y^{(\lambda)} = S(\lambda; y).$$

Поэтому с точностью до константы отношение максимального правдоподобия равно

$$L_{max}(\lambda) = -\tfrac{1}{2}n\log S(\lambda, y) + (\lambda-1)\Sigma \log y_i. \qquad (4)$$

Бокс и Кокс [2] предлагают определить значение $\bar{\lambda}$, максимизирующее это отношение правдоподобия, непосредственно

по графику зависимости $L_{max}(\lambda)$ от λ.

Интересный метод для выбора преобразования независимой переменной предложили Бокс и Гидвелл [5]. Основой в их методе является разложение X^{λ} в ряд Тейлора.

$$X^{(\lambda)} = X^{\lambda_0} + (\lambda - \lambda_0) X^{\lambda_0} \log X. \qquad (5)$$

Если мы уже имеем какое-то приближение для λ, например, полагаем, что степень должна находиться около единицы, то используя формулу (5), можно оценить уравнение линейной регрессии

$$y = BX + B(\lambda - \lambda_0) X \log X = BX + \gamma X \log X,$$

параметры которой B и γ определяют степень $\bar{\lambda}$. Названную процедуру можно итерировать, получая на каждом шаге все более точное приближение для $\bar{\lambda}$.

При выборе преобразований необходимо обратить внимание еще на одну сторону вопроса, а именно, на робастность процедуры оценивания. Статистическую процедуру принято считать робастной, когда она сохраняет эффективность при отклонении от предпосылок, принятых за основу оценки. Например, оценки метода наименьших квадратов весьма чувствительны к наличию выбросов среди обрабатываемых данных. Многие исследования [6, 7, I5] позволяют сделать вывод, что на практике 5–I0 % аномальных значений в общей массе данных – это скорее правило, чем исключение.

Метод Бокса и Кокса подвергался критике за его чувствительность к аномалиям в распределении обрабатываемых данных. Эндрьюс [8] предложил метод, который базируется на F-критерии и является сравнительно несувствительным к выбросам. Но Аткинсон [9], используя эксперименты Монте-Карло, показал, что при нормальном распределении метод Бокса и Кокса является более мощным, чем метод Эндрьюса. В этой ситуации оказалось, что методы, которые при нормальной модели являются мощными, не обладают робастностью и, наоборот, те, которые являются робастными, не так мощны при нормальной модели.

В настоящее время Кэролл [I0] предложил метод, который обладает достаточной мощностью и работоспособностью.

В методе Кэролла центральное место, как и в методе Бокса и Кокса, занимает функция максимального правдоподобия

$$L(B,\sigma,\lambda)=\sigma^{-n}\sum_{i=1}^{n}\exp\left\{-\psi\left(\frac{y_i^{(\lambda)}-X_iB}{\sigma}\right)+(\lambda-1)\log y_i\right\}. \qquad (6)$$

Для заданного λ находятся $\hat{B}(\lambda)$ и $\hat{\sigma}(\lambda)$ стабильным методом регрессии [II], находится такое значение λ, при котором функция (6) достигает максимума.

3. Автоматизация поиска линеаризующих преобразований

Широкий класс нелинейных зависимостей можно привести к линейным, используя преобразования зависимой, а возможно, и независимой переменной. Семейства преобразований, которые подходят для этих целей, приводились выше (I, 2).

В настоящей работе автоматический подбор преобразований производится поэтапно.

Первый этап базируется на изучении остатков методами Тьюки и позволяет осуществить быстрый поиск начального приближения. На следующем этапе производится уточнение полученного результата по критерию максимального правдоподобия Бокса и Кокса. Для придания используемым процедурам устойчивости используются два приема:

I) исследуемые последовательности перед обработкой сглаживают;

2) при оценке регрессионных уравнений используются устойчивые методы [II].

Итак, автоматическая процедура поиска преобразований состоит из следующих шагов:

I. Сглаживание последовательности.

2. Поиск начального приближения методом Тьюки.

3. Уточнение преобразования по критерию максимального правдоподобия Бокса и Кокса (4) методами оптимизации.

3.I. Процедура сглаживания последовательностей

Основным определяющим компонентом предлагаемой методики является процедура сглаживания (фильтрации). Для сглаживания зависимые переменные y_i, $i=1,n$ сначала упорядочивают по значению независимой переменной x_i. Затем для каждой y_i, $i=2,\ldots,n-1$ применяется метод скользящего медианного сглаживания, при этом y_i заменяется медианой от $y(i-1)$, $y(i)$ и $y(i+1)$. Открытым остается вопрос, как поступить с крайними точками $y(1)$ и $y(n)$. Для них используется правило, предложенное Тьюки в [3]. Крайний член $y(1)$ заменяют медианой от $y(1)$, $y(2)$ и $3y(2)-2y(3)$. Аналогично $y(n)$ заменяют медианой от $y(n-1)$, $y(n)$ и $3y(n)-2y(n-1)$. Правило крайних точек вытекает из следующих соображений. Наклон прямой, соединяющей вторую и третью точки, не должен резко отличаться от наклона прямой, проходящей через первую и вторую точки. При этом естественно возможны и некоторые различия наклонов. Принимая крайним значением медиану от $y(1)$, $y(2)$ и $3y(2)-2y(3)$, наклон может изменяться до двух раз.

Описанную процедуру сглаживания повторяют до тех пор, пока в сглаживаемой последовательности не происходит изменений, т.е. пока процедура не сходится.

Описанная процедура сглаживания обладает одной особенностью, а именно, в точках, где находится локальный минимум или максимум последовательности, образуются плоские области, в которых элементы имеют одинаковые значения. Процедура сглаживания как бы срезает пики максимумов и минимумов. Во многих случаях такое поведение процедуры сглаживания неприемлемо. Чтобы устранить названную особенность, последовательность делят в локальных минимумах и максимумах на части и используют правило крайних точек для обеих сторон разреза. Опять-таки процедуру повторяют до схождения.

Следует отметить, что в процессе сглаживания из последовательности устраняют случайные выбросы. Причем величина выбросов не влияет на результат сглаживания.

3.2. Выбор преобразования на основе изучения остатков

Как было уже сказано, Тьюки предложил семейство преобразований

$$y^{(\lambda)} = \begin{cases} y^{\lambda} & \lambda \neq 0, \\ \log y & \lambda = 0. \end{cases} \qquad (7)$$

Для выбора преобразований данные необходимо разложить на основе равенства:

$$y_i = f(x_i, \lambda) + \varepsilon_i ,$$

где f – является функцией, на основе которой получают оценки для y_i;

ε_i – остатки, т.е. величины, которые нельзя объяснить с помощью функции f.

Если функция f хорошо аппроксимирует данные, то остатки должны быть независимыми, иметь нулевое среднее, одинаковую (постоянную) дисперсию σ^2 и подчиняться нормальному распределению. Отклонение от этих требований указывает на неправильность выбираемой модели. Изучение остатков дает нам ключ к выбору преобразований.

Для выбора преобразований самым информативным является изучение распределения остатков в зависимости от оцененного значения $\bar{y}_i$.

Подбор требуемого преобразования происходит по следующему правилу. Выбирается начальное значение для λ. Изучается график остатков. Если график остатков имеет вогнутость вниз, то при выборе преобразования надо двигаться в направлении больших значений λ, в противном случае – меньших. Открытым остается вопрос, какой длины шаг должен быть.

Шкала преобразований (7) является непрерывной, но в ней можно выделить ряд опорных пунктов. Например, двигаясь от значения $\lambda = 1$ к меньшим значениям λ, можно выделить следующие опорные пункты:

$$y,\ \sqrt{y},\ \log y,\ y^{-\frac{1}{2}},\ y^{-1},\ \ldots$$

Исходя из вышесказанного, можно выбрать следующую тактику поиска преобразований:

I. Двигаясь по опорным пунктам определяются нижние и верхние границы требуемого преобразования.

2. Затем происходит дальнейший поиск преобразования в названных границах. Поиск ведется до получения случайного распределения остатков или достижения заданной точности.

3.3. Уточнение преобразования методами оптимизации

Для уточнения значения степени λ используется функция максимального правдоподобия Бокса и Кокса (4). Поиск максимума функции правдоподобия производится методом, приведенным в [I4]. Названный метод относится к прямым методам оптимизации, т.е. к методам, где не используются производные. Эти методы оправдывали себя в ситуациях, когда интервал поиска сравнительно узок. В основе используемого алгоритма лежит комбинация метода золотого сечения и последовательной параболической интерполяции.

Начальными данными для алгоритма являются нижняя и верхняя границы интервала поиска, максимизируемая функция и граница погрешности. Нижняя и верхняя границы интервала поиска определялись на предыдущем шаге методики. Максимизируемая функция, которой в данном случае является функция максимального правдоподобия, задается в виде подпрограммы, которая вычисляет ее значение при данном аргументе λ. При вычислении функции правдоподобия используются робастные методы [II].

4. Реализация процедур автоматического поиска преобразований

Предложенная процедура поиска преобразований реализована в виде модулей, написанных на Фортране и включенных в состав системы анализа данных, которая разработана и эксплуатируется в Таллинском политехническом институте.

Процедуры успешно применялись для решения многих практических задач, в том числе для анализа загрязненности в малых равнинных реках (на примере Эстонской ССР) и для выявления функций рождаемости и смертности в Эстонской ССР.

Литература

1. Tukey J.W. On the comparative anatomy of transformations. - Ann. Math. Stat., 1957, vol. 28, pp. 602-632.

2. Box G.E.P., Cox B.R. An analysis of transformations. - J. R. Stat. Soc., 1964, ser. B., vol. 26, pp. 211-252.

3. Tukey J.W. Exploratory data analysis. New York, Addison-Wesley, 1977.

4. Schlesselman J. Power families: a note on the Box and Cox transformation. - J. R. Stat. Soc., 1971, ser. B., vol. 33, pp. 307-311.

5. Box G.E.P., Tidwell P.W. Transformation of the independent variables. - Technometrics, 1962, vol. 4, pp. 531-550.

6. Tukey J.W. The future of data analysis. - Ann. Math. Stat., 1967, vol. 33, pp.1 - 67.

7. Box G.E.P., Andersen S.L. Permutation theory in the derivation of robust criteria and the study of departures from assumption of normality - J. R. Stat. Soc., 1955, ser. B., vol. 17, pp. 1-34.

8. Andrews D.F. A note on the selection of data transformations. - Biometrika, 1971, vol. 58, pp. 249-254.

9. Atkinson A.C. Testing transformations to normality. - J. R. Stat. Soc., 1973, ser. B., vol. 35, pp. 473-479.

10. Carroll R.J. A robust method for testing transformations to achieve approximate normality. - J. R. Stat. Soc., 1980, ser. B., vol. 42, pp. 71-78.

11. Huber P.J. Robust statistical procedures. SIAM. Philadelphia, 1977.

12. Себер Дж. Линейный регрессионный анализ. М., Мир, 1980.

13. Ершов А.А. Стабильные методы оценки параметров (обзор). Автоматика и телемеханика, 1978, № 8, с. 66-100.

I4. Форсайт Дж., Мальколь м М., Моулер К. Машинные методы математических вычислений. М., Мир, I980.

Е. Õunapuu

Automatic Curve Fitting of Monotonic Functions

Summary

In this paper the usage of data transformations for fitting the nonlinear functions is considered. Special attention is paid to robustness of the procedures.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3

Л.К. Выханду, М. Раху, Х. Хурт

ЭСТОНСКИЙ РЕГИСТР РАКА: ТЕХНОЛОГИЯ ОБРАБОТКИ ДАННЫХ

В данной работе дается краткое описание некоторых процедур Эстонского регистра рака (ЭРР) при хранении и обработке данных за 1976-1980 гг. ЭРР опирается на автоматизированную систему хранения и обработки дискретной информации (СОДИ), реализованную на ЭВМ "Раздан-3". Рассматриваемые вопросы охватывают централизацию информации, подготовку данных к обработке, структуру базы данных, автоматизированный контроль качества данных, виды выходной информации и перспективы развития регистра.

I. Введение

Регистрация рака (злокачественных опухолей) представляет собой деятельность, при которой централизуется выборочная информация о всех диагностированных случаях заболевания на точно определенной территории. Дальнейшая судьба централизованной информации с точки зрения ее обработки зависит от многих обстоятельств: количество данных, наличие кадров, заказов потребителей, вычислительной техники и прочее. В регистрах рака используется ручная, механическая или электронная обработка информации.

В Эстонии учет больных раком был более или менее централизован к 1967 г. Вплоть до 1971 г. ручной способ обработки документации был единственным. Попытка к переходу обработки данных на ЭВМ была предпринята в 1971-1975 гг. по договору между Республиканским Таллинский онкологическим диспансером (РТОД) и ВЦ ЦСУ ЭССР. Попытка оказалась неудачной, но поучительной: наряду с определенной ограниченностью проектного задания четко выявилась решающая роль системы обработки. Использованной системой были введены не-

удобные предписания для перфорации, исправление массива данных проводилось нерационально, а запрос любых неотчетных таблиц требовал непрерывного дополнительного программирования и финансирования [3].

Единственным разумным выходом из положения было привлечь созданную в 1969-1970 гг. коллективом кафедры вычислительной математики (ныне кафедра обработки информации) ТПИ систему СОДИ, реализованную на ЭВМ "Раздан-3"[2, 5]. Первым шагом в этом направлении стал перенос информации приблизительно о 26 тысячах случаев рака, сохраняемой на МЛ ЭВМ "Минск-32" (ВЦ ЦСУ ЭССР) на МЛ "Раздан-3" (ВЦ Эстонского Радио). Ввиду технических причин перенос осуществлялся постепенно - сначала на МЛ ЭВМ ЕС-1022, затем на МЛ ЭВМ 1010 В, а оттуда по внутренней линии связи в память "Раздан-3". Таким образом, начиная с 1976 г. на базе СОДИ действует система сбора, хранения и обработки онкологических статистических данных. Успешное функционирование системы и явилось толчком к официальному созданию Эстонского регистра рака в январе 1978 г. Последующее описание некоторых процедур регистра отражает состояние системы к концу 1980 г.

