АНАЛИЗ ДАННЫХ В ЭКСПЕРТНЫХ СИСТЕМАХ
(Вычислительные системы)
1986 год Выпуск 117

УДК 519.68

К ФОРМАЛЬНОЙ МОДЕЛИ СИСТЕМЫ ПРОГРАММИРОВАНИЯ РИДЭР

М.А.Смоян

I. В настоящее время мы являемся свидетелями активного проникновения идей и концепций математической логики в теорию и практику программирования. В качестве примеров можно привести такие направления, как синтез и верификация программ, базы данных и базы знаний, абстрактные типы данных, функциональное программирование, искусственный интеллект. Появились концепции и соответствующие им средства программирования, базирующиеся практически полностью на логико-математических идеях. Примером могут служить концепция логического (лучше говорить, хорновского) программирования и язык ПРОЛОГ [I,2]. В качестве программ здесь выступают конечные наборы так называемых хорновских дизъюнктов (специального вида формул языка исчисления предикатов I-го порядка). Исполнение логической программы сводится к выполнению специальной процедуры поиска доказательства теорем. Отметим, что уровень языка ПРОЛОГ, если сравнивать его с традиционными языками программирования, достаточно высокий. Это связано с тем, что в логическом программировании пользователь описывает только условия решаемой задачи, а само решение находится по этому описанию с помощью некоторого универсального способа. В то же время при традиционном подходе программист обязан описать в виде императива сам способ решения задачи (вычислительная машина только реализует данный способ при конкретных входных данных).

Однако если сравнивать тот же язык ПРОЛОГ со всем языком исчисления предикатов, то его уровень окажется весьма низким. Поэтому вполне понятными являются попытки либо дополнять саму концепцию новыми идеями (отрицание невозможностью not, предикат DEMO, параметризация и т.п.), либо встраивать в язык ПРОЛОГ новые кон -

струкции, улучшающие его выразительную силу и повышающие эффективность исполнения программ (оператор управления выполнением cut (сечение), встроенные функции и предикаты, модульность, параметризация и т.п.). Другой путь - создание новых логических концепций. Примером может служить семантическое программирование [3].

В семантическом программировании в качестве логической основы программ для описания эффективных отношений взят в определенном смысле максимальный класс формул языка исчисления предикатов I-го порядка. Это так называемый класс Σ-формул. Данный класс формул обладает многими замечательными свойствами. Так, он вполне естественно допускает механизм параметризации (естественность здесь понимается в том смысле, что теоретико-модельная семантика Σ-формул с параметрами строится аналогично подобной семантике для Σ-формул без параметров). В классе Σ-формул возможно эффективное решение "рекурсивных уравнений". Наконец, данный класс формул допускает естественное расширение до языка, выражениями которого являются предикаты высших типов (т.е. предикаты, аргументами которых вновь могут быть предикаты). Исполнение семантической программы заключается в проверке ее теоретико-модельной истинности в фиксированной модели (задаваемой также семантической программой). Подобная проверка может осуществляться самыми различными способами, в том числе с помощью процедуры доказательства. Выбор конкретного способа проверки истинности (т.е. выбор Σ-машины) существенно определяется решаемым классом задач. При этом сам язык семантического программирования может существенно видоизменяться от класса к классу. Подобная ориентация на классы задач является одним из центральных методологических принципов концепции семантического программирования. В настоящей статье данный принцип получит свое "материальное" воплощение. Ниже будет описана математическая модель языка семантического программирования РИДЭР, ориентированного на решение задач обработки табличных данных [4]. Точнее, речь будет идти о структуре РИДЭР-программ, а также об описании и формировании данных.

2. Пусть $\mathcal{M}$ - многосортная модель сигнатуры σ_0. Положим $\sigma \Leftarrow \sigma_0 \cup \{\underline{\text{head}}, \underline{\text{tail}}, \underline{\text{cons}}, \underline{\text{tc}}, \underline{\text{nil}}, \sqsubseteq, \in, U\}$ и $\sigma^* \Leftarrow \sigma \cup \{P_i\}_{i \in \omega}$, где head tail, cons, tc, nil - списочные функции; $\sqsubseteq$, $\in$ - отношения на списках, а $(P_i)_{i \in \omega}$ - предикатные параметры. Напомним, что head - это операция взятия последнего элемента списка, tail - операция отбрасывания последнего элемента списка, cons - операция до-

бавления к списку нового элемента в качестве последнего, tc - транзитивное замыкание списка, т.е. список, элементами которого являются все элементы данного списка, элементы его элементов и т.д., nil - пустой список, $\subseteq$ - отношение "быть началом списка", $\in$ - "быть элементом списка", U - "быть праэлементом".

