ТЕОРИЯ АЛГОРИТМОВ И ЕЕ ПРИЛОЖЕНИЯ
(Вычислительные системы)
1989 год Выпуск 129

УДК 510.25:517.11:518.5:519.681

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ В ШИРОКОМ СМЫСЛЕ

С.С.Гончаров, Д.И.Свириденко

В в е д е н и е

В теории и практике программирования одной из центральных проблем является проблема качества программ. Среди многочисленных методов и подходов к решению этой проблемы особое место занимают идеи автоматизации построения программ и, в частности, идея автоматического синтеза программ по спецификациям, идея автоматической верификации программ и идея исполняемых спецификаций. Предлагаемые средства реализации этих идей носят в основном логико-математический характер, а успешное их применение позволяет говорить о возникновении в программировании нового перспективного направления, которое мы называем "логическое программирование (в широком смысле)". Анализу этого направления и посвящена настоящая статья. Структура ее следующая.

В §1 будут проанализированы основные нетрадиционные модели вычислений, имеющие логико-математическое происхождение, - функциональная, хорнова и Σ-модель. §2 посвящается обзору методов так называемого доказательного программирования - логико-математическим методам построения программ, в которых реализован принцип создания программ "правильных по построению". Сразу же отметим, что настоящий обзор не претендует на пол-

ноту*), поскольку преследовалась совершенно другая цель - продемонстрировать концептуальную общность рассматриваемых моделей и методов. Последнее обстоятельство естественно приводит к выводу о возможности создания некой единой логико-математиче - ской концепции, в рамках которой нашло бы свое место данное разнообразие моделей и методов или, другими словами, создание оснований логического программирования в широком смысле. В [106] предпринята попытка найти подобные основания.

§1. Не фон неймановские модели вычислений

Модель вычислений - это набор формальных средств для описания вычислений (т.е. язык с точным (формальным) синтаксисом) вместе с точными указаниями, как исполнять такие описания, т.е. описаниями того, что называется операционной семантикой языка модели. Такая семантика может задаваться в терминах элементарных действий (актов вычислений), выполняемых некоторой маши - ной, описание устройства которой также включается в модель вычислений.

Важным требованием, предъявляемым к модели вычислений, является простота и обозримость ее описания, достаточная для того, чтобы рассматривать модель и порождаемые ею вычисления как математические объекты. Примерами моделей вычислений могут служить машины Тьюринга и Поста, нормальные алгоритмы Маркова, язык рекурсивных функций, λ-определимые функции, системы равенств Эрбрана-Гёделя, алгоритмы Колмогорова-Успенского и др.

---

*) Остались нерассмотренными такие подходы, как реляционные и дедуктивные базы данных, абстрактные типы данных и, в частности, алгебраический подход, индуктивный синтез программ, продукционное программирование, логические системы, используемые для анализа свойств программ и в искусственном интеллекте (динамическая, временные логики, логики знаний, логики процессов и т.п.).

Многие из этих моделей уже нашли свое применение в программировании.

Дескриптивность модели вычислений означает, что ее язык допускает другую, отличную от операционной семантику, определяющую не как, а что надо вычислять. Это и позволяет смотреть на язык такой модели как на язык спецификации задач. Дескриптивных семантик у языка модели может быть несколько (так же, как и операционных). Важно, чтобы они были согласованы с выделенной операционной семантикой, т.е. чтобы дескриптивная и операционная семантики определяли бы, по возможности, один и тот же класс объектов.

Характерной чертой обсуждаемых ниже функциональной и хорновой моделей является их теоретико-доказательное происхождение; это касается и языка, и его операционной семантики, которая представляет собой фактически процедуру построения доказательств в формальном исчислении, что позволяет задачу для таких моделей мыслить как пару (условие, цель), где условие - это набор аксиом, а цель - теорема.

Функциональные модели вычислений - это модели, основанные на языке равенств и на операционном понимании равенств как правил редукций или правил переписывания термов. При такой интерпретации равенств, т.е. выражений вида $l = r$, где l и r - термы, программа предстает как пара $\langle E, t \rangle$, где E - конечный набор равенств, t - исходный терм, а вычислительный процесс, порождаемый этой программой, состоит из повторяющихся переписываний исходного терма до тех пор, пока это возможно. В последнем случае говорят о нормализуемости терма, а его нормальную форму (т.е. итоговый терм, к которому не применима никакая аксиома из E) рассматривают как итоговый результат вычислений.

В равенства могут вкладываться самые различные математические смыслы. На равенства,например,можно смотреть как на утверждения, описывающие свойства объектов,или как на уравнения. Заметим, что в последнем случае неизвестными могут считаться функции и тогда равенства выступают в роли определений функций. Подобное многообразие смыслов порождает многообразие дескриптивных семантик языка равенств.Мы можем смотреть на равенство как на формулу некоторого логического (эквационального) исчисления,и в связи с этим нас может интересовать,скажем,либо факт выводимости формулы из заданного набора аксиом,либо факт ее истинности в соответствующей математической структуре (теоретико-модельная семантика).Взгляд на равенства как на определения функций позволяет определить другую дескриптивную семантику - семантику наименьшей неподвижной точки или, как ее еще называют, - денотационную семантику [6,40,43,47,63,70,168,169].

Функциональные модели вычислений по своему логическому статусу можно разделить на два класса - модели первого порядка и модели высших порядков. Среди функциональных моделей первого порядка наиболее известной является бестиповое λ-исчисление [6]*), используемое в программировании, главным образом, в двух целях: для описания точных семантик (операционной и денотационной) языков программирования высокого уровня и для построения различных моделей вычислений и языков программирования (например, LISP [42], SHEME [160], GEDANKEN [155], ML [120, 147], HOPE [88] и др. [85, 173]).

Бестиповое λ-исчисление является одним из первых логико-математических формализмов, где функция трактуется как вычислительное предписание,которое,будучи примененное к аргумен-

*) Отметим, что схожая трактовка бестипового λ-исчисления содержится в [149]: предлагается на операции λ-абстракции и аппликации смотреть как на бинарные операции.

ту, выдает вполне определенный результат. В λ-исчислении и близкой ему комбинаторной логике все вычислительные предписания образуют единую неструктурированную совокупность, где основной операцией является операция применения (аппликации) функции к аргументу (имеется еще операция λ-абстракции). В силу такой неструктурированности дескриптивные семантические аспекты для λ-исчисления являются хотя и не самыми главными*), но чрезвычайно трудными для их точной формулировки из-за возникающего эффекта самоприменимости (допускаются выражения $x(x)$). Только в конце 60-х - начале 70-х годов Ю.Л.Ершову [30] и независимо Д.Скотту [166] удалось описать первые естественные модели для бестипового λ-исчисления, что и послужило одной из причин резкой активизации в области функционального программирования и, в частности, привело к созданию математической теории вычислений [6,30,43,168,169].

У λ-исчисления есть много общего с языками программирования высокого уровня. Оно допускает непротиворечивые обогащения, при которых оказывается возможным моделировать абстрактные типы данных, параллельные и недетерминированные вычисления и т.п. В этом смысле на λ-исчисление следует смотреть как на средство, с помощью которого можно точно формулировать и строго решать многие задачи программирования.

Помимо бестипового λ-исчисления в теории и практике программирования используются и типовые варианты, на которые, кстати, целесообразно смотреть как на модификации бестипового варианта, удовлетворяющие некоторым типовым ограничениям. Счи-

*) Так ли уж важно, например, знать, какова область определения тождественной функции $\lambda x.x \leftrightarrows I$, определяемой соотношением $(\lambda x.x)x = x$, т.е. $Ix = x$; более того, если себя не связывать заранее требованием существования теоретико-множественной семантики, то нет и никакой несбходимости разделять области и операции над ними - в этом и заключается бестиповость.

тается, что подобные ограничения возникают из-за желания иметь удобную и ясную теоретико-множественную семантику, поскольку от типов требуется, чтобы они образовывали иерархию (если равенство типов понимается как синтаксическое совпадение, то наличие функциональных типов и позволяет образовывать иерархию; в этом случае мы имеем дело с эквациональными логиками высоких порядков). В действительности же иерархичность не спасает от необходимости решать весьма сложные семантические проблемы, например, проблему семантики рекурсивных определений. Представляется, что основная роль типов заключается в задаваемой ими определенной дисциплине работы с вычислительными предписаниями и их аргументами. При этом на равенство типов можно смотреть как на отношение эквивалентности, согласованное с операциями конструирования типов из базисных. При таком взгляде бестиповое λ-исчисление оказывается частным случаем типового, у которого имеется ровно один тип τ, удовлетворяющий равенству $\tau = (\tau \to \tau)$. Равенство типов $\tau = \sigma$ можно рассматривать как требование считать объекты типа τ одновременно объектами типа σ, и наоборот. Вообще говоря, язык типов может быть весьма сложным. Например, это могут быть формулы некоторого логического языка (подход "формулы - как - типы" [81, 129]) или типовые λ-термы [156]. На типах может быть определено отношение "быть подтипом" $\subseteq$, что приводит к очень интересной проблеме полиморфности типов [139,146,156]. Наконец, равенство типов, понимаемое как эквивалентность, хотя и усложняет соответствующие модели вычислений, позволяет расширить их многообразие, включая в том числе и различные варианты конструктивных λ-исчислений. Последнее обстоятельство позволяет установить точную связь λ-теорий с декартово замкнутыми категориями [61,136,167].

С λ-исчислением тесно связана другая функциональная модель вычислений (кстати, исторически более ранняя) - комби -

наторная логика [6,98,161]. Комбинатор - это вычислительное предписание, позволяющее строить (комбинировать) другие предписания. Примерами комбинаторов служат уже знакомый нам тождественный комбинатор I, комбинаторы K и S, где $(K \cdot x) \cdot y = x$ и $(((S \cdot x) \cdot y) \cdot z) = (x \cdot z) \cdot (y \cdot z)$. Имеет место свойство комбинаторной полноты: любой комбинатор выразим через K и S. Например, $I = (S \cdot K) \cdot K$. Не всегда ясен смысл результата применения комбинатора к предписанию, и в этом язык комбинаторов проигрывает λ-языку в естественности. С другой стороны, теория комбинаторов технически более проста, чем λ-исчисление (хотя и эквивалентна последнему) в силу того, что в ней используется только одна операция аппликации и, следовательно, отпадает необходимость анализа области действия оператора λ-абстракции. Поэтому считается, что с точки зрения "машинной" реализации комбинаторная модель предпочтительнее. В силу высказанного возникает желание построить смешанную " λ-комбинаторную" модель вычислений, где входным языком был λ-язык, а внутренним, "машинным" - язык комбинаторов. Схема трансляции λ-термов в комбинаторные термы была известна давно. Однако она обладала существенным недостатком: длина результирующего выражения экспоненциально зависела от глубины вложенности оператора λ-абстракции. В конце 70-х годов в [175] была предложена схема трансляции с линейной оценкой, что и позволило реализовать идею смешанной модели [173,174].

