ЛОГИЧЕСКИЕ МЕТОДЫ В ПРОГРАММИРОВАНИИ
(Вычислительные системы)
1987 год Выпуск 120

УДК 510.25:519.68

# ЛОГИЧЕСКИЙ ЯЗЫК ТИПОВ ДАННЫХ

А.В.Манцивода

Одной из важнейших проблем современного программирования является проблема типов данных и их представления [7]. Существует множество подходов к ее решению, и практически все они в той или иной мере связаны с математической логикой.

Важной частью этой проблемы является представление одного типа данных в другом. В настоящей работе изучается эта задача на основе теоретико-модельного подхода с использованием многосортных моделей [4]. Базисом логического языка типов данных, предлагаемого в статье, является концепция Σ-программирования, разработанная С.С.Гончаровым и Д.И.Свириденко [5]. Σ-программирование обладает рядом важных достоинств. Например, в нем естественно решаются многие вопросы, связанные с распараллеливанием вычислений. Кроме того, в основе Σ-программирования лежат принципы, обеспечивающие легкую настройку Σ-языка на конкретную предметную область. Таким образом, можно получить целый спектр Σ-языков, инвариантная часть которых основана на ясных и строгих принципах. Наконец, в Σ-программировании заложены предпосылки для успешной работы с типами данных.

Настоящую работу следует рассматривать как попытку сделать шаг в направлении построения практического языка Σ-программирования. Все неопределяемые в данной работе понятия можно найти в книгах по математической логике, например, в [1].

## §1. Списочные надстройки и типы данных

Зафиксируем две модели:

$$\mathcal{M} = \langle M_1, \ldots, M_k, \Omega_1 \rangle ,$$
$$\mathcal{K} = \langle K_1, \ldots, K_m, \Omega_2 \rangle ,$$

сигнатуры $\Omega_1$ и $\Omega_2$ соответственно; $K_i$ и $M_i$ - основные множества (сорта). Будем рассматривать их как абстрактные типы данных [10]. Нашей задачей является представление одного абстрактного типа, скажем $\mathcal{M}$, в другом типе $\mathcal{R}$. Для этого мы должны определить в $\mathcal{R}$ образы элементов, сигнатурных операций и отношений модели $\mathcal{M}$. Займемся первым пунктом: как нам представить сами элементы? Чаще при построении такого представления в качестве образов элементов $\mathcal{M}$ удобнее использовать не сами элементы основных множеств модели $\mathcal{R}$, а элементы более сложной структуры, "сконструированные" определенным образом из элементов $\mathcal{R}$. В программировании почти любой язык предоставляет средства для построения объектов, обладающих сложной внутренней структурой. Иногда эти средства называются конструкторами типов. Например, в языке ПАСКАЛЬ это массивы и записи, в СЕТЛ - множества, в ЛИСП - списки и т.д. Такие же средства рассматриваются и в теоретических работах, посвященных типам данных и основаниям программирования. К примеру, в работе [3] сложные объекты строятся с помощью понятия множества, а в [5] - списка. Мы в дальнейшем будем придерживаться концепции Σ-программирования и поэтому в качестве основного конструктора будем рассматривать списки.

Для этого воспользуемся следующей конструкцией. Построим над моделью $\mathcal{R}$ так называемую списочную надстройку $S(\mathcal{R})$: из элементов $\mathcal{R}$ строим списки, над ними снова списки и т.д. Особое место занимает пустой список nil, который в принципе может рассматриваться как атом. Над списком введем только одну операцию cons, которая присоединяет к списку новый элемент в качестве первого.

Естественно, для нужд программирования нам достаточно рассматривать не произвольные списочные надстройки, а надстройки, состоящие только из конечных списков. Такие надстройки мы будем в дальнейшем называть наследственно конечными и определять так:

$S^0(\mathcal{R}) \rightleftharpoons \{$конечные линейные списки из элементов $K = \bigcup_{j=1}^{m} K_j\}$,

$S^{i+1}(\mathcal{R}) \rightleftharpoons \{$конечные линейные списки из элементов $S^i(\mathcal{R}) \cup K\}$.

Пусть $S(\mathcal{R}) = \bigcup_j S^j(\mathcal{R}) \cup K$. Множество $S(\mathcal{R})$ и будет наследственно конечной списочной надстройкой над $\mathcal{R}$. Заметим, что в $S(\mathcal{R})$ у нас входят и сами элементы модели $\mathcal{R}$, что отличается от традиционного определения. Мы будем часто представлять списки из

$S(\mathfrak{A})$ в виде $\langle a_1,\ldots,a_1\rangle$ . Тогда операция cons определяется следующим образом: $\underline{cons}(a,\langle a_1,\ldots,a_1\rangle) = \langle a,a_1,\ldots,a_1\rangle$ . Мы будем использовать два равноправных обозначения пустого списка: nil и $\langle\ \rangle$ .

Определим по индукции ранг $r(a)$ элемента $a \in S(\mathfrak{A})$. Ранг всех атомов, за исключением nil, равен нулю. Ранг nil равен единице. Если $a = \underline{cons}(a_1,a_2)$ и $r(s_1) \geq r(a_2)$, то $r(a) = r(a_1)+1$, в противном случае, если $r(a_1) < r(a_2)$, то $r(a) = r(a_2)$. Грубо говоря, ранг есть максимальное число скобок в "глубину".

