**Российская академия наук**
**Сибирское отделение**
**Институт систем информатики**
**имени А. П. Ершова**

**В.И. Шелехов**

# ИСЧИСЛЕНИЕ ВЫЧИСЛИМЫХ ПРЕДИКАТОВ

**Препринт**
**143**

**Новосибирск 2007**

Исчисление вычислимых предикатов определяется набором базисных вычислимых предикатов и шестью видами вычислимых формул: суперпозицией, альтерацией, параллельной композицией, порождением предиката, применением предиката и конструктором массива. Набор базисных вычислимых предикатов определяет систему типов данных. Исчисление является математической моделью класса программ, спецификация которых представима в виде формулы на языке исчисления предикатов. Исчисление вычислимых предикатов является минимальным базисом для построения языков функционального программирования.

**Siberian Division of the Russian Academy of Sciences**
**A. P. Ershov Institute of Informatics Systems**

**V.I. Shelekhov**

# THE CALCULUS OF COMPUTABLE PREDICATES

**Preprint**
**143**

**Novosibirsk 2007**

The calculus of computable predicates is defined by a collection of basic predicates and the following six computable formulas: superposition, alteration, parallel composition, predicate generation, predicate application, and array constructor. The collection of basic predicates defines a data type system. The calculus is a mathematical model of the class of programs whose specification may be written as a formula of the higher-order predicate calculus. The calculus of computable predicates may be used to construct pure functional programming languages.

## 1. ВВЕДЕНИЕ

Имеются два определяющих свойства программы: *автоматическая вычислимость* и существование *спецификации* программы. Определены три формы спецификации программы: предикатная, процессная и процессорная [1]. Предикатная спецификация может быть представлена в виде формулы на языке исчисления предикатов[1]. Процессная спецификация описывает взаимодействие параллельных процессов в системе реального времени. Процессорная спецификация определяет функционирование процессора некоторого языка программирования.

Автоматическая вычислимость определяется как существование *процессора*, способного в автоматическом режиме исполнять произвольную программу на некотором языке программирования [1]. В настоящей статье представлена математическая модель программ с предикатной спецификацией. Она является конкретизацией содержательно определенного понятия автоматической вычислимости. Предлагаемая модель существенно проще по сравнению с моделями, используемыми для классов программ с процессной и процессорной спецификацией.

Модель программ с предикатной спецификацией представлена в виде *исчисления вычислимых предикатов*, определяющего множество всех вычислимых формул исчисления предикатов. Исчисление вычислимых предикатов определяет набор базисных вычислимых предикатов и шесть вычислимых формул, используемых для порождения остальных вычислимых формул. Базисные вычислимые предикаты определяют операции со значениями типов данных.

В разд. 2 данной статьи описывается общая схема построения программ с предикатной спецификацией. Программы этого класса имеют математическую природу. Они строятся на базе математических свойств, доказуемых из спецификации программы. В разд. 3 показывается, что автоматическая вычислимость программы предопределена вычислимостью свойств, доказуемых из спецификации. Исследование автоматической вычислимости достаточно провести для формул исчисления предикатов. В разд. 4 определяется, что множество вычислимых формул имеет форму исчисления, названного *исчислением вычислимых предикатов*. Сигнатура исчисления содержит набор базисных вычислимых предикатов, определяемых системой типов данных, и шестью вычислимыми формулами, называемыми ба-

[1] Здесь и далее имеется в виду исчисление предикатов высших порядков.

зисными композициями и используемыми для порождения всего множества вычислимых формул. Оказывается, что программа на языке функционального программирования конвертируется в набор вычислимых формул исчисления предикатов. Поэтому исчисление вычислимых предикатов является базисом для всех языков функционального программирования.

Далее определяется исчисление вычислимых предикатов. В разд. 5 описывается математическая модель системы типов данных. Предложенная система типов покрывает системы типов многих известных языков программирования. Определение рекурсивных типов дано с использованием аппарата наименьшей неподвижной точки. В разд. 6 описываются базисные вычислимые композиции: суперпозиция, альтерация, параллельная композиция, порождение предиката, применение предиката и конструктор массива. Разд. 7 определяет семантику рекурсивной системы определений предикатов в виде наименьшей неподвижной точки системы рекурсивных уравнений. В заключении рассматривается проблема выбора системы типов в качестве базиса исчисления вычислимых предикатов и постулируется возможность построения произвольного языка функционального программирования из исчисления вычислимых предикатов.

## 2. ПРОГРАММЫ С ПРЕДИКАТНОЙ СПЕЦИФИКАЦИЕЙ

Спецификация программы называется *предикатной*, если она эксплицируема в виде формулы на языке исчисления предикатов. Такая экспликация возможна для спецификации, представленной в математической форме. Для программы с предикатной спецификацией характерна простая схема взаимодействия программы с окружением: ввод входных данных происходит в начале исполнения программы, вывод результирующих данных — в конце, а других взаимодействий с окружением нет. Спецификация определяет функцию, отображающую значения входных данных в значения результатов.

Предикатная спецификация есть условие математической задачи, исходными данными которой являются входные данные программы, а неизвестными — результаты программы. Программа является реализацией алгоритма решения математической задачи. Алгоритм строится с использованием *свойств* (утверждений, лемм, теорем), доказуемых из условия задачи строгими математическими методами. Описание алгоритма обычно является содержательным, хотя доказательство правильности алгоритма остается

математически строгим. Общая схема, отражающая связь предикатной спецификации и программы, представлена на рис. 1.

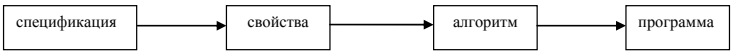


*Рис. 1.* Схема построения программы из предикатной спецификации

Программа является результатом кодирования алгоритма на языке программирования. Логические связи алгоритма, кодируемые в конструкциях языка программирования, принимают существенно более сложную форму. Практика императивного программирования такова, что при необходимости определить свойства программы, программист действует не в математическом стиле — он пытается анализировать эти свойства по результатам исполнения программы; в том числе с помощью отладчика. Следствия этого общеизвестны: неизбежные ошибки, далеко не всегда устраняемые посредством тестирования и отладки.

Итак, алгоритм решения задачи имеет математическую природу, но формулируется содержательно. Формальное представление алгоритма реализуется в программе, однако императивная программа де-факто не является математическим объектом.