Бэкусом [77] предложен вариант модели, у которой и внешним, и внутренним языком являются языки комбинаторов, но разных уровней. Вот как, например, в языке Бэкуса выглядит определение функции факториал:

$$! = \mathrm{eq} \circ [\mathrm{id}, 1] \rightarrow 1;\ x \circ [\mathrm{id}, ! \circ - \circ [\mathrm{id}, 1]] \ .$$

Естественно, что некоторые исследователи выражают сомнение в непосредственном использовании подобных языков в качестве практических языков программирования.

В последние годы появились исследования по категорным моделям вычислений [97], в основе которых лежит теория категорий [140]. В этой теории функция также интерпретируется как вычислительное предписание.Но в отличие от предыдущих моделей здесь основной операцией над функциями является операция композиции. Отметим подход, развиваемый в [56,57] и называемый композиционным программированием. Разрабатываемую в данном подходе модель вычислений также можно отнести к функциональным. Основная идея этого подхода - свести средства конструирования вычислений к алгебраическим операциям над функциями.

Среди отечественных функциональных языков программирования несомненный интерес представляет язык РЕФАЛ [5,60,172], в основе которого лежит модель вычислений, предложенная А.А.Марковым. Приведем описание на этом языке функции "!":

```
§k 'F'0 ~ '1'
§k 'F'S1 ~k'MUL'(k'F'k'SUB'(S1)' 1 11) S1 1.
```

Здесь MUL и SUB - базисные функции умножения и вычитания, ~ есть другое обозначение (ориентированного) равенства. Операционная семантика в основе своей следует семантике исполнения алгоритмов Маркова, используя при этом стратегию самой левой внутренней подстановки и специальный механизм односторонней унификации.

Если L- функциональный язык, то,выбрав некоторую строку true в качестве обозначения логического значения "истина", на функциональную программу p(x,y), вырабатывающую истинностное значение, можно смотреть как на реализацию некоторого предиката. Будем говорить, язык L допускает инверсную

семантику, если существует такое частичное отображение inverse: $L \times D \to D$, где D - предметная область языка L (множество входных и выходных строчек программ из L), что для всех $p \in L$ и $d \in D$ справедливы следующие условия:

(i) если существует $d' \in D$ такое, что $p(d,d') =$ = true, то inverse (p,d) определено и $p(d,$ inverse $(p,d)) =$ true;

(ii) если не существует d' такого, что $p(d,d') =$ = true, то inverse(p,d) не определено.

Если L допускает инверсную семантику и существует $p \in L$, реализующая ее, то на L уже можно смотреть и как на язык, близкий по своим свойствам к языку предикатного программирования, т.е. к языку, где основным объектом программирования являются не функции, а отношения. Такими языками являются языки хорново- и Σ-программирования, о которых будет идти речь ниже.

Известна реализация инверсной семантики для ограниченного подмножества языка РЕФАЛ [172]. В [2,59] описаны различные методы автоматической реализации инверсной семантики для функциональных языков, что позволяет не только смотреть на эти языки как на языки описания предикатов,но и расширять их новыми конструкциями. О других формах взаимоотношения функциональных и хорновых языков будет идти речь ниже. А сейчас перейдем к рассмотрению собственно хорновых моделей вычислений.

Концептуальную основу хорновых моделей вычислений составляет тезис, согласно которому доказательство некоторых формул исчисления предикатов первого порядка можно рассматривать как вычисление. Предположим, что нас интересует следующая проблема: для данных формул P и Q, где P является формульным представлением множества фактов и правил, выполняющихся в некоторой области, а Q содержит свободные переменные $z_1,\ldots,z_n$ и интерпретируется как запрос, нужно узнать, верна ли формула

$P \supset \exists z_1 \ldots z_n . Q$, и найти набор термов $t_1 \ldots t_n$, на которых выполняется формула $P \supset Q[z_1 \leftarrow t_1, \ldots, z_n \leftarrow t_n]$. Легко показать, что в такой общей постановке данная проблема не имеет решения: существуют P и Q такие, что $\models P \supset \exists z_1, \ldots, z_n . Q$, но искомых термов нет. Поэтому естественно так ограничить класс формул, чтобы, во-первых, избежать подобного эффекта, а во-вторых, чтобы существовал эффективный метод построения подтверждающих термсв. Этим условиям удовлетворяют так называемые хорновы формулы, т.е. формулы, конъюнктивные нормальные формы которых представляют собой конъюнкции хорновых дизъюнктов универсальных формул вида $\forall x_1 \ldots x_n . \Phi$, где Φ - свободная от кванторов дизъюнкция элементарных формул (литералов), содержащих самое большое одну позитивную атомную формулу. Таким образом, хорнов дизъюнкт может принимать одну из следующих форм:

$$A(t_1, \ldots, t_n); \tag{1}$$

$$\neg A_1(\bar{t}^1) \vee \ldots \vee \neg A_n(\bar{t}^n) \vee A(\bar{t}), \quad n \geq 1; \tag{2}$$

$$\neg A_1(t^1) \vee \ldots \vee \neg A_n(t^n), \quad n \geq 0, \tag{3}$$

где A_i, A - атомные формулы.

Формулы вида (1) называются также фактами, формулы (2) - правилами, формулы вида (1) или (2) - определяющими дизъюнктами, или квазитождествами, и, наконец, формулы (3) - целями, или негативными дизъюнктами. Для хорновых дизъюнктов имеет место следующий фундаментальный факт: если P - конъюнкция универсальных замыканий хорновых дизъюнктов, сигнатура которых содержит хотя бы одну константу, то

(i) для каждых ϕ_i $(i=1,2)$ вида $\exists \bar{y} \cdot \varphi$,где φ - конъюнкция атомных формул, из того, что $\models P \supset \phi_1 \vee \phi_2$, следует, что либо $\models P \supset \phi_1$, либо $\models P \supset \phi_2$;

(ii) для каждого ϕ вида $\exists \bar{y} \cdot \varphi$, где φ - конъюнкция атомных формул, из того, что $\models P \supset \exists \bar{y} \cdot \varphi$, следует, что существует набор замкнутых термов $\bar{t}$ такой, что $\models P \supset \varphi[\bar{y} \leftarrow \bar{t}]$.

Другими словами, на классе хорновых дизъюнктов теоретико-модельная семантика, по Тарскому, ведет себя как конструктивная (например, как семантика типа реализуемости [37]). Кроме того, и классическое исчисление предикатов, ограниченное этим классом формул, является конструктивным в том смысле, что

(i') $P \vdash \phi_1 \vee \phi_2 \Rightarrow P \vdash \phi_1$ или $P \vdash \phi_2$;

(ii') по доказательству $P \vdash \exists \bar{y} \cdot \varphi$ можно эффективно построить набор замкнутых термов $\bar{t}$ такой, что $P \vdash \varphi[\bar{y} \leftarrow \bar{t}]$. (Здесь $P, \phi_1, \phi_2, \varphi$ - формулы, как в (i) и (ii).)

Итак, имея в своем распоряжении некоторый метод автоматического доказательства теорем языка исчисления предикатов первого порядка (полный для хорновых дизъюнктов), мы можем смотреть на него как на основу операционной семантики языка хорновых дизъюнктов. В качестве же дескриптивной семантики этого языка можно рассматривать теоретико-модельную семантику. Такая модель вычислений и носит название хорновой.

Наиболее известной разновидностью хорновой модели является модель, у которой в качестве формульного (аксиоматического) представления проблемной области выступают конечные наборы определяющих дизъюнктов языка (не содержащего предикат равенства, но содержащего хотя бы одну константу) в роли запросов - формулы вида $\exists \bar{z} \cdot \varphi$, где φ - конъюнкция атомных формул, а в роли операционной семантики - SLD-резолюция (процедура оп -

ровержения, приспособленная к классу хорновых дизъюнктов [9,38, 76,127,134,138]), снабженная средствами извлечения из SLD-до - казательств требуемых наборов замкнутых термов (посредством композиции наиболее общих унификаторов). Таким образом, в отличие от исполнения функциональных программ, где вычисление реализуется "прямым" методом, без возвратов и пространства поиска, в данной модели вычисление протекает от цели к подцелям, с возвратами, тупиками и поиском. Однако и на SLD-процедуру можно смотреть как на недетерминированный процесс пере - писывания (но уже не термов, а формул), использующий для этого не сравнение по образцу, а унификацию. Следуя [9], мы эту модель будем называть классической. Эта модель обладает естественной процедурной семантикой, согласно которой формула $\neg A_1 \vee \ldots \vee \neg A_n \vee A$, записываемая так же, как $A \leftarrow A_1, \ldots, A_n$ или $A :- A_1, \ldots, A_n$, понимается как процедура сведения задачи A к подзадачам $A_1, \ldots, A_n$, а формула $A(\bar{t})$ используется для непосредственного решения задачи. Эта модель обладает дескриптивной семантикой, аналогичной семантике наименьшей неподвижной точки. В [177] показано, что семантики классической хорновой программы корректны и полны относительно друг друга (в качестве стандартной модели при теоретико-модельной семантике рассматривается свободная система $\mathcal{U}(P, \emptyset)$ - инициальная модель в квазимногообразии P).

Появление классической модели обязано языку PROLOG [10, 38], который быстро завоевал популярность во всем мире и сейчас выступает как один из основных языков искусственного ин - теллекта [74,84,134]. В японском проекте ЭВМ первого поколе - ния PROLOG выбран прототипом внутренних языков (Kernel language) KL-0, KL-1 и KL-2 логических машин и их "макроассембле - ров" MANDALA и ESP [90]. Нужно отметить, что между языком

PROLOG и классической моделью существует большое различие из-за наличия в первом программистских конструкций, не имеющих логического объяснения. Причиной встраивания таких конструкций является желание увеличить выразительную мощь языка квазитождеств. Поэтому большое число исследований по хорновому программированию посвящено либо попыткам придать этим конструкциям логический статус, либо увеличению выразительной силы классической модели за счет расширения класса выполнимых формул-спецификаций. Рассмотрим некоторые из этих исследований.