2. Главная перфотека ЭРР

Основным документом при регистрации рака является "Контрольная карта диспансерного наблюдения". ЭРР использует модифицированную контрольную карту в виде ручной перфокарты (формат 207x297 мм) с верхней краевой перфорацией.

Применяемый в настоящее время вариант карты был введен в начале 1978 г. с учетом всесоюзных и международных[I] инструкций по регистрации рака. На карту вписывается основная информация о случае заболевания раком, поступающая из разных источников. Цветными рейтерами, прикрепляемыми к верхнему краю перфокарты, обозначается ряд признаков, нужных для ежедневной работы с картами.

[I] Для внедрения многих признаков по [7] получена субсидия от Всемирной организации здравоохранения (договор CAN/C2/181/92).

Главная перфотека находится в кабинете статистики РТОД. Большинство перфокарт имеет дублет, сохраняемый в т.н. местном регистре рака (например, в онкологическом отделении или кабинете).

Данные, содержащиеся на картах главной перфотеки, кодируются в кабинете статистики. На дублеты коды не переносятся. Карта главной перфотеки служит единицей ввода, хранения и выдачи информации на основе базы данных ЭВМ.

3. Подготовка данных к вводу в ЭВМ

После кодирования контрольные карты передаются на перфорацию. Перфорация проводится на перфоленту при помощи телетайпа. Всю ответственность за качество перфорирования несет кабинет статистики РТОД, где помещен телетайп. Соблюдение общего принципа, при котором о подготовке данных к вводу в ЭВМ заботится заказчик, а не ВЦ другого ведомства, оказалось эффективным решением и в работе ЭРР.

Любые дополнения (и/или исправления) данных на контрольной карте одновременно записываются на специальный бланк. Для последующего автоматического слияния основных и дополнительных записей на МЛ, на бланке повторяются регистрационный номер больного и номер диагноза. При необходимости добавляется еще код признака, указывающего на наличие первично-множественных опухолей у больного. Впоследствии бланки с кодами поступают на перфорацию, а затем перфоленты передаются в ВЦ Эстонского Радио.

4. Структура базы данных

По состоянию на декабрь 1980 г. на МЛ "Раздан-3" отдельными массивами сохранялись следующие данные ЭРР:

1. Больные, взятые на учет и снятые с учета в 1978-1979 гг. - 11 083 объекта (карт).

2. Больные, находящиеся под диспансерным наблюдением по состоянию на 31 декабря 1979 г. (за исключением зарегистрированных в 1978-1979 гг.) - 14 445 объектов.

3. Больные, снятые с учета в 1976-1977 гг. - 7 529 объектов.

4. Больные, снятые с учета в 1973-1975 гг. - 5 114 объектов.

Целью разделения базы данных на подобные массивы является, главным образом, экономия времени при составлении годовых отчетов. Четвертый массив - это наследие ВЦ ЦСУ ЭССР: объекты в нем с ошибками и нередко имеют дублеты. По мере проверки и исправления каждого объекта в четвертом массиве на основе главной перфотеки и других источников этот массив соединяется с третьим массивом.

Хранимые на МЛ объекты относятся к разным периодам времени. Поэтому, например, объект, зарегистрированный в 1974 г., характеризовался гораздо меньшим количеством признаков, чем объект 1979 г. Используя аппаратуру СОДИ для создания новых признаков [4], все старые версии перфокарты были видоизменены и приведены в соответствие со структурой новой карты 1978 г. Ранее не учтенным признакам было присвоено значение "нуль" ("неопределенность").

Один объект в базе данных имеет 599 признаков. Поскольку признаки упакованы, такой объект занимает 62 машинного слова.

Признаки №№ 560-599 были выделены в качестве резерва для новых признаков, образуемых на основе существующих. В зависимости от необходимости, новый признак создается постоянным (сохраняется на МЛ) или временным (сохраняется только во время обработки). Среди постоянных новых признаков назовем "возраст при взятии на учет", "возрастная группа", "длительность выживания" (в месяцах). Аналогично временные признаки предусмотрены для дополнительной характеристики объектов при выборе их для таблиц.

Благодаря гибкости СОДИ, все изменения, связанные с расширением объекта и отдельных его признаков, легко реализуются в системе. Необходимо только описать новые максимальные значения признаков и связи между значениями признаков - и в базе данных автоматически перераспределяется структура объекта. Так, запланированный ЭРР переход на новую Международную классификацию болезней (МКБ-онкология, топографический раздел) и ожидаемое вместе с тем требование предъявлять часть отчетов по рубрикам МКБ-9 не требует дополнительного программирования.

5. Автоматизированный контроль качества информации

Автоматизированный контроль качества информации ЭРР проводится в два этапа: при вводе и после ввода данных в ЭВМ.

При вводе новых объектов проверяется синтаксис. Объекты, в которых обнаружена ошибка, не записываются на МЛ, они выводятся на широкую печать со всеми значениями признаков и словесным описанием ошибки.

Аналогичный контроль синтаксиса имеет место и при вводе дополнений.

На втором этапе проводится содержательный логический контроль, в ходе которого сопоставляются признаки одного и того же объекта, а также сравниваются признаки разных объектов. Здесь, как на первом этапе, используется аппаратура логических условий СОДИ [4]. В настоящее время для проверки правильности данных одного объекта ЭРР применяет более 80 тестов, выраженных при помощи языка логических условий.

Среди прочих тестов при проверке внутренней совместимости объекта проверяется соответствие сельского совета и сельского административного района в адресе, наличие недопустимых номеров в четырехзначных рубриках МКБ-8 и в морфологическом разделе МКБ-онкология, принадлежность к клинической группе в зависимости от объективного состояния больного.

Все тесты пронумерованы. Если при проверке правильности объектов найдены ошибки, то эти объекты полностью выводятся на широкую печать с указанием соответствующих номеров тестов. По напечатанным объектам, каталогу тестов (где приведены символическая и словесная характеристики каждого вида ошибки), перфокартам и, при необходимости, исходным медицинским документам проводится корректировка.

В начале 1980 г., когда перед составлением отчетов двухэтапному контролю подвергались объекты, зарегистрированные ЭРР в 1979 г., у 14 % объектов были обнаружены ошибки. По количеству ошибок эти объекты распределялись сле-

дующим образом: одна ошибка - 76 %, две - 18 %, три - 4 %, четыре и больше - 2 %. Из всех ошибок 42 % были пропущены при перфорировании, 58 % при заполнении и кодировании контрольной карты диспансерного наблюдения.

Сравнению подлежат и регистрационные номера (ошибкой считается, если один и тот же номер присвоен разным лицам) и ряд других признаков - фамилия, пол, дата рождения, национальность, место жительства, диагноз - разных объектов (для выявления дублетов). Если у одного и того же лица две или больше опухолей, то проверяется, имеют ли соответствующие объекты сходные значения следующих признаков: фамилия, пол, дата рождения, место рождения, национальность, место жительства, субъективное состояние, дата и причина снятия с учета, дата, место и причина смерти.

Имеющимися тестами исчерпываются далеко не все возможности автоматизированного контроля качества базы данных. Список тестов нельзя считать окончательным - он постоянно дополняется в ходе работы ЭРР.

6. Выходная информация

Наиболее непосредственное и понятное соприкосновение с главным оружием СОДИ - нами уже упомянутым языком логических условий - имеет пользователь системы при требовании выходной информации. На основе опыта работы с базой данных ЭРР следует здесь выделить следующие достоинства системы:

1. Заказчик (научный сотрудник, практический врач) может самостоятельно описать желаемые таблицы и списки с помощью логических условий, правила записи которых легко усваиваются.

2. Предусмотренный СОДИ способ выполнения арифметических действий над элементами таблиц позволяет в "обычном" порядке вычислить и включить в таблицу разного рода статистики, принятые в онкологии.

3. При изменении формы и содержания отчетов заказчик просто задает описание новых отчетов.

4. СОДИ функционирует как ИПС.

5. Возможность содержательного описания запросов заказчиком и способность ЭВМ на основе запросов самостоятельно программировать вывод нужной информации исключает рутинное программирование и ведет в конечном итоге к большой экономии денежных средств.

Нужная для ЭРР выходная информация делится на две группы - рутинная и нерутинная информация. Рутинная информация выдается к более или менее строго определенным срокам по формам, ранее предусмотренным всесоюзными или местными требованиями. Сюда относятся представляемые Министерству здравоохранения ЭССР годовые отчеты, предназначенные местным регистрам рака, годовые отчеты и списки больных, таблицы для (главных) специалистов и для стандартных публикаций ЭРР. Описания таких отчетных и неотчетных таблиц хранятся на МЛ, а для составления их следует задавать запрос, указав номер таблицы и логическое условие с датой (датами). При желании на базе записанного на МЛ можно автоматически составлять новые отчеты, дополнительно указывая в запросе, из каких таблиц следует выбрать логические условия, описавшие строки и столбцы новой таблицы.

К рутинной выходной информации относится и распечатка объектов во время контроля качества.

Нерутинная информация предназначена для удовлетворения остальных запросов, частота или непредвиденность которых не позволяет вносить их в список стандартных заказов. В настоящий момент подобная информация чаще всего нужна для текущей работы по улучшению регистрации рака в Эстонии и для научных разработок.

Как правило, заказываемые в нерутинном порядке таблицы не нуждаются в сопроводительных текстах, что в определенной мере упрощает и ускоряет их выдачу.

Приведем простой пример. Нужна таблица, отражающая в абсолютных числах и процентах распределение по полу (признак I5) и семейному положению (2I) впервые зарегистрированных (5.I) в Эстонии в I978-I979 гг. (7.78; 7.79) больных раком желудка (42.I5I). Логическое условие для таблицы задается в конъюктивной нормальной форме

K5.I.Λ42.I5I.Λ7.78.V7.79.Э
C I5
R 2I
Таблица имеет следующий вид:

	1:	0:	1:	2:	3:	4:	:
	0:	:	:	:	:	:	:
PER	:	:	:	:	:	:	:
PER R:		:	:	:	:	:	:
PER C:		:	:	:	:	:	:
	1:	27:	518:	32:	38:	18:	633:
PER	:	272:	42,9:	2,6:	3,1:	1,5:	52,4:
PER R:		473:	84,8:	5,1:	6,0:	2,8:	:
PER C:		4279:	67,1:	50,0:	13,7:	54,6:	:
	2:	36:	254:	32:	239:	15:	576:
PER	:	370:	21,0:	2,6:	19,8:	1,2:	47,6:
PER R:		673:	44,1:	5,6:	41,5:	2,6:	:
PER C:		5711:	32,9:	50,0:	86,3:	45,5:	:
	:	63:	772:	64:	277:	33:	1209:
	:	572:	63,9:	5,3:	22,9:	2,7:	SUM:

Видно, например, что из общего количества 633 мужчин у 4,3 % семейное положение не указано, 8I,8 % женаты, 5,I % холостяки, 6,0 % вдовы и 2,8 % разведены.

Все выходные данные ЭВМ, полученные ЭРР, разделяются по форме следующим образом:

I. Таблицы. В зависимости от цели, они выводятся с текстом или без текста, имеют более или менее сложную структуру.

2. Списки. Включают выборочные признаки, число и порядок следования которых определены назначением. Объекты расположены в алфавитной расстановке фамилий.

3. Диаграммы. Сначала заказывается таблица, потом она визуализируется диаграммой.

4. Картограммы и картосхемы. Отражают пространственное распределение абсолютных и относительных величин на территории Эстонии. Картограммы заболеваемости раком построены по методу вероятностного картографирования [5].

Нужно добавить, что объем рутинной информации нельзя считать достаточным: он должен увеличиваться по мере улучшения качества данных и уточнения заказов от врачей и научных сотрудников.

7. О дальнейшем развитии системы

Действующая на базе СОДИ система сбора, хранения и обработки онкологических статистических данных не является пассивным производителем таблиц и прочих выходных форм. В ходе внедрения системы и по мере накопления опыта работы складывалась четко прослеживаемая обратная связь с методологией и методикой регистрации рака.

Заказчик, нуждающийся в описании желаемой таблицы с помощью логических условий, должен четко представить себе содержание запроса. Нередко такое дополнительное размышление над предметом выявляет потребность в более четком определении регистрируемых в рутинном порядке признаков. Кроме того, возможность быстро получить из базы данных ответы на довольно сложные вопросы, оставляет больше времени для рассуждений о качестве накапливаемой информации.

Наряду с необходимостью ликвидировать ряд давно известных недостатков организационного порядка в регистрации рака, перспективы развития ЭРР непосредственно зависят и от обновления технической базы. Обратная связь базы данных с диспансеризацией больных была бы гораздо эффективнее при переходе на режим работы в реальном масштабе времени. Это дало бы, прежде всего, возможность ввести в ЭВМ даты вызова больного и последнего контакта с ним и, таким образом, лучше следить за работой местных регистров. Целесообразным был бы переход на ЭВМ Единой Серии, так как для них создана последняя версия автоматизированной системы программ хранения и обработки дискретной информации – СХОДИ [3].

В то же время залог успеха деятельности ЭРР не зависит только от улучшения организации и технической базы регистрации. Наша система не может развиваться изолированно от других систем. Уже сейчас нуждаемся в автоматизированном регистре населения и в базе данных, хранящей информацию

свидетельств о смерти. Более рациональному развертыванию эпидемиологических исследований рака и других заболеваний содействовало бы наличие регистров в других службах, занимающихся массовой диспансеризацией больных. Однако как установлено [I], специализированные медицинские службы - психиатрическая, противотуберкулезная и кожно-венерологическая - еще не готовы приступить к автоматизированной обработке данных.