Теперь мы получили возможность при определении типа $\mathcal{M}$ в типе $\mathfrak{A}$ выбирать представителей для элементов модели не только среди самих элементов из $\mathfrak{A}$ , а в наследственно конечной списочной надстройке над $\mathfrak{A}$ . Здесь возникает следующая проблема. Пусть нам даны две модели: $\mathbb{Q} = \langle Q, +, \cdot\rangle$ – рациональные числа со сложением и умножением и $\mathbb{Z} = \langle Z, +, \cdot\rangle$ – целые числа с теми же операциями. Нашей задачей является представление типа $\mathbb{Q}$ в типе $\mathbb{Z}$. Поступая предложенным выше способом, мы легко найдем в наследственно конечной списочной надстройке над $\mathbb{Z}$ представителей элементов $\mathbb{Q}$. Это будут просто пары вида $\langle z_1, z_2\rangle$, определяющие рациональные числа $z_1/z_2$ . Но списочная надстройка над $\mathbb{Z}$ содержит списки сколь угодно большой сложности, а нам нужны только пары, т.е. некоторое подмножество $S(\mathbb{Z})$. Остальные элементы нам не нужны. Таким образом, мы естественным путем приходим к определению структурного типа над моделью $\mathfrak{A}$ как подмножества ее наследственно конечной списочной надстройки $S(\mathfrak{A})$. Понятно, что с практической точки зрения нас интересуют не произвольные подмножества $S(\mathfrak{A})$, а "хорошие": было бы, например, неплохо, если бы мы могли по любому элементу из $S(\mathfrak{A})$ и типу T узнать, принадлежит ли элемент типу или нет и т.д.

В типизированных алгоритмических языках, например в языке ПАСКАЛЬ, можно выделить подъязыки, описывающие типы данных. Мы также в дальнейшем рассмотрим язык типов данных, основанный уже на логических средствах.

Вообще язык типов данных должен отвечать двум весьма противоречивым требованиям:

I) выразительности – нам хотелось бы иметь возможности строить как можно больше разнообразных типов данных;

2) конструктивности - ключевые проблемы, возникающие при работе с таким языком, в частности проблема принадлежности элемента типу, должны быть разрешимыми.

Поэтому любой такой язык есть некоторый компромисс между I-м и 2-м, и понятно, что в этом случае нет единого решения, которое всех бы удовлетворило. Автором была предпринята попытка построить такой язык, который, обладая широкими выразительными возможностями, оставался бы в большей мере в рамках конструктивности. Но прежде чем перейти к его определению, рассмотрим еще два важных аспекта.

## §2. Инкапсуляция и ссылочные типы

Предположим, что мы определили абстрактный тип данных $\mathcal{M}$ в типе $\mathcal{R}$, пользуясь наследственно конечной списочной надстройкой над $\mathcal{R}$. Чаще всего представители элементов из $\mathcal{M}$ обладают, как уже говорилось, сложной внутренней структурой, т.е. являются списками, "собранными" из элементов модели $\mathcal{R}$. Поэтому при работе с таким представлением модели $\mathcal{M}$ нам все время придется учитывать сложность структуры элементов (кстати, так же, как формульных операций и отношений, представляющих операции и отношения из $\mathcal{M}$). Это как раз тот случай, когда "слишком много знать весьма вредно" - было бы гораздо удобнее считать эти элементы, операции и отношения н е д е л и м ы м и. В программировании такими вопросами - так называемой инкапсуляцией типов данных - занимаются уже достаточно давно и предложено много вариантов решения этой проблемы (см. обзор [7]).

Кратко опишем простую идею, которой будем придерживаться в настоящей работе.

Чтобы смоделировать некоторую область реального мира, мы при определении типов данных должны будем где-то остановиться, признать некоторые объекты неделимыми, первичными. Смоделировать же первичные объекты на ЭВМ можно только с помощью чисел и заново вводимых имен, например скалярных типов в языке ПАСКАЛЬ. Это имена, которые внутри данной модели на самом деле не являются именами н и к а к и х объектов, т.е. имена, не имеющие денотата. Из первичных объектов можно "собрать" объекты более сложные, снова упрятав их за некоторыми именами и т.д. С объектами типа ИМЯ можно работать и как с символами, и как с элементами, указывающими на свой денотат.

В качестве примера элементов такого типа приведем пропозициональные переменные исчисления высказываний. В формуле $A \& B \to C$ переменные A, B и C можно рассматривать двояко: с одной стороны (синтаксически), это просто символы, не имеющие внутреннего содержания, с другой (семантически) - их можно рассматривать как и м е н а истинностных значений. В первом случае они ведут себя аналогично элементам скалярного типа, во втором - так же, как элементы ссылочного типа [II]. Заметим, что в математике (как, впрочем, и в любой другой области человеческой деятельности) понятие имени используется очень широко и плодотворно. Тщательное исследование этих проблем содержится во введении книги [I2].

Понятие имени будет представлено на выбранном нами языке многосортных моделей следующим образом.

С настоящего момента будем считать, что одним из основных множеств модели $\mathfrak{A}$ является множество натуральных чисел $\mathbb{N}$; если это не так, то можно ввести их перед построением списочной надстройки. Определим нумерацию $\nu: \mathbb{N} \to S(\mathfrak{A})$ с единственным пока условием $\nu(0) = 0$. Будем говорить, что натуральное число $n \in \mathbb{N}$ является именем элемента $t \in S(\mathfrak{A})$, если $t = \nu n$.

Здесь следует подчеркнуть, что введение дополнительного множества $\mathbb{N}$ на самом деле совершенно не обязательно: язык списочных надстроек достаточно богат, чтобы без привлечения дополнительных средств реализовать идею именования. Натуральные числа добавлены из чисто технических соображений. Кроме того, нам легко считать, что натуральные числа не обладают внутренней структурой, что важно для понятия имени, в то время как при кодировании имен списками нам потребуется приложить для этого некоторые усилия.

Оказалось также, что аппарат именования может быть использован для моделирования так называемых ссылочных типов, или указателей, которые являются важной составной частью таких языков программирования, как ПАСКАЛЬ и СИ, и используются для определения рекурсивных типов. В отличие от обычных переменных, содержимым которых, грубо говоря, являются сами объекты, содержимым ссылочной переменной является адрес, т.е. номер ячейки памяти, в которой данный объект находится.