Для сигнатур σ_0, σ и σ^* обычным образом определяются классы Δ_0, Δ и Σ-формул. Через Σ^+ будем обозначать класс Σ-формул сигнатуры σ^*, у которых элементарные формулы вида $P(\bar{t})$, где $P \in (P_i)_{i\in\omega}$, входят только позитивно. Далее для Σ^+-формул будет предполагаться выполнимость аксиомы Σ^+-индукции [5], и тем самым мы можем считать, что в нашем распоряжении находится теория списочных надстроек GES_1^+ [5].

Пусть $S(\mathcal{M})$ - списочная надстройка над моделью $\mathcal{M}$, удовлетворяющая теории GES_1^+. Определим понятие числа*) в GES_1^+. Полагаем $0 \Leftrightarrow nil$, $1 \Leftrightarrow cons(nil, nil)$, ..., $n \Leftrightarrow cons(nil, n-1)$, ... Заметим, что в данном случае функция cons выступает как операция "прибавления единицы". Воспользовавшись теоремой о примитивной рекурсии для теории GES_1^+ (см. [5,6]), мы можем определить операцию Sum над числами. Напомним, что в семантическом программировании определяемыми конструкциями являются отношения. Поэтому и функции задаются в виде отношений. Таким образом, операция Sum будет задаваться в виде следующей Δ_0-программы:

```
Sum(x,y) = z def Nat(x)& Nat(y)& Nat(z) &
&[if y=0 then x=z else(Sum(x head(y))=w &Cons(w,1)=z);
Nat(x) def if x=0 then 0=0 else(List(x) &
        & x=Cons(0,head(x)) & Nat(head(x)); end;
```

Здесь $List \Leftrightarrow TU$.

В дальнейшем мы часто при определении функций будем пользоваться другой функциональной записью. Так, например, функция Sum может быть определена с помощью следующей системы функциональных равенств:

```
Sum(n,0) = n;
Sum(n,m) = Cons(Sum(n,m),1); end;
```

*) Заметим, что числа и операции над ними в РИДЭР являются сигнатурными (встроенными). Мы приводим здесь описание этих понятий для полноты изложения, не привязываясь к устройству конкретной модели $\mathcal{M}$.

Нетрудно видеть, что эта система определений и подобные ей функциональные определения легко транслируются в Δ_0-программы (в общем случае в Σ-программы).

В дальнейшем операцию $\lambda x.\underline{Cons}(\underline{nil},x)$ (при условии $\underline{Nat}(x)$) будем обозначать более привычным способом: $x+1$. То же касается и записи $\lambda x.\lambda y.\underline{Sum}(x,y)$; если $\underline{Nat}(x)$ и $\underline{Nat}(y)$, то будем писать $x+y$ вместо $\underline{Sum}(x,y)$. Определим теперь понятие длины списка $\underline{ln}(x)$, полагая

$$\underline{lh}(nil) = 0;$$

$$\underline{lh}(\underline{Cons}(\alpha,\delta)) = \underline{lh}(\alpha)+1;\ \underline{end};$$

Далее нам понадобится также понятие глубины списка $dh(x)$. Предварительно введем еще отношение $\leq$ и простую операцию $\underline{max}$ на числах. Полагаем:

$$x \leq y \ \underline{def}\ \underline{Nat}(x)\ \&\ \underline{Nat}(y)\ \&\ x \in \underline{tc}(y);\ \underline{end}.$$

Напомним, что операция $\underline{tc}$ может быть задана следующей функциональной схемой:

$$\underline{tc}(\underline{nil}) = \underline{nil};$$

$$\underline{tc}(\underline{cons}(\alpha,\delta)) = \underline{if}\ U(\delta)\ \underline{then}\ \underline{Cons}(\underline{tc}(\alpha),\delta)\ \underline{else}\ \underline{Cons}(\underline{tc}(\alpha),\ \underline{Cons}(\underline{tc}(\delta),\delta));\ \underline{end},$$