## 3. АВТОМАТИЧЕСКАЯ ВЫЧИСЛИМОСТЬ

Автоматическая вычислимость [1] присуща программе, но не алгоритму, представленному содержательным описанием. Для автоматической вычислимости алгоритма, по меньшей мере, необходима его формализация. Поэтому алгоритм автоматически вычислим лишь потенциально. Оказывается, что потенциальной автоматической вычислимостью обладает не только алгоритм, но и математические свойства, доказуемые из условия задачи и используемые для построения алгоритма. Чтобы эти свойства были автоматически вычислимы, необходимо описать их формально и придать им форму программы. Подобный стиль представления математических формул в виде программы характерен для функционального программирования.

Программа на чистом[2] языке функционального программирования может быть конвертирована в виде набора формул на языке исчисления предикатов. Это следует из того, что для языковой конструкции любого вида можно представить соответствующий эквивалент в исчислении предикатов. Формулы, полученные из функциональной программы, обладают свойством автоматической вычислимости: можно построить процессор, который будет исполнять эти формулы как программу, причем исполнение будет эквивалентно исполнению исходной функциональной программы.

Свойства, доказуемые из условия задачи и используемые для построения алгоритма, могут быть записаны в виде формул исчисления предикатов. Автоматическая вычислимость этих формул определяется существованием процессора, исполняющего их.

Таким образом, исследование автоматической вычислимости программ сводится к определению автоматической вычислимости для формул исчисления предикатов. Множество всех вычислимых формул является универсальным базисом для всех языков функционального программирования, поскольку каждому языку функционального программирования соответствует некоторое подмножество вычислимых формул.

## 4. СТРУКТУРА МНОЖЕСТВА ВЫЧИСЛИМЫХ ФОРМУЛ

Закономерности построения множества вычислимых формул попытаемся проследить на примере известной формулы, определяющей суперпозицию двух функций:

$$A(x, y) \equiv (\exists z)\,(B(x, z)\ \&\ C(z, y)). \qquad (*)$$

Здесь предикаты A, B и C обозначают функции. Аргументом функции является первый параметр соответствующего предиката, а результатом функции — второй параметр предиката. Далее для нас будет необходимо явно указывать, что одни параметры предиката являются аргументами, а другие — результатами. Для этой цели в списке параметров предиката используется двоеточие “**:**” в качестве разделителя: слева от двоеточия находятся аргументы, а справа — результаты. Определение (*) переписывается следующим образом:

$$A(x\mathbf{:}\ y) \equiv (\exists z)\,(B(x\mathbf{:}\ z)\ \&\ C(z\mathbf{:}\ y)). \qquad (**)$$

[2] Практически все языки функционального программирования содержат расширения, включающие конструкции императивного программирования в целях повышения эффективности программ. Такие языки не являются чистыми.

Предположим, что предикаты B и C являются автоматически вычислимыми: т.е. по значению аргумента можно вычислить соответствующее значение результата, удовлетворяющее предикату. Пусть $P_B$ и $P_C$ — процессоры, умеющие автоматически вычислять предикаты B и C. Тогда программа процессора $P_A$ для автоматического вычисления предиката A конструируется очевидным образом:

$$P_A(x, y) \quad \{ P_B(x, z); P_C(z, y) \}$$

В программе процессора $P_A$ сначала запускается процессор $P_B$ с входным аргументом x. Далее полученный результат подается аргументом процессору $P_C$, и его результат y будет являться результатом процессора $P_A$.

Итак, формула $(\exists z)\,(B(x{:}\,z)\ \&\ C(z{:}\,y))$ является автоматически вычислимой для произвольных вычислимых предикатов B и C. Данную формулу для суперпозиции функций обозначим через Sup(B, C), где B и C рассматриваются как параметры, на место которых можно подставлять имена любых вычислимых предикатов. Действительно, если $\varphi(x{:}\,z)$ и $\psi(z{:}\,y)$ — вычислимые предикаты, то $\mathrm{Sup}(\varphi, \psi)$ является вычислимой формулой. Таким образом, формула Sup(B, C) определяет механизм порождения новых вычислимых формул.

Формулу вида $K(B_1, B_2, \ldots, B_n)$ будем называть *вычислимой композицией*, если она является автоматически вычислимой для произвольных входящих в формулу K вычислимых предикатов $B_1, B_2, \ldots, B_n$, которые называются *параметрами композиции*. Примером вычислимой композиции является формула Sup(B, C). Вычислимая композиция $K(B_1, B_2, \ldots, B_n)$ определяет механизм порождения новых вычислимых формул: если $\varphi_1, \varphi_2, \ldots, \varphi_n$ — вычислимые предикаты, то $K(\varphi_1, \varphi_2, \ldots, \varphi_n)$ определяет новую вычислимую формулу.

Вычислимая формула, полученная из вычислимой композиции подстановкой некоторых вычислимых формул на место параметров композиции, в свою очередь, может оказаться вычислимой композицией[3]. Вычислимая композиция называется *базисной*, если она не может быть получена из других вычислимых композиций. В данной работе определяется шесть базисных вычислимых композиций: суперпозиция, альтерация, параллельная композиция, порождение предиката, применение предиката и конструктор массива (см. разд. 6).

В качестве вычислимых формул могут использоваться вычислимые предикаты, которые невозможно получить из вычислимых композиций.

[3] Доказательство того, что полученная формула является вычислимой, достаточно провести лишь для нескольких базисных вычислимых композиций.

Такие предикаты называются *базисными*. Базисный предикат определяет элементарную операцию со значениями некоторого типа данных. Каждый тип данных однозначно определяется некоторым набором базисных предикатов. Совокупность всех базисных предикатов определяет систему типов данных.

Множество вычислимых формул имеет форму исчисления, которое назовем *исчислением вычислимых предикатов*. Сигнатура исчисления определяется набором базисных вычислимых предикатов и шестью базисными вычислимыми композициями. Будем предполагать существование некоторого абстрактного процессора, реализующего автоматическое вычисление любого предиката, принадлежащего исчислению.