## §3. Хорошие нумерации

Как уже было сказано выше, введение нумераций позволяет моделировать два важных понятия: инкапсулирование и ссылочные типы. С помощью ссылок можно определять элементы сложной структуры, мы их будем в дальнейшем называть динамическими. Динамический элемент является на самом деле ориентированным графом, вершины которого сами обладают сложной структурой. В нашем случае вершинами таких графов являются элементы $S(\mathfrak{A})$ и существует ребро из элемента a в b, если $a = \langle \ldots n \ldots \rangle$ и $b = \nu n$; в таких случаях мы говорим, что a содержит ссылку (имя) на b. Итак, динамическим элементом назовем множество $\hat{a}$ элементов наследственно конечной списочной надстройки $S(\mathfrak{A})$ такое, что если b, $b = \langle \ldots, n, \ldots \rangle$, принадлежит $\hat{a}$, то элемент $\nu n$ также принадлежит $\hat{a}$. Здесь n – натуральное число. Другими словами, множество $\hat{a}$ замкнуто относительно ссылок. Динамический элемент $\hat{a}$ назовем конечно-порожденным, если существуют $a_1, \ldots, a_l \in S(\mathfrak{A})$ такие, что $\hat{a}$ является замыканием множества $\{a_1, \ldots, a_l\}$ относительно ссылок.

Очевидно, структура динамических элементов полностью зависит от нумерации. Нам бы хотелось иметь такую нумерацию, чтобы с ее помощью определялись всевозможные конечные динамические элементы (именно они наиболее интересны с точки зрения программирования), т.е. чтобы были любые элементы, представимые конечным ориентированным графом с допустимой структурой вершин. Не все нумерации обладают таким свойством. К примеру, гёделева нумерация, как несложно показать, будет определять только динамические элементы, представляющие собой цепочки элементов множества $S(\mathfrak{A})$, и не будет допускать циклических элементов вида: $a_1 \to a_2 \to \ldots \to a_l \to a_1$.

С другой стороны, было бы плохо, если бы нумерация допускала динамические элементы бесконечной длины: $a_1 \to a_2 \to a_3 \to \ldots$

Нумерацию назовем хорошей, если выполняются три условия:

1) с ее помощью в модели определяются все возможные конечные динамические элементы;

2) не допускаются конечно-порожденные бесконечные динамические элементы;

3) модель $\langle \mathfrak{A}, \nu \rangle$ является конструктивной [2].

**ТЕОРЕМА 1.** *Для любой конструктивизируемой модели* $\mathfrak{A} = \langle K_1, \ldots, K_m, \mathbb{N}; \Omega \rangle$ *существует хорошая нумерация множества* $S(\mathfrak{A})$.

Такая нумерация строится конструктивно с использованием нумерации всех конечных ориентированных графов.

Хорошие нумерации будут иметь принципиально важную роль для дальнейшего изложения.

Теперь перейдем к определению языка типов данных $\mathcal{L}_D$.

## §4. Логический язык типов данных

Определим язык $\mathcal{L}_D$ следующим образом:

1. Символы $K_1, K_2, \ldots, K_m$, NIL, NULL, NUM являются атомными формулами $\mathcal{L}_D$. Счетное множество символов $A_1, A_2, \ldots$ также принадлежит $\mathcal{L}_D$.

2. Если $F, G \in \mathcal{L}_D$, то $F \& G$, $F \vee G$, $\neg F$, $\uparrow F$, $\overline{\text{CONS}}[F,G] \in \mathcal{L}_D$.

3. Других формул нет.

Теперь определим понятие истинности формулы $\mathcal{L}_D$ на элементах множества $S(\mathcal{L})$. Символы $K_i$ интерпретируются одноместными предикатами, истинными только на элементах основного множества $K_i$, а NUM истинен только на положительных натуральных числах, NIL – на пустом списке, NULL – на нуле.

Если $F = F_1 \& F_2$, то F истинна на элементе a тогда и только тогда, когда $F_1$ и $F_2$ истинны на a.

Дизъюнкция и отрицание определяются аналогично. Формула $\overline{\text{CONS}}[F,G]$ интерпретируется как $\overline{\text{CONS}}[F,G](x) \rightleftharpoons \exists x_1 \exists x_2 (F(x_1) \& G(x_2) \& x = \overline{\text{cons}}(x_1, x_2))$, $\uparrow F$ интерпретируются как $\uparrow F(n) = F(\nu n)$.

Таким образом, тип $\uparrow F$ содержит все имена элементов типа F.

Для определения рекурсивных типов введем понятие контекстного множества.

Контекстное множество $\mathcal{A}$ есть конечное множество формул вида $A_i \equiv F_i$, называемых контекстами, $F_i \in \mathcal{L}_D$. Заметим, что контексты не принадлежат языку $\mathcal{L}_D$.

Будем говорить, что G получена из H подстановкой относительно $\mathcal{A}$, если G равно H, за исключением некоторого вхождения $A_i$, которое в формуле G заменено на $F_i$, и контекст $A_i \equiv F_i$ принадлежит $\mathcal{A}$.

Формула $A_i$ входит в формулу $F_i$, если существует последовательность $G = G_1, G_2, \ldots, G_n$ подстановок, возможно, пустая и такая, что $A_i$ входит в формулу $G_n$ явно.

В дальнейшем будем рассматривать не произвольные контекстные множества, а те, которые удовлетворяют следующим трем условиям:

I. Контекст $A_i \equiv F_i$ может принадлежать $\mathcal{A}$, если $A_i$ либо не входит в $F_i$, либо входит в $F_i$ строго позитивно, т.е. не находится под влиянием ни одной связки отрицания.