Действующая в Эстонии с 1976 г. система сбора, хранения и обработки онкологических данных является в данный момент единственной системой в СССР, имеющей в своем распоряжении пакет прикладных программ для удобного и рационального манипулирования информацией о больных раком на уровне популяционного регистра. Хотя многие процедуры системы нуждаются в усовершенствовании, приобретенный опыт может оказаться полезным при создании регистров отдельных заболеваний как в Эстонии, так и в других регионах СССР.

Литература

I. Аарелейд Т.П. Сравнительная оценка готовности к автоматизированной обработке данных онкологической и других специализированных медицинских служб Эстонии. - Экспериментальная и клиническая онкология, вып. 4, Таллин, ИЭКМ, 1981, с. 173-177.

2. Выханду Л.К. Архитектура системы "СОДИ". Опыт использования ЭВМ в исследованиях культуры. - НИИкультуры, Труды 32. М., 1976.

3. Пурде М.К., Раху М.А. Усовершенствование системы учета заболеваемости и смертности от злокачественных опухолей в ЭССР (заключительный отчет). Таллин, ИЭКМ МЗ ЭССР, 1975. 21 с. № Госрегистрации 74056810.

4. Система хранения и обработки дискретной информации (СХОДИ). Сост. Выханду Л.К., Микли Т.И., Тепанди Я.Я. и др. Таллин, ТПИ, 1980. 38 с.

5. Труды Вычислительного Центра, вып. 30, Тарту, ТГУ, 1974.

6. R a h u, M. Some Aspects of Nosogeographical Mapping. - Estonia. Regional Studies. Tallinn, 1976, 176-180.

7. WHO Handbook for Standardized Cancer Registries. Geneva, WHO, 1976.

L. Vyhandu, M. Rahu, H.Hurt

Estonian Cancer Registry: Technology of Data Processing

Summary

This paper concentrates on some procedures of the Estonian Cancer Registry in data storing and processing in 1976-1980. The system bases on SODI (Automated System for Discrete Information Storage and Processing) planned for a Rhazdan-3 computer. Problems related to data centralization, input preparing, data base structure, quality control, output and future developments are described.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.142

А.О. Вооглайд, М.В. Лепп, Д.Б. Лийб

ВХОДНЫЕ ЯЗЫКИ СИСТЕМЫ ELMA

1. Введение

Описанная в данной статье инструментальная система является четвертой версией системы, выросшей из универсального анализатора, реализованного более десяти лет назад. Данная система ELMA вобрала в себя опыт создания инструментальных средств на трех разных типах ЭВМ. Названная инструментальная система позволяет проектировать пакеты прикладных программ (ППП) и принадлежащие им проблемно-ориентированные языки с соответствующими трансляторами. В результате проектирования ППП получается описание пакета на метаязыке. Далее из этих метаописаний система ELMA автоматически генерирует прикладные программы.

С помощью данной системы реализовано около 40 проблемно-ориентированных языков с соответствующими трансляторами и универсальных генераторов ввода данных.

Наиболее крупной, реализованной в последнее время на базе системы ELMA, является система ведения баз данных и манипулирования данными, называемая ПАРЕС [3].

1.1. Единый идеологический базис системы ELMA

Вся система и создаваемый ею продукт построены по единому принципу - на любом уровне представления алгоритма строго разграничиваются структура управления (СУ) и выполняющие операторы. Это дает возможность использовать при проектировании системы технику "оптимизации на уровне алгоритма" и при реализации - каскадную технику [5].

Принцип расслоенности метаописания

Все метаописания в системе ELMA можно составлять по частям - по принципу сверху вниз, а затем автоматически генерировать в единое описание.

I.2. Структура управления

В системе ELMA все описания и реализации проводятся по технологии, которая представляет собой развитую идею структурного программирования Джексона [I3].

В системе можно использовать четыре разных способа определения СУ. Статическая СУ соответствует структуре, введенной Джексоном. Динамическая СУ - это статическая СУ, значение каждой компоненты которой можно динамически изменить в атрибутной технике во время выполнения пакета.

Под компонентами подразумеваются компоненты структуры Джексона, выполняемые операторы и предикаты, связанные с конкретным компонентом структуры. В ходе решения доступ к компонентам достигается с помощью фиксированных атрибутов. Система управления, определенная входными данными - этот способ аналогичен описанному в работах [II, I5]. По существу это означает описание СУ грамматическими формализмами, причем само управление осуществляется синтаксически управляемой трансляцией. Синтезированная СУ - нахождение СУ пакета осуществляется с помощью системы ПРИЗ [2].

I.3. Система построения трансляторов

При реализации проблемно-ориентированных языков применяются следующие технологии.

Во-первых, описывается абстрактный синтаксис создаваемого языка как множество неприведенных неоднозначных регуляризованных КС-грамматик [II, I7].

Во-вторых, семантика языка описывается в атрибутной технике. При этом семантические действия и атрибуты непосредственно связываются с абстрактным синтаксисом. Для этого создана теория абстрактных атрибутных грамматик [I0] и реализованы соответствующие инструментальные средства (ге-

нератор преобразователей из конкретного синтаксиса в абстрактный; генератор вычислителя, использующего синтезированные итеративные и простые атрибуты, инициализированные и унаследованные атрибуты, вычисление которых можно планировать статически, и глобальные атрибуты).

В-третьих, семантические действия отлаживаются отдельно (от конкретного анализатора) в интерпретирующем режиме, причем имитируется выход преобразователя из конкретного синтаксиса в абстрактный и среду атрибутов.

В-четвертых, описывается конкретный синтаксис.

В-пятых, описывается генерация кода.

При генерации кода создаваемого языка используется специальный язык ФОРТ. Данный язык позволяет создавать из старых понятий новые и во время решения выбирать нужный режим интерпретации или компиляции. Язык ФОРТ дает возможность использовать машинный код наряду со стандартными структурными конструкциями языка, полученными из машинного кода с помощью раскрутки. Для того, чтобы получить генерированный код в терминах машинного кода, после отладки транслятора следует провести антираскрутку языка ФОРТ.

I.4. Проходы создаваемого транслятора

Структура создаваемого транслятора состоит из разных, определяемых динамической СУ, последовательных прохождений. При этом каждое отдельное прохождение может быть одним из следующих вариантов:

а) $T_{Lex}^{\mathcal{L}i}\ T_{Syn}^{\mathcal{L}i}\ T_{Trans}^{\mathcal{L}i}\ T_{Sem}^{\mathcal{L}i}\ T_{Code}^{\mathcal{L}i}$;

в) все компоненты пункта а) в отдельности;

с) произвольное подмножество компонентов пункта а) сохраняя и их порядок, при этом начиная от внутрисистемного лексического анализатора.

В элементе $T_{x}^{\mathcal{L}i}\ \mathcal{L}_i$ обозначает язык

x - часть транслятора, созданного для этого языка, причем в роли x может выступать либо:

Lex - лексический анализатор,
Syn - синтаксический анализатор,
Trans - преобразователь конкретного синтаксиса в абстрактный,
Sem - семантический анализатор,
Code - генератор кода объектного языка,
Lexsys - внутрисистемный лексический анализатор.

Особо следует отметить, что при переходе от одного прохождения к другому имеется возможность динамической замены языка.

Для описания всех перечисленных действий разработаны метаязыки ELMAMETA и ELMAGUIDE и системные средства, которые из описаний автоматически генерируют текст прикладных программ на языке АССЕМБЛЕР или ФОРТРАН. Основными компонентами данной инструментальной системы являются системы построения трансляторов и система создания структур управления. Таким образом, систему ELMA можно успешно использовать при создании пакетов как для ЭВМ ЕС, так и для мини- и макро-ЭВМ.

2. ЯЗЫК УПРАВЛЕНИЯ

Язык ELMAGUIDE является языком спецификации системы ELMA, теоретический базис которого представляет собой развитую идею структурного программирования Джексона [13]. Язык ELMAGUIDE предназначен для применения в следующих целях.

а. Для автоматизации проектирования и производства ПШ. С этой целью разработаны средства, поддерживающие проектирование и реализацию пакетов по технологии сверху вниз.

б. Для проектирования и автоматической реализации транслирующих систем проблемно-ориентированных языков (ПОЯ). В данном случае язык ELMAGUIDE используется для дополнения описания ПОЯ до полного получения его транслирующей системы.

Описания на языках ELMAGUIDE и ELMAMETA совместно дают схему трансляции. При этом описания на языке ELMAMETA определяют подсхемы трансляции. Такие средства позволяют реализовать трансляторы с несколькими прохождениями.

в. Для структурного программирования при помощи грамматических формализмов. В данной возможности заключается идея описания входных данных с помощью формальных грамматик и манипуляции данными - с помощью семантических действий [II, I5]. С этой целью в язык ELMAGUIDE включена переменная, значением которой может быть описание на языке ELMAMETA.

В общем случае текст на языке ELMAGUIDE представляет собой описание управления, выполняемых действий и информационной среды пакета программ. В результате трансляции из этого описания генерируется организующая программа пакета [I] или транслятора, которая выполняется в интерпретирующем режиме, исходя из структуры управления, представляющей из себя дерево Джексона [I3, I5] и значений некоторых системных атрибутов.

Подробнее данные вопросы рассматриваются в статье [8].

2.I. Базис языка ELMAGUIDE

В данном пункте рассматриваются принципы, положенные в основу проектирования языка ELMAGUIDE. В целях сжатого изложения под объектом подразумевается организационная часть либо пакета программ, либо транслятора, либо генератора ввода [I].

а. Разграничение управления и выполняемых действий

Разграничение осуществляется как на уровне описания, так и на уровне выполнения программы, сгенерированной из данного описания.

Управление является той частью описания, которая определяет время и порядок выполнения действий. Действия же могут быть либо модулями на некотором языке программирования, либо описаниями на языке ELMAMETA или ELMAGUIDE.

б. Деление управления на две компоненты - структурную и функциональную

Структурная компонента (структура управления) представляет собой дерево Джексона объекта. Структура описывается при помощи правил регуляризованной контекстно-свобод-

ной грамматики [II, I7]. Таким образом, структурная компонента определяет бесконечное множество управлений. Функциональная компонента конкретизирует управление в ходе выполнения объекта. Более точно, функциональная компонента управляет выполнением итерации и выбора. Она представляет собой связанную структурой совокупность функций управления.

в. Динамическое изменение элементов объекта

В практике часто возникают задачи, при которых невозможно или неразумно статически описать все входящие в пакет действия. Необходимые действия часто зависят от хода решения задачи. В связи с этим введено понятие динамически изменяемого действия, а также динамически изменяемой функции управления. Изменение соответствующих элементов осуществляется при помощи самих действий.

г. Атрибутирование вершин структуры управления

Атрибуты, связанные с вершинами структуры - это системные глобальные атрибуты для динамических изменений, описанных в пункте 3, и для конкретизации управления. Каждая вершина имеет следующие атрибуты:

I) EXEC - атрибут логического типа, значение которого определяет участие подструктуры в управлении;

2) SEL - атрибут целочисленного типа для осуществления выбора;

3) LOG - атрибут типа ссылки на предикат, значение которого определяет предикат; присваивающий значение атрибуту значение EXEC;

4) FNSEL - атрибут типа ссылки на функцию выбора, значение которого определяет функцию, присваивающую значение атрибуту SEL;

5) ACT - атрибут типа ссылки на выполняемое действие, значение которого определяет выполняемое в данной вершине действие.

д. Параметризированный буферный интерфейс

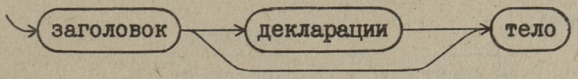
Интерфейс между выполняемыми действиями, осуществляется на системном буфере. Настраивание входной и выходной

информации действия на буфер обеспечивается с помощью параметров.

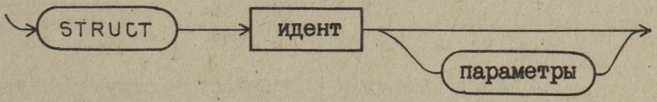
2.2. Синтаксис и семантика языка ELMAGUIDE

Синтаксис языка ELMAGUIDE описывается при помощи схем, использованных для описания синтаксиса языка PASCAL в [I6].

объект:

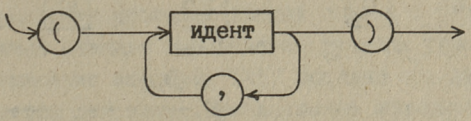


заголовок:

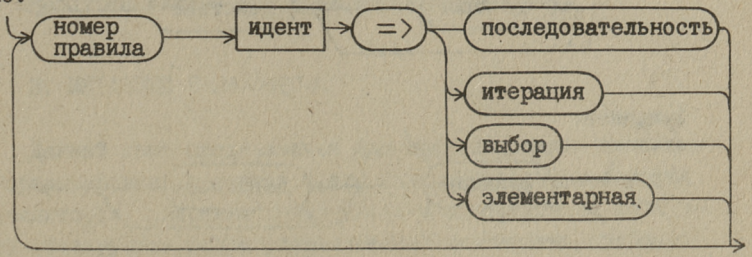


Описание на языке ELMAGUIDE состоит из информационной части и "тела" объекта. Существующий в "заголовке" идентификатор является именем описания. Он используется системой для именования программ, организующих выполнение объекта. Конструкцией "декларации" представляется описание параметров выполняемых действий и управляющих функций. Ее синтаксис должен быть согласован с синтаксисом языка, выбранного для действий.