где

$$\underline{Cons}(\alpha,\underline{nil}) = \alpha;$$

$$\underline{Cons}(\alpha,\underline{Cons}(\beta,\delta)) = \underline{Cons}(\underline{Cons}(\alpha,\beta),\delta);\ \underline{end}.$$

Используя отношение $\leq$, определим операцию $\underline{max}$: $\underline{max}(x,y)=\underline{if}\ x \leq y\ \underline{then}\ y\ \underline{else}\ x;\underline{end}$. Теперь мы готовы определить глубину списков

$$\underline{dh}(\underline{nil}) = 0;$$

$$\underline{dh}(\underline{Cons}(\alpha,\delta)) = \underline{if}\ U(\delta)\ \underline{then}\ \underline{max}(\underline{dh}(\alpha),1)\ \underline{else}\ \underline{max}(\underline{dh}(\alpha),\ dh(\delta)) + 1;\ \underline{end};$$

Располагая $\underline{lh}$ и $\underline{dh}$, мы теперь в состоянии ввести в рассмотрение основные структуры данных, характерные для системы РИДЭР.

3. Основной структурой данных языка РИДЭР является таблица, которая, в свою очередь, может быть элементом другой основной структуры данных языка - набора данных или файла. Логически организованную совокупность наборов данных можно рассматривать как свое -

образную реляционную базу данных, программно представляющую ту проблемную область, к которой относятся решаемые задачи.

Примером традиционного понимания таблицы может служить следующая запись:

ФАМИЛИЯ	ИМЯ	ОТЧЕСТВО	ЦЕХ	ЗАРПЛАТА
Иванов	Иван	Иванович	I	220
Петров	Петр	Петрович	I	280
Дмитриев	Дмитрий	Дмитриевич	2	I80
. . . .	. . . .	. . . . .	. . .	. . .

Мы будем работать со списочным представлением таблицы. Так, например, списочным представлением вышеприведенной таблицы будет список

((Фамилия, Имя, Отчество, цех, зарплата),
(Иванов,Иван,Иванович, I, 220),
(Петров, Петр, Петрович, I, 280),
(Дмитриев, Дмитрий, Дмитриевич, 2 , I80),
(.....)

Первый элемент таблицы-списка будем называть заголовком таблицы, а элемент заголовка - реквизитом. Формальное определение понятия "таблица" вводится следующей Δ_0-программой:

$$\underline{\text{table}}(x) \ \underline{\text{def}} \ \underline{\text{dh}}(x) = 2 \ \& \ (\forall \beta \in x)(\beta \neq \text{nil}) \ \&$$
$$\& \ (\forall \beta \in x)(\forall \gamma \in x)(\underline{\text{lh}}(\beta) = \underline{\text{lh}}(\gamma)); \ \underline{\text{end}}.$$

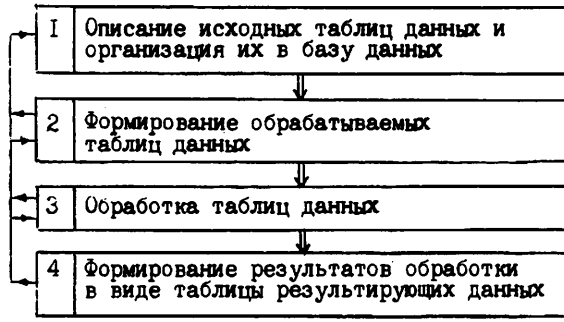
Далее, через TAIL и HEAD будем обозначать функции, аналогичные tail и head , но с обычной,традиционной семантикой,т.е., например, TAIL ((x,y,z)) = (y,z) и HEAD((x,y,z)) = x . Если x - таблица и TAIL(x) ≠ x , то будем говорить, что x - непустая таблица,и обозначать этот факт ftable(x).