2. $A_i$ может входить в $F_i$ только "внутри" связок $\overline{CONS}$, в частности, запрещаются такие контексты, как

$$A_i \equiv A_i \vee F_i',$$

$$A_i \equiv \neg A_i \& F_i, \ldots$$

3. Если существует контекст, в который $A_i$ входит в правую часть, то существует ровно один контекст, в который $A_i$ входит слева, другими словами, любой предикатный символ $A_i$, входящий в контексты из $\mathcal{A}$, должен быть однозначно определен в $\mathcal{A}$.

Контекст $A_i \equiv F_i$ на самом деле есть не что иное, как определение нового типа данных с именем $A_i$.

Семантика контекстов не так прозрачна, как в предыдущих случаях. Мы могли бы интерпретировать символ $A_i$ как предикат - наименьшую неподвижную точку монотонного определенного формулой $F_i$ оператора $\Gamma_{F_i}$, так как $A_i$ входит в $F_i$ строго позитивно. Но в этом случае, как несложно убедиться, $A_i$ не будет выполняться ни на одном динамическом элементе, содержащем цикл, даже если по всем другим параметрам этот элемент и удовлетворяет определению типа. Для более точного определения семантики контекстов введем конструкцию "наибольшей" неподвижной точки. Пусть контекстное множество $\mathcal{A}$, состоящее из контекстов

$$A_1 \equiv F_1,$$

$$\ldots$$

$$A_l \equiv F_l,$$

удовлетворяет вышеприведенным требованиям. Тогда с помощью $\mathcal{A}$ можно определить оператор $\Gamma$, действующий на списках предикатов. Пусть $\langle \tilde{A}_1, \ldots, \tilde{A}_l \rangle$ - список некоторых одноместных предикатов. Тогда $\Gamma$ переводит $\langle \tilde{A}_1, \ldots, \tilde{A}_l \rangle$ в $\langle \tilde{A}_1', \ldots, \tilde{A}_l' \rangle$, где $\tilde{A}_i'$ эквивалентен формуле $F_i$ с интерпретацией всех $A_i$ как $\tilde{A}_i$. Наибольшая неподвижная точка оператора $\Gamma$ определяется так:

1) первоначально имеем список тождественно истинных предикатов $\langle \tilde{A}_1^0, \ldots, \tilde{A}_l^0 \rangle$;

2) пусть на $i$-м шаге получен список $\langle \tilde{A}_1^i, \ldots, \tilde{A}_l^i \rangle$. Тогда $\langle \tilde{A}_1^{i+1}, \ldots, \tilde{A}_l^{i+1} \rangle = \Gamma(\langle \tilde{A}_1^i, \ldots, \tilde{A}_l^i \rangle)$.

Определяем $\tilde{A}_i$ как $\bigcap_j \tilde{A}_i^j$. Это и будет искомая интерпретация предикатного символа $A_i$.

Отметим, что корректность данного определения следует из условий, накладываемых на контекстное множество $\mathcal{C}$ и нумерацию $\nu$.

В дальнейшем с помощью $SK(\mathcal{C})$ будем обозначать модель $SK(\mathcal{C}) = \langle S(\mathfrak{A}); NUM, NULL, NIL, K_1, \ldots, K_m, A_{i_1}, \ldots, A_{i_1} \rangle$ языка $\mathcal{L}_D$, где $A_{i_1}, \ldots, A_{i_1}$ определены в $SK(\mathcal{C})$ контекстным множеством $\mathcal{C}$.

В заключение параграфа приведем примеры определений типов с помощью контекстов.

ПРИМЕР I. Выше уже рассматривалось представление рациональных чисел $Q$ с помощью пар целых чисел. Если $Z(x)$ - предикатный символ, интерпретируемый как предикат, истинный на целых числах, то тип $Q$ рациональных чисел может быть определен с помощью следующего контекста: $Q \equiv \overline{CONS}[Z, \overline{CONS}[Z, NIL]]$. При этом тип $\uparrow Q$ - тип "имен" рациональных чисел, т.е. натуральных чисел, кодирующих пары $\langle z_1, z_2 \rangle$, $z_i \in \mathbb{Z}$.

ПРИМЕР 2. Пусть $Z(x)$ - предикат целых чисел. Тогда контекст $LISTZ = NIL \vee \overline{CONS}[Z, LISTZ]$ определяет тип LISTZ всех списков, элементами которых являются целые числа и только они.

ПРИМЕР 3. Можно определить тип "натуральные числа" N с помощью контекста $N \equiv NIL \vee \overline{CONS}[N, NIL]$. Его элементами будут списки вида $\langle\langle \ldots \langle\rangle \ldots \rangle\rangle$. Числу $n$ соответствует список, содержащий $2n$ скобок. Определим $\nu: \{x \mid N(x)\} \to S(\mathfrak{A})$. Элементы типа N можно рассматривать теперь как имена объектов из $S(\mathfrak{A})$. Здесь мы сталкиваемся с некоторыми трудностями, связанными со "смешением" типов, например, $\langle\rangle$ принадлежит типу LISTZ из примера 2 и там рассматривается совсем не как ссылка, а как список, не содержащий элементов.

## §5. Исчисление типов данных

Введенный в предыдущем параграфе язык $\mathcal{L}_D$ выбран таким образом, что его формулы не "различают" два неравных элемента одного основного множества $K_i$: для любых $a, b \in K_i$, если формула $F \in \mathcal{L}_D$ выполняется на $a$, то она обязательно будет выполняться и на элементе $b$. Этот выбор был сделан преднамеренно, чтобы сфокусировать внимание именно на структуре элемента, не касаясь вопроса, из каких атомов этот элемент состоит. В $\mathcal{L}_D$ также не входят предикатные и функциональные символы сигнатуры модели $\mathfrak{A}$. Поэтому вместо произвольной m-сортной модели достаточно рассматривать очень простую модель с пустой сигнатурой $\Omega_0$: $T = \langle \{r_1\}, \{r_2\}, \ldots$

$\ldots,\{r_m\},\mathbb{N};\Omega_0\rangle$, где i-е основное множество состоит из единственного символа $r_i$; $\mathbb{N}$ - множество натуральных чисел.