Прежде всего возникает желание снять ограничение на вид программ и запросов, существующих в классической модели. Этому посвящена работа [117], где описывается так называемая обобщенно-логическая модель вычислений и базирующийся на этой модели язык HORNLOG. В этой модели программа P может содержать любые хорновы дизъюнкты, а запрос имеет вид $Q = \exists z_1,\ldots,z_n \cdot Q'$, , где $Q' = \neg H_1 \vee \ldots \vee \neg H_m$, где H_i - хорновы дизъюнкты, чьи множества свободных переменных попарно не пересекаются. Заметим, что $\models P \supset \exists \bar{z}.(\neg H_1 \vee \ldots \vee \neg H_m)$ тогда и только тогда, когда формула $P \wedge \forall \bar{z}.(H_1 \wedge \ldots \wedge H_m)$ опровержима; последняя же формула эквивалентна конъюнкции хорновых дизъюнктов. Процедура опровержения, описанная в [117], отлична от метода резолюций и использует графическое представление формул в виде H-графов [115]. Авторы подхода отмечают следующие его достоинства: 1) существенно расширяется класс логических программ и запросов к ним; 2) представление программы и запроса к ней в виде H-графа, состоящего из атомных формул в качестве вершин, позволяет дать естественную параллельную интерпретацию операционной семантике; 3) в языке выразимы некоторые формы отрицания (без необходимости использовать конструкцию not из PROLOG'а и его семантику); 4) ответы имеют вид наборов n-ок термов (не обязательно замкнутых) $\langle t_1^1,\ldots,t_n^1\rangle,\ldots,\langle t_1^l,\ldots,t_n^l\rangle$ таких, что

$$\vdash P \supset (Q'[z_1 \leftarrow t_1^1, \ldots, z_n \leftarrow t_n^1] \vee \ldots$$
$$\ldots \vee Q'[z_1 \leftarrow t_1^1, \ldots, z_n \leftarrow t_n^1]),$$

что выглядит весьма естественно для некоторых приложений (например, в задачах диагностики). Для данной модели остается проблемой построение ее денотационной семантики.

Выше мы упомянули связку not, позволяющую в PROLOG'е в определяющие дизъюнкты вида $A \leftarrow A_1, \ldots, A_n$ программы вводить "отрицания" в посылки импликаций. Операционное толкование этой связки таково - not(A) выполнено, если неудачны попытки установить выполнимость A (отрицание неудачей). Появившаяся статья Фиттинга [109] позволила придать not точный логический статус, однако описание ее семантики требует уже трех истинностных значений: истина, ложь, неопределенно.

Р.Ковальский [134,135] предлагает расширить язык квази-тождеств, разрешив использовать в посылках импликаций другие импликации, т.е. формулы вида $A \leftarrow A_1, \ldots, [B \leftarrow B_1, \ldots, B_k], \ldots, A_n$. Такие спецификации достаточно часто встречаются в математике: $X \subseteq Y \leftarrow [u \in Y \leftarrow u \in X]$. Однако подобное расширение при стандартной операционной семантике зачастую приводит к очень неэффективным вычислениям [135]. В отдельных случаях может выручить использование not, но возникает проблема корректности программ. Отметим, что подобное расширение имеет непосредственное отношение к проблеме модуляризации хорновых программ. Вообще говоря, модули давно используются в логическом программировании (MPROLOG, MICROPROLOG, см. приложения в [38]), но только совсем недавно появились теоретические работы, посвященные обоснованию введения понятия модуль в логическое программирование [110,144].

К проблеме модуляризации имеет самое непосредственное отношение параметризация хорновых программ. Использование предикатных параметров, с одной стороны, позволяет повысить универсальность модулей, а с другой стороны - вводить в язык конструк-

ции высоких порядков. Идея введения параметров используется, например, в MICROPROLOG'е, но наиболее последовательное и полное решение эта проблема получила в [9,11], где указываются операционная, теоретико-модельная и денотационная семантики параметрических хорновых программ и показывается их согласованность. Заметим, что в отличие от [11], где обсуждается проблема параметризации классической модели, в [9] описывается еще одна версия хорновой модели - так называемая однородная модель (там же приводится описание еще одной модели - окрестностной). Рассмотрим несколько подробнее однородную модель, на которую можно смотреть как на "бестиповый" вариант хорновой модели вычислений. В языке этой модели отсутствует разница между термами (объектами) и атомными формулами (отношениями), поскольку используется единая сигнатура как для построения объектов, так и для обозначения отношений на этих объектах. Если ω - сигнатурный символ, $t_1, \ldots, t_n$ - объекты, то терм $\omega(t_1, \ldots, t_n)$, с одной стороны, обозначает, что объекты $t_1, \ldots, t_n$ связаны отношением ω, а с другой - он может вновь рассматриваться как новый объект. Таким образом, однородная модель - это пример объектно-ориентированной модели вычислений, где все есть объекты и вся ответственность за структуризацию "мира" ложится на программиста. В таком подходе есть свои недостатки (главным образом, методологические), но и есть достоинства. В частности, эта модель частично снимает претензии, обычно предъявляемые к классической модели (при сравнении ее с функциональными моделями) в связи с тем, что в ней нельзя определять предикаты высоких типов. В то же время однородную модель трудно считать чисто хорновой; скорее она занимает промежуточное положение между функциональными и логическими моделями вычислений. Похожую характеристику можно дать и модели вычислений высокого порядка, предлагаемой в [124,145]. Здесь в роли объек -

тов (структур данных) выступают типовые λ-термы, в которых могут встречаться (как свободные переменные) функциональные переменные. Основу этой модели составляет хорновоподобная логика высокого порядка, для которой оказываются верными многие свойства, характерные для логик первого порядка: теорема Эрбрана, выполнимость свойств (i') и (ii') для конструктивных логик, унификация термов и т.п. Однако, приобретя действительно высокую выразительную силу, эта модель и соответствующий ей язык λ-PROLOG проигрывают в главном - в эффективности.

Интерес к конструкциям высокого порядка в хорновом программировании во многом объясняется появляющейся возможностью управлять исполнением хорновой программы. Именно с этой целью в [134] вводится мета-предикат доказуемости ДЕМО. О различных вариантах использования этого мета-предиката для построения стратегий исполнения классических хорновых программ см. в [4].

Большое внимание сейчас уделяется построению моделей, сочетающих в себе достоинства функциональных и хорновых моделей: из функциональных моделей можно взять достаточно развитые средства управления вычислениями и такие конструкции,как функционалы, ленивые и потоковые вычисления, процессы, типы данных,полиморфизм типов, богатое программное и операционное окружение и, в частности, хорошо развитую технику интерпретации и компиляции, а из хорновых моделей позаимствовать преимущества работы с логическими переменными, позволяющими определять обратимые процедуры и частично определенные структуры данных. Наличие в рамках одной модели и функций, и отношений существенно увеличивает выразительную мощь и удобство языка.

Большинство подходов к интеграции моделей базируется на одном из следующих технических решений.

1. Расширение хорновых моделей конструкциями, позволяющими моделировать функциональные определения.

В данном подходе есть два основных приема. В [170] приводится описание расширения классической модели, где допускаются импликации вида $B_1 \wedge \ldots \wedge B_n \leftarrow A_1 \wedge \ldots \wedge A_m$, где A_i, B_i - атомные формулы, и дополнительно к обычным переменным ($\forall$-квантифицированным) вводятся так называемые "выводимые" (в действительности, $\exists$-квантифицированные) переменные. Подобное расширение позволяет корректно транслировать в язык модели не только определения функций (что возможно и для классической модели), но и равенства, выражающие свойства функций. В качестве операционной семантики используется интерпретация импликаций как редукционных правил.

Другой путь моделирования функциональных определений - это расширение классической модели предикатом равенства "=". При этом можно работать с "=" непосредственно, разработав для этого новый метод резолюции, как это делается, скажем, в [96] (см. также [126]), либо можно предварительно равенства переписывать в хорновы дизъюнкты, а затем использовать обычную SLD-резолюцию [79] (см. [78 и обзор 82]).

2. Ограничение операционного поведения логических переменных в хорновых моделях (и соответствующих механизмов работы с ними) с целью придать хорновым моделям большую "функциональность".

Основная идея - это, рассматривая логические переменные в классической модели как неориентированные каналы обмена информацией между задачами (предикатами), сделать их ориентированными. Это приводит к возникновению понятия процесса и возможности параллельного исполнения процессов, что требует соответствующих механизмов синхронизации. Основной прием ориентации переменных и синхронизации - это аннотация переменных. Различают статический способ аннотации, принятый, например, в [92], и динамический [93,159,176]. Соответствующим образом модифицируется SLD-резолюция (см. обзор [82]).

3. Моделирование в функциональной модели механизма работы с логическими переменными.

Имеется несколько способов реализации этого решения. Первый способ [103] - это трансляция хорновых дизъюнктов в равенства и последующее преобразование равенств в конфлюентные редукционные системы с использованием техники пополнения (алгоритм Кнута-Бендикса; см. обзор [86]). Однако есть сомнения в адекватности метода пополнения целям логического программирования [82]. Интересное обобщение метода пополнения для языка квазитождеств вида $t_1 = t'_1 \wedge \ldots \wedge t_n = t'_n \supset t = t'$ и использование его для построения операционной семантики хорновой модели с единственным предикатом "=" (остальные предикаты представляются как булевозначные функции) содержится в [113,114].

Второй способ заключается в том, чтобы средствами языка функциональной модели промоделировать механизм работы с логическими переменными. Здесь возможны два варианта. Первый вариант сводится к тому, чтобы в рамках некоторого традиционного функционального языка программирования (чаще всего используется LISP и его диалекты) определить функции, имитирующие SLD-резолюцию. В этом случае функциональная модель выступает в роли метамодели по отношению к смоделированной хорновой модели. Подобная комбинация очень привлекательна с точки зрения возможностей расширения средств вывода и управления выводом. Но есть и недостатки, привносимые тем, что приходится работать с двумя языковыми компонентами; не ясна дескриптивная семантика такой интегральной модели, сложен интерфейс между языковыми компонентами. Примером подобного решения может служить LOGLISP [157].

Второй вариант заключается в разработке специальной функциональной модели, позволяющей некоторые определяемые функции естественно интерпретировать как отношения. Такой подход взят за основу в методе инверсной семантики (в частности, для

РЕФАЛа), о котором уже шла речь выше, в проекте SUPERLOGLISP [84, 99]. Основная идея проекта SUPERLOGLISP заключается в том, чтобы в язык встроить теоретико-множественную абстракцию, позволяющую определять множественнозначные функции:

$$\text{append}(x1,x2,x3)=\{(x1,x2,x3) \mid \exists\, x,y,z,w\ [(x1 = \text{nil} \wedge x2 = x \wedge x3 = x) \vee (x1 = \text{cons}(x,y) \wedge x2 = z \wedge x3 = \text{cons}(x,w) \wedge \text{append}(y,z,w)]\}.$$

Ясно, что для такой функциональной модели нужны соответствующие редукционные правила, позволяющие справляться с $\exists$-квантифицированными переменными и унификацией. Соответствующий метод ($\mathcal{E}$-редукция) предложен в [83].

4. Встраивание в функциональные модели механизмов унификации термов и поиска с возвратами.

Добавление к функциональной модели механизма работы с логическими переменными в действительности означает возможность вычисления значений переменных, встречающихся в термах. Такие переменные есть не что иное, как $\exists$-квантифицированные переменные, работать к которыми можно, если заменить в операционной семантике исходной функциональной модели поиск по образцу унификаций. Полученная семантика для эквациональных моделей носит название сужения (narrowing [112,154]). Заметим,что сужение может быть эффективно реализовано как собственное расширение метода редукций. На сужении базируется целый ряд проектов, в которых в роли языков хорново программирования используются эквациональные языки [102, 104, 119, 153, 162,164,178].