*Предикат* $\varphi(d, e)$, принадлежащий исчислению, определяется именем $\varphi$ и наборами переменных $d$ и $e$, где $d = d_1, d_2, ..., d_n$, $e = e_1, e_2, ..., e_m$, причем $n \geq 0$, $m > 0$. Перечисленные переменные попарно различны. Процессор по значениям переменных набора $d$ вычисляет значения переменных набора $e$. Чтобы разграничить исходный набор $d$ от вычисляемого набора $e$, используется запись предиката в виде $\varphi(d: e)$. В случае, когда требуется вычислить логическое значение предиката $\varphi$ (т.е. набор $e$ пуст), будем использовать в качестве результата дополнительный параметр $b$ логического типа и записывать предикат в виде $\varphi(d: b)$. Переменная $b$ далее будет обозначать логическую переменную. Вместе с предикатной формой $\varphi(d: e)$, являющейся канонической, будем также использовать эквивалентную ей функциональную форму записи $e = \varphi(d)$.

Предикат с именем $\varphi$ принадлежит исчислению, если либо он является базисным, либо $\varphi = K(\varphi_1, \varphi_2, ..., \varphi_n)$, где $K$ — одна из шести базисных вычислимых композиций, $\varphi_1, \varphi_2, ..., \varphi_n$ — предикаты, принадлежащие исчислению. Предикат $\varphi$ может определяться рекурсивно (см. разд. 7).

## 5. СИСТЕМА ТИПОВ ДАННЫХ

*Тип* определяет множество значений. Система типов исчисления вычислимых предикатов включает примитивные типы, подмножество произвольного типа и структурные типы. Для каждого типа набор операций со значениями типа представлен в виде базисных предикатов.

*Примитивными типами* являются: логический, целый, вещественный и литерный. Для них соответственно будем использовать имена: BOOL, INT, REAL и CHAR. Набор базисных предикатов для примитивных типов соответствует распространенным арифметическим и логическим операциям. На-

пример, базисные предикаты: +(x, y: z), -(x, y: z), -(x: y), <(x, y: b) соответствуют операциям: z = x + y, z = x - y, y = -x , b = x < y.

Для каждого примитивного типа определены два вида предикатов равенства: =(x: y) и =(x, y: b). Для предиката =(x: y) процессор присваивает переменной y значение переменной x. Для предиката =(x, y: b) процессор определяет значение отношения x = y и присваивает его логической переменной b. Предикат =(d: e) определен также для наборов переменных d и e.

Имеется набор базисных предикатов для задания констант. Предикаты ConsIntZero( : x) и ConsIntOne( : x) определяют целочисленные значения 0 и 1 в качестве значения переменной x.

*Подмножество типа* T определяется как множество истинности некоторого предиката P(x, d), где x — переменная типа T, а d — произвольный, возможно пустой, набор переменных. Подмножество типа T на базе предиката P есть S = Subset(T, d) = {x∈T | P(x, d)}. Переменные набора d являются параметрами типа S. Предикат P(x, d: b) должен принадлежать исчислению, т.е. быть вычислимым. Примером подмножества типа является тип натуральных чисел: NAT = Subset(INT) = {x∈INT | x ≥ 0}. Другим примером является диапазон целых чисел:

$$DIAP(n) = Subset(INT, n) = \{x \in INT \mid x \geq 1 \;\&\; x \leq n\}.$$

*Структурный тип* определяется в виде композиции других типов, называемых *компонентными* по отношению к структурному. Структурными типами являются: произведение типов (кортеж), объединение типов, массив, множество подмножеств типа и предикатный тип. Базисные предикаты для структурного типа подразделяются на конструкторы, деструкторы и другие операции со значениями типа. *Конструктор* по значениям компонентных типов строит значение структурного типа. *Деструктор* для значения структурного типа определяет соответствующие значения компонент.

*Произведение типов* $Z = X \times Y$ определяет тип Z в виде множества кортежей (x, y), т.е. $Z = \{(x, y) \mid x \in X, y \in Y\}$. Конструктором является предикат ConsStruct(x, y: z), где $x \in X$, $y \in Y$ и $z \in Z$, а деструктором — CompStruct(z: x, y). Для конструктора и деструктора истинно отношение z = (x, y).

*Объединение*[4] *типов* $Z = X + Y = \{(1, x) \mid x \in X\} \cup \{(2, y) \mid y \in Y\}$ определяет множество элементов (1, x) и (2, y) для типов X и Y. Первая компонента в этих кортежах (1 или 2) является *тегом* значения. Конструктор ConsUnion1(x: z) строит структурное значение z по некоторому $x \in X$, т.е. z = (1, x). Аналогично, для конструктора ConsUnion2(y: z) имеет место

[4] Размеченное объединение в терминологии Т. Хоара [2].

z = (2, y). Деструктор CompUnion(z: i, x, y) для z ∈ Z определяет, что либо z = (1, x) и i = 1, либо z = (2, y) и i = 2. Если i = 1, то значение y не определено. Если i = 2, то значение x не определено.

Тип Z = Set(X) = {z | z ⊆ X} есть *множество подмножеств* конечного типа X. Конструктор ConsSetEmpty( : z) определяет пустое множество в качестве значения переменной z. Конструктор ConsSetElem(x : z) определяет z = {x} для x ∈ X. Предикат CompSet(z, x: b) определяет истинность формулы x ∈ z.

Тип A = Array(X, L) = {a: L → X | ∀i∈L. a(i) ∈ X} определяет множество *массивов* с *элементами* типа X и *индексами* конечного типа L. Конструктор массива будет введен в разд. 6.6. Деструктор CompArray(a, i: x) определяет x равным i-му элементу массива a, т.е. истинно x = a(i), что эквивалентно a(i: x).

*Предикатный тип* Z = Pred(D: E) = {φ(d: e) | d ∈ D, e ∈ E} есть множество вычислимых предикатов вида φ(d: e) для непересекающихся списков переменных d и e. Типы переменных определяются списками типов D и E. Конструктор предиката описан в разд. 6.4.

Тип, принадлежащий исчислению вычислимых предикатов, либо является примитивным, либо вводится следующим определением:

<имя типа> = <типовый терм>

Типовый терм включает все определенные выше способы конструирования новых типов:

<типовый терм> ::= Subset(<имя типа> [, <список переменных>]) |
(<имя типа> × <имя типа>) |
(<имя типа> + <имя типа>) |
Set(<имя типа>) |
Array(<имя типа>, <имя типа>) |
Pred(<список имен типов>: <список имен типов>) |

<список имен типов> ::= <имя типа> |
<список имен типов>, <имя типа>

В качестве <имени типа> здесь используется либо имя примитивного типа, либо имя определяемого типа.