Наследственно конечной надстройкой над T будет множество термов $S(T)$, определяемое так:

I) $r_i \in S(T)$; $\mathbb{N} \subseteq S(T)$: $\underline{nil} \in S(T)$ и является списком, где $\underline{nil}$ - новый символ, понимаемый нами как пустой список;

2) если $t_1, t_2 \in S(T)$ и $t_2$ - список, то терм $\underline{cons}(t_1,t_2) \in S(t)$ и является списком;

3) в $S(T)$ входят только элементы, определенные в пп. I и 2.

Очевидно, существует естественное склеивающее отображение из $\mathcal{A}$ на T, индуцирующее отображение из $S(\mathcal{A})$ на $S(T)$. Можно также показать, что нумерация $\nu$ множества $S(\mathcal{A})$ некоторым образом индуцирует нумерацию $\nu_0$: $\mathbb{N} \to S(T)$, причем если $\nu$ - хорошая, то $\nu_0$ также является хорошей нумерацией.

Сформулируем понятие представителя элемента $S(\mathcal{A})$ в $S(T)$: $r_i$ есть представитель любого $a \in K_i$, элемент $\underline{nil} \in S(T)$ есть представитель $\langle\,\rangle \in S(\mathcal{A})$. Натуральные числа представляют самих себя, $\underline{cons}(t_1,t_2)$ есть представитель $\langle a_1,\ldots,a_1\rangle$, если $t_1$ - представитель $a_1$ и $t_2$ - представитель $\langle a_2,a_3,\ldots,a_1\rangle$.

Зафиксируем некоторое контекстное множество $\mathcal{C}$. Обозначим через $ST(\mathcal{C})$ модель языка $\mathcal{L}_D$ с основным множеством $S(T)$ и аксиомами из $\mathcal{C}$.

ЛЕММА I. $F \in \mathcal{L}_D$ выполняется на элементе $a \in S(\mathcal{A})$ тогда и только тогда, когда $ST(\mathcal{C}) \models F(t)$, где t - представитель a.

Введем теперь секвенциальное исчисление с "от доказываемой формулы к аксиомам" языка $\mathcal{L}_D$. Термами будут элементы множества $S(T)$. Буквы $\Delta,\Sigma$ обозначают последовательности формул.

Аксиомами будут:

$\Delta \to K_i(r_i),\Sigma$; $\quad \Delta \to NIL(\underline{nil}),\Sigma$;

$\Delta \to NUM(n),\Sigma$; $\quad \Delta \to NULL(t) \to \Sigma$; $t \neq 0$;

$\Delta, K_i(t) \to \Sigma$; $\quad \Delta \to NULL(0),\Sigma$;

$\Delta, NUM(t) \to \Sigma$; $\quad \Delta, NIL(t) \to \Sigma$; $t \neq \underline{nil}$;

$\Delta, \overline{CONS}[F,G](t) \to \Sigma$; $\quad t$ не список;

$\Delta, \uparrow F(t) \to \Sigma$; $t \notin \mathbb{N}$.

Правила вывода:

Iа) $\dfrac{NIL(\underline{nil}),\Delta \to \Sigma}{\Delta \to \Sigma}$ ,

Iб) $\dfrac{NUM(n),\Delta \to \Sigma}{\Delta \to \Sigma}$ ,

Iв) $\dfrac{NULL(0),\Delta \to \Sigma}{\Delta \to \Sigma}$ ,

Iг) $\dfrac{K_i(r_i),\Delta \to \Sigma}{\Delta \to \Sigma}$ ,

2а) $\dfrac{\Delta \to F \,\&\, G,\Sigma}{\Delta \to F,\Sigma \quad \Delta \to G,\Sigma}$ ,

2б) $\dfrac{\Delta, F \,\&\, G \to \Sigma}{\Delta, F, G \to \Sigma}$ ,

3а) $\dfrac{\Delta \to F \vee G,\Sigma}{\Delta \to F,\ G,\Sigma}$ ,

3б) $\dfrac{\Delta, F \vee G \to \Sigma}{\Delta, F \to \Sigma \quad \Delta, G \to \Sigma}$ ,

4а) $\dfrac{\Delta \to \uparrow F(n),\Sigma}{\Delta \to F(\nu n),\Sigma}$ .

4б) $\dfrac{\Delta, \uparrow F(n) \to \Sigma}{\Delta,\ F(\nu n) \to \Sigma}$ ,

5а) $\dfrac{\Delta, \overline{CONS}[F,G](\underline{cons}(t_1,t_2)) \to \Sigma}{\Delta, F(t_1),\ G(t_2) \to \Sigma}$ ,

5б) $\dfrac{\Delta \to \overline{CONS}[F,G](\underline{cons}(t_1,t_2)) \to \Sigma}{\Delta \to F(t_1),\Sigma \quad \Delta \to F(t_2),\Sigma}$ ,

6а) $\dfrac{\Delta \to \mathbf{A}(t),\Sigma}{\Delta \to F(t),\Sigma}$ ,

6б) $\dfrac{\Delta,\ \mathbf{A}(t) \to \Sigma}{\Delta, F(t) \to \Sigma}$ ,

если $A \equiv F$ принадлежит $\mathcal{A}$ ;

7а) $\dfrac{\Delta \to \overline{CONS}[F,G]\ (t),\Sigma}{\Delta \to \Sigma}$ , $t$ не список;

7б) $\dfrac{\Delta \to \uparrow F(t),\Sigma}{\Delta \to \Sigma}$ , $t \notin \mathbf{N}$;

8а) $\dfrac{\dfrac{\dfrac{\Delta \to F(t),\Sigma}{\vdots}}{\Delta_1 \to F(t),\Sigma_1}}{\Delta_1 \to NIL(\underline{nil}),\Sigma_1}$