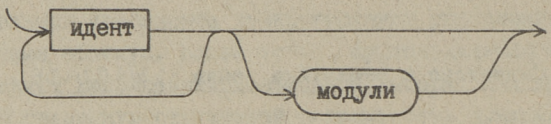
параметры:



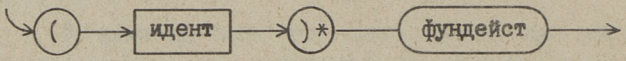
тело:



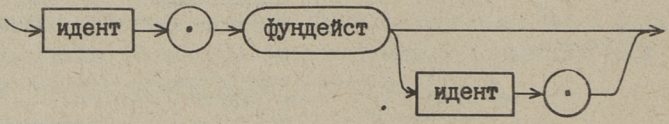
последовательность:



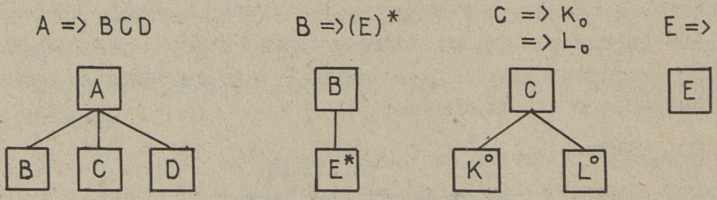
итерация:



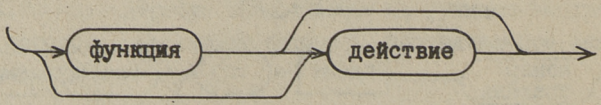
выбор:



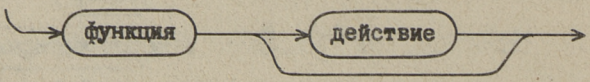
В "теле" объекта определяется структура управления и связываемые с нею действия и / или функции управления. При этом СУ дается с помощью правил четырех видов, где каждое из правил эквивалентно одному из типов компонентов дерева Джексона.



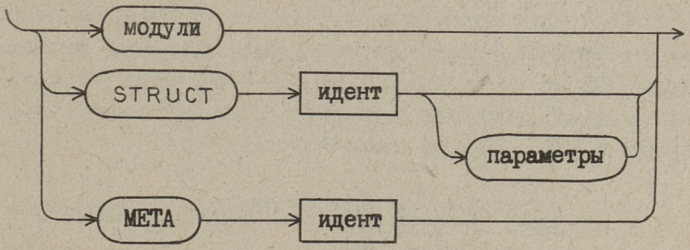
модули



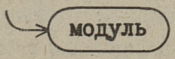
Фундейст:



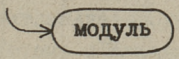
элементарная:



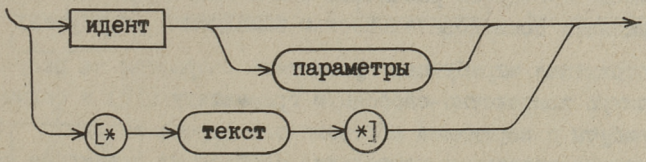
функция:



действие:



модуль:



В описании объекта и действия и функции управления следует оформлять как модули, которые можно непосредственно записать в виде текста на некотором языке программирования или на которые можно ссылаться при помощи имени. При выполнении программы, сгенерированной из описания, структуру управления проходят сверху вниз и слева направо. В каждой вершине сначала выполняется функция управления, а затем, если значение атрибута EXEC истинно при данной вершине, выполняется действие. Присваивание значений системным атрибутам должно обеспечиваться действиями.

3. МЕТАЯЗЫК ELMAMETA

Данный язык предназначен для проектирования проблемно-ориентированных языков и для описания соответствующих трансляторов. Описания конкретного и абстрактного синтаксиса базируются на регуляризованных контекстно-свободных грамматиках [II, I7]. Описание семантики базируется на абстрактных атрибутных грамматиках [I0].

3.I. Описание синтаксиса создаваемого языка

Поскольку объем данной статьи не позволяет привести язык ELMAMETA полностью, то на фиг. I в виде диаграммы приведена та часть языка, которая определяет конкретный и абстрактный синтаксис создаваемого языка. Более точное представление о языке можно получить из [7].

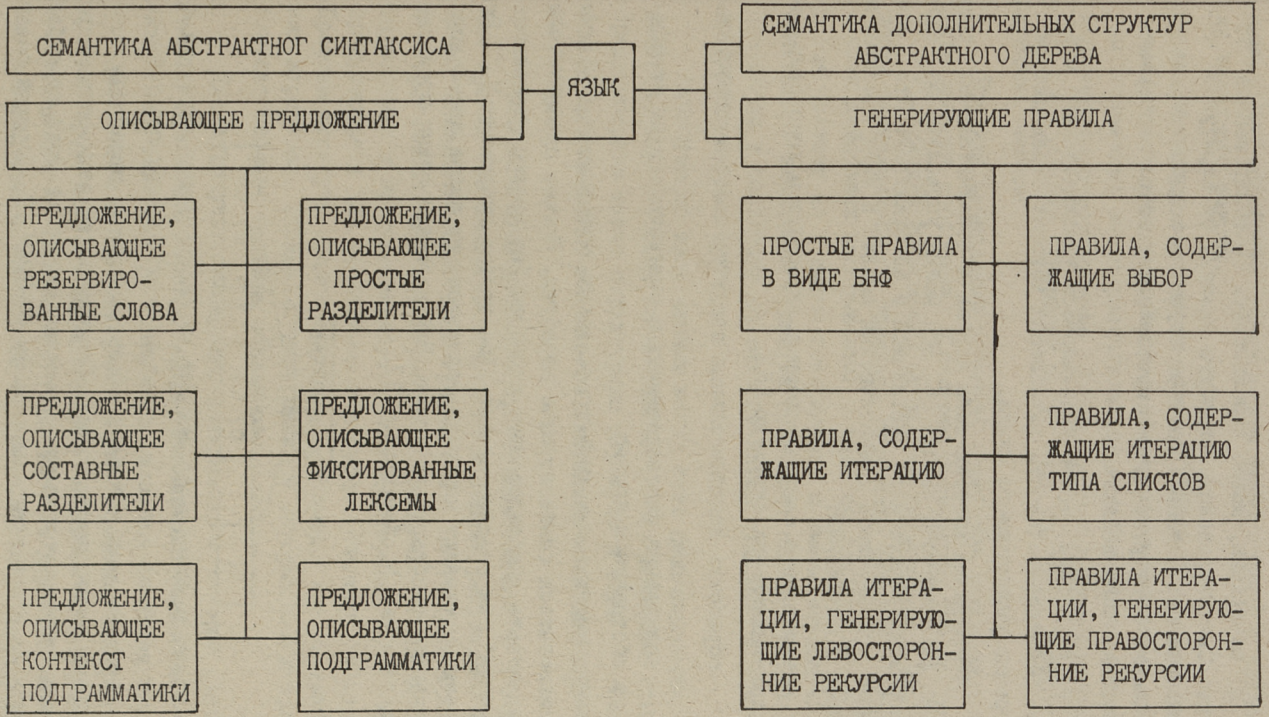
3.2. Семантический анализ. Описание семантики создаваемого языка

Семантический анализ состоит из процесса вычислений и из управления последовательностью вычислений. Для описания семантического анализа в системе построения трансляторов [5] используется абстрактная атрибутная грамматика [I0, I2]. Наличие атрибутов различных типов позволяет пользователю управлять последовательностью выполнения атрибутов.

Абстрактная атрибутная грамматика опирается на регуляризованную контекстно-свободную грамматику [I7] и связывает атрибуты с вершинами абстрактного дерева вывода. С частью регуляризованных контекстно-свободными правилами сопоставляются правила вычисления значений атрибутов. Те части, которые связаны с вхождениями атрибутов и упорядоченные множества поддеревьев дерева вывода определяют абстрактный синтаксис языка. Правила вычисления значений атрибутов, соответствующие данному грамматическому правилу, представляются при помощи фиксированной схемы и применяется для всех вхождений этого правила.

Управление вычислением атрибутов зависит от выбора атрибутов различных по области действия:

а) во-первых, можно использовать унаследованные атрибуты. Значения унаследованных атрибутов зависят от значений атрибутов непосредственного предка соответствующего абстрактного поддерева вывода. В системе построения трансляторов ELMA схема вычислений унаследованных атрибутов рассматривается как статическая схема, по которой дается доступ к корням упорядоченных поддеревьев абстрактного дерева вывода. Истинность последовательности выполнения вычисления унаследованных атрибутов остается на совести пользователя.



Фиг. 1. Компоненты метаязыка ELMAMETA.

б) во-вторых, можно использовать синтезированные атрибуты. Синтезированные атрибуты зависят от атрибутов непосредственных потомков соответствующего поддерева вывода. Вычисление атрибутов производится "снизу вверх" во время синтаксического анализа.

в) в-третьих, можно использовать унаследованные атрибуты, последовательность вычисления которых определяется на уровне управляющего языка.

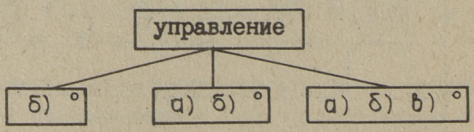
Построением абстрактного дерева и дополнительных структур из разреженного вывода [6, I8] можно управлять во время перевода конкретного синтаксиса в абстрактный. Это позволяет организовать семантический анализ в виде комбинации из трех прохождений:

- вычисление синтезированных атрибутов и конструирование дополнительных структур [5],

- обработка дополнительных структур абстрактного дерева,

- вычисление унаследованных и синтезированных атрибутов на абстрактном дереве.

При описании семантики объектного языка всегда должны использоваться синтезирующие атрибуты. Причем прежде чем использовать унаследованные атрибуты "на уровне управляющего языка", их следует описать "на уровне грамматики". Эти условия определяют иерархию использования типов атрибутов, различных по области действия. Эту иерархию можно представить по схеме М. Джексона [I3]



Окончание вычислений осуществляется описанием семантических действий согласно атрибутной грамматике. Для описания семантических действий объектного языка выработан специальный семантический язык. Семантический язык описан средствами системы построения трансляторов ELMA. Этот язык базируется на языке программирования ФОРТРАН IУ, который расши-

ряется средствами описания вхождений атрибутов, средствами символьной обработки, операторами формирования и распечатки выходного текста.

При записи операторов языка ФОРТРАН IУ не надо соблюдать фиксированного формата перфорации, предложения описания переменных могут располагаться в любом месте в тексте, мнемонические сокращенные формы записи логических операторов и операции отношений могут быть заменены соответствующими знаками.

Правила вычислений значений атрибутов можно записывать разными способами. В системе построения транслятор ELMA правила вычисления атрибутов записываются отдельно, после синтаксического правила. Вхождение атрибут в семантических действиях записывается в виде составного имени, которое состоит из имени нетерминала и имени атрибута. Такая запись вхождения атрибута позволяет описать и отладить синтаксис и семантику объектного языка отдельно и затем соединить в одно целое.

Атрибуты по типу данных могут быть целочисленные (INT), вещественные (REAL), логические(LOGICAL)и литерные (CHAR). Предусмотрены также структурные атрибуты в виде списка (LIST). Элементами списка могут быть все вышеуказанные типы данных.

Вхождения атрибутов как прототипов элементов абстрактных правил (по аналогии с правилами записи регулярных выражений контекстно-свободной грамматики) могут быть простыми и итеративными. Простые атрибуты подразделяются на инициализированные и простые атрибуты. Инициализированные атрибуты предусмотрены для присвоения атрибутам начального значения. Простые атрибуты предназначены для описания правил вычисления атрибутов, вхождение нетерминалов которых вне итерации синтаксического правила объектного языка.

Для атрибутов, вхождение нетерминалов которых входит в итерацию синтаксического правила объектного языка, имеется итеративный атрибут.

Для описания правила вычисления в системе построения трансляторов ELMA сперва описываются имена атрибутов,соответствующие им типы данных, тип области действия атрибутов и вхождение нетерминалов (в определении атрибутов

ATTRIBUTES). По умолчанию просматривается первая буква имени атрибута. Если она является одной из букв I, J, K, L, M, N, тогда берется целочисленный тип данных, а если нет, тогда вещественный тип данных. Тип области действия по умолчанию будет синтезированным. Отдельно описывается вхождение фиксированных лексем в качестве инициализированных атрибутов (в определении фиксированных лексем FIXLEXEMS). Имя инициализированного атрибута зафиксировано (VAL).

Далее рассмотрим синтаксис записи вхождений атрибутов различных типов.

Вхождение инициализированных атрибутов записывается следующим образом:

$$\left\{\begin{matrix}\text{CONST}\\ \text{IDENT}\\ \text{STRING}\end{matrix}\right\}. \quad [n.] \quad \left\{\begin{matrix}\text{VAL}\\ \text{REAL}\\ \text{INT}\end{matrix}\right\}, \quad \text{где}$$

$\left\{\begin{matrix}\text{CONST}\\ \text{IDENT}\\ \text{STRING}\end{matrix}\right\}$ – обозначают имена зафиксированных лексем;

VAL – значение лексемы передается в литерном представлении;

REAL – значение лексемы преобразуется из литерного в вещественное;

INT – значение лексемы преобразуется из литерного в целое,

если в синтаксическом правиле имеется несколько одинаковых зафиксированных лексем, тогда n показывает номер необходимой лексемы по порядку слева направо (по умолчанию берется первая лексема).

Вхождение простых атрибутов независимо от типов данных записывается следующим образом:

⟨ имя нетерминального символа ⟩. [n.] ⟨ имя атрибута ⟩

Первым компонентом должно быть одно из имен нетерминала соответствующего синтаксического правила и оно должно находиться в декларации атрибутов. Второй компонент ведет себя аналогично описанию инициализированного атрибута. Имя атрибута дается согласно определению атрибутов. Итеративный атрибут пишется следующим образом:

⟨имя нетерминального символа⟩.[n.]⟨имя атрибута⟩$\left(\left\{\begin{matrix}*\\ \text{FIRST}\\ \text{LAST}\\ \text{PRED}\\ \text{SUCC}\end{matrix}\right\}\right)$,

где * - означает текущий элемент итерации;
FIRST - означает первый элемент итерации;
LAST - означает последний элемент итерации;
PRED - означает предыдущий элемент итерации относительно текущего (* -1);
SUCC - означает последующий элемент итерации относительно текущего (* +1).

Другие компоненты те же, что у вхождений простых атрибутов.

Для описания упорядоченных поддеревьев дерева вывода предназначается следующее предложение:

$$k\left[\left\{\begin{matrix}\text{HASH } m\\ \text{LIST } n\end{matrix}\right\}\right],$$

где k - номер вершины поддерева вывода;
m - номер HASH таблицы;
n - номер корня поддерева вывода.