И в заключение параграфа рассмотрим коротко Σ-модель [18-23, 32-34, 62-69, 106, 122, 123]. В отличие от аксиоматической природы предыдущих моделей здесь используется элементарная определимость одной модели в другой. Чтобы данный способ спецификации задач мог стать основой некоторой модели вычисле-

ний, необходимо, во-первых, теоретико-модельную семантику, по Тарскому, превратить в алгоритм вычисления истинности формул на модели и, во-вторых, научиться автоматически извлекать те значения переменных, при которых формулы исчисления предикатов первого порядка становятся истинными. Для этого необходимо, прежде всего, потребовать конструктивности или, в более общем случае, позитивности тех моделей, в языке которых будут элементарно определяться другие модели. Но и в этом случае, очевидно, не все формулы допускают алгоритмическую проверку их истинности (одновременно с извлечением требуемых значений переменных). Поэтому, помимо требования конструктивности базовых моделей, нужно указать и тот класс формул, которые бы допускали эффективную проверку их истинности.

Пусть $(\mathfrak{A}, \nu)$, где $\mathfrak{A} = \langle A; P_1, \ldots, P_n, f_1, \ldots, f_k, c_1, \ldots \rangle$, - конструктивная модель, т.е. на P_i можно смотреть как на рекурсивные предикаты, а на f_j - как на рекурсивные функции. Для формулы φ языка логики предикатов первого порядка сигнатуры модели $\mathfrak{A}$ через $[\bar{x};\ \varphi](\mathfrak{A})$ обозначим предикат:

$$[\bar{x};\ \varphi](\mathfrak{A}) \rightleftharpoons \{\bar{a} \mid \bar{a} \in A^n,\ \mathfrak{A} \models \varphi(\bar{a})\},$$

где $\bar{x} = x_1, \ldots, x_n$ - набор различных переменных, содержащий все свободные переменные формулы φ. Оказывается, если φ - $\exists$-формула, то предикат $[\bar{x};\ \varphi](\mathfrak{A})$ является рекурсивно-перечислимым. Данное свойство сохранится, если вместо $\exists$-формул рассматривать так называемые Σ-формулы, в построении которых можно использовать ограниченные кванторы $\forall x \in y$, $\exists x \in y$, где y - либо конечное множество, либо конечный список. В Σ-модели по причинам, указанным в [23], используются в качестве ограничений наследственно конечные списки [20, 68]. Σ-формулы обладают еще одним хорошим свойством. Если $\varphi(P)$ - Σ-формула, в записи которой участвуют элементарные

формулы вида $P(\bar{t})$, где P - n-местная предикатная переменная, и эти элементарные формулы входят в φ позитивно, то наименьшая неподвижная точка монотонного преобразователя предикатов $\Gamma_\varphi: P \mapsto [\bar{x}; \varphi](\mathfrak{A}, P)$ является также рекурсивно-перечислимым предикатом. Такая замечательная особенность сохранения рекурсивной перечислимости Σ-формулами позволяет смотреть на них как на язык исполняемых спецификаций. Σ-программа - это пара (S, q), где S - конечный набор рекурсивных определений предикатов (и предикатов функций) вида

$$P \leftrightarrow \varphi(\bar{x}, \ldots; P, \ldots),$$

а q - "запрос" к S, имеющий вид $[!\bar{x}.?\bar{y}.\psi(\bar{x}, \bar{y})]$, интерпретируемый как указание найти те значения переменных из $\bar{x}$, которые для заданных значений переменных из $\bar{y}$ делают формулу φ истинной в модели $\mathfrak{A}$. Здесь φ и ψ - Σ-формулы $\{\bar{x}\} \cap \{\bar{y}\} = \emptyset$ и $\{\bar{x}\} \cup \{\bar{y}\}$ - свободные переменные формулы φ.

Σ-программы допускают естественную параметризацию (параметрами могут служить не только предметные, но и предикатные переменные) и, следовательно, модуляризацию. Между Σ-программами и хорновыми программами существует тесная связь: каждая хорнова программа тривиальным образом записывается в виде Σ-программы [34,123]. Обратное также верно при условии, что базовая модель изоморфна инициальной модели для некоторой конечно аксиоматизируемой хорновой теории [106].

Язык Σ-программ допускает различные обогащения. В [69] описано расширение языка, допускающее использование определяемых функций как термообразующих, что позволяет в Σ-программировании определить функциональный стиль программирования. Кроме того, у языка Σ-программ существует процедурное расширение, для которого возможно построение семантики в терминах про-

цессов*). Наконец, Ю.Л.Ершов в [34] построил язык Σ-выражений, который является языком описания вычислимых предикатов конечных типов. Денотационная семантика этого языка описана в [33,62]. К сожалению, до сих пор не известна операционная семантика языка Σ-выражений.

§2. Доказательное программирование

В данном параграфе мы рассмотрим логико-математические подходы к реализации перехода от спецификации задач к программам (для различных моделей вычислений), гарантирующих корректность такого перехода в том смысле, как понимается правильность доказательства в математике. Отсюда и название - доказательное программирование [23,27,29,50,51,64,65,165].

Следует отметить, что необходимость создания более убедительных (по сравнению с традиционной отладкой) способов гарантии правильности программ была осознана еще в 60-е годы, когда появились первые работы, приведшие к созданию верификационного подхода. Основная идея верификации программ заключается в том, чтобы вместо отладки и тестирования предъявлять математическое доказательство правильности соответствия текста программы ее исходной спецификации. Первые работы в этом направлении [111, 126] предлагали связывать с началом и концом работы программы P логические утверждения φ и ψ и доказывать, что если для входных данных выполняется условие φ и P на этих данных заканчивает работу, то выходные данные будут удовлетворять условию ψ (в данном случае говорили о частичной корректности P по отношению к φ и ψ - в формальной записи $\varphi\{P\}\psi$). Если же удавалось доказать, что P не только частично корректна, но и заканчивает свою работу при всех данных, удовлетворя-

*) Описание процедурного расширения и его семантика будут опубликованы в одном из последующих сборников серии "Вычислительные системы".

ющих φ, то P была названа тотально корректной по отноше - нию к φ и $\psi(\varphi[P]\psi)$. Основной трудностью при верификации является выявление закономерностей,относящихся к значениям переменных, входящих в циклические и рекурсивные конструкции (инварианты циклов и рекурсии). Если же такие закономерности выявлены, то тот факт, что они сохраняются при выполнении соответствующих им операторов, можно уже доказывать методом,аналогичным методу математической индукции.В этом и состояла первоначальная идея Флойда и Наура. И сами формулировки индуктивных утверждений, и процесс доказательства в их работах осуществлялись неформальным образом. Чтобы строить "механические" системы верификации программ, необходимы были формальные дедуктивные системы в языке, средствами которых можно было бы как записывать индуктивные утверждения, так и строить доказательства. Первую такую систему указал Хоар [126]. Дальнейшие ис - следования в области верификационного программирования осуще - ствлялось в следующих трех направлениях (см. обзоры [3,53,75] и работы [13,54]):

а) в поиске новых методов доказательства частичной и полной корректности программ;

б) в поиске ограничений на структуру программ и создание новых языков программирования, для которых задача верификации решалась бы достаточно просто;

в) в автоматизации процесса верификации программ и создании практических верификационных систем.

Что же касается верификации логических (хорновых), программ, то здесь можно выделить два подхода (см. [127]). Первый подход по своим установкам очень напоминает подход к верификации традиционных программ, описанный выше. Другой подход ("метод вывода процедур"), имеющий также непосредственное отноше - ние к трансформационному синтезу логических программ (см.ниже),

предложили независимо Кларк и Хоггер (см. [127]). Основная идея данного подхода заключается в следующем.

Пусть P - логическая программа и S - набор формул, специфицирующих предикаты из P. Считается, что каждое предложение из S имеет вид: $R \leftrightarrow D$, где R - определяемый предикат, а D - определение. Если теперь известно, что $S \models D \leftrightarrow D_1$, то, очевидно, имеет место $S \models R \leftrightarrow D_1$. Подобная замена одного определения в S другим называется трансформацией определений. Используя подобные трансформации, мы в конце концов можем получить спецификационное предложение $R \leftrightarrow D_n$, где D_n имеет вид дизъюнкции $B_1, \ldots, B_n$, в которой каждый дизъюнктивный член может быть использован при аксиоматическом описании в логической программе:

$$R \leftarrow B_1$$
$$\ldots$$
$$R \leftarrow B_n.$$

Будем говорить, что данные процедуры выведены из S. Если при этом оказалось, что полученная логическая программа совпадает с исходной P, то тем самым мы доказали корректность P относительно S *). Заметим, что здесь гарантируется не только частичная корректность программы P, но и ее тотальная корректность, имея в виду, что для каждого определяемого предиката R справедливо

УТВЕРЖДЕНИЕ. *Для всех подстановок* θ

$$[P = R(\bar{t}\theta) \leftrightarrow S = R(\bar{t}\theta)] .$$

Независимость логических аспектов от аспектов управления приводит к необходимости анализа не только логических программ, но и так называемых логических алгоритмов: вместе с логической программой P и целью G рассматривается и страте -

*) Нетрудно видеть, что очевидная модификация этого подхода может быть развита и для Σ-программ.

гия управления ее исполнением C. Для анализа логических алгоритмов, помимо таких свойств, как частичная и тотальная корректность, полнота, непустота, имеет большое значение нётеровость: исполнение логического алгоритма (P,G,C) нётерово, если при этом порождается конечное число вычислений и все они имеют конечную длину. Заметим, что концепция нётеровости для логических алгоритмов является более сложной в сравнении с аналогичным понятием для традиционных и функциональных программ.

В заключение этого параграфа отметим, что идея верификации логических программ [100,108,158] естественно может быть оформлена как своеобразная точка зрения на процесс обучения логическому программированию [101].

Работы по верификационному программированию послужили толчком для исследований в области аксиоматической семантики языков программирования, которые, в свою очередь, привели к созданию Дейкстрой нового метода доказательного программирования, который мы назовем методом Дейкстры [1,24,27].

В данном подходе каждый оператор Op в фон неймановских программах интерпретируется не только как преобразователь данных, но и как преобразователь условий (предикатов), причем, образно говоря, преобразование предикатов обратно по своему направлению к преобразованию данных. Это наблюдение позволило поставить задачу поиска для оператора Op и выходного условия ψ такого предусловия $wp(Op,\psi)$, чтобы выполнялось утверждение $wp(Op,\psi)\{Op\}\psi$, и если $\varphi\{Op\}\psi$, то была истинна импликация $(\varphi \supset wp(Op,\psi))$ (в силу этих условий предикат $wp(Op,\psi)$ называется слабейшим предусловием для Op и ψ). Понятие слабейшего предусловия позволило Дейкстре создать метод программирования, где правильность программы доказывается одновременно с ее конструированием. Основу этого метода составляет процедура последовательного преобразования целевого предиката ψ к последовательности слабейших предус-

ловий вместе с соответствующими этим условиям программными конструкциями и доказательством того, что эти предусловия логически следуют из заданного спецификационного предусловия Φ на входные данные.