Тип, определяющий подмножество типа, может быть параметризован некоторым набором переменных d. Структурный тип является параметризованным, если параметризован один из его компонентных типов. В качестве параметра может выступать также имя типа.

Если <имя типа>, встречающееся в правой части определения типа, не есть имя примитивного типа и этот тип не объявлен как параметр, то долж-

но существовать определение для этого типа. Таким образом, мы рассматриваем *замкнутую* совокупность определений типов, где любой встречающийся тип либо является примитивным, либо имеет определение, либо считается параметром для некоторых типов, определяемых в данной совокупности определений.

Тип является *рекурсивным*, если определение типа прямо или косвенно, через совокупность определений, использует этот тип.

Дадим два примера рекурсивных типов. Типовый терм вида $(X + Y) + Z$ условимся записывать без скобок: $X + Y + Z$. Введем специальный тип: $NIL = \{$ **nil** $\}$. Имя **nil** обозначает пустую последовательность. Тип последовательности, составленной из элементов типа $T$, вводится следующим рекурсивным определением:

$$Seq(T) = NIL + (T \times Seq).$$

Тип списка LIST для языка Лисп определяется следующим набором рекурсивных определений:

$$LIST = Seq(ATOM)$$
$$ATOM = NIL + NUM + SYMBOL + LIST$$
$$NUM = INT + REAL$$
$$SYMBOL = STRING$$
$$STRING = Seq(CHAR).$$

Рассмотрим замкнутую систему определений типов:

$$X_k = \varphi_k(X_1, X_2,..., X_n);\ k = 1,...,n;\ n > 0,$$

где $\varphi_k$ — типовый терм, зависящий от имен $X_1, X_2,..., X_n$, являющихся именами определяемых типов, примитивных типов и типов, объявленных параметрами. Обозначим $X = (X_1, X_2,..., X_n)$ — вектор типов и $\varphi = (\varphi_1, \varphi_2,..., \varphi_n)$ — вектор типовых термов. Систему определений типов запишем в векторной форме: $X = \varphi(X)$.

Определим отношение $\subseteq$ на множестве всех типов. Для двух разных примитивных типов отношение $\subseteq$ не определено. Если $X = Subset(Y)$, то $X \subseteq Y$. Например, $NAT \subseteq INT$. Отношение $\subseteq$ естественным образом определяется для каждой формы построения типов. Это очевидно для объединения, произведения, подмножества и множества подмножеств. Мы можем распространить определение отношения $\subseteq$ на предикатные типы и типы массивов, моделируя их как специальные формы произведения соответствующих компонентных типов. Отношение $\subseteq$ является отношением порядка. Пустой тип $\varnothing$ является минимальным элементом.

Введем отношение $\sqsubseteq$ на векторах типов:

$$X \sqsubseteq Y \cong \forall k=1..n\ (X_k \subseteq Y_k).$$

Отношение $\sqsubseteq$ является отношением порядка. Вектор типов $\Theta = (\varnothing, \varnothing,..., \varnothing)$ является минимальным элементом. Множество типов и множество векторов типов являются полными решетками [6, 7] относительно $\subseteq$ и $\sqsubseteq$ соответственно.

Рассмотрим последовательность векторов типов $\{X^m\}_{m\geq 0}$, определяемую следующим образом:

$$X^0 = \Theta,\ X^{m+1} = \varphi(X^m),\ m \geq 0.$$

Естественно ожидать, что предел последовательности $\{X^m\}_{m\geq 0}$ (если он существует) даст нам неподвижную точку — решение системы $X = \varphi(X)$. Рассмотрим построение неподвижной точки для рекурсивного типа Seq. Получим следующую последовательность типов:

$\mathrm{Seq}^0 = \varnothing$,

$\mathrm{Seq}^1 = \mathrm{NIL} + (T \times \mathrm{Seq}^0) = \mathrm{NIL} + \varnothing = \{1\} \times \mathrm{NIL} \cup \{2\} \times \varnothing = \{(1, \mathbf{nil})\}$,

$\mathrm{Seq}^2 = \mathrm{NIL} + \{(t, (1, \mathbf{nil}))\}$, где $t \in T$,

$\mathrm{Seq}^3 = \mathrm{NIL} + (\{t_2\} \times (\mathrm{NIL} + \{(t_1, (1, \mathbf{nil}))\}))$ , где $t_1, t_2 \in T$, и т.д.

**Лемма**. Оператор $\varphi$ является *монотонным*, т.е.

$\forall\ Y, Z\ (Y \sqsubseteq Z \Rightarrow \varphi(Y) \sqsubseteq \varphi(Z))$.

Пусть $Y \sqsubseteq Z$ и требуется доказать, что $\varphi(Y) \sqsubseteq \varphi(Z)$, т.е. для всякого $k$ требуется доказать, что $\varphi_k(Y) \subseteq \varphi_k(Z)$. Таким образом, достаточно доказать монотонность для каждого из способов конструирования типов Действительно, если $Y_1 \subseteq Y_2$ и $Z_1 \subseteq Z_2$, то реализуется: $Y_1 + Z_1 \subseteq Y_2 + Z_2$, $Y_1 \times Z_1 \subseteq Y_2 \times Z_2$ и т.д.

Последовательность типов $\{Y^m\}_{m\geq 0}$ является *возрастающей цепью*, если $Y^0 \subseteq Y^1 \subseteq ... \subseteq Y^m \subseteq ...$ . Аналогично, последовательность векторов типов $\{B^m\}_{m\geq 0}$ является *возрастающей цепью*, если $B^0 \sqsubseteq B^1 \sqsubseteq ... \sqsubseteq B^m \sqsubseteq ...$ .

*Верхней гранью* подмножества Q типов является тип Z такой, что $Y \subseteq Z$ для всех $Y \in Q$. Тип Z является *наименьшей верхней гранью*, если $Z \subseteq Z'$ для любой другой верхней грани $Z'$ подмножества Q. Для наименьшей верхней грани цепи типов $\{Y^m\}_{m\geq 0}$ будем использовать обозначение $\cup_{m\geq 0} Y^m$. Аналогичным образом определяется понятие наименьшей верхней грани для подмножества векторов типов. Для наименьшей верхней грани цепи векторов типов $\{B^m\}_{m\geq 0}$ используется обозначение $\cup_{m\geq 0} B^m$.