Множество упорядоченных поддеревьев дерева вывода называется разреженным деревом дерева вывода, т.е. в дереве вывода имеются только те вершины, которым дана семантика.

Семантический язык расширяет язык ФОРТРАН IУ средствами символьной обработки. Для этого предусмотрен оператор конкатенации

$$a_0 = a_1 !! a_2 !! \ldots !! a_n,$$

где $a_0 \ldots a_n$ - либо идентификатор, либо вхождение атрибута. В правой части выражения может также быть литерная константа.

Для формирования выходного текста предлагается оператор

PRINT → ⟨вхождение атрибута⟩
или
PRINT → ⟨литерная константа⟩

Такой оператор формирует запись в специальном буфере. Для распечатки сформированного предложения предлагается оператор:

PRINT LINE

Если буфер заполняется, тогда запись выпечатывается автоматически.

Во время генерирования транслятора объектного языка из семантических действий генерируется один общий модуль. Поэтапно сперва проверяется правильность синтаксиса семантических действий, далее автоматически генерируется фортранная программа и, наконец, она транслируется в объектный код.

При появлении синтаксических ошибок выпечатываются соответствующие сообщения и работа прекращается.

Во время генерирования фортранной программы выпечатываются ошибки о несоответствии вхождений атрибутов с определением.

В заключение хотелось бы коснуться некоторых проблем, которые обнаружились в ходе эксплуатации данной системы. Условно можно их подразделить на три группы.

Первая группа связана с отладкой создаваемого языка и его транслятора. Эти проблемы невозможно решить без специального математического и программного обеспечения. В системе ELMA данные средства занимают одну треть объема трансляторов ELMAMETA и ELMAGUIDE.

Вторая группа проблем возникает в связи с непригодностью классических методов анализа (LR(к)-грамматика) и вычисление атрибутов для наших машин. Обычно эта проблема связана с неэффективным использованием памяти машины. Для преодоления этого недостатка используется техника оптимизации на уровне алгоритма и каскадная техника - в реализации.

Третья проблема состоит в том, что для практических работ нецелесообразно создавать только трансляторы создаваемого языка, но следует создавать систему программирования, базирующуюся на данном языке. Для этих целей в системе ELMA разработан язык ELMAGUIDE и обращение со всей системой и с ее программным продуктом организовано как макросистема. Это означает, что с любой точки обрабатываемого текста можно обращаться по имени к системным и созданным трансляторам.

В настоящее время система ELMA эффективно используется в конфигурации, где в роли инструментальной машины выступают ЭВМ ЕС и в роли объектной машины - все машины с фортранным транслятором.

Литература

1. Тамм Б.Г., Тыугу Э.Х. Пакеты программ. - Техническая кибернетика, 1977, № 5, с. 111-124.

2. Тыугу Э.Х. Программы и система программирования. Система программирования ПРИЗ. Вып. I, общее описание. Таллин, 1977, с. 24.

3. Крахт В.А., Эйвак Ю.Э. Система ведения баз данных и манипулирования данными "ПАРЕС". Общее описание. Таллин, 1977, с. 37.

4. Вооглайд А.О., Томбак М.О. О проблемах редуцирования в грамматиках предшествования. - Тр. Таллинск. политехн. ин-та, 1975, № 386, с. 23-37.

5. Вооглайд А.О., Томбак М.О. Система построения трансляторов с LR(к)-грамматикой. - Программирование, 1976, № 5.

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

7. Вадер А.Р., Вооглайд А.О. Описание метаязыка в системе построения трансляторов. - Тр. Таллинск. политехн. ин-та, 1978, № 439, с. 83-92.

8. Лепп М.В. Схема трансляции, реализуемая при помощи СПТ ТПИ. - Тр. Таллинск. политехн. ин-та, 1980, № 482, с. 31-40.

9. Лепп М.В., Вооглайд А.О. Опыт внедрения классических методов синтаксического анализа. - Тр. Таллинск. политехн. ин-та, 1979, № 464, с. 3-20.

10. Ershov A.P., Grushetsky V.V. An implementation oriented method for describing algorithmic languages. - IFIP-77. Proc. IFIP Congr. Ed. B.Gilchrist,

North-Holland, 1977, p. 117.

11. S i l v e r b e r g B.A. Using a grammatical formalism as a programming language. - Techn. Rep. CSRG-88, Univ. Toronto, 1978.

12. K n u t h D.E. Semantics of context-free languages. - Math. Systems Theory, 1968, 2, N 2, p. 127.

13. J a c k s o n M.A. Principles of program design. London - New York - San Francisco, Academic Press, 1977.

14. K o w a l s k i R. Algorithm = Logic + Control. - Communications of the ACM, 1979, 22, N 7, p. 424.

15. C o l e m a n D., H u g h e s J.W., P o w e l l M.S. A method for the syntax directed design multiprograms. - IEEE Transactions on Software Engineering, 1981, 7, N 2, p. 189-196.

16. J e n s e n K., W i r t h N. PASCAL - User manual and report. Springer-Verlag, 1978, p. 196.

17. L e w i J., D e V l a m i n c k K., H u e n s J., H u y b r e c h t s M. A programming methodology in compiler construction. Pt. 1; Concepts. Amsterdam - New York - Oxford, North-Holland Publishing Company, 1979, p. 308.

18. G r a y J.N., H a r r i s o n M.A. On the covering and reduction problems for context-free grammars. - J. ACM, 1972, 19, 2, p. 225-243.

A. Vooglaid, M. Lepp, D. Liib

Metalanguages of the System "ELMA"

Summary

The description of metalanguages used in compiler writing system for ES computers is presented. The rules for generating context-free grammar from the object language description are given.

№ 524
TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 51:801

Д.Б. Лийб

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ МОДУЛЬНЫХ СИСТЕМ

I. Статья является продолжением работы [8], в которой предлагается синтаксически ориентированный генератор ввода. По накопленному опыту реализации этой системы предлагается технология программирования, ориентированная на конкретный класс задач.

Предлагаемая технология программирования дает средства организации надстройки сетевой базе данных (БД) и гарантирует сопряжение с системой управления базой данных (СУБД), дает универсальность средств, являющихся независимыми от сложности схемы базы данных и от конкретной СУБД.

В качестве средств реализации используется система построения трансляторов (СПТ) для описания языков и трансляторов. Она дает возможность использовать грамматические формализмы как прямые средства программирования классов задач структурной обработки данных.

Описание необходимых структур данных, позволяющее производить манипулирование данными в БД, рассматривается на трех этапах:

- описание схемы базы данных на проблемно-ориентированном языке;

- описание входной информации (данных) на метаязыке СПТ [3];

- отбор данных из сетевой БД.

2. Описание схемы базы данных. Схема базы данных пишется на проблемно-ориентированном языке, который дает возможность удобного написания информации:

- описывающую логическую структуру базы данных;
- некоторую семантику данных (отраженную в структуре БД), определяющую сужения путем локализации записей БД [12].

Схему БД как логическую структуру данных можно отображать в виде конечного связного ориентированного мультиграфа Y с помеченными вершинами и дугами [7]. Вершинами $V=\{v_0, v_1, \ldots, v_n\}$ являются имена типов записей, и дугами – набор упорядоченных пар смежных вершин $X=\{x_0, x_1, \ldots, x_m\}$, отображающих отношения между типами записей. Дуги помечены именами отношений и семантикой, т.е. инструкцией, характеризующей взаимосвязь смежных вершин на логическом уровне. Эти инструкции показывают, какие действия нужно совершить, чтобы перейти с одного логического уровня данных на другой.

Маршрутом в графе Y называется чередующаяся последовательность вершин и дуг $v_0 x_0 v_1, \ldots, v_i x_j v_{i+1}, \ldots, v_{n-1} x_m v_n$. Путь – это маршрут, в котором все вершины различны [10]. Рассматриваем только такие пути в графе, которые начинаются с указанных вершин – множества $V_A \subset V$, $V_A \neq \phi$. Местонахождение искомых данных можно сравнить движением с одной или нескольких вершин графа $v_i \in V_A$ до какой-то вершины $v_j \in V$, выполняя на каждом шаге указанные инструкции.

Независимо от структуры вводимых данных можно определить все возможные пути движения на графе. С некоторыми ограничениями на схему БД выбор путей локализации записей предлагается в [9]. Для порождения графов можно использовать контекстно-свободные грамматики (КСГ) [7, 11].

Далее определим язык $L_1(G_1)$, предложения которого являются путями в графе Y. Определим КСГ $G_1=(N, \Sigma, P, S)$ [11]. Алфавитом языка L_1 является непустое множество троек терминальных символов

$$\Sigma = \{v_0 x_0 v_1, v_1 x_1 v_2, \ldots, v_i x_j v_{i+1}, \ldots, v_{n-1} x_m v_n\},$$

где N – непустое конечное множество символов нетерминалов, $N \cap \Sigma \neq \phi$;

P – называется множеством порождающих правил, выводом которых является последовательность $\gamma_0, \gamma_1, \ldots, \gamma_n \in N \cup \Sigma$ и $\gamma_0 = S \in N$.

Каждое слово этого языка обозначает какой-то путь в графе. Предложение такого промежуточного языка можно интерпретировать специальной управляющей программой и системой подпрограмм [2]. Результатом интерпретации становится отбор дан-

ных из БД. Управляющая программа активирует подпрограммы, написанные для конкретной СУБД. Это делает систему универсальной и независимой от конкретной СУБД.

3. Описание структур вводимых данных. Метаязык СПТ $L_2(G_2)$ представляет регуляризованную КСГ $G_2=(N_2,\Sigma_2,P_2,S_2)$. Он позволяет использовать грамматический формализм не только для описания языков программирования, но и для описания структур данных [I4].

Описание структур данных дается в виде формальных правил (в виде БНФ) на трех уровнях:

- данные разделяются на логические группы (на записи);
- указываются правила, левая часть которых является именем записи БД и правая часть включает имена элементов этой записи. Все имена, имеющие семантику, входят в N_2;
- каждое имя элемента БД устанавливается в соответствии с лексемой метаязыка (константа, цепочка, идентификатор $\leqslant \Sigma$).

Поскольку СПТ работает синтаксически ориентированной схемой трансляции и по смешанной стратегии [4, 5], то она позволяет совместить функции анализа с функциями управления. Для этого некоторые синтаксические понятия в грамматике снабжены семантикой.

По описанию структуры данных в метаязыке с помощью СПТ атоматически генерируется транслятор. Такой транслятор состоит из зависимой и независимой от объектного языка [3] частей. Зависимая от объектного языка часть называется описанием документа. Такое разделение транслятора на две части позволяет произвести замену описания документа во время отбора данных для различных структур данных. Для формирования независимой части транслятора следует:

- во-первых, определить, какое синтаксическое понятие в описании структур данных совпадает с именем записи БД. Совокупность указанных имен записей БД определяет область поиска в БД [I2]. По количеству информации для поиска $v_0 \subseteq V$ можно строить из алфавита Σ предложения. С каждой вершиной $v_0, v_1, \ldots, v_j$ можно сопоставить два символа Σ, поскольку она входит в состав этих символов. Из всех выбранных символов можно строить какой-то набор предложе-

ний языка L_1, которые и являются путями локализации записей БД, но и в то же время алгоритмом отбора данных из БД;

– во-вторых, по имени записи и по именам элементов этой записи можно сформировать сводные записи, структура которых будет соответствовать записям базы. Поскольку имена записей базы данных уникальные, то составное имя, состоящее из имени записей и ее элемента, дает возможность однозначно определить относительный адрес элемента в записи;

– в-третьих, можно активировать семантические модули. Действие, отраженное в семантическом модуле, связывается с синтаксическим правилом и пишется вслед за ним в метаязыке СПТ. Семантическими действиями являются:

– активирование управляющих программ манипулирования данными, последовательность выполнения которых отражает режим обновления;
– активирование программ логического контроля данных;
– активирование программ пользователя.

4. Отбор данных из БД. Отбор данных начинается с загрузки описания документа по имени и соответственного ему семантического модуля в память ЭВМ. Во входном потоке могут быть данные, соответствующие различным описаниям документов. Поэтому надо произвести замену описания документа и семантического модуля. Для этого предусмотрен специальный загрузчик, функциями которого являются:

– загрузка семантического модуля по имени;
– соединение общих областей с уже имеющимися;
– передача управления загруженному модулю и ее секциям по именам.

После загрузки зависимой и независимой частей транслятора может начаться анализ входных данных. На каждом шаге анализа строят синтаксическое дерево вывода и выполняются семантические действия.

При распознавании синтаксических понятий, имеющих семантику, дается управление модулем, которое формирует сводную запись БД. Для этого предусматривается поддерево дерева вывода [6] и составляют вышеуказанное составное имя.

Пример. По синтаксическим правилам

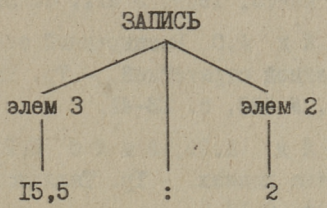
ЗАПИСЬ ⟶ элем 3 : элем 2

ЭЛЕМ 2 ⟶ с

ЭЛЕМ 3 ⟶ с ¤ константа ¤

над входными данными I5,5 : 2

производится синтаксический разбор [I] по следующей схеме:



При прохождении дерева разбора составляются составные имена ЗАПИСЬ. ЭЛЕМ 3. и ЗАПИСЬ. ЭЛЕМ 2, по которым можно найти информацию для внесения соответствующих реквизитов в область памяти ЭВМ, отведенной для сводной записи БД. Для локализации конкретной записи БД имеется заранее выбранный путь локализации, т.е. предложение языка L_1 в виде алгоритма. После обработки текущей записи БД продолжается синтаксический анализ вводных данных. При появлении синтаксических ошибок в данных включается система анализа синтаксических ошибок [I3]. Эта система исключает ошибочную запись, выпечатывает ошибочную запись, выпечатывает сообщение о характере и месте нахождения ошибки. После этих действий анализ продолжается со следующей записи. Такие же действия предусматриваются при обнаружении логических ошибок в данных.