Для реализации вышеописанной идеи Дейкстре потребовалось не просто механически соединить логические утверждения с программными конструкциями и построить соответствующую общему замыслу дедуктивную систему, но и серьезно модифицировать само понятие программы. Заметим, что выбор подходящей логической системы для метода также не был случаен: ею оказалась интуиционистская логика [24]. Таким образом, опыт создания метода Дейкстры позволяет сформулировать вывод: в стиле программирования метод (логика) построения программ и логика доказательства их правильности должны быть концептуально согласованными с моделью вычислений.

В методе Дейкстры возникает последовательность программистских конструкций, каждая из которых является "уточнением" предыдущей. А нельзя ли мыслить себе программу как результат серии преобразований исходной спецификации? Если при этом для каждого преобразования (трансформации) гарантировалась бы его правильность (в том смысле, что результат применения преобразования был бы корректен по отношению к тому, к чему преобразование применялось), то тем самым гарантировалась бы правильность полученной таким образом программы в целом. Это и есть центральная идея трансформационного программирования [55,87, 105, 123]. Естественно, что успех перехода от спецификации к программе зависит от имеющегося в распоряжении программиста набора трансформаций и выбора правильного направления преобразований. В свою очередь, как нетрудно понять, сама система преобразований во многом определяется как синтаксисом выбранных языков спецификаций и программирования, так и их семантикой. Таким образом, можно говорить о трансформационном программирова-

нии для фон неймановской, функциональной или хорновской модели вычислений и понимать построение программ, скажем, как процесс преобразований грамматик входного и выходного текстов, или как переход от абстрактного типа данных к его реализации, или как процесс решения функциональных уравнений. Заметим, что и метод Дейкстры также можно считать за один из вариантов трансформационного подхода.

В СССР исследования в области трансформационного программирования (для традиционных программ) осуществляются в ВЦ СО АН СССР, где реализуются два проекта: Новосибирская Трансфор - мационная Машина [28,105] и СКАТ [35]. Среди зарубежных проектов особо стоит выделить Мюнхенский проект [7,148,150], в ко - тором предпринята попытка объединить функциональное,хорновское и фон неймановское программирования. В основу подобного объединения положена идея построения единой "трансформационной" семантики для различных моделей вычислений.

Трансформационное программирование для хорновской модели вычислений, обсуждаемое в [127], идейно следует работе Бар - стелла и Дарлингтона [87], где была предложена трансформационная техника построения функциональных программ.

Основная здесь идея - это в точности идея "метода выводимых процедур": используя трансформации спецификационных опре - делений,привести последние к такому виду, который позволил бы тривиальным образом переписать их в виде логических программ*). На практике используются различные уточнения и модификации этого метода. Примером может служить целеориентированный вывод процедур, где вывод процедуры для некоторого спецификационного предиката S начинается с цели $G_1: ?R(\bar{t})$, а заканчивается целью $G_n: ?R_1,\ldots,R_k$. Если при этом для каждого $i \geq 1$ имело место $S,G_i \models G_{i+1}$, то мы можем считать, что выведена им-

*) Эта идея применима и к Σ-программам.

пликация $R \leftarrow R_1, \ldots, R_k$. Заметим, что промежуточные цели G_i, $1 < i < n$, могут быть произвольными формулами языка исчисления предикатов. Для целеориентированного вывода процедур нетривиальной является проблема полноты набора выведенных процедур.

При автоматизации вывода логических программ из спецификаций могут использоваться различные системы автоматического поиска доказательств теоремы и, в частности, системы, основанные на методе резолюций [58]. Для целей модификации логических программ (связанной, например, с необходимостью модифицировать представление структур данных), возможно, предпочтительнее использовать нерезолюционную технику [127].

В заключение рассмотрим метод так называемого дедуктивного программирования (в терминах [28] - логический подход). Основу этого подхода составляет следующий тезис конструктивной математики [80]: в ходе доказательства теоремы, утверждающей существование объекта, такой объект обязательно должен быть построен (хотя бы в принципе). Если на спецификацию задачи смотреть как на формулировку теоремы, утверждающей существование решения массовой задачи, то конструирование программы при дедуктивном подходе заключается в поиске доказательства этой теоремы в подходящей конструктивной логике, поскольку тогда, согласно основному тезису, мы можем построить и искомую программу - решение. Процесс построения такой программы может быть осуществлен различными способами, и его конкретный вид существенно зависит от той логики, в которой осуществляется поиск доказательства. Можно, например, иметь дело с логикой, чьи дедуктивные правила в точности соответствуют правилам композиции программ, т.е. непосредственно генерировать и преобразовывать аннотированные программные тексты (как это делается в подходе Дейкстры), либо в начале строить доказательство и лишь затем извлекать из него программу [8, 14-16, 41, 44, 46, 48-51,

66, 67, 71-73, 78, 81, 94, 121, 125, 141,143], либо,рассматривая его как саму программу,"исполнять" его [118]. При "исполнении" доказательства или при извлечении из него программы можно столкнуться с неэффективностью, которая является следствием разных обстоятельств [44, 49-51]. С одной стороны,доказатель - ство может "кодировать" не только нужные алгоритмические сущ - ности, но и различные построения, имеющие вспомогательный характер.С другой - либо масштабы доказательства могут не позволить эффективно его исполнять, либо извлекаемая программа может оказаться слишком сложной, либо, наконец, сложным окажется сам алгоритм извлечения. Все это заставляет при ориентации на дедуктивное программирование обращать самое серьезное вни - мание на выбор соответствующей конструктивной логики как логики построения программ, что, в свою очередь, приводит к таким проблемам [48, 67]: а) какие логики считать конструктивными? б) как строить нужные конструктивные логики?

Для многих исследователей термин "конструктивная логика" обычно ассоциируется с понятием "интуиционистская логика". Последняя возникла как результат критики Брауэром [7] законов классической логики. Традиционный способ конструктивной интерпретируемости аксиом и правил вывода интуиционистского исчис - ления предикатов и интуиционистской арифметики, восходящий к Колмогорову [39] и называемый реализуемостью, был впервые точно сформулирован Клини [37]. Использовать эту интерпретацию в качестве теоретической основы алгоритмов извлечения программ из конструктивных доказательств впервые было предложено в [94].

В последние годы в доказательном программировании активно начинает применяться уже упоминавшаяся конструктивная семантика "формулы - как - типы", восходящая к Ховарду [129]. Согласно этой семантике, каждая формула ассоциируется с определенным типом данных: интуитивно с типом всех конструктивных доказа - тельств этой формулы. Так, например, формуле $(\varphi \wedge \psi)$ соот -

ветствует декартово произведение типов, ассоциированных с φ и ψ, поскольку с конструктивной точки зрения любое доказательство формулы $(\varphi \wedge \psi)$ - это есть пара (x,y), где x - доказательство формулы φ, y - доказательство формулы ψ.

Мартин-Лёф подобный взгляд на тип данных оформил в виде логического исчисления (интуиционистская теория типов) [142] и предложил в последующем использовать модификацию этого исчисления в программировании [143]. На базе этой модификации Констейбл и его группа [95] спроектировали язык программирования логического типа PL/CV3.

Основными синтаксическими конструкциями исчисления Мартин-Лёфа являются выражения вида: $t: \varphi$ ("t есть объект типа φ"); φ тип ("φ есть тип данных"); $s = t: \varphi$ ("s и t как объекты типа φ равны"); $\varphi = \psi$ ("φ и ψ равны как типы данных"). В исчислении имеется богатый набор типовых конструкторов, позволяющий, начиная с базовых типов, строить весьма сложные типы. В формализации Мартин-Лёфа либо разрешается иметь некий "универсальный" тип U ("тип всех малых типов"), либо построение иерархии типов управляется схемами аксиом, определяющими индуктивно порождаемые типы. Отметим, что хотя здесь нет привычных логических связок и кванторов, но логика присутствует в том смысле, что логические конструкции являются в исчислении Мартин-Лёфа определимыми; например, тип произведения $(\Pi x: \varphi)\psi(x)$ соответствует ограниченному квантору всеобщности $\forall x \in \varphi . \psi$, а тип суммы $(\Sigma x: \varphi)\psi(x)$ - ограниченному квантору существования $\exists x \in \varphi . \psi$.

На объекты типа ψ в семантике Ховарда или в формализме Мартина-Лёфа можно смотреть не только как на конструктивные доказательства, но и как на λ-термы и читать выражение $t: \varphi$ как "терм t имеет тип φ" или "терм t есть реализация типа (формулы) φ". Здесь обыгрывается хорошо известное в

математической логике обстоятельство, что между конструктивными доказательствами (скажем, доказательствами в интуиционистской логике) и λ-термами имеется тесная связь [161]. Это обстоятельство и позволяет смотреть на доказательство как на программу и исполнять его [118]. Роль шагов "вычислений", как и в функциональных моделях, играют так называемые нормализационные преобразования доказательств или редукции Правитца [151, 152]. Нормализованные выводы обладают следующими замечательными свойствами:

(i") нормализованное доказательство замкнутой формулы $(\varphi \vee \psi)$ имеет вид:

либо $\dfrac{\begin{matrix}\Pi\\ \varphi\end{matrix}}{(\varphi \vee \psi)}$, либо $\dfrac{\begin{matrix}\Pi'\\ \psi\end{matrix}}{(\varphi \vee \psi)}$;

(ii") нормализованное доказательство замкнутой формулы $\mathbf{x}.\varphi$ имеет вид:

$$\frac{\begin{matrix}\Pi\\ \varphi(t)\end{matrix}}{\exists\, \mathbf{x}.\varphi}\ .$$

Следовательно, если спецификация задачи имеет вид $\forall \bar{x}.\exists y.\varphi(\bar{x}, y)$ и в нашем распоряжении имеется доказатель - ство Π этой формулы, то на него действительно можно смотреть как на программу: положив $\bar{x} := \bar{t}$, мы вначале получаем доказательство-программу $\Pi(\bar{t})$ формулы $\exists y.\varphi(\bar{t}, y)$ и, нормализуя (исполняя) его, приходим к доказательству вида:

$$\frac{\begin{matrix}\Pi'\\ \varphi(\bar{t}, q)\end{matrix}}{\exists y.\varphi(\bar{t}, y)}\ .$$

Тогда терм q мы можем считать "вычисленным" значением "программы" Π при $\bar{x} := t$.

Корректность исполнения программ-доказательств следует из того, что система редукций Правитца обладает свойством Чёрча-Россера [152]. Это и неудивительно, если вспомнить наше замечание о существовании между доказательствами и λ-термами тесной связи. Данное обстоятельство является центральным моментом в подходе Гоада [118], поскольку доказательства (как программы) он кодирует с помощью подходящих λ-термов так, что нормализационные преобразования становятся обычнями редукциями λ-термов [6,161]. Правда, учитывая специфику построения доказательств в системе натурального вывода, он предлагает дополнительные преобразования выводов (а следовательно, и соответствующих им λ-термов), которые способствуют более эффективному их "исполнению" (преобразования типа "подрезки" или "прополки").