Если x - таблица, γ ∈ HEAD(x) , т.е. γ является реквизитом и занимает i-ю позицию в заголовке (i ≤ lh(HEAD(x)) , то каждый i-й элемент списка β(отличного от заголовка) из таблицы x называется значением реквизита γ для β в таблице x. Введем предикат State(α, β,γ, δ) , содержательно означающий следующее: δ есть значение реквизита γ в строке β таблицы α. Например, если α - вышеприведенная таблица, β - ее вторая строка, то State (α, β , Имя, Иван) и State (α,β, Цех,I)- истинные высказывания, а значение State(α,β Цех, Андрей) ложно. Формально предикат State описывается следующей Δ_0-программой:

State($\alpha,\beta,\gamma,\delta$) def ftable(α) & ($\beta\in\alpha$) & ($\gamma \in$ HEAD(α)) &
&($\delta\in\beta$) & ($\exists\varepsilon \subseteq$ HEAD(α))($\exists\xi\sqsubseteq\beta$) (HEAD($\varepsilon$) = γ & HEAD(ξ)=δ)); end.

Имея в своем распоряжении предикат State, мы можем с его помощью решать самые различные задачи поиска нужной информации в базе данных. Заметим, что предикат State в языке РИДЭР представлен набором конструкций, что, вообще говоря, более удобно пользователю. Ниже мы подробнее остановимся на реализационных моментах модели языка РИДЭР.

4. Как уже отмечалось ранее, РИДЭР представляет собой язык обработки табличных данных и предназначен для адекватного программирования решения задач, допускающих следующую схему действий:



Нетрудно понять, что в эту схему укладывается большое число задач обработки, скажем, экономической или управленческой информации.

Описанную выше последовательность шагов естественно отразить и в структуре алгоритмов решения задач, и в последовательности технологических этапов написания соответствующих программ. Таким образом, проектирование РИДЭР-программы в соответствии с приведенной выше схемой разбивается на следующие стадии.

На первом шаге разрабатываются и уточняются описание и способ представления данных. В силу этого в РИДЭР-программе выделены конструкции, совокупность которых можно рассматривать как язык описания данных.

На втором шаге описываются требуемые логические связи между элементами данных, которые будут обрабатываться, а также указываются (если это необходимо) способы их выборки. Выборке элементов

из таблиц, хранящихся в базе данных, соответствует часть языка, которую естественно назвать языком манипулирования данными. Семантика этого языка, как мы убедимся ниже, полностью описывается предикатом State (см. п.3). Это обстоятельство, а также желание освободить пользователя от необходимости рутинного описания алгоритмов поиска данных и других технических деталей, возникающих при этом (например, обработка ситуации "конец файла"), позволяет придать языку манипулирования данными полностью дескриптивный логический характер.

На третьем этапе описывается уже собственно задача обработки данных. Здесь же описываются и спецификации результатов обработки. Для этих целей служит подъязык языка РИДЭР – язык обработки и отчета. Для подразумеваемого класса задач обработки табличных данных наиболее адекватным оказалось представление способов обработки данных не в виде Σ-определимых отношений, а в функционально-терминальном виде. Более того, из соображений эффективности реализации языка рекурсивные описания представлены циклическими конструкциями. В связи с этим по своей структуре и виду язык обработки и отчета напоминает традиционные, фон-неймановские языки программирования, хотя и имеет ряд отличительных особенностей (см. приложение). Однако для нас наибольший интерес будет представлять язык манипулирования данными.

5. Для каждой из таблиц, использующихся в РИДЭР-программе, необходимо указать условие выбора ее строк. Это выполняется одним из двух операторов: УВГФ или УВНФ (Условие Выборки Главного или соответственно Неглавного Файла). Если выбор строки таблицы не зависит от содержимого других таблиц, то набор данных, хранящий эту таблицу, объявляется главным. Кроме того, по содержимому главного файла определяется алгоритм обработки данных. Таким образом, объявление файла главным или неглавным не влияет на логическую структуру описаний, но способствует большей эффективности выполнения программ. Наличие таких управляющих конструкций вообще характерно для языка РИДЭР*). Некоторые другие управляющие конструкции будут указаны ниже. Влияние объявления файла главным или неглавным на эффективность выполнения программы заключается хотя бы в том, что главный файл загружается (обычно частями) в оперативную память, а для неглавных файлов условие загрузки надо указывать отдельно.

*) С необходимостью введения нелогических примитивов для управления процессом поиска вывода столкнулись и разработчики языков логического программирования (см., например, [2]).