Оператор $\psi$ является *непрерывным*, если для любой возрастающей цепи векторов типов $\{B^m\}_{m\geq 0}$ имеет место равенство: $\psi(\cup_{m\geq 0} B^m) = \cup_{m\geq 0} \psi(B^m)$. Аналогичным образом определяется понятие непрерывности оператора, определенного на типах и реализующего некоторый способ конструирования новых типов.

**Лемма**. Произведение $Y \times Z$ и объединение $Y + Z$ непрерывны относительно вхождений компонентных типов Y и Z.

Пусть $\{Y^m\}_{m\geq 0}$ и $\{Z^m\}_{m\geq 0}$ — возрастающие цепи типов. Необходимо доказать, что:

$$\cup_{m\geq 0}(Y^m \times Z^m) = (\cup_{m\geq 0}Y^m) \times (\cup_{m\geq 0}Z^m).$$

Докажем сначала, что тип левой части равенства содержится в типе правой части. Пусть $(y, z) \in \cup_{m\geq 0}(Y^m \times Z^m)$. Существует такое k, что $(y, z) \in Y^m \times Z^m$ для всех $m \geq k$. Тогда $y \in Y^m$ и $z \in Z^m$ для $m \geq k$. Далее, $y \in \cup_{m\geq 0}Y^m$ и $z \in \cup_{m\geq 0}Z^m$. И как следствие, $(y, z) \in (\cup_{m\geq 0}Y^m) \times (\cup_{m\geq 0}Z^m)$. Допустим теперь, что $(y, z) \in (\cup_{m\geq 0}Y^m) \times (\cup_{m\geq 0}Z^m)$, и докажем, что $(y, z) \in \cup_{m\geq 0}(Y^m \times Z^m)$. Из определения произведения $y \in \cup_{m\geq 0}Y^m$ и $z \in \cup_{m\geq 0}Z^m$. Существуют такие k1 и k2, что $y \in Y^m$ для $m \geq k1$ и $z \in Z^m$ для $m \geq k2$. Пусть $k = \max(k1, k2)$. Тогда $(y, z) \in Y^m \times Z^m$ для всех $m \geq k$. Далее, $(y, z) \in \cup_{m\geq 0}(Y^m \times Z^m)$. Непрерывность произведения доказана. Непрерывность объединения доказывается аналогичным образом.

Таким образом, если система определений типов $X = \varphi(X)$ построена с использованием только произведения и объединения, то оператор $\varphi$ является монотонным и непрерывным. В соответствии с теоремой Клини о неподвижной точке [4] решение системы $X = \varphi(X)$ есть $\cup_{m\geq 0}X^m$ и оно является наименьшей неподвижной точкой оператора $\varphi$.

Разумеется, другие способы конструирования типов, отличные от произведения и объединения, могут использоваться для определения типов. Однако типы, определяемые другими способами, должны определятся автономно от системы $X = \varphi(X)$. Например, тип Y, определяемый как Y = Subset(Z), не может участвовать в рекурсии, но может быть использован в качестве параметра определяемой системы рекурсивных типов. Тем самым мы запрещаем использование следующих экзотических форм рекурсии:

```
Y = Q(Subset(H(Y)))
Z = Q(Set(H(Z)))
I = Q(Array(H(I), ...))
E = Q(Array(..., H(E)))
A = Q(PRED(H(A), ...: ...)
B = Q(PRED(...: H(B), ...),
```

где H и Q — некоторые произвольные операторы. Эти формы, кроме двух последних, кажутся бесполезными и абсурдными. Тем не менее, некоторые из них (например, I = Q(Array(H(I), ...))) анализируются в книге [3].

## 6. БАЗИСНЫЕ ВЫЧИСЛИМЫЕ КОМПОЗИЦИИ

Ниже предлагается набор из шести базисных композиций. Далее в формулах, определяющих композиции, предикаты B и C являются параметрами определяемых композиций. По умолчанию, переменные x, y и z обозначают произвольные непустые наборы переменных, не пересекающиеся в рамках одной формулы.

### 6.1. Суперпозиция

Композиция вида “суперпозиция” определяется формулой:

$$A(x{:}\ y) \equiv \exists z\ (B(x{:}\ z)\ \&\ C(z{:}\ y)).$$

Здесь, x, y и z — произвольные непересекающиеся наборы переменных, причем набор x может быть пустым. Алгоритм вычисления композиции состоит в последовательном исполнении предикатов B и C. Значения переменных набора z, вычисленные при исполнении предиката B, используются при вычисления предиката C. Для композиции введем следующее обозначение:

$$A(x{:}\ y) \equiv B(x{:}\ z);\ C(z{:}\ y) \qquad (1)$$

Переменные набора z являются *локальными* переменными.

### 6.2. Параллельная композиция

Параллельная композиция определяется формулой:

$$A(x{:}\ y, z) \equiv B(x{:}\ y)\ \&\ C(x{:}\ z).$$

Набор x может быть пустым. Алгоритм вычисления композиции слагается из вычисления предикатов B и C. Поскольку эти два вычисления между собой не взаимодействуют, они могут проводиться параллельно. Параллельную композицию будем записывать в виде:

$$A(x{:}\ y, z) \equiv B(x{:}\ y)\ ||\ C(x{:}\ z) \qquad (2)$$

### 6.3. Альтерация

Композиция вида “альтерация” определяется формулой:

$$A(b, x{:}\ y) \equiv (b \Rightarrow B(x{:}\ y))\ \&\ (\neg b \Rightarrow C(x{:}\ y)).$$

Здесь b — переменная логического типа, не встречающаяся в наборах x и y. Набор x может быть пустым. Алгоритм вычисления композиции зависит от значения переменной b. Если значение b — истинно, вычисляется предикат B. В противном случае вычисляется предикат C. Композицию будем записывать в виде:

$$A(b, x: y) \equiv \textbf{if } b \textbf{ then } B(x: y) \textbf{ else } C(x: y) \textbf{ end} \qquad (3)$$