Помимо вышеупомянутых концепций в дедуктивном программировании используются и другие идеи математической логики. Так, в работах [48,49] активно используется семантика Шанина или, как ее еще называют, - метод конструктивной расшифровки, которая (как и семантика Клини) может быть отнесена к семантикам типа реализуемости.

К семантикам типа реализуемости относится и семантика Гёделя, где в качестве реализации формул интуиционистской арифметики рассматриваются примитивно-рекурсивные функционалы конечных типов [17]. Известна экспериментальная система GDLO дедуктивного программирования, базирующаяся на этой семантике [121].

В [125] приводится описание метода дедуктивного синтеза ЛИСП-программ, теоретическую основу которого составляет теория операций и классов, построенная с целью формализации представлений Бишопа о конструктивной семантике [107]. Несомненный ин-

терес для дедуктивного программирования (во всяком случае, для его теоретических аспектов) представляют также логические системы, построенные в [81].

Семантика, описанная в [52], представляет собой синтез идей семантики "формулы - как - типы" и семантики типа реализуемости (в терминах вычислимых функционалов конечных типов). В качестве денотационной области здесь рассматривается декартово замкнутая подкатегория нумерованных множеств [30]. Заметим, что данная семантика может быть достаточно просто переформулирована в терминах f_0-пространств Ершова, либо эквивалентным образом в терминах информационных систем [169]. Последняя возможность была использована в [16,67] для дедуктивного синтеза хорновых и Σ-программ. Интересный и, на наш взгляд, весьма перспективный подход к автоматическому построению хорновых программ (который условно можно отнести к дедуктивным методам) сформулирован в [110]*). Основная здесь идея - связать с каждым оператором композиции сложных предикатов из простых (декартово произведение, объединение, проекции и т.п.) некоторую стандартную схему построения хорновой программы (из программ для простых предикатов).

Из практических систем дедуктивного программирования прежде всего необходимо отметить систему ПРИЗ [36,72]. В данной системе структура извлекаемых (синтезируемых) программ изоморфна структуре конструктивного доказательства, из которого она извлекается. Следует отметить одно интересное обстоятельство - логическое обоснование системы ПРИЗ появилось вслед за ее созданием [12,46,71,72]**).

---

*) Представляет интерес развить этот подход к построению Σ-программ.

**) В настоящее время разрабатывается новый, объектно-ориентированный язык НУТ [171], синтезирующий в себе идеи таких языков как УТОПИСТ, ПРОЛОГ и СМОЛТОК.

З а к л ю ч е н и е

Заканчивая анализ логико-математических моделей вычисле - ний и методов доказательного программирования, хотелось бы указать на одно важное обстоятельство: хотя за каждым подходом, моделью и методом стоит вполне определенная теоретическая база, возникает такое ощущение, что есть нечто общее,что способно объединить все эти направления. И,естественно,возникает вопрос - нельзя ли разработать такую достаточно большую логико-математическую концепцию программирования, в рамках которой нашли бы свое естественное место и объяснение все вышеупомя - нутые методы и модели. Об одном из возможных подходов в решении этой проблемы и будет идти речь в следующей публикации.

Л и т е р а т у р а

1.АБРАМОВ С.А. Элементы анализа программ. - М.: Наука, 1986. - 128 с.

2. АБРАМОВ С.М. Метавычисления и логическое программиро - вание //Семиотические аспекты формализации интеллектуальной деятельности (Боржоми-88) - М., 1988. - С. 23-26.

3. АНДЕРСОН Р. Доказательство правильности программ.-М.: Мир, 1982. - 163 с.

4. АЙЕЛЛО Л., ЧЕККИ К., САРТИНИ Д. Представление и использование метазнаний //Труды института инженеров по электронике и радиоэлектронике. - 1986. - Т.74, №10. - С. 12-31.

5. Базисный РЕФАЛ и его реализация на вычислительных машинах (методические рекомендации) - М.: ЦНИПИАСС Госстроя СССР, 1977. - 300 с.

6. БАРЕНДРЕГТ Х. Ламбда-исчисление. Его синтаксис и семантика. - М.: Мир, 1985. - 606 с.

7. БРАУЭР Ф., БРЕЙ М. и др. На пути к языку широкого спектра для поддержки спецификации и разработки программ //Требования и спецификации в разработке программ. - М., 1984. С. 28-46.

8. БЕЛЬТЮКОВ А.П. Язык дедуктивного программирования //Теория языков программирования. - Ижевск, 1983. - С. 3-18.

9. БОРЩОВ В.Б. Логическое программирование //Изв. АН СССР. Техническая кибернетика. - 1986. - № 2. - С. 89-109.

10. Его же. ПРОЛОГ - основные идеи и конструкции //Прикладная информатика. - 1986. - Вып. 2 (11).- С. 49-76.

11. Его же. Семантика параметрических конструкций в логическом программировании //Логические методы в программировании. - Новосибирск, 1987. - Вып. 120: Вычислительные системы. - С. 3-23.

12. ВОЛГИ Б.Б. и др. Система ПРИЗ и исчисление высказываний /Б.Б.Волги, М.Б.Мацкин, Г.Е.Минц, Э.Х.Тыугу //Кибернетика. - 1982. - №6. - С. 65-70.

13. ВОРОБЬЁВ С.К. О применении условных систем подстановок термов в верификации программ //Программирование. - 1986. - №4. - С. 3-14.

14. ВОРОНКОВ А.А. Логические программы и их синтез. - Новосибирск, 1986. - 32 с. - (Препринт/АН СССР, Сиб. отделение. Ин-т математики, № 23).

15. Его же. Синтез логических программ. Новосибирск,1986. - 42 с. - (Препринт/ АН СССР Сиб. отделение. Ин-т математики, № 24).

16. Его же. Реализуемость и синтез программ /Автореф.дис. на соискание ученой степени канд. физ.-мат. наук. - Новосибирск, 1987.

17. ГЁДЕЛЬ К. Об одном еще не использованном расширении финитной точки зрения //Математическая теория логического вывода. - М., 1967. - С. 299-304.

18. ГОНЧАРОВ С.С. Σ-программы и эффективные реализации списочной надстройки //Труды Всесоюзной конференции по прикладной логике. - Новосибирск, 1985. - С. 57-60.

19. Его же. Замечания об аксиомах списочной надстройки GES //Логические вопросы теории типов данных. - Новосибирск, 1986. - Вып. 114: Вычислительные системы. - С. 11-15.

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

21. Их же. Математические основы семантического программирования //Докл. АН СССР. - 1986. - Т. 289, №6. - С. 1324-1328.

22. Их же. Σ-программы и их семантики //Логические методы в программировании. - Новосибирск, 1987. - Вып. 120: Вычислительные системы. - С. 24-51.

23. ГОНЧАРОВ С.С., ЕРШОВ Ю.Л., СВИРИДЕНКО Д.И. Методоло - гические аспекты семантического программирования //Научное значение: логика, понятия,структура. - Новосибирск, 1987. -С.154-184.

24. ГРИС Д. Наука программирования.-М.: Мир,1984.-416 с.

25. ДАРЛИНГТОН Дж. Синтез нескольких алгоритмов сортировки //Кибернетический сб. - М., 1986. - Вып. 18. - С. 141-176.

26. ДЕГТЯРЕВ А.И., ВОРОНКОВ А.А. Методы управления равенством в автоматическом доказательстве теорем //Кибернетика. - 1986. - №3. - С. 56-72.

27. ДЕЙКСТРА Э. Дисциплина программирования. - М.: Мир, 1978. - 280 с.

28. ЕРШОВ А.П. Трансформационный метод в технологии программирования //Тез. докладов 1 Всесоюз. конф. "Технология программирования" - Киев, 1979. - С. 12-26.

29. Его же. Научные основы доказательного программирования //Вестн.АН СССР. - 1984. - №10. - С. 7-18.

30. Его же. Теория нумераций. - М.: Наука, 1977. - 414 с.

31. Его же. Проблемы разрешимости и конструктивные модели. - М.: Наука, 1980. - 415 с.

32. Его же. Динамическая логика над допустимыми множествами //Докл. АН СССР. - 1983. - Т. 273, № 5. - С. 1045-1048.

33. Его же. Σ-предикаты конечных типов над допустимыми множествами //Алгебра и логика. - 1985.- Т.24,№5.-С.563-602.

34. Его же. Язык Σ-выражений //Логические вопросы теории типов данных. - Новосибирск, 1986. -Вып. 114: Вычислительные системы. - С. 3-10.

35. КАСЬЯНОВ В.Н. Редуцированные преобразования программ //Трансляция и оптимизация программ. - Новосибирск, 1983. - С. 86-98 (ВЦ СО АН СССР).

36. КАХРО М.И., КАЛЬЯ А.П., ТЫУГУ Э.Х. Инструментальная система программирования ЕС ЭВМ (ПРИЗ). - М.: Финансы и статистика. - 1981.

37. КЛИНИ С.К., ВЕСЛИ Р.Ю. Основания интуиционистской логики. - М.: Наука, 1978. - 272 с.

38. КЛОКСИН В., МЕЛЛИШ К. Программирование на языке ПРОЛОГ. - М.: Мир, 1987. - 336 с.

39. КОЛМОГОРОВ А.Н. К толкованию интуиционистской логики //А.Н.Колмогоров. Избранные труды. Математика и механика. -М., 1985. - С. 142-148.

40. ЛАВРОВ С.С. Методы задания семантики языков программирования //Программирование. - 1978. - №6. - С. 3-10.

41. Его же. Синтез программ //Кибернетика. - 1982. - №6. - С. 11-16.

42. ЛАВРОВ С.С., СИЛАГАДЗЕ Г.Г. Автоматическая обработка данных. Язык ЛИСП и его реализация. - М.: Наука, 1978.

43. МАННА З. Теория неподвижной точки программ //Кибернетический сб. - 1978. - Вып. 15. - С. 38-100.

44. МИНЦ Г.Е. Логические основы синтеза программ. - Таллинн, 1982. - 42 с. - (Препринт/АН ЭССР Ин-т кибернетики).

45. Его же. Некоторые формальные системы логического программирования //Кибернетика. - 1987. - №6. - С. 97-98.

46. МИНЦ Г.Е., ТЫУГУ Э.Х. Обоснование структурного синтеза программ //Автоматический синтез программ. - Таллинн, 1983. - С. 52-60.

47. НЕПЕЙВОДА Н.Н. Семантика алгоритмических языков //Итоги науки и техники, сер. Теория вероятностей, мат. статистика, теор. кибернетика. - М.: ВИНИТИ, 1983. - Т. 20. - С. 95-166.

48. Его же. Математическая теория синтеза программ (обзор) //Синтез программ. - Устинов, 1985. - С. 4-8.

49. Его же. Анализ и методы доказательного программирования в конструктивных логиках /Автореф.дис. на соискание ученой степени доктора физ.-мат.наук./- Ижевск, 1988.