8б) $\dfrac{\dfrac{\dfrac{\Delta, F(t) \to \Sigma}{\vdots}}{\Delta_1, F(t) \to \Sigma_1}}{\Delta_1 \to \Sigma_1}$ ,

если нижнее вхождение $F(t)$ есть потомок верхнего;

9а) $\dfrac{\Delta \to \neg F,\Sigma}{\Delta,\ F \to \Sigma}$ ,

9б) $\dfrac{\Delta, \neg F \to \Sigma}{\Delta \to F,\Sigma}$ ,

а также обычные структурные правила. Условия на применимость: сначала применяются правила 8а и 8б и аксиомы; если они неприменимы, то используется любое другое правило. Как видим, это не

совсем обычное секвенциальное исчисление: во-первых, вывод зависит от нумерации, во-вторых, правила ∀а и ∀б не традиционны для исчисления секвенций, хотя в натуральном выводе есть их аналоги, например:

$$\frac{\begin{array}{c}[\,A\,]\\ \vdots\\ B\end{array}}{A \supset B}\ .$$

Нелогическими аксиомами служат элементы контекстного множества $\mathcal{A}$ .

ТЕОРЕМА 2. $ST(\mathcal{A}) \models F(t)$ тогда и только тогда, когда $\mathcal{A} \vdash_c F(t)$.

Назовем дерево вывода в этом исчислении с-деревом; с-доказательство есть с-дерево, все терминальные вершины которого являются аксиомами. Далее, с-дерево формулы F будет ее с-опровержением, если среди его терминальных вершин встретится пустая секвенция $\to$ .

ЛЕММА 2. Если $\nu_0$ - хорошая нумерация $S(T)$, то для любой формулы $F(t)$ число ее с-деревьев конечно.

ДОКАЗАТЕЛЬСТВО заключается в нахождении оценки длины максимальной нити с-дерева формулы $F(t)$ с использованием условий на применимость правил вывода.

ТЕОРЕМА 3. Если $\nu$ - хорошая нумерация множества $S(\mathfrak{K})$, то проблема принадлежности элемента типу разрешима.

Эта теорема является следствием теоремы 2 и предыдущей леммы.

Действительно, пусть $a \in S(\mathfrak{K})$ и нам надо проверить, принадлежит ли элемент а типу, определяемому формулой F. Пусть $t \in S(T)$ - представитель а. Если с-дерево формулы $F(t)$ является ее с-доказательством, то $a \in F$, в противном случае $a \notin F$ .

## §6. Позитивность типов данных

Цель настоящего параграфа - показать, что любое контекстное множество $\mathcal{A}$ можно видоизменить таким образом, что для каждой формулы $F \in \mathcal{L}_D$ будет существовать позитивная, т.е. без связок $\neg$ , формула $F^+ \in \mathcal{L}_D$ такая, что формулы F и $F^+$ определяют один и тот же тип данных. Иными словами, любой тип данных, определимый на языке $\mathcal{L}_D$ , может быть определим и позитивной формулой.

Предварительно сформулируем несколько простых лемм.

ЛЕММА 3. Следующие типы определимы позитивными контекстами, т.е. контекстами вида $A \equiv F$, где $F$ позитивна:

1) тип $U$ всех атомов $U = K_1 \vee \ldots \vee K_m$;

2) тип $S$ всех списков из $S(\mathcal{A})$ $S \equiv NIL \vee \overline{CONS}[U \vee S \vee N, S]$;

3) тип $N$ всех натуральных чисел $N \equiv NUM \vee NULL$;

4) тип $L$ всех элементов $S(\mathcal{A})$ $L \equiv U \vee S \vee N$;

5) тип $P_i$, выделяющий все элементы $S(\mathcal{A})$, кроме элементов основного множества $K_i$, $P_i \equiv K_1 \vee \ldots \vee K_{i-1} \vee K_{i+1} \vee \ldots \vee K_m \vee S \vee N$;

6) тип NON, выполняющийся на всех элементах $S(\mathcal{A})$, кроме nil, $NON \equiv U \vee \overline{CONS}[L,S] \vee N$;

7) тип NPOS, выполняющийся нв всех элементах, кроме положительных чисел: $NPOS \equiv U \vee S \vee NULL$;

8) тип NNU, выделяющий все элементы $S(\mathcal{A})$, кроме нуля, $NNU \equiv U \vee S \vee NUM$.

В дальнейшем мы будем неоднократно пользоваться именами нововведенных типов.

ЛЕММА 4. Для всех $F,G \in \mathcal{L}_D$ формулы $\neg\overline{CONS}[F,G]$ и $U \vee N \vee \overline{CONS}[\neg F,G] \vee \overline{CONS}[L,\neg G \,\&\, S]$ эквивалентны на $SK(\mathcal{A})$.

ЛЕММА 5. Для любой формулы $F \in \mathcal{L}_D$ формулы $\neg\neg F$ и $(\neg\neg F) \,\&\, NNU \vee U \vee S$ эквивалентны на $SK(\mathcal{A})$.

Несколько менее тривиальной является

ЛЕММА 6. Для любого контекста $A \equiv F$ существуют позитивные контексты $B_1 \equiv F_1$ и $B_2 \equiv F_2$ такие, что $t$ принадлежит типу $B_1$ тогда и только тогда, когда $A(t)$, и $t \in B_2$ тогда и только тогда, когда выполняется $\neg A(t)$.

Построим с помощью $\mathcal{A}$ контекстное множество $\mathcal{B}$ следующим образом: все контексты, определенные в лемме 3, принадлежат $\mathcal{B}$. Если $A \equiv F \in \mathcal{A}$, то $B_1 \equiv F_1$ и $B_2 \equiv F_2$, определенные, как в лемме 6, принадлежат $\mathcal{B}$.