### 6.4. Порождение предиката

Композиция вида "порождение предиката" определяется формулой:

$$ConsPred(x, B: A) \equiv (\forall y)(\exists z)(A(y: z) \equiv B(x, y: z)).$$

Предикат B, являющийся параметром базисного предиката ConsPred, должен быть вычислимым. Наборы x и y могут быть пустыми. Пусть $x^{\sim}$ — список значений переменных набора x. Результатом исполнения композиции является создание определения предиката:

$$A_x(y: z) \equiv =(x^{\sim}: x); B(x, y: z)$$

Здесь $A_x$ — новое имя предиката, отличное от имен других предикатов. Переменной A предикатного типа присваивается имя $A_x$. Для композиции используется следующая форма записи:

$$ConsPred(x, B: A) \equiv A = \textbf{lambda}(y: z)\ B(x, y: z) \qquad (4)$$

### 6.5. Применение предиката

Композиция вида "применение предиката" определяется формулой:

$$RunPred(y, A: z) \equiv A(y: z) \qquad (5)$$

RunPred — базисный предикат. Набор y может быть пустым, A — переменная предикатного типа. Допустим, значением переменной A является имя предиката $A_x$. Исполнение композиции есть исполнение вызова $A_x(y: z)$.

Отметим, что в реализации композиции (4) не обязательно строить определение предиката $A_x$. Достаточно запомнить кортеж $(B, x^{\sim})$ как значение переменной A. А в реализации композиции (5) следует реализовать вызов B с параметрами $x^{\sim}$ и y.

### 6.6. Конструктор массива

Композиция вида "конструктор массива" определяется формулой:

$$ConsArray(x, G, T, L: a) \equiv (\forall i \in L)\ (\exists y \in T)\ G(i, x: y)\ \&\ a(i: y).$$

Набор x может быть пустым. Переменная a является массивом типа Array(T, L) с элементами типа T и индексами конечного типа L. Предикат G является параметром композиции ConsArray и должен быть вычислимым. Исполнение композиции реализуется следующим образом. В памяти создается массив как значение типа Array(T, L). Для типа L реализуется итерация элементов множества L. Для каждого значения индекса i вычисляется предикат G(i, x: y), и значение присваивается элементу массива a(i). Вычисление предиката G(i, x: y) для разных индексов i может проводиться независимо, т.е. параллельно. Когда вычисление будет завершено для всех индексов, созданный массив присваивается переменной a. Для композиции используется следующая форма записи:

$$\mathrm{ConsArray}(x, G{:}\ a) \equiv \textbf{forAll}\ i \in L\ \textbf{do}\ a[i] = G(i, x)\ \textbf{end} \qquad (6)$$

## 7. РЕКУРСИВНЫЕ ОПРЕДЕЛЕНИЯ ПРЕДИКАТОВ

*Программа* на языке исчисления вычислимых предикатов есть замкнутый набор определений предикатов вида (1) - (3). Имя определяемого предиката должно быть отлично от имен базисных предикатов, а также от имен типов и имен других определяемых предикатов.

Набор определений предикатов является *замкнутым*, если для всякого вхождения имени предиката в правой части некоторого определения это имя является именем базисного предиката либо предикат с этим именем определен в данном наборе определений.

Определение предиката *рекурсивно*, если определение прямо или косвенно, через совокупность определений, использует этот предикат. Рассмотрим замкнутую систему определений предикатов:

$$A_i(x_i{:}\ y_i) \equiv K_i(A_1, A_2,..., A_n);\ i = 1...n;\ n > 0,$$

где $x_i$ и $y_i$ — наборы переменных, $K_i$ — композиция вида (1) - (3), параметрами которой могут быть имена определяемых предикатов $A_1, A_2,..., A_n$ и имена базисных предикатов. Обозначим через $A = (A_1, A_2,..., A_n)$ — вектор имен предикатов и $K = (K_1, K_2,..., K_n)$ — вектор композиций. Систему определений предикатов запишем в векторной форме: $A = K(A)$. Таким образом, решением системы является неподвижная точка оператора K.

Введем отношение $\sqsubseteq$ на векторах предикатов:

$$B \sqsubseteq C \cong \forall i=1..n\ \forall x_i\ \forall y_i\ (B_i(x_i{:}\ y_i) \Rightarrow C_i(x_i{:}\ y_i)).$$

Отношение $\Rightarrow$ на множестве предикатов с наборами переменных $x_i$ и $y_i$ является отношением *частичного порядка*, т.е. является рефлексивным, антисимметричным и транзитивным. Как следствие, отношение $\sqsubseteq$ также является отношением частичного порядка.

Определим вектор $\Phi = (F_1, F_2, ..., F_n)$, где $F_1, F_2, ..., F_n$ — тотально ложные предикаты, т.е. $\forall i=1..n\ \forall x_i\ \forall y_i\ (\neg F_i(x_i: y_i))$. Вектор $\Phi$ является минимальным элементом, т.е. выполняется: $\forall B\ (\Phi \sqsubseteq B)$.

Последовательность предикатов $\{C^m\}_{m\geq 0}$ является *возрастающей цепью*, если $C^0 \Rightarrow C^1 \Rightarrow ... \Rightarrow C^m \Rightarrow ...$ . Аналогично, последовательность векторов предикатов $\{B^m\}_{m\geq 0}$ является *возрастающей цепью*, если $B^0 \sqsubseteq B^1 \sqsubseteq ... \sqsubseteq B^m \sqsubseteq ...$ .

*Верхней гранью* подмножества $Q$ предикатов является предикат $p$ такой, что $q \Rightarrow p$ для всех $q \in Q$. Предикат $p$ является *наименьшей верхней гранью*, если $p \Rightarrow p'$ для любой другой верхней грани $p'$ подмножества $Q$. Для наименьшей верхней грани цепи предикатов $\{C^m\}_{m\geq 0}$ будем использовать обозначение $\cup_{m\geq 0} C^m$. Аналогичным образом определяется понятие наименьшей верхней грани для подмножества векторов предикатов. Для наименьшей верхней грани цепи векторов предикатов $\{B^m\}_{m\geq 0}$ используется обозначение $\cup_{m\geq 0} B^m$.