Литература

I. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Синтаксический анализ. Том I, М., Мир, I978. 6I6 с.

2. Берестовая С.М., Перевозчикова О.Л., Романов В.Н., Ющенко Е.Л. Конструирование систем программирования обработки данных. М., Статистика, I979, с. 270.

3. В а д е р А.Р., В о о г л а й д А.О. Описание метаязыка системы построения трансляторов. – Тр. Таллинск. политехн. ин-та, 1978, № 439, с. 57–82.

4. В о о г л а й д А.О. Семантическое равенство распознавателей, работающих на грамматике LR(к) и грамматике предшествования с (1/1) ограниченным контекстом. – Тр. Таллинск. политехн. ин-та, 1976, № 411, с. 39–56.

5. В о о г л а й д А.О. Расширенный анализатор предшествования со смешанной стратегией. – Тр. Таллинск. политехн. ин-та, 1977, № 423, с. 23–41.

6. В о о г л а й д А.О., Л и й б Д.Б. Древовидная переменная в обработке данных. – Тр. Таллинск. политехн. ин-та, 1980, № 482, с. 3–14.

7. Г о с т е в Ю.Г. Описание структур данных с помощью графопорождающих грамматик. – Программирование, М., 1981, № 2, с. 44–51.

8. Л и й б Д.Б., Р е н з е р А.В. Синтаксически ориентированный генератор ввода. – Тр. Таллинск. политехн. ин-та, 1981, № 511, с. 71.

9. Р е н з е р А.В. Об автоматическом интерфейсе между базами данных типа CODASYL и системами ввода-вывода. См. наст. сб., с. 29.

10. Х а р а р и Ф. Теория графов. М., Мир, 1973, с. 300.

11. Х о л о н г р е н А. Определение языков программирования интерпретирующими автоматами. М., Мир, 1977, с. 289.

12. Т е п а н д и Я.Я. О методах трансляции языков с проблемной ориентацией. – В кн.: Всесоюзная конференция по методам трансляции. Тезисы докладов, Новосибирск, 1981, с. 75–77.

13. Р о х т л а Х.Х. Локализация синтаксических ошибок в системе построения трансляторов ТПИ. См. наст. сб., с. 119.

14. M a u r e r H., S t u c k y W. Ein Vorschlag für die Verwendung syntax-orientierter Methoden in höheren Programmiersprachen. Angewandte Informatik, Nr. 5, p. 189-195.

D. Liib

Modul Systems Programming Technology

Summary

In this article a technology of programming for a certain class of tasks is presented.

Means of construction of a complementation to CODASYL data bases are shown.

For implementation facilities grammatical formalisms are used as directed means of programming for classes of structured data processing tasks. This approach allows universal compatibility with data base management systems and automatical synthesis of algorithms for data localization.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 681.3.06

Л.К. Выханду, В.А. Йокк

ОБ ОПЫТЕ ИСПОЛЬЗОВАНИЯ ПРОГРАММНЫХ ХАРАКТЕРИСТИК ХОЛСТЕДА

I. Введение

В связи с использованием инженерно-математических структур данных и определенной методики обучения программированию возникает потребность в более или менее объективной оценке программной реализации алгоритмов. Подходящей методикой для этих целей является теория Холстеда [6], которая названа началами науки о программах [5]. Эта теория может быть названа и программометрикой, так как изучает метрические характеристики программы и подходит как для предсказания параметров составляемой программы, так и для анализа готовой программы. Для наших целей представляют интерес возможности программометрики для анализа уже составленной программы.

При анализе программы М. Холстед исходит из текста программы на конкретном языке и из представления, что реализация алгоритма на некотором языке программирования состоит только из операторов и операндов. Эти величины порождают четыре основных метрических характеристики программы и могут быть найдены простым подсчетом из текста программы, если правила счета операторов и операндов известны.

При помощи основных метрических характеристик программы определяется целый ряд важных для практического использования понятий и соотношений, таких как оценка длины программы, объем и минимальный или потенциальный объем программы, уровень и интеллектуальное содержание программы и др. С понятием интеллектуального содержания программы связано другое существенное с точки зрения практики понятие - это несовершенство программы. Устранение несовершенств(ком-

пенсирующих операторов, ненужных присвоений и др.) улучшает программу и увеличивает значение ее интеллектуального содержания.

Исследование большого количества программ вручную слишком трудоемкий процесс и вероятность возникновения ошибок достаточно велика. По этой причине для нахождения метрических характеристик программы в ВЦ ТПИ была построена автоматизированная система анализа программ при помощи системы построения трансляторов (СПТ). В данной статье эта система анализа не описывается, а приводятся результаты исследования студенческих программ по этой системе.

Основные метрические характеристики программы и их вычисление рассматриваются в пункте 2. В пункте 3 приводятся отношения между основными характеристиками программы и возможности их применения. В пункте 4 перечисляются классы несовершенств программы. Результаты анализа студенческих программ приводятся в пункте 5.

2. Основные метрические характеристики программы

В программометрике выделяются следующие основные метрические характеристики реализации алгоритма или программы:

1) количество различных операторов, появляющихся в данной реализации - η_1;

2) количество различных операндов, появляющихся в данной реализации - η_2;

3) количество всех операторов, появляющихся в данной реализации - N_1;

4) количество всех операндов, появляющихся в данной реализации - N_2.

Эти параметры являются исходными величинами для вычисления других метрических характеристик и могут быть найдены простым подсчетом из текста программы. При этом операторами считаются те инструкции программы, которые связаны с реализацией алгоритма, и операндами - переменные и константы программы. Подробнее правила счета операторов и операндов рассматриваются ниже.

2.I. Подсчет операторов и операндов

В качестве примера для подсчета операторов и операндов рассмотрим ФОРТРАН-программу. Для определения операторов и операндов такой программы предлагаются следующие правила:

I) рассматривается только та часть программы, которая связана с выполнением алгоритма;

2) операторами считаются:

а) символы присваивания (=),
б) арифметические операции (+,-, * ,/, **),
в) логические операции(.LT.,.LE.,.EQ.,.GT.,.GE.),
г) операторы языка ФОРТРАН (IF( ),DO, ...),
д) имена функции (ABS( ),SIN( ),...),
е) разделители (" , " и пары скобок в арифметических выражениях),
ж) ограничитель оператора языка ФОРТРАН (обозначаем через " ; ");

3) операндами считаются имена переменных и константы;

4) метка считается составной частью оператора GOTO, а не операндом, так как не является ни переменной, ни константной;

5) операторами не считаются:

а) описания и операторы ввода/вывода как не связанные непосредственно с выполнением алгоритма,

б) операторы RETURN и STOP, являющиеся выходом из алгоритма,

в) оператор END как физический конец программы.

2.2. Пример подсчета основных метрических характеристик ФОРТРАН-программы

Приведем вычисление основных метрических характеристик ФОРТРАН-программы на примере программы построения треугольника Паскаля из задачника [I] (решение N° 4, с. I5):

```
DO 31 I = 1,17
N (I,1) = 1
```

```
      DO 31 J = 2,17
      N(I,J) = ϕ
      IF(J.LE.I)N(I,J) = N(I-1, J-1) + N(I-1,J)
   31 CONTINUE
```

Операторы, операнды и частота их появления представлены в таблице I.

Таблица I

№	Оператор	Частота появления операторов в программе	№	Операнд	Частота появления операндов в программе
1.	DO	2	1.	I	7
2.	=	5	2.	1	6
3.	,	7	3.	17	2
4.	IF( )	1	4.	N( )	5
5.	.LE.	1	5.	J	6
6.	-	3	6.	2	1
7.	+	1	7.	0	1
8.	;	5			

$\eta_1 = 8$ $N_1 = 25$ $\eta_2 = 7$ $N_2 = 28$

3. Отношения между основными характеристиками программы

При помощи основных метрических характеристик (η_1, η_2, N_1, N_2) программы определяется емкость словаря программы

$$\eta = \eta_1 + \eta_2$$

и длина программы

$$N = N_1 + N_2.$$

Для вычисления оценки длины программы Холстед вводит следующее выражение

$$\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2.$$

Оценка длины была проверена несколькими авторами [7, 8, 9] на опубликованных в печати программах, составленных опытными программистами. Данные измерений показали, что $\hat{N}$

дает довольно точную оценку длины N программы. Таким образом, сравнение величин N и $\hat{N}$ может быть использовано для предварительной оценки качества программы.

При помощи выражения оценки длины можно даже решать столь сложную проблему как оптимальное разбиение программы на модули, так как $\hat{N} \approx N$ справедливо и для отдельных логических модулей и для полных программ.

Одной важной характеристикой алгоритма является его размер. При переводе алгоритма с одного языка на другой размер программы либо увеличивается, либо уменьшается. Но и алгоритмы, написанные на одном языке, имеют различные размеры. Соответствующая метрическая характеристика размера любой программной реализации какого-либо алгоритма, называемая объемом V, определяется как

$$V = N \log_2 \eta$$

и измеряет минимальное число битов, необходимое для задания программы. Объем программы - это величина, которая меняется при трансляции программы с одного языка программирования на другой.

Потенциальным или минимальным называется объем наиболее компактной программы, которая написана на языке с заранее заданными требуемыми действиями. Для реализации алгоритма на таком языке требуются лишь имена параметров ввода/вывода (η_2^*), один оператор для имени функции или процедуры и второй - для присваивания или группировки.

Таким образом, подобно объему V, потенциальный объем V^* определяется выражением

$$V^* = (2 + \eta_2^*) \log_2 (2 + \eta_2^*).$$

Отсюда видно, что параметр V^* не зависит от языка программирования и является постоянным для данного алгоритма.

При таком определении объемов V и V^* программы отношение

$$L = \frac{V^*}{V}$$

задает уровень программы. Величина уровня программы ($L \leq 1$) является мерой компактности программы и играет двойственную роль в определении легкости или трудности ее понимания. Специалист, которому известны все используемые тер-

мины, может уловить идею тем быстрее, чем выше уровень ее представления и наоборот, для того, чтобы сообщить подобную идею лицу, менее сведущему в данной области, требуется большой объем и более низкий уровень.

Перестроив формулу уровня программы следующим образом

$$V^* = L \times V, \qquad (*)$$

получим своеобразный закон сохранения, связывающий уровень программы с ее объемом. Это показывает, что для данного алгоритма при увеличении объема программы ее уровень уменьшается и наоборот.

Для определения уровня программы непосредственно из реализации, не зная, чему равен ее потенциальный объем, можно применять следующее соотношение:

$$\hat{L} = \frac{2}{\eta_1} \frac{\eta_2}{N_2}.$$

Холстед определяет закон сохранения, связывающий уровень программы с ее объемом (*), как интеллектуальное содержание I реализации:

$$I = \hat{L} \times V = \frac{2}{\eta_1} \frac{\eta_2}{N_2} \times N \log_2 \eta \approx V^*.$$

Величина параметра I (как и V^*) не зависит от языка программирования, на котором написана программа. Она определяется характером самой проблемы и, как показывают эксперименты, лишь увеличивается по мере роста сложности проблемы. Например, увеличение степени сложности проблемы на 25 % приводит к усложнению программы в два раза [10].

Таким образом, параметр I может быть использован для оценки степени сложности программы. Тем самым в процессе обучения программированию он может служить критерием для выбора такой последовательности задач, где бы сложность составленных программ непрерывно увеличивалась. Следует отметить, что при этом использование параметра I имеет смысл только для совершенных программ. Классы несовершенств рассматриваются ниже в пункте 4.

Трудности программирования возрастают с увеличением объема программы и уменьшаются с повышением уровня программы. В силу этого, предложенное Холстедом отношение

$$E = V/\hat{L}$$

характеризует интеллектуальные усилия, необходимые для составления программы (работа в программировании).

3.I. Пример вычисления метрических характеристик программы)

В сборнике [I] приведены 9 различных программ вычисления треугольника Паскаля. Очевидно, не все из них совершенны. Попытаемся формально оценить эти программы на основании вычисленных характеристик, представленных в таблице 2.

Таблица 2

Номер программы	2	3	4	5	6	7	8	9	I0
η_1	6	6	8	6	6	6	6	6	6
η_2	7	7	7	6	8	8	8	9	9
$\eta = \eta_1+\eta_2$	I3	I3	I5	I2	I4	I4	I4	I5	I5
N_1	34	3I	25	23	3I	29	22	22	3I
N_2	36	3I	28	26	35	33	26	25	3I
$N = N_1+N_2$	70	62	53	49	66	62	48	47	62
$\hat{N}=\eta_1\log_2\eta_1+\eta_2\log_2\eta_2$	35	35	44	3I	40	40	40	44	44
$V = N\log_2\eta$	259	259	207	I75	25I	236	236	283	24I
$\hat{L}=2\eta_2/\eta_1 N_2$	0,06	0,08	0,06	0,08	0,08	0,08	0,I	0,I2	0,I
$I=\hat{L}/V$	I6	I8	I2	I4	20	I9	24	22	24
$E=V/\hat{L}$	4320	2870	3430	2200	3I40	2960	2360	I530	2420

Анализ этих и некоторых других программ [2] показывает, что если N и $\hat{N}$ значительно отличаются друг от друга, то соответствующая программа составлена не лучшим образом и содержит несовершенства, которые рассматриваются в следующем пункте.

4. Классы несовершенств программы

Холстед показывает, что в случае совершенной программы совпадение длины N программы и оценки длины $\hat{N}$ достаточно точное [8]. Если эти величины значительно отличаются друг от друга, то соответствующая программа, как правило, содержит несовершенство, которое подразделяется на шесть классов:

1) компенсирующие операторы (один оператор в той или иной мере уничтожает действие другого оператора);

2) неоднозначные операнды (один и тот же операнд используется для представления двух или более переменных в программе);

3) синонимичные операнды (два или несколько имен представляют ту же самую переменную);

4) общие подвыражения (одно и то же подвыражение используется более одного раза);

5) ненужное присваивание (значение подвыражения присваивается промежуточной переменной, которая используется лишь один раз);

6) выражения, не представленные в виде произведения множителей.