Оператор,задающий условие выбора информации из главного файла, предназначен для формирования массива подлежащих обработке строк в оперативной памяти. Формат оператора:

$$\langle \text{имя файла} \rangle \quad \text{УВГФ} \left\{ \begin{array}{l} \text{ОЧЕ} \quad (\text{з. n}) \\ \text{ВСЕ} \quad (\text{з. n}) \end{array} \right\} \quad (\langle \text{условное выражение} \rangle),$$

где n - имя того же файла (таблицы).

Для неглавных файлов формат соответствующих операторов может быть использующим:

$$\langle \text{имя файла} \rangle \ \text{УВНФ} \left\{ \begin{array}{l} \text{ОЧЕ (з. n)} \\ \text{ВСЕ (з. n)} \\ \text{ОЧЕ (ф. n)} \end{array} \right\} (\langle \text{условное выражение} \rangle).$$

Содержательная семантика всех этих выражений одна и та же: выбрать все строки из таблицы, имя которой указано в операторе, удовлетворяющие приведенному условному выражению.Обозначения ОЧЕ, ВСЕ, з. и ф. не влияют на логику программы;единственное их назначение состоит в указании дополнительной информации о структуре таблицы: ОЧЕ означает, что строка,удовлетворяющая условному выражению,в таблице ровно одна; ВСЕ - что таких строк может быть,вообще говоря,несколько; з. означает,во-первых,что все строки,удовлетворяющие условному выражению,находятся после выделенной строки с указателем current, и,во-вторых, что все такие строки должны находиться в таблице одна за другой (т.е., например, если таблица упорядочена по фамилиям,то между двумя Сергеевыми может находиться только Сергеев); ф. - что связи между условным выражением и структурой таблицы, вообще говоря, не имеется, и таким образом, поиск надо вести по всей таблице.

После того, как некоторая часть строк выбрана в оперативную память,значение константы current устанавливается равным последней выбранной строке.Перед выполнением программы значение current не определено.

Укажем теперь точную семантику этих конструкций в терминах предиката State (см.п.3). Пусть задана конструкция языка РИДЭР:

$$\begin{array}{l} \text{ВСЕ} \quad (\text{з. } a_0)\, \varphi_0 \\ \quad \ldots \\ \text{ВСЕ} \quad (\text{з. } a_n)\, \varphi_n \ , \end{array} \qquad (I)$$

где a_0 - имя главного файла. Введем новые формулы ψ_i, $i = 0, \ldots, n$, полученные из φ_i заменой выражений $b \,.\, a_k$ на $\underline{\text{State}}(a_k, x_k, b, y_k)$,

$k = 0, \ldots, n$. Заметим, что выражение $b.a$ – это конструкция языка РИДЭР, содержательно означающая множество значений реквизита b у строк таблицы a. Таким образом, $b.a$ можно рассматривать как функцию, которая по строке x таблицы a в качестве значения выдает значение y реквизита b для этой строки. Естественно, что такое выражение формально эквивалентно State (b, x, a, y). При замене $b.a_k$, $k=0, \ldots, n$, (I) эквивалентно следующей Σ-схеме:

$R_0(x_0)$ def ϕ_0;

$R_1(x_1)$ def $(\exists x_0 \in a_0)(\phi_0 \,\&\, \phi_1)$;

. . . . . .

$R_n(x_n)$ def $(\exists x_0 \in a_0) \ldots (\exists x_{n-1} \in a_{n-1})(\phi_0 \& \ldots \& \phi_n)$; end.

ПРИМЕР. Пусть имеются таблицы ТО и ТI. Конструкция

ВСЕ (з. ТО) (ФАМИЛИЯ. ТI = ИВАНОВ)
ВСЕ (ф. ТI) (ЗАРПЛАТА. Т2 = ЗАРПЛАТА. ТI) (2)

является указанием для выборки всех объектов из ТО, имеющих фамилию Иванов, и всех объектов из ТI, имеющих ту же зарплату, что некоторый Иванов из ТО. Конструкция (2), согласно указанному выше алгоритму, переводится в Σ-схему

$R_0(x_0)$ def (State (ТО, x_0, ФАМИЛИЯ, ИВАНОВ);

$R_1(x_1)$ def $(\exists x_0 \in$ ТО$)$(State (ТО, x_0, ФАМИЛИЯ, ИВАНОВ) &

& State (Т1, x_1, ЗАРПЛАТА, z) &

& State (ТО, x_0, ЗАРПЛАТА, y) & y=z; end;