**Лемма**. Пусть $\{B^m\}_{m\geq 0}$ — возрастающая цепь векторов предикатов. Тогда:

$$\cup_{m\geq 0} B^m = (\cup_{m\geq 0} B_1^m, ..., \cup_{m\geq 0} B_n^m).$$

**Лемма**. Множество векторов предикатов с отношением $\sqsubseteq$ является полной решеткой.

График предиката $B_i(x_i: y_i)$ есть $\text{график}(B_i) = \{(x_i, y_i) \mid B_i(x_i: y_i)\}$. Реализуется следующее тождество:

$$B_i(x_i: y_i) \Rightarrow C_i(x_i: y_i) \equiv \text{график}(B_i) \subseteq \text{график}(C_i).$$

Таким образом, отношение $\Rightarrow$ эквивалентно отношению $\subseteq$ на множестве подмножеств произведения типов переменных, входящих в наборы $x_i$ и $y_i$. Множество подмножеств с отношением $\subseteq$ является полной решеткой [6, 7]. Поэтому множество предикатов с отношением $\Rightarrow$ также является полной решеткой. Далее это свойство распространяется на множество векторов предикатов.

Рассмотрим последовательность векторов предикатов $\{A^m\}_{m\geq 0}$, определяемую следующим образом:

$$A^0 = \Phi,\ A^{m+1} = K(A^m),\ m \geq 0.$$

Естественно ожидать, что предел последовательности $\{A^m\}_{m\geq 0}$ (если он существует) даст нам неподвижную точку — решение системы A = K(A). Построение неподвижной точки рассмотрим на примере следующей программы для вычисления факториала натурального числа n:

Factorial(n: f) ≡ **if** n = 0 **then** f = 1 **else** f = n ∗ Factorial(n - 1) **end**

Запишем эту программу на языке исчисления вычислимых предикатов.

A0(n: f) ≡ ConsIntZero( : c0); A1(n, c0: f) (7)
A1(n, c0: f) ≡ ConsIntOne( : c1); A2(n, c0, c1: f)
A2(n, c0, c1: f) ≡ =(n, c0: b); A3(n, c1, b: f)
A3(n, c1, b: f) ≡ **if** b **then** =(c1: f) **else** A4(n, c1: f) **end**
A4(n, c1: f) ≡ -(n, c1: n1); A5(n, n1: f)
A5(n, n1: f) ≡ A0(n1: f1); ∗(n, f1: f)

Вектор предикатов данной программы A = (A0, A1, A2, A3, A4, A5). Проследим схему построения неподвижной точки. Вначале $A^0 = \Phi$, т.е. каждый из шести предикатов устанавливается ложным (пустым), не заданным ни на одном наборе аргументов. Подставляя эти значения предикатов в правые части определений, получим значения предикатов на первом шаге, остающиеся пустыми, кроме $A3^1$, которому присваивается альтернатива "**then**" условного оператора. Анализируя предыдущие определения, получим значение предиката $A3^1$**:** 0, 1, **true** → 1. На втором шаге изменится только значение предиката A2, на третьем — A1. В итоге получаем следующую цепочку модификаций, в которой на каждом шаге мы указываем лишь изменившиеся компоненты вектора A:

$A3^1$**:** 0, 1, **true** → 1
$A2^2$**:** 0, 0, 1 → 1
$A1^3$**:** 0, 0 → 1
$A0^4$**:** 0 → 1
$A5^5$**:** 1, 0 → 1 — параметр n1 = 0 как результат подстановки $A0^4$, n = 1 устанавливается анализом правой части A4;
$A4^6$**:** 1, 1 → 1
$A3^7$**:** 0, 1, **true** → 1; $A3^7$**:** 1, 1, **false** → 1;
$A2^8$**:** 0, 0, 1 → 1; $A2^8$**:** 1, 0, 1 → 1; и т.д.

Оператор K является *монотонным*, если истинно:

$$\forall B, C\ (B \sqsubseteq C \Rightarrow K(B) \sqsubseteq K(C)).$$

Монотонность оператора K реализуется, если имеет место монотонность композиций, используемых в правых частях определений системы A = K(A). Поэтому достаточно установить монотонность базисных композиций.

**Лемма**. Суперпозиция (1), параллельная композиция (2) и альтерация (3) монотонны относительно вхождений предикатов B и C.

Рассмотрим суперпозицию (1). В соответствии с определением суперпозиция обозначает формулу: $\exists z\,(B(x{:}\ z)\ \&\ C(z{:}\ y))$. Пусть $B_1$, $C_1$, $B_2$ и $C_2$ — произвольные предикаты, удовлетворяющие отношениям $B_1 \Rightarrow B_2$ и $C_1 \Rightarrow C_2$, т.е.:

$$\forall x,z.\ B_1(x{:}\ z) \Rightarrow B_2(x{:}\ z) \text{ и } \forall z,y.\ C_1(z{:}\ y) \Rightarrow C_2(z{:}\ y). \qquad (8)$$

Необходимо доказать, что:

$$\exists z\,(B_1(x{:}\ z)\ \&\ C_1(z{:}\ y)) \Rightarrow \exists z\,(B_2(x{:}\ z)\ \&\ C_2(z{:}\ y)). \qquad (9)$$

Допустим, истинна посылка: $\exists z\,(B_1(x{:}\ z)\ \&\ C_1(z{:}\ y))$. Тогда для некоторого набора $z_0$ истинно: $B_1(x{:}\ z_0)\ \&\ C_1(z_0{:}\ y)$. Из условий (8) следует истинность $B_2(x{:}\ z_0)$ и $C_2(z_0{:}\ y)$. Следовательно, истинно $\exists z\,(B_2(x{:}\ z)\ \&\ C_2(z{:}\ y))$ и соотношение (9). Доказательство монотонности альтерации и параллельной композиции проводится аналогично.

Справедливо более общее утверждение: операции конъюнкции и дизъюнкции монотонны относительно своих аргументов. Однако операция отрицания $\neg B$ не коммутативна относительно B. По этой причине альтерация **if** b **then** B(x: y) **else** C(x: y) **end** не коммутативна относительно вхождения b.

Далее будем считать, что оператор K является монотонным. Это накладывает ограничения на формы рекурсии системы $A = K(A)$. В частности, недопустима рекурсия, обусловленная вхождением b в альтерации: если b является результатом некоторого предиката в составе системы $A = K(A)$, то определение этого предиката можно вынести из системы $A = K(A)$, т.е. провести автономно. В приведенном выше примере программы факториала вхождению b соответствует нерекурсивный предикат $n = 0$, определение которого можно провести автономно от системы определений (7).