Эти классы несовершенств характеризуют недостатки метода составления программы.

5. Анализ студенческих программ

Для анализа качества составленных программ были рассмотрены сотни студенческих программ на ФОРТРАНЕ. Так как при исследовании такого количества программ вручную вероятность возникновения ошибок достаточно велика, то нахождение параметров оцениваемых программ было автоматизировано. Используя пакет программ, составленный в ВЦ ТПИ при помощи системы построения трансляторов [4], все параметры каждой ФОРТРАН-программы были подсчитаны автоматически. Для предварительной оценки качества большого количества программ использовалось сравнение величин N и $\hat{N}$. Анализ показал, что если для некоторой программы N и $\hat{N}$ значительно отличаются друг от друга, то действительно, либо эта про-

грамма содержит какие-то из перечисленных несовершенств, либо программа состоит только из вычисления формул, либо в программе используются неподходящие для данной задачи конструкции языка программирования. По нашему мнению, последнее указывает на то, что предложенные преподавателями задачи были выбраны неудачно. Таким образом, появляется возможность оценить задачу с точки зрения пригодности использования в ней тех или иных конструкций языка. Такого типа недостатки нигде раньше не указывались и явились неожиданностью для нас самих.

Рассматриваемые студенческие программы были настолько простые, что наблюдение увеличения степени сложности проблемы по параметру I оказалось невозможным.

Рассмотрим теперь наиболее часто встречающиеся несовершенства в студенческих программах.

I) Наличие компенсирующих операторов.

В общем случае, компенсирующие операторы встречаются как в выражениях, так и в программах. В последнем случае они влияют на управление программой и затрудняют ее понимание, как, например, в таком фрагменте:

```
      DO 2 I = 1,30
         IF(A(I).LT.Ø)GOTO 3
         GOTO 2
    3    L=L+1
    2 CONTINUE
```

Этот фрагмент программы может быть заменен более "прозрачным":

```
      DO 2 I = 1,30
         IF(A(I).LT.Ø)L=L+1
    2 CONTINUE
```

Следующий фрагмент программы

```
      IF(MOD(K,N).NE.Ø)GOTO 1Ø
        WRITE(6,4)K
   1Ø ...
```

следует заменить оператором

```
      IF(MOD(K,N).NE.Ø)WRITE(6,4)K.
```

Нерациональное использование управляющих операторов приводит к труднопонимаемым программам. Таким образом, студентов следует более тщательно обучать тому, как применять управляющие структуры, и особенно, как их представлять в языках программирования, где отсутствуют конструкции

```
IF-THEN
IF-THEN-ELSE
DO-WHILE
```

Например, конкретные правила для представления таких конструкций на языке ФОРТРАН приведены в [3].

2) Наличие синонимичных операндов.

Встречались программы, в которых два или несколько имен представляют ту же самую переменную:

```
X1 = P
X2 = X1 + Q
R  = X1 + X2
```

Для устранения такого типа несовершенства следует опустить лишние присваивания, что проясняет смысл выполняемых программой действий:

```
X2 = P + Q.
```

3) Наличие общих подвыражений.

Если подвыражение используется в программе более одного раза, то его следует обозначить новым именем. Это приводит к увеличению скорости вычислений и к уменьшению сложности программы. Например, в случае

```
      MIN = A(1)
      MAX = A(1)
      DO 10 I = 1,50
        IF (A(I).GT.MAX) MAX = A(I)
        IF (A(I).GT.MIN) MIN = A(I)
   10 CONTINUE
```

при неоднократном использовании элемента массива A(I) его целесообразно обозначить новым именем. Тогда фрагмент, выполняемый в цикле, запишется так:

```
Z = A(I)
IF(Z.GT.MAX)MAX = Z
IF(Z.LT.MIN)MIN = Z
```

4) Наличие ненужных присваиваний.

В некоторых программах промежуточной переменной присваивали подвыражéние, которое используется лишь один раз.

Авторы книги [I2] приводят правило "Избегайте промежуточных переменных" и считают, например, что следующие три оператора

```
F1 = X1 - X2*X2
F2 = 1.ϕ - X2
FX = F1 + F2
```

следует заменить оператором

```
FX=X1 - X2**2 + 1.ϕ - X2.
```

Чем меньше промежуточных переменных в программе, тем меньше возможность их неоднозначного использования.

Для устранения такого несовершенства следует опустить лишние присвоения.

Однако Гордон показывает[II] , что в программах, объем словаря которых больше чем 46, целесообразно использовать промежуточную переменную для обозначения сложных выражений даже тогда, когда она используется лишь один раз. Это приводит к большей ясности программы. Анализ студенческих программ показал, что эти программы были настолько просты, что использование в них промежуточных переменных не разумно.

6. Заключение

Метод Холстеда для формальной оценки программ, основанный на общих информационно-теоретических и статических идеях, многим кажется странным и невероятным и находит в литературе как сторонников [7, 9, II, I0, 5], так и противников [I3]. В статье [I3], где использованы студенческие программы, отмечается, что не все параметры Холстеда являются стабильными, что однако нельзя сказать о N и $\hat{N}$, но

авторы этой статьи не считают их важными параметрами. Наш опыт все-таки показывает, что именно в таких студенческих программах, для которых N и $\hat{N}$ существенно отличались, были обнаружены несовершенства разного типа. Их устранение привело к сближению величин N и $\hat{N}$ и улучшению качества программ. Вместо того, чтобы исправлять готовые программы, следовало бы при обучении началам программированию обратить больше внимания на выполнение следующих правил:

1) использовать управляющие структуры в строгом соответствии с их назначением;

2) одно и то же значение вычислять не более одного раза;

3) каждая переменная программы должна иметь свое определенное значение;

4) излишние промежуточные переменные должны быть исключены.

Кроме того, теория Холстеда оказывается полезной и для преподавателя. Почти каждая задача имеет несколько решений, и сравнение величин N и $\hat{N}$ дает возможность оценки и выбора лучшего варианта решения. Если не находится достаточно точного соответствия между величинами N и $\hat{N}$, то разумно отказаться вообще от данной задачи, так как она не подходит для использования тех или иных конструкций языка программирования.

Помимо этого параметр I может быть использован для оценки степени сложности программы. Таким образом, использование программометрики дает возможность

1) формальной оценки качества составленных программ;

2) сравнения различных техник программирования;

3) организации процесса преподавания программирования таким образом, чтобы сложность составляемых программ имела бы тенденцию плавного роста.

В силу различия методов программирования и обучения у каждого преподавателя описанная выше методика может служить основанием для оценки стиля программирования с точки зрения его улучшения, а также для проверки и сравнения действительного содержания и уровня обучения.

Литература

I. Дрейфус М., Ганглоф К. Практика программирования на ФОРТРАНЕ. Упражнения с комментариями. М., Мир, 1978. 224 с.

2. Ламуатье Ж.-П. Упражнения по программированию на ФОРТРАНЕ IV. М., Мир, 1978. 162 с.

3. Выханду Л.К., Йокк В.А. Методические рекомендации по составлению программ. Таллин, Ротапринт ТПИ, 1981. 21 с.

4. Вадер А.Р., Вооглайд А.О. Описание метаязыка в системе построения трансляторов. - Тр. Таллинск. политехн. ин-та, 1978, № 439, с. 83-92.

5. Холстед М.Х. Начала науки о программах. М., Финансы и статистика, 1981. 228 с.

6. Halstead M.H. Elements of Software Science. New York, Elsevier, 1977. 127 p.

7. Elshoff J.L. Measuring commercial PL/I programs using Halstead's criteria. - ACM SIGPLAN Notices, 1976, vol. 7, N 5, p. 38-46.

8. Halstead M.H. Natural laws controlling algorithm structure. - ACM SIGPLAN Notices, 1972, vol. 7, N 2, p. 19-26.

9. Bulut N. Invariant Properties of Algorithms. Ph.D.Thesis, Purdue University, 1973. 149 p.

10. Woodfield S.N. An experiment on unit increase in problem complexity. - IEEE Transactions on Software Engineering, 1979, vol. SE-5, N 2, p. 76-79.

11. Gordon R.D. A Qualitative justification for a measure of program clarity. - IEEE Transactions on Software Engineering, 1979, vol. SE-5, N 2, p. 121-128.

12. Kernighan B.W., Plauger P.J. The Elements of Programming Style. New York, McGraw-Hill, 1974. 147 p.

13. J o h n s o n B.D., L i s t e r A.M. An experiment in software science. - Lecture Notes in Computer Science. Language Design and Programming Methodology, 1980, vol. 79, p. 195-215,

L. Vyhandu, V. Jokk

Experience of Using Halstead's Metries of Programs

Summary

The paper describes the possibilities of using Halsted's metries in the realm of the students' programming. The rules for counting operators and operands of a program, and calculating the measurable properties of programs are regarded in this paper. Classis of impurities and the ways of removing them as well as the rules for constructing "pure" programs are discussed in the paper.

№ 524

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

УДК 51:801

Х.Х. Рохтла

НЕЙТРАЛИЗАЦИЯ СИНТЕТИЧЕСКИХ ОШИБОК В СИСТЕМЕ ПОСТРОЕНИЯ ТРАНСЛЯТОРОВ ТПИ

I. Введение

В настоящее время уже не вызывает сомнения необходимость эффективных средств обработки синтаксических ошибок как в отдельных трансляторах, так и в системах построения трансляторов (СПТ). Подобные средства должны предоставить возможность не прерывать синтаксического анализа независимо от числа ошибок в транслируемом тексте. Также они должны обнаруживать в тексте возможно больше из имеющихся синтаксических ошибок и давать о них возможно точную и полную диагностику (желательно в терминах исходного языка).

В средствах такого рода нуждается и СПТ, создаваемая в ТПИ [2]. В статье [4] нами было предложено создание каскадных средств обработки синтаксических ошибок. Эта идея хорошо согласуется с идеологией СПТ ТПИ, где построение всего транслятора производится по каскадной технике.

На каждом последующем уровне каскада средства обработки ошибок становятся более мощными, но в то же время возрастает их объем и трудоемкость. Пользователь же имеет возможность выбирать подходящие средства из предлагаемого пакета программ для конкретного языка, конкретной стадии овладения языком и т.д. (здесь и далее под пользователем понимается создатель языка при помощи СПТ). В некоторых случаях имеется также возможность автоматического определения нужных средств.

Каждый конкретный уровень из каскада опирается на все нижестоящие уровни, его реализация возможна лишь при внедренных и уже оправдавших себя более низких уровнях. Отсюда вы-

текает необходимость разработки каскадных средств снизу-вверх. Так как самый низкий уровень - это практически отсутствие какой-либо обработки, то базовым можно считать второй уровень - нейтрализацию ошибки в виде пропуска части входного текста, содержащего ошибку. Исследованию этого уровня каскадных средств обработки синтаксических ошибок и посвящена данная статья, в которой решают две основные задачи:

I. Определение данных для локализации частей текста, подлежащих пропуску.

2. Реализацию данного уровня в СПТ ТПИ, в том числе:

I) автоматическую генерацию таблиц левых и правых границ пропускаемых конструкций. (Реализуется во время генерации транслятора с использованием информации, имеющейся в распоряжении данной части СПТ, плюс возможность получения дополнительной информации от пользователя);

2) реализацию метода нейтрализации. (Производится во время работы транслятора со средствами синтаксического анализатора предшествования с привлечением (I,I) -ограниченного канонического контекста (ОКК), плюс генерированные таблицы левых и правых границ).

2. Методы нейтрализации синтаксических ошибок с помощью пропуска ошибочной части текста

Названный тип методов нейтрализации ошибок - самый простой и распространенный. Он осуществляется несколькими разными способами.

В одном случае грамматика расширяется дополнительными правилами подстановки, содержащими наиболее распространенные синтаксические ошибки в конкретном языке [5, 7]. При появлении ошибки во время синтаксического анализа вводный текст пропускается до тех пор, пока не наступит возможность использования одного из таких правил, затем анализ продолжается. Такой метод порождает множество проблем. Это покрытие всех "самых вероятных" ошибок "своими" правилами, быстрое возрастание объема грамматики при этом и опасность возникновения ее неоднозначности.

Вторая группа методов - локализация потенциального выражения, содержащего ошибки и его замена предполагаемым понятием [6, 7, 8, I0]. Такая техника дает хорошие результаты, но так как она требует более точного анализа ситуации ошибки и возможных исправляющих действий, то ее принципы включены в третий уровень каскадных средств обработки ошибок и в данной работе далее не рассматриваются.

Третья возможность - т.н. метод паники, описанный в [8]. По этой схеме после обнаружения ошибки входной текст пропускается до появления одного из символов, обозначающих конец выражения (например ";", "END" и т.д.). Далее из магазина анализа элементы исключаются до тех пор, пока не найдется такой элемент, за которым может следовать найденный символ, затем анализ продолжается. Немного отличающаяся техника изложена в [II].

По [9] метод паники имеет лишь одно преимущество(кроме его простоты) - это невозможность бесконечного зацикливания. Главным его недостатком считается невозможность обнаружения других ошибок в пропускаемой части текста, которая может оказаться довольно большой. Все же маловероятно, что группы ошибок скапливаются именно в одни конструкции. Это утверждение поддерживается в [I2], где произведен статистический анализ 589 действительных программ в языке PASCAL. Выясняется, что 79,2 % из ошибочных предложений содержали только одну ошибку. Это значит, что пропуск этих предложений не "скрывал" бы ни одной дополнительной ошибки. А возможный пропуск других конструкций языка, содержащихся в предложении, уменьшил бы еще количество необнаруженных ошибок.

В СПТ ТПИ на втором уровне каскадных средств обработки синтаксических ошибок используется метод, относящийся к последнему типу. Проблема настраиваемости для любого применяемого языка, касающаяся всех СПТ, при этом методе вполне разрешима.