Для любой формулы $F$ определим $F^+$ так: с помощью эквивалентностей из лемм 4 и 5 и обычных тождеств для отрицания в классической логике "вносим" связку отрицания вглубь к атомным формулам. Теперь если атомная формула имеет вид $\neg NULL$, $\neg NILL$, $\neg K_i$, $\neg NUM$, то заменяем ее соответственно на $NNU$, $NON$, $P_i$, $NPOS$, пользуясь леммой 3; если же атомная формула имеет вид $A$ или $\neg A$ и $A \equiv F \in \mathcal{A}$, то заменяем ее соответственно на $B_1$ или $B_2$, пользуясь леммой 6.

Из этих построений следует

ТЕОРЕМА 4. *В $SK(\mathcal{A})$ истинна формула $F(a)$ тогда и только тогда, когда $SK(\mathcal{A}) \models F^+(a)$ для любого $a \in S(\mathcal{A})$.*

В действительности из теоремы 4 следует, что мы всегда можем выбрать контекстное множество $\mathcal{A}$ так, что уже внутри него будем обладать всеми возможностями для построения позитивных формул, эквивалентных на $SK(\mathcal{A})$ произвольной данной формуле. В таких случаях $\mathcal{A}$ будем называть замкнутым. Использование замкнутых контекстных множеств позволит нам в дальнейшем ограничиться рассмотрением только позитивных формул языка.

## §7. О предикате включения одного типа в другой

При моделировании на ЭВМ некоторой реальной обстановки, создании типов данных, соответствующих внешним объектам, мы сталкиваемся со следующей ситуацией. Элементы внешнего мира каким-то образом взаимодействуют друг с другом, и это, естественно, должно быть отражено в машинной модели: на элементах различных типов и на самих типах вводятся некоторые операции и отношения. В каждом конкретном случае это, конечно, свои локальные операции и свои локальные отношения. Но есть моменты и глобального характера. Например, вхождение одного типа в другой. Всем известно, что натуральные числа входят в целые, действительные - в комплексные, лампы накаливания - в множество приборов для освещения. Можно выразить, все это, введя на типах отношение включения. Как это будет выглядеть в нашем случае?

Будем говорить, что тип $F$ включен относительно контекстного множества $\mathcal{A}$ в тип $G$ ($F \subseteq G$), если на $SK(\mathcal{A})$ верна импликация $\forall x(F(x) \supset G(x))$.

Естественно, возникает вопрос о разрешимости проблемы включения одного типа в другой. Прежде чем перейти непосредственно к

этой задаче, сформулируем вспомогательный результат, имеющий, с нашей точки зрения, самостоятельный интерес.

Назовем тип, определенный формулой $F(x)$, пустым, если не существует такого элемента $a \in S(\mathfrak{A})$, что $SK(\mathcal{A}) \models F(a)$.

ТЕОРЕМА 5. Проблема пустоты типа разрешима.

Наметим доказательство теоремы.

ЛЕММА 7. Следующие эквивалентности истинны на $SK(\mathcal{A})$:

1) $\overline{CONS}[F \,\&\, G, H] \equiv \overline{CONS}[F,H] \,\&\, \overline{CONS}[G,H]$;

2) $\overline{CONS}[F \vee G, H] \equiv \overline{CONS}[F,H] \vee \overline{CONS}[G,H]$;

3) $\overline{CONS}[F,G \,\&\, H] \equiv \overline{CONS}[F,G] \,\&\, \overline{CONS}[F,H]$;

4) $\overline{CONS}[F,G \vee H] \equiv \overline{CONS}[F,G] \vee \overline{CONS}[F,H]$;

5) $\neg(F \,\&\, G) \equiv F \,\&\, \neg G$; 6) $\neg(F \vee G) \equiv F \vee \neg G$.

Пользуясь леммой 7, мы можем ввести частичную дизъюнктивную нормальную форму (ЧДНФ) на формулах $\mathcal{L}_D$, представляя их в виде дизъюнкции конъюнкций, причем конъюнкты содержат только связки $\neg$ и $\overline{CONS}$ (мы рассматриваем только позитивные формулы). Операцию взятия ЧДНФ формулы $F$ обозначим через $[F]_D$, т.е. $[F]_D$ – формула в ЧДНФ, определяющая тот же тип, что и $F$.

Определим позитивное исчисление е (на этот раз исчисление высказываний). Оно будет иметь форму натурального вывода, хотя доказательство, так же как и в исчислении с из §5, будет проводиться от доказываемой формулы к аксиомам. Правила вывода исчисления определены только на формулах в ЧДНФ. Формулы будем рассматривать с точностью до коммутативности конъюнкций. Выражение $\ldots \& F \& \ldots$ обозначает конъюнкцию, в множество конъюнктов которой входит формула $F$.

Аксиомами будут атомные формулы $NUM$, $NULL$, $NIL$, $K_1, \ldots, K_m$.

Правила вывода следующие:

Iа) $\dfrac{F \vee G}{F}$; Iб) $\dfrac{F \vee G}{G}$;

2) $\dfrac{\neg F_1 \,\&\ldots\&\, \neg F_k}{NULL}$;

3) $\dfrac{\uparrow F_1 \,\&\ldots\&\, \uparrow F_e \,\&\, \mathrm{NUM} \,\&\, \uparrow F_{e+1} \,\&\ldots\&\, F_k}{F_1 \&\ldots\&\, F_e \,\&\, F_{e+1} \,\&\ldots\&\, F_k}$ ;

4) $\dfrac{\ldots \&H\& \uparrow F \,\&\, G \&\ldots\&\, \mathrm{NULL} \,\&\ldots}{\ldots \&H \,\&\, G \,\&\ldots\&\, \mathrm{NULL} \,\&\ldots}$ ;

5) $\dfrac{\ldots \,\&\, A \,\&\ldots}{[\ldots \&F \&\ldots]_D}$ , $A \equiv F \in \mathcal{A}$ ;