50. НЕПЕЙВОДА Н.Н., СВИРИДЕНКО Д.И. Программирование с логической точки зрения. Новосибирск, 1981. - 50 с. - (Препринт/АН СССР Сиб. отделение. Ин-т математики, Часть 1).

51. Их же. Логическая точка зрения на программирование. Новосибирск, 1981. - 50 с. - (Препринт/АН СССР Сиб. отделение. Ин-т математики. Часть 2).

52. Их же. К теории синтеза программ //Труды Ин-та математики/АН СССР Сиб. отделение. - 1982. - Т.2: Математическая логика и теория алгоритмов. - С. 159-175.

53. НЕПОМНЯЩИЙ В.А. Практические методы верификации программ /Кибернетика. - 1984. - №2. - С. 21-28, 43.

54. Его же. О проблемно-ориентированной верификации программ //Программирование. - 1986.-№2. - С. 3-12.

55. НЕПОМНЯЩИЙ В.А., САБЕЛЬФЕЛЬД В.К. Трансформационный синтез корректных программ //Прикладная информатика. - 1986. - Вып. 2(11). - С. 19-38.

56. РЕДЬКО В.Н. Основания композиционного программирования //Программирование. - 1979. - №3. - С. 3-13.

57. РЕДЬКО В.Н., НИКИТЧЕНКО Н.С. Композиционные аспекты программологии. 1. //Кибернетика. - 1987. - №5. -С. 49-56.

58. РОБИНСОН Дж. А. Машинно-ориентированная логика, основанная на принципе резолюции //Кибернетический сб. (новая серия). - 1970. - Вып. 7. - С. 194-218.

59. РОМАНЕНКО А.Ю. Построение обратных функций //Семиотические аспекты формализации интеллектуальной деятельности (Боржоми-88). - М., 1988. - С. 62-65.

60. РОМАНЕНКО С.А. РЕФАЛ 4 - расширение РЕФАЛа-2, обеспечивающее представление результатов прогонки. - М.,1987.- (Препринт/АН СССР. Ин-т прикладной математики им. Келдыша, №147).

61. САЗОНОВ В.Ю. Категория многосортных алгебраических теорий, эквивалентная категории категорий с конечными произведениями //Тез. IУ Всесоюзн. конф. по применению методов мат. логики. - Таллинн, 1986. - С. 167-169.

62. САЗОНОВ В.Ю., СВИРИДЕНКО Д.И. Денотационная семантика языка Σ-выражений //Логические вопросы теории типов данных.- Новосибирск, 1986. - Вып. 114: Вычислительные системы. -С. 16-34.

63. СВИРИДЕНКО Д.И. Некоторые вопросы математической семантики языков программирования //Эмпирическое предсказание и распознавание образов. - Новосибирск, 1976. - Вып. 67: Вычис - лительные системы. - С. 93-105.

64. Его же. О природе программирования //Математическое обеспечение вычислительных систем из микро-ЭВМ. - Новосибирск, 1983. -Вып. 96: Вычислительные системы. - С. 51-74.

65. Его же. Предпосылки логического подхода к программи - рованию //Философские основания научной теории. - Новосибирск, Наука, 1985. - С. 91-107.

66. Его же. Проектирование Σ-программ. Постановка проблемы //Методы анализа данных. - Новосибирск, 1985. -Вып. 111: Вычислительные системы. - С. 108-127.

67. Его же. Проектирование Σ-программ. Σ-оцениваемость //Логические вопросы теории типов данных. - Новосибирск, 1986. - Вып. 114: Вычислительные системы. - С. 59-83.

68. Его же. Об одном варианте теории списочных надстроек //Прикладная логика. - Новосибирск, 1968. - Вып. 116: Вычислительные системы. - С. 64-76.

69. Его же. Об одном обогащении языка Σ-программ //Логические методы в программировании.-Новосибирск, 1987. -Вып.120: Вычислительные системы. - С. 97-104.

70. Семантика языков программирования: Пер. с англ. -М.: Мир, 1980. - 396 с.

71. ТЫУГУ Э.Х. Синтез программ (обзор) //Тез. докл. Всесоюз. конф. по методам математической логики в проблемах искусственного интеллекта и семантическое программирование. - Вильнюс, 1980. -С. 70-89.

72. Его же. Концептуальное программирование. -М.: Наука, 1984. - 256 с.

73. Его же. ПРОЛОГ или логическое программирование //Тез. докладов 1У Всесоюз. конф. по применению методов мат. логики. - Таллинн,1986. -С. 11-15.

74. ЭВМ пятого поколения. Концепции, проблемы, перспективы: Пер. с японского.-М.: Финансы и статистика. - 1984. -110 с.

75. ЮЩЕНКО Е.Л., КАСАТКИНА И.В. Современные методы доказательства правильности программ //Кибернетика. - 1980. - № 6. - С. 37-62.

76. APT K.R., Van EMDEN M.H. Contributions to the theory of logic programming //J.ACM-1982. -Vol. 29. -P. 841-863.

77. BASKUS J. Can programming be liberated from the von Neumann Style? A functional style and its algebra of programs //Comm. ACM. - 1978. - Vol. 21, N 8. -P. 613-641.

78. BATES J.L., CONSTABLE R.L. Proofs as programs // ACM Trans. Programming Languages and Systems. - 1985. -Vol. 7,N 1. - P. 113-136.

79. BARBUTI R., BELLIA M., LEVI G., MARTELLY M. On the integration of logic programming and functional programming// Proc. of the 1984 Internat. Symp. on Logic Programming. - N.Y. Atlantic City. - 1984. -P. 160-166.

80. BEESON M. Foundation of constructive mathematics // Mathematical studies. - Berlin: Springer. - 1985. -P. 180.

81. BEESON M. Proving programs and programming proofs // Logic, methodology and philosophy of science VII (Studies in logic and foundations of mathematics, vol. 114). - Amsterdam: North-Holland. - 1986. -P. 51-82.

82. BELLIA M., LEVI G. The relation between logic and functional languages: a survey //J. logic programming. - 1986. - Vol. 3, N 3. - P. 217-236.

83. BERKLING K. Epsilon-reduction: another view of unification //Syracuse University, CASE Center, 1985.

84. BERKLING K., ROBINSON J.A., SIBERT E.E. A proposal for a 5th generation logic and functional programming system based on highly parallel reduction machines architecture //Syracuse Unuversity, CASE Center, Syracuse, 1982.

85. BERKLING K., FEHR E. A modification of the lambda calculus as a base for functional programming languages //Lect.Notes in comp. sci. - 1982. - Vol. 140. -P. 35-47.

86. BUCHBERGER B. History and basis feature of the critical-pair/complction procedure //J.symb. comput. - 1987. -N 3. - P. 3-38.

87. BURSTALL R.M., DARLINGTON J. Transformation for developing recursive programs //J. ACM. - 1977. - Vol. 24. N 1. - P. 44-67.

88. BURSTALL R.M., MacQUEEN D.B., SANNELLA D.T. HOPE: An experemental applicative language //Conferense record of the 1980 LISP Conference, Stanford. - 1980. -P. 136-143.

89. CARLSSON M. On implementing PROLOG in functional programming //Proc. IEEE Symp. on logis programming. - 1984. - P. 154-159.

90. CHIKAYAMA T. ESP - Extendent self-contained PROLOG as a preliminary kernel language of fifth generation computer // New generation computing. - 1983. - Vol. 1, N. 1.-P. 11-24.

91. CLARK K.L., DARLINGTON J. Algorithm classification trough synthesis //The computer joural. - 1980. - Vol.23, N 1. - P.61-65.

92. CLARK K.L., GREGORY S. A relational language for paraller programming //Proc. ACM Conf. on functional programming languages and computer architecture. - 1981. -P. 171-178.

93. Idem. PARLOG: a parallel logic programming language. - London: Imperial College, 1983. - PR-83/5.

94. COSTABLE R.L. Constructive mathematics and automatic program writes //Proc. of IEIP'71. - Amsterdam: North-Holland. - 1971. -P. 229-233.

95. CONSTABLE R.L., ZLATIN D.R. The type theory of PL/CV3 //Lect.notes in comp.sci. - 1982. -Vol.131. -P. 72-93.

96. COX P.T., PIETRZYKOWSKI T. Incorporating equality into logic programming via surface deduction //Ann. pure and appl. log. - 1986. - Vol.31. -P. 177-189.

97. COUSINEAU G., CURIEN P.L., MAUNY M. The categorical abstract machine //Lect. notes in comp. sci. - 1985. -Vol.201. - P. 50-64.

98. CURRY H.B., FEYS R. Combinatory logic. - Amsterdam: North-Holland, 1968.

99. DARLINGTON J., FIELD A.J., PULL H. The unification of functional and logic language //Logic programming: Functions, Relations and Equations. - Prentice-Hall, Englewood.Cliffs,N.Y. - 1986. -P. 37-70.

100. DERANSART P., FERRAND G. Detection d'erreurs en programmation logique //Rapport de Recherche. Lab.d'Informatique, Univ. D'ORLEANS. - 1985. - 33 p.

101. Idem. Logic programming, methodology and teaching // Actes du Seminaire, 1986. CNET Tregastel. - 1986. -P. 75-90.

102. DERSHOWITZ N. Commputing with rewrite systems //Inform. and control. - 1985. - Vol.65. -P. 122-157.

103. DERSHOWITZ N., JOSEPHSON N.A. Logic programming by completion //Proc. of the 2nd Internat. logis programming conference. - 1984. -P. 313-320.

104. DERSHOWITZ N., PLAISTED D.A. Logic programming cum applicative programming //Proc. of the 1985. Symp. on Logic programming, Boston. - 1985. -P. 54-66.

105. ERSHOV A.P. The transformational machine, theme and variations //Lect. notes in comp. sci. - 1982. -Vol.118.-P.16-32.

106. ERSHOV Yu.L., GONCHAROV S.S., SVIRIDENKO D.I. Semantic foundations of programming //Lect. notes in comp.sci.-1987. -Vol. 278. -P. 116-122.

107. FEFERMAN S. Constructive theories of functions and classes //Logic Collog'78. - 1978. -P. 159-224.

108. FERRAND G. Error diagnosis in logic programming: an adaptation of E.Y.Shapiro's method //RR-375, Paris, INRIA. - 1985. - 36 p.

109. FITTING M. A Kripke-Kleene semantics for logic programs //J.logic programming. - 1985. - Vol. 2, N 4.-P.295-312.

110. FITTING M. Enumeration operators and modular logic programming //J.logic programming.-1987. -Vol.3,N 1.-P.11-22.

111. FLOYD R.W. Assigning meaning to programs //Proc.Symp. Appl. Math. Vol.19. Mathematical aspects of computer science.- Providence. - 1967. -P. 19-32.

112. FRIBOURG L. A narrowing procedure for theories with constructors //Lect. notes in comp. sci. - 1984. - Vol. 170. -P. 259-281.