3. Определение данных, необходимых для метода нейтрализации ошибок

Для полноты следующего изложения приведем некоторые необходимые понятия.

Обозначим через A^* множество всех слов в алфавите A и $A^+ = A^* \setminus \{\lambda\}$, где λ – пустое слово. Если $x \in A^*$, то $|x|$ – длина слова x. Контекстно-свободная грамматика (КС-грамматика) – это упорядоченная четверка $G = (V_N, V_T, P, S)$, где V_N – конечный алфавит нетерминальных символов (понятий) языка, V_T – конечный алфавит терминальных символов такой, что $V_T \cap V_N = \emptyset$, P – множество правил подстановки вида $A \to x$, где $A \in V_N$, $x \in (V_T \cup V_N)^*$ и S – начальный символ. Обозначим $V = V_N \cup V_T$. Пусть $x, y \in V^*$. Слово y непосредственно выводимо из слова x $(x \Rightarrow y)$, если $x = uAv$, $y = uzv$ и $A \to z \in P$ $(u \in V^*, v \in V_T^*)$. Слово y выводимо из слова x $(x \overset{*}{\Rightarrow} y)$, если существует последовательность слов $z_0, z_1, \ldots, z_k$ $(z_i \in V^*$, $\emptyset \leq i \leq k)$ такая, что $x = z_0$, $y = z_k$ и $z_i \Rightarrow z_{i+1}$ $(0 \leq i \leq k)$. Цепочка u называется сентенциальной формой, если $S \overset{*}{\Rightarrow} u$. $L(G) = \{x : x \in V_T^*, S \overset{*}{\Rightarrow} x\}$ есть язык, определяемый грамматикой G. Каноническим (m,k) – контекстом нетерминального символа A называется $C_A^{m,k} = \{(x,y) \mid \exists S \overset{*}{\Rightarrow} uxAyv;\ u, x \in V^*,\ (|x| = m, |y| = k) \cup |ux| < m \cup |yv| < k\}$.

Множествами к-префиксов и к-суфиксов называем соответственно:

$$PREF_k(A) = \{x \mid A \overset{*}{\Rightarrow} xB,\ |x| = k \cup A \overset{*}{\Rightarrow} x,\ |x| < k;\ x \in V^*\}$$

$$SUF_k(A) = \{x \mid A \overset{*}{\Rightarrow} Bx,\ |x| = k \cup A \overset{*}{\Rightarrow} x,\ |x| < k;\ x \in V^*\}.$$

При применении метода паники возникает оправданный вопрос: какие конструкции вообще можно исключить из текста в конкретном языке без нарушения его синтаксиса, и какие нет (здесь и далее слово "конструкция" обозначает терминальную цепочку, выводимую из какого-то понятия языка).

В языках можно найти конструкции двух типов:

1) конструкции, исключение которых из текста вызывает появление синтаксической ошибки;

2) конструкции, исключение которых допустимо по синтаксису языка.

Именно конструкции последнего типа являются в языке объектами возможного пропуска при появлении в них синтаксических ошибок.

Рассмотрим, например, фрагмент языка, данный следующими правилами:

```
ФРАГМЕНТ → begin  ТЕЛО end
ФРАГМЕНТ → begin  end
   ТЕЛО → НАЧАЛО     КОНЕЦ
 НАЧАЛО → СЛОВО      НАЧАЛО
 НАЧАЛО → СЛОВО
  СЛОВО → . . .
  КОНЕЦ → . . .

        . . .
```

Слова begin и end обозначают терминальные, другие слова - нетерминальные символы. Как видно из правил, конструкции, выводимые из понятий НАЧАЛО и КОНЕЦ, относятся к первому типу, а другие, выводимые из понятий ТЕЛО и СЛОВО - ко второму.

При реализации метода паники возникает две проблемы:

I) определение конструкции, пропуск которых синтаксически допустим;

2) определение символов, по которым в тексте возможно было бы узнавать конструкции из I).

Одна возможность - оставить решение этих проблем на долю пользователя, другая - решить проблемы автоматическими средствами.

В СПТ ТПИ существует непосредственная возможность автоматического решения поставленных проблем. Все повторные конструкции языка, которые в определенном месте текста могут встречаться любое количество раз, рекомендуется описывать в метаязыке [I] с помощью правил итерации - применением звездочки Клини.

Например, понятие НАЧАЛО из предыдущего примера может быть описано как

$$\text{НАЧАЛО} \rightarrow (\text{СЛОВО})^{*}$$

Конструкции, выводимые из понятия, стоящего в скобках в правой части правила итерации, относятся к числу пропускаемых по необходимости. В правилах со звездочкой Клини они легко узнаваемы. В случае, если в скобках стоит больше, чем один символ, и среди них есть терминальные, символы в скобках заменяются новым нетерминалом и нужная конструк-

ция выводима из этого понятия. Так с помощью правил итерации решается первая из поставленных проблем. Приступаем к решению второй задачи.

Определение I

Множеством левых к-границ конструкции, выводимых из понятия А, называется множество

$$LG_k(A)=\{x \mid x\in C_A^{k,\phi} \cup x\in PREF_k(A)\}.$$

Определение 2

Множеством правых к-границ конструкции, выводимых из понятия А, называется множество

$$RG_k(A)=\{x \mid x\in C_A^{0,k} \cup x\in SUF_k(A)\}.$$

Осмысливаем эти определения неформально. Левая граница конструкции - это к-символьная цепочка символов, которая в магазине анализа может стоять непосредственно перед конструкцией или в ее начале. Правая граница - это к-символьная цепочка символов, которая может во входном тексте быть в конце конструкции или стоять непосредственно после нее. Таким образом, в LG(A) и RG(A) содержатся цепочки символов, по которым можно всегда узнавать начало конструкции, выводимой из А, и ее конец.

Следует отметить, что из вышеприведенных определений вытекает одна трудность. При пропуске конструкции по ее границам неизвестно, принадлежат ли сами границы к пропускаемой конструкции или являются ее контекстом, т.е. приходится ли пропускать их самих. или нет. Для решения этой проблемы можно ограничить определение I.

Определение I'

Множеством левых к-границ конструкции, выводимых из понятия А, называется множество

$$LG_k(A)=\{x \mid x\in C_A^{k,0}\} = C_A^{k,0}.$$

Другими словами, ко множеству относятся только к-символьные цепочки символов, стоящие перед конструкцией, выводимой из А. При таком определении всегда известно, что левая граница пропускаться не будет. Возможный пропуск правой границы решается при помощи средств синтаксического анализатора.

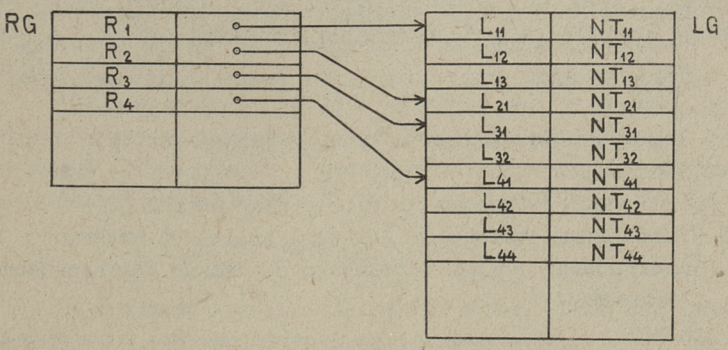
В СПТ ТПИ доступны множества $C^{1,0}$, $C^{0,1}$, $PREF_1$ и SUF_1. Из этого следует, что вычисляемы и множества LG_1 и RG_1 - т.е. односимвольные границы. Точнее, они вычисляются для понятий, описанных при помощи правил итерации. Таким образом, решается и вторая поставленная проблема.

Как было сказано уже выше, в решении этих проблем свое слово может сказать и пользователь. Такая возможность предоставлена ему и в СПТ ТПИ. Пользователь может описывать в метаязыке такие понятия языка, которые не стоят в правилах итерации, но пропуск которых все же разрешается. Множества левых и правых I-границ конструкций, выводимых из таких понятий, определяются описанным выше способом.

По определениям I' и 2 можно вычислять границы пропускаемых конструкций для применения в любом анализаторе, использующем (m,k)-контекст, а также $LR(k)$-контекст. Все же следует отметить, что и по методу предшествования с привлечением (I,I)-ОКК, анализируются все детерминированные языки [2].

4. Реализация метода нейтрализации ошибок в СПТ ТПИ

По принципам, изложенным в предыдущем пункте, в стадии генерации транслятора таблицы левых и правых I-границ строятся следующим образом.



Фиг. 1. Таблицы RG и LG.

R_i - обозначает код символа, являющегося правой I-границей какой-то пропускаемой конструкции;

L_{ij} - обозначает код символа, являющегося одной из левых I-границ той же конструкции;

NT_{ij} - обозначает символьное представление названия понятия (данное пользователем в метаязыке), из которого выводима та же конструкция. NT_{ij} используется в сообщении об ошибке и о пропуске конструкции.

Подобные таблицы используются во время работы транслятора, когда при обнаружении синтаксической ошибки (и только тогда) включается программа, работающая по следующему алгоритму.

Алгоритм I нейтрализация синтаксических ошибок.

A1 Выдача объявления об ошибке.
Выключение дальнейших семантических действий.

A2 Поиск во входном тексте символа, принадлежащего во множестве правых I-границ (R_i в таблице RG).

A3 Если правая I-граница нашлась, поиск в магазине анализа символа, принадлежащего во множество левых I-границ, соединенный с найденной правой I-границей (L_{ij} в таблице LG).

A4 Если левая I-граница нашлась, имеется ли отношение предшествования (I), между левой и правой I-границей.

A5 Если отношение (I) имеется, переход в A8.

A6 Включение правой I-границы в пропускаемую часть.
Имеется ли отношение предшествования (2) между левой I-границей и символом, следующим за правой I-границей.

A8 Пропуск части текста, стоящего между найденными границами.
Выдача объявления о пропуске конструкции.
Конец.

A9 Если левая I-граница не нашлась, переход в A2.

A10 Если правая граница не нашлась, выдача объявления о заканчивании входного текста и окончании синтаксического анализа.
Конец.

Алгоритм I требует некоторых пояснений.

I) Выключение дальнейших семантических действий в AI необходимо по следующей причине. По ходу работы алгоритма из магазина исключаются элементы, по которым уже могут быть выполнены семантические действия. Ввиду невозможности их заканчивания и приходится выключить все действия.

2) В A8, если имеется отношение (I), пропускаются элементы магазина, следующие за левой I-границей, и входной текст до правой I-границы. Если отношения (I) не имеется, а отношение (2) имеется, пропускается и сама правая I-граница.

3) В A7, если отношения (2) не имеется, то пропуск невозможен - очевидно была попытка недопустимого пропуска конструкции.

4) Переходы из A7 и A9 в A2 обозначают начало поиска границ новой конструкции, выводимой из понятия более высокого уровня.

5) После конца алгоритма в A8 синтаксический анализ продолжается, в AIO - кончается.

Программа по данному алгоритму, так же как и генерирующие программы, введены в СПТ ТПИ. Они испытывались на специальных языках, созданных для отладки средств, а также на языках системы ПАРЕС [3]. Результаты получились удовлетворительные. Их качество зависело в основном от числа в языке конструкции, описанных при помощи правил итерации, причем желательно, чтобы они взаимно содержали бы друг друга. Такие конструкции представляют возможность пропуска на разных уровнях и тем самым выбора из них самого узкого, что и является признаком качественной нейтрализации синтаксической ошибки.

Рассмотренные средства и образуют второй уровень каскадных средств обработки синтаксических ошибок. На нем базируются высшие уровни - реализуемые или находящиеся пока ь стадии проектирования.

Литература

I. Вадер А.Р., Вооглайд А.О. Описание метаязыка в системе построения трансляторов. - Тр. Таллинск. политехн. ин-та, 1978, № 439, с. 57-82.

2. Вооглайд А.О., Томбак М.О. Система построения эффективных многопроходных трансляторов с LR(к) семантикой. - Программирование, 1976, № 5, с. 28-36.

3. Разработка системы ведения баз данных и манипулирования данными. Отчет НИР. № госрегистрации 78037194. Таллинский политехнический институт, 1979.

4. Рохтла Х.Х. Каскадные средства обработки синтаксических ошибок. - Всесоюзная конференция по методом трансляции. Тезисы докладов. Новосибирск, 1981, с. 91-93.

5. Aho A.V., Johnson S.C. LR Parsing. - Computer Surveys, 1974, vol. 6, N 2, p. 99-124.

6. Ciesinger J. Generating error recovery in a computer writing system. - Fachtagung über Programmiersprachen, Springer-Verlag, 1976, p. 185-193.

7. Graham S.L., Haley S.P., Joy W.N. Practical LR error recovery. - ACM SIGPLAN Notices, (August 1979) 14:8, p. 168-175.

8. Graham S.L., Rhodes S.P. Practical syntactic error recovery. - Comm. ACM, 1975, 18, N 11, p. 639-650.

9. Horning J.J. What the compiler should tell the user. - Compiler Construction. An advanced course. 1976, chapt. 5, p. 525-548.

10. Krol J., Wyrostek P. Pewna metoda neutralizacij bledow w parserze precedensyjnym. - Podstawa Sterowania, Tom 10, 1980, z. 2, p. 93-108.

11. Ripley G.D. A simple recovery - only procedure for simple precedence parsers. - Comm. ACM, 1978, 31, N 11, p. 928-930.

12. Ripley G.D., Druseikis F.C. A statistical analysis of syntax errors. - Computer Languages, vol. 3, 1978, p. 227-240.

H. Rohtla

Syntax-Error Recovery in the Compiler-Writing System of TPI

Summary

This paper deals with a syntax-error recovery method, based on deleting the erroneous constructions. The algorithms for finding the borders of constructions and for error-recovery are given. The implementation of this technique in the compiler writing system of TPI is described.

Содержание

руб. 1.10