6) $\dfrac{\ldots \&\, H \,\&\, F \,\&\, G \,\&\ldots\&\, F \,\&\ldots}{\ldots \&\, H \,\&\, G \,\&\ldots\&\, F \,\&\ldots}$ ;

7) $\dfrac{\overline{\mathrm{CONS}}[F_1, G_1] \,\&\ldots\&\, \overline{\mathrm{CONS}}[F_k, G_k]}{[F_1 \&\ldots\&\, F_k]_D \qquad [G_1 \&\ldots\&\, G_k \,\&\, S]_D}$ ,

где S из леммы 3;

8) $\dfrac{\ldots \,\&\, P_i \,\&\ldots\&\, P_j \,\&\ldots}{\mathrm{false}}$ ,

$P_i$, $P_j$ – неравные формулы из NUM, NULL, NIL, $K_1, \ldots, K_m$;

9) $\dfrac{\ldots \,\&\, P_i \,\&\ldots\&\, \overline{\mathrm{CONS}}[F,G] \,\&\ldots}{\mathrm{false}}$ ,

$P_i \in \{\mathrm{NIL}, \mathrm{NUM}, \mathrm{NULL}, K_1, \ldots, K_m\}$;

10) $\dfrac{\ldots \,\&\, P_i \,\&\ldots\&\, \uparrow F \,\&\ldots}{\mathrm{false}}$ ;

11) $\dfrac{\ldots \,\&\, \overline{\mathrm{CONS}}[F,G] \,\&\ldots\&\, \uparrow F \,\&\ldots}{\mathrm{false}}$ ;

12) $\dfrac{\dfrac{\dfrac{F}{\vdots}}{F}}{\mathrm{false}}$

(false – логическая константа, интерпретируемая как ложь). Условия на применение: первым применяется правило I2, в последнюю очередь – правило 5. Назовем e-доказательством дерево e-вывода, все терминальные вершины которого есть аксиомы. Выводимость в e обозначим через $\vdash_e$.

ЛЕММА 8. Выводимость в $\vdash_e F$ тогда и только тогда, когда F определяет непустой тип.

ЛЕММА 9. Для любой формулы $F^+ \in \mathcal{L}_D$ число ее e-деревьев конечно.

Теорема 5 является следствием лемм 7,8 и 9: для проверки типа F на пустоту строим формулу $F^+$, эквивалентную F и находящуюся в ЧДНФ, а затем перебираем все e-деревья формулы $F^+$. Если $\vdash_e F^+$, то тип F непуст, в противном случае F определяет пустой тип.

ТЕОРЕМА 6. Проблема включения одного типа в другой разрешима.

Действительно, $F \subseteq G$ тогда и только тогда, когда тип $F \& \neg G$ пуст. Пусть H - позитивная формула, эквивалентная $F \& \neg G$. Если $\vdash_e H$, то $F \not\subseteq G$; если $\not\vdash_e H$, то $F \subseteq G$.

СЛЕДСТВИЕ. Проблема равенства типов разрешима.

В заключение, отметим, что результаты данной работы можно использовать в ряде интересных предложений, например для построения точных семантик языков программирования. Основным из возможных приложений, как было сказано во введении, автор считает создание алгоритмического языка, основанного на концепции Σ-программирования [5].

Естественно, язык $\mathcal{L}_D$, являясь по своей сути логическим, требует некоторого аппарата логической поддержки. Возможным решением этой проблемы является метод инвариантных преобразований [8]. Этим целям могут служить и представимые преобразования [9].

Отметим также, что существуют различные возможности расширения языка $\mathcal{L}_D$, в частности с помощью введения ограниченной квантификации по предикатам и др.

Автор искренне признателен А.А.Воронкову за многочисленные обсуждения работы, оказавшиеся очень плодотворными. Ряд ценных замечаний было высказано А.И. Кокориным, за что автор ему глубоко благодарен.

## Литература

1. ЕРШОВ Ю.Л., ПАЛЮТИН Е.А. Математическая логика. -М.: Наука, 1979.

2. ЕРШОВ Ю.Л. Проблемы разрешимости и конструктивные модели. -М.: Наука, 1980.

3. ЕРШОВ Ю.Л. Динамическая логика над допустимыми множествами //Докл. АН СССР. - 1983. - Т.273, № 5. - С.1045-1048.

4. КОКОРИН А.И., ПИНУС А.Г. Вопросы разрешимости расширенных теорий //Успехи мат.наук. - 1978. - Т.33, №2. - С.49-84.

5. ГОНЧАРОВ С.С., СВИРИДЕНКО Д.И. Σ-программирование //Логико-математические основы проблемы МОЗ. - Новосибирск, 1985. - Вып. 107: Вычислительные системы. - С. 3-29.

6. ВОРОНКОВ А.А. Способы выполнения программ в Σ-программировании //Тез.докл. 4 Всесоюзн. конф. по применению методов мат. логики, Таллин, 1986 г. - Таллин, 1986. - С.40-42.

7. АГАФОНОВ В.Н. Типы и абстракции данных в языках программирования //Данные в языках программирования. - М., 1982.-С.265-327.

8. МАРТЬЯНОВ В.И. О методах задания и частичного построения теории на ЭВМ //Кибернетика. - 1982.-№6. - С. 102-110.

9. МАНЦИВОДА А.В. О представимых преобразованиях //Тез.докл. Всесоюзн. конф. по прикладной логике, Новосибирск, 1985. - Новосибирск, 1985. - С.137-138.

10. ГОНЧАРОВ С.С. Модели данных и языки их описания //Логико-математические основы проблемы МОЗ. - Новосибирск, 1985.-Вып.107: Вычислительные системы. - С. 52-70.

11. ВИРТ Н. Алгоритмы + структуры данных = программы. - М.: Мир, 1985.

12. ЧЁРЧ А. Введение в математическую логику. - М.: Иностранная литература. 1960.

Поступила в ред.-изд.отд.
3 марта 1987 года