113. FRIBOURG L. SLOG: A logic programming language interpreter based on clausal superposition and rewriting //Proc. IEEE Symp.on logic programming. IEEE Comp.soc.press. - 1985. - P. 172-184.

114. FRIBOURG L. Oriented Equational clauses as a programming language //J.logic programming. - 1984. -Vol. 1, N 2. - P. 165-177.

115. DOWLING W.P., CALLIER J.H. Linear-Time algorithms for testing the satisfiability of propositional Horn formulae //J.logic programming.-1984. -Vol.1, N 3. -P. 267-284.

116. GABBAY D.M., REYLE V. N-PROLOG: An extension to PROLOG with hypothetical implications.I //J.logic programming.- 1984.- Vol.1, N 4.- P.319-355.

117. GALLIER J.H., RAATZ S. HORNLOG: A Graph-based interpreter for general Horn clauses //J.logic programming. - 1987. - Vol. 4, M 1. -P. 119-155.

118. GOOD C.A. Computational uses of the manipulation of formal proofs //Stanford Univ. Report N STAN-CS-80-819.-1980. - 122 p.

119. GOGUEN J.A., MESEGUER J. Equality, types modulas and (why not?) generics for logic programming //J.logic program - ming. - 1984. -Vol.1, N 2. -P. 179-210.

120. GORDON M.J., MILNER A.J., WADSWORTH C.P. Edinburgh LCF //Lect. notes in comp.sci. - 1979. -Vol. 78. - 160 p.

121. GOTO S. Program synthesis from natural deduction proofs //Proc. of the 6th Internat. Joint Conf. on Artificial Intelligence. -Vol. 1. - Amsterdam, 1979. -P. 339-341.

122. GONCHAROV S.S., SVIRIDENKO D.I. Theoretical aspect of Σ-programming //Lect. notes in comp. sci. - 1986. -Vol.215. -P. 169-179.

123. GONCHAROV S.S., ERSHOV Yu.L., SVIRIDENKO D.I. Semantic programming //Proc. 10th World Congress Information Processing 86. -Amsterdam, 1986. - P. 1093-1100.

124. GOPALAN Nadathur. A Higher-order logis as the basis for logic programming //Univ. of Pensylvania,Philadelpha, MS-CIS-87-48, LINC 69. - 1987. - 112 p.

125. HAYASHI S. Extracting LISP-programs from proofs // PRIMS, Kyoto University. - 1983. -Vol. 19, N 1. -P. 169-191.

126. HOARE C.R. An axiomatic basis for computer prog - ramming //Comm. SCM. - 1969. - Vol. 12, N 10. -P. 576-583.

127. HOGGER C.J. Introduction to logic programming. -London: Academic Press, 1984. - 278 p.

128. HOFFMANN C.M., O'DONNELL M.J. Programming with equations //ACM Trans. Programming languages and systems. - 1987. - Vol. 4, N 1. - P. 83-112.

129. HOWARD W.A. The formulae-as-types notion of construction //Festschrift on the occasion of H.B.Curry's 80th birthday. - N.Y. - 1980. -P. 479-490.

130. JOUANNAUD J.-P., KIRCHNER C., KIRCHNER H. Incremen - tal construction of unification algorithms in equational theories //Lect.notes in comp. sci. - 1983. -Vol. 154. -P.331-346.

131. KORNFELD W.A. Equality for PROLOG //Proc. of 8th Internat. Joint Conf. on Artificial Intelligence. - Karlsruhe: West Germany. - 1983. -P. 514-519.

132. KIRCHNER C. Computing unification algorithms //Proc. of the 1st Symp. on Logic in Comp. Sci. -Cambridge, Mass.,1986. - P. 206-216.

133. KOWALSKI R.A. Predicate logic as programming language //Proc. IEIP'74.-Amsterdam: North-Holland. - 1974. -P.569-574.

134. KOWALSKI R.A. Logic programming //Proc. IFIP'83. - Amsterdam: North-Holland. - 1983. -P. 133-145.

135. KOWALSKI R.A. The relation between logic programming and logic specification //Philosoph. trans. of the Royal soc. of London. - 1985. - Vol. 312. -P. 345-357.

136. LAMBEK J. From Lambda-calculus to cartesian closed categories //To H.B.Curry Essays on Combinatery logic, lambda-calculus and formalism. -Academic Press, N.Y. - 1980.

137. LINSTRÖM G. Functional programming and the logical variable //RR-357, Paris, INRIA. - 1985. - 32 p.

138. LLOYD J.W. Foundation of logic programming. -Berlin: Springer, 1984. - 124 p.

139. MacQUEEN D., PLOTKIN G., SETHI R. An ideal model for recursive polymorphic types //Proc. Principles of Programming Languages Symp. -1984. - P. 165-174.

140. MacLANE S. Categories for the working mathematician. - Berlin: Springer, 1971. - 262 p.

141. MANNA Z., WALDINGER R. Synthesis: dreams - programs //IEEE Trans. on Software Eng. - 1979. -Vol.SE-5, N 4.-P.294 - 328.

142. MARTIN-LÖF P. An intuitionistic theory of types; predicative part //Logic Collaquium'73. - Amsterdam: North-Hol - land. - 1975. -P. 73-118.

143. MARTIN-LÖF P. Constructive mathematics and computer programming //Logic, methodology and philosophy of science.VI. - Amsterdam: North-Holland. - 1982. -P. 153-179.

144. MILLER D.A. A theory of modulas for logic programming //Proc.of the Symp. on logic programming. - 1986. -P.106-115.

145. MILLER D.A., NADATHUR G. Higher-order logic programming //Proc. 3th Internat. Conf. on Logic programming.-1986.

146. MILNER R.A. A theory of type polymorphism in prog - ramming //J.comp.syst. sci. - 1978. -Vol. 17. -P. 348-375.

147. MILNER R.A. A proposal for standart ML //ACM Symp. on LISP and functional programming. - 1984. -P.184-197.

148. The Munich project CIP. -Vol. 1: The wide spectrum language CIP-L //Lect. notes in comp. sci. - 1985. - Vol. 183. - 276 p.

149. O'DONNELL M.J. Equational logic as a programming language //MIT Press, Cambridge, Mass. - 1985.

150. PARTSCH H. The CIP transformation system //Program transformation and program environment. - Rept. Workshop NATO Adv. Res (Munich,1983). -Berlin a.o. - 1984. -P. 305-322.

151.PRAWITZ D. Natural deduction. - Stocholm: Aguist and Wiksel, 1965. - 150 p.

152. PRAWITZ D. Ideals and results in proofs theory // Proc. 2nd Scand. Logic Symp. - Amsterdam: North-Holland.- 1973. - P. 235-308.

153. REDDY U.S. Narrowing as the operational semantics of functional language //Proc. of the 1985. Symp. on logic prog - ramming. -Boston. - 1985. -P. 138-151.

154. RETY P. et al. Narrower: a new algorithm for unifi - cation and its application to logic programming //Lect. notes in comp. sci. - 1985. -Vol. 202. -P. 141-157.

155. REYNOLDS J.C. GEDANKEN - A simple typeless language based on principle of completeness and reference concept // Comm. ACM. - 1970. - Vol. 13, N 5. -P. 308-319.

156. REYNOLDS J.C. Types, abstraction and parametric polymorphism //IEIP'83. - Amsterdam: North-Holland.-1983.-P.513-523.

157. ROBINSON J.A., SIBERT E.E. LOGLISP: an alternative to PROLOG //Machine Intelligence. - 1982. -Vol. 10.-P.399-419.

158. SHAPIRO E.Y. Algorithmic program debugging.-Boston. - MIT Press. - 1983.

159. SHAPIRO E.Y., TAKEUSHI A. Object-oriented program - ming in concurrent PROLOG //New Generation Computing. - 1983. - Vol.1, N 1. -P. 25-48.

160. STEELE G.L., SUSSMAN G.J. CHEME: An interpreter for the extended lambda calculus //AI memo 349, Artificial Intel - ligence Lab., MIT, Cambridge, Mass, 1975.

161. STENLUND S. Combinators, lambda-terms and proof theory. - Dordrecht: D.Reidel, 1972. - 183 p.

162. SUBRAHMANYAM P.A., YOU J.-H. Conceptual basis and evaluation strategies for inegration functional and logic programming //Proc. of the 1984 Internat. Symp. on Logic Programming. - N.Y.,Atlantic City, 1984. -P. 144-153.

163. SATO M. Towards a mathematical theory of program synthesis //Proc. Internat. Joint Conf. on Artificial Intelligence. - Tokyo. - 1979. -P. 757-762.

164. SATO M., SAKURAI T. QUTE: A functional language based on unification //Proc. of FGCS'84. - 1984. -P. 157-165.

165. SCHERLIS W.L., SCOTT D.S. First stepts towards inferentical programming //IFIP'83. -Amsterdam: North-Holland. - 1983. -P. 199-212.

166. SCOTT D.S. Continuous lattices //Toposes, algebraic geometry and logic. -Vol. 274: Lect. Notes in Math. - 1972.- P. 97-136.

167. SCOTT D.S. Relating theories of the lambda-calculus //To H.B.Curry: Essay on Combinatory logic, lambda-calculus and formalism. - Academic Press, N.Y. - 1980. -P.403-450.

168. SCOTT D.S. Lectures on a mathematical theory of computation. -Oxford: Oxford Univ. Computing Lab., Programming Reseach Group, PRG-19. - 1981. - 148 p.

169. SCOTT D.S. Domains for denotational semantics //Lect. notes in comp.sci. - 1982. -Vol. 140. -P. 577-610.

170. TOGASHI A., NOGUSHI S. A program transformation from equational programs into logic programs //J.logic programming. - 1987. -Vol. 4, N 2. -P. 85-103.

171. TYUGU E.H., MATSKIN M.B., PENJAM J.E., EOMOIS P.U. NUT - an object-oriented language //Computer and artificial intelligence. - 1986. -Vol. 5, N 6. -P. 521-542.

172. TURCHIN V. The language REFAL,the theory compilation and metasystem analysis //Technical Report N 18, Currant Institute of Mathematics. - N.Y., 1980.

173. TURNER D.A. SASL Language manual. -St. Andrews Uni - versity, 1976.

174. TURNER D.A. A new implementation technique for applicative language //Software: Practice and Experience. - 1979. -Vol. 9. -P. 31-49.

175. TURNER D.A. A nother algorithm for bracket abstrac - tion //J.symbol. log. - 1979. -Vol. 44, N 2. -P. 267-270.

176. VEDA K.K. Guarded Horn clauses //Technical report TR-103, ICOT. - 1985.

177. Van EMDEN M.H., KOWALSKI R.A. The semantics of pre - dicate logic as a programming language //J.ACM.-1976. -Vol.23. N 4. -P. 733-742.

178.Van EMDEN M.H., YUKAWA K. Logic programming with equations //J.logic programming. - 1987. -Vol. 4, N 4. -P.265-288.

Поступила в ред.-изд.отд.

24 июня 1988 года