Как мы уже указывали, выражения ВСЕ и ОЧЕ, а также з. и ф. не влияют на логику конструкций языка РИДЭР, они служат подсказками для более эффективного выполнения программы. В [3] отмечено, что концепция Σ-программирования позволяет записывать стратегии выполнения Σ-программ также в виде Σ-схем. Здесь мы покажем, что указанные выражения также допускают адекватное представление в виде Σ-формул. Использование конструкции ОЧЬ означает, что элемент, который описывается этой конструкцией, всего один, т.е. запись

ОЧЬ (ф. a_i) φ_i или ОЧЬ (з. a_i) φ_i

означает, что выполняется Δ_0-формула $R_i(x)$ & $R_i(y) \to x = y$, а запись

ВСЕ (з. a_i) φ_i или ОЧЕ (з. a_i) φ_i

означает, что таблица a_i упорядочена, как указывалось в п.3, т.е. имеет место формула

$$(R_i(x) \& R_i(y) \to (\forall z \in a_i)[\underline{After}(a_i, x, z) \& \underline{After}(a_i, z, y) \to$$

$$\to R_i(z)]) \& (\forall z \in a_i)(R_i(z) \to \underline{After}(a_i, \underline{current}, z)).$$

где предикат After (a,x,y) (в таблице a строка y встречается не раньше строки x) описывается Σ-схемой:

After (a,x,y) def table (a) & x∈a & y∈a &

&(∃z ⊆ a)(y = head(z) & x ∈ z); end,

Таким образом, все конструкции логической части языка РИДЭР (как чисто логические, так и управляющие) являются просто другой записью соответствующих им конструкций Σ-языка.

6. Как видно из предыдущего, семантика основных конструкций языка манипулирования данными весьма прозрачна и допускает простую формализацию. Это же относится и к остальным фрагментам языка РИДЭР. Необходимость подобных описаний очевидна. Во-первых, появляется возможность ставить и решать вопросы непротиворечивости концепции языка в целом. Во-вторых, появляется возможность отделить главное от несущественных деталей; более отчетливо видны различные огрехи. В-третьих, видно, в каком направлении можно улучшать выразительную силу языка непротиворечивым образом. Например, предикат State дает возможность ставить и решать следующую задачу поиска: найти все те имена таблиц, для которых реквизит β имеет для всех (или некоторых, или ровно одной и т.п.) строк значение ε.

И таких задач существует множество. Все дело в том, чтобы найти для них наиболее простые и в то же время адекватные языковые конструкции. Заметим, что при этом необходимо учитывать, что язык РИДЭР ориентирован на пользователя, не обладающего достаточными знаниями ни по математической логике, ни по программированию. Поэтому синтаксис этого языка и должен включать в себя конструкции, близкие по своему устройству к выражениям естественного языка. Нужно, чтобы пользователь главным образом указывал "что делать", а не "как делать": он должен описывать лишь семантику своего алгоритма решения, не вдаваясь в детали реализации. Это описание, с одной стороны, является программой для ЭВМ (и весьма

эффективной, как показала практика), и с другой - оно может рассматриваться одновременно и как проектная документация на задачу. Тем самым РИДЭР позволяет как бы совместить этапы спецификаций, проектирования и кодирования программ. Отличительной особенностью РИДЭР-программ является их "фрагментарность": каждую программу можно рассматривать как совокупность нескольких самостоятельных частей, каждая из которых имеет свою конкретную замкнутую семантику и поэтому может составляться отдельно. Это обстоятельство позволяет в полной мере использовать преимущества коллективного написания программ. И, что очень важно, делает возможным легкую адаптацию и модификацию программ. Отметим еще раз, что все эти достоинства языка - следствие четкой ориентации на класс задач и на особенности алгоритмов решения задач этого класса.

Литература

1. KOWALSKI R.A. Logic Programming.- In: Proc.IFIP-83. Amsterdam, 1983, p.133-145.

2. CLOCKSIN W.F., MELISH C.S. Programming in PROLOG.-Berlin-Heidelberg-New York: Springer-Verlag, 1981.- 280 p.

3. ГОНЧАРОВ С.С., СВИРИДЕНКО Д.И. Σ-программирование.-В кн.: Логико-математические проблемы МОЗ (Вычислительные системы, вып. 107). Новосибирск, 1985, с. 3-29.

4. СМОЛН М.А. РИДЭР - логический язык для разработки систем обработки данных. -В кн.: Труды Всесоюз. конф. по прикладной логике. Новосибирск, 1985, с. 196-199.

5. СВИРИДЕНКО Д.И. Об одном варианте теории списочных надстроек. -В кн.: Прикладная логика (Вычислительные системы, вып. 116). Новосибирск, 1986, с. 67-69.

6. ГОНЧАРОВ С.С. Замечания об аксиомах списочной надстройки GES. -В кн.: Логические вопросы теории типов данных (Вычислительные системы, вып.114). Новосибирск, 1985, с.11-15.

Поступила в ред.-изд.отд.
17 июля 1986 года

Опишем решение задачи, на которой будут проиллюстрированы основные алгоритмические конструкции РИДЭР.

Требуется распределить стипендию студентам по итогам экзаменов, с учетом общественной активности и семейного положения каждого студента.

Размер стипендии устанавливается равным 57 рублям, если все оценки "отлично"; если средний балл не меньше 4,5, то стипендия - 50 рублей.Если же средний балл не меньше допустимого значения и количество оценок "удовлетворительно" не больше указанного, то студент получает стипендию в размере 45 рублей.Студент, получивший оценку "неудовлетворительно", не получает стипендию.

Общественная активность учитывается при определении среднего балла и количества оценок "удовлетворительно".

Введем обозначения реквизитов и файлов, которые необходимы для решения поставленной задачи.

Файл 01 - "Входные данные" состоит из реквизитов:

SG - шифр группы;

SS - шифр студента;

OC - оценка.

Файл 02 - "Справочные данные" состоит из реквизитов:

SG - шифр группы;

SS - шифр студента;

FIO - фамилия студента;

BAL - средний балл;

LB - количество допустимых оценок "удовлетворительно".

Файл 03 - "Выходные данные" включает реквизиты:

SG - шифр группы;

SS - шифр студента;

FIO - фамилия;

BAL - средний балл;

SUM - размер стипендии.

Программа манипулирования данными выглядит следующим образом:

```
C01 CX  (0101)
 01 УВГФ  ВСЕ(з.01)(SG.01=SG.01 & SS.01=SS.01)
 02 УВНФ  ОЧЕ(з.02)(SG.02=SG.01 & SS.02=SS.01)
 03 ПЗНФ
```

В этой программе конструкция СХ определяет программу обработки данных, код которой равен 0101. Конструкция ПЗЫФ указывает, что 03 является выходным файлом. УВГФ задает условие выборки данных из главного файла, а УВНФ определяет условие выбора данных из файла 02.

Программа обработки данных описывается следующим образом:

```
100  БУС  ВСЕ(з.01)(SG .01=SG .01 & SS .01=SS .01)
110  УС   OC.01=2
110  ДЕ   PR=1
120  УС   OC.01=3
120  БДЕ  L=L+1
100  БДЕ  S=S+OC .01
200  УС   SG 01'=SG.01!SS .01'=SS .01
210  УС   PR=1
210  ДЕ   SUM .03=0
210  ДЕ   BAL .03=0
220  УС   PR'=1
220  ДЕ   BAL .03=S/L
221  УС   BAL .03<BAL .02! LB .02<L
221  ДЕ   SUM .03=0
222  УС   BAL .03>=4.5 & LB .02>=L
222  ДЕ   SUM .03=50
223  УС   BAL .03=5
223  ДЕ   SUM .03=57
224  УС   BAL .03<=4.5 & LB.02<=L
224  ДЕ   SUM .03=45
230  ДЕ   SG .03=SG .01, SS .03=SS .01
230  ДЕ   FIO .03=FIO .02
```

В операторах с номерами, начинающимися на "1", происходит обработка информации об одном студенте; определяются количество оценок "удовлетворительно" и сумма оценок, если же у студента есть неудовлетворительная оценка, то этот студент выделяется.

В операторах с номерами, начинающимися на "2", происходит формирование выходных данных, определяется средний балл студента (оператор с номером 220), затем сравнивается значение среднего балла со справочным и определяется размер стипендии. Если же студент имеет оценку "неудовлетворительно", то размер стипендии равен 0 (оператор с номером 210).