**Лемма**. Пусть $\{C^m(x{:}\ y)\}_{m\geq 0}$ является возрастающей цепью предикатов и $C^{\sim} = \cup_{m\geq 0} C^m$. Тогда $\forall x\ \forall y.\ C^{\sim}(x{:}\ y) \Rightarrow \exists k\ C^k(x{:}\ y)$.

Доказательство от противного. Допустим, что для некоторых $x^0$ и $y^0$ истинно $C^{\sim}(x^0{:}\ y^0)$, однако для всех k ложно $C^k(x^0{:}\ y^0)$. Определим предикат $C^{\#}(x{:}\ y)$, совпадающий с $C^{\sim}(x{:}\ y)$ всюду, за исключением набора $(x^0, y^0)$, на котором он ложный. Вектор $C^{\#}$ является верхней гранью последовательности $\{C^m\}_{m\geq 0}$, причем $C^{\#} \sqsubseteq C^{\sim}$ и $C^{\#} \neq C^{\sim}$, а тогда грань $C^{\sim}$ не является минимальной, что приводит к противоречию.

Оператор K является *непрерывным*, если для любой возрастающей цепи векторов предикатов $\{B^m\}_{m\geq 0}$ имеет место равенство: $K(\cup_{m\geq 0} B^m) = \cup_{m\geq 0} K(B^m)$.

Аналогичным образом определяется понятие непрерывности композиции предикатов.

**Лемма**. Оператор K, составленный из непрерывных композиций $K_i$ (i = 1...n), является непрерывным. Действительно, если $\{B^m\}_{m\geq 0}$ — возрастающая цепь векторов предикатов, то имеет место цепочка равенств:

$$\cup_{m\geq 0}K(B^m) = \cup_{m\geq 0}(K_1(B_1^m), ..., K_n(B_n^m)) = (\cup_{m\geq 0}K_1(B_1^m), ..., \cup_{m\geq 0}K_n(B_n^m)) =$$
$$= (K_1(\cup_{m\geq 0}B_1^m), ..., K_n(\cup_{m\geq 0}B_n^m)) = K(\cup_{m\geq 0}B^m).$$

**Лемма**. Суперпозиция (1), параллельная композиция (2) и альтерация (3) непрерывны относительно вхождений предикатов B и C.

Рассмотрим суперпозицию (1), определяемую формулой: $\exists z\,(B(x{:}\ z)\ \&\ C(z{:}\ y))$. Пусть $\{B^m\}_{m\geq 0}$ и $\{C^m\}_{m\geq 0}$ — возрастающие цепи предикатов. Необходимо доказать, что:

$$\cup_{m\geq 0}\exists z\,(B^m(x{:}\ z)\ \&\ C^m(z{:}\ y)) \equiv \exists z\,(\cup_{m\geq 0}B^m(x{:}\ z)\ \&\ \cup_{m\geq 0}C^m(z{:}\ y)). \qquad (10)$$

Допустим, левая часть формулы (10) истинна для некоторых x и y. Существует такое k, что формула $\exists z\,(B^m(x{:}\ z)\ \&\ C^m(z{:}\ y))$ истинна для всех $m \geq k$. Тогда для $m \geq k$ и некоторого $z_0$ истинны $B^m(x{:}\ z_0)$ и $C^m(z_0{:}\ y)$. Далее, очевидно, будет истинной правая часть формулы (10).

Допустим теперь, что для некоторых x и y истинна правая часть формулы (10), и докажем истинность левой части. Для некоторого набора $z_0$ истинны $\cup_{m\geq 0}B^m(x{:}\ z_0)$ и $\cup_{m\geq 0}C^m(z_0{:}\ y)$. Далее, существует такое k1, что $B^m(x{:}\ z_0)$ истинно для всех $m \geq k1$. Аналогично, существует такое k2, что $C^m(z_0{:}\ y)$ истинно для всех $m \geq k2$. Пусть $k = \max(k1, k2)$. Формула $B^m(x{:}\ z_0)\ \&\ C^m(z_0{:}\ y)$ истинна для всех $m \geq k$, из чего следует истинность левой части формулы (10).

Доказательство непрерывности альтерации и параллельной композиции проводится аналогично.

Напомним, что последовательность векторов предикатов $\{A^m\}_{m\geq 0}$ определяется следующим образом:

$$A^0 = \Phi,\ A^{m+1} = K(A^m),\ m \geq 0.$$

**Лемма**. Последовательность $\{A^m\}_{m\geq 0}$ является возрастающей цепью. Доказательство использует монотонность оператора K и проводится по индукции.

Таким образом, если система определений предикатов A = K(A) построена с использованием суперпозиции, альтерации и параллельной ком-

позиции, то оператор K является монотонным и непрерывным. В соответствии с теоремой Клини о неподвижной точке [4, 6, 11] решение системы A = K(A) есть $\cup_{m\geq 0}A^m$, и оно является наименьшей неподвижной точкой оператора K.

Разумеется, базисные композиции (4)–(6) могут использоваться в программе. Однако предикаты, использующие композиции (4)–(6), должны определятся автономно от системы A = K(A). Тем самым мы запрещаем использование следующих трех экзотических форм рекурсии:

C ≡ H(**if** Q(C) **then ... else ... end**)
B ≡ H(**lambda**(y**:** z) Q(B(x, y**:** z)))
G = H(**forAll** i∈L **do** a[i] = Q(G(i, x)) **end**),

где H и Q — некоторые произвольные операторы. Допустимость и полезность этих форм не исследуется в данной статье.

## ЗАКЛЮЧЕНИЕ

Исчисление вычислимых предикатов является математической моделью определенного в [1] понятия автоматической вычислимости для класса программ с предикатной спецификацией. Исчисление определяет набор базисных вычислимых предикатов и шесть базисных вычислимых композиций: суперпозицию, альтерацию, параллельную композицию, порождение предиката, применение предиката и конструктор массива.

Набор базисных вычислимых предикатов определяет систему типов данных. Предлагаемая в настоящей статье система типов покрывает системы типов многих известных языков программирования. В том числе, допускаются часто используемые рекурсивные типы — последовательности и деревья, однако запрещаются типы с экзотическими видами рекурсии, рассматриваемыми в λ-исчислении и теории доменов [3, 5]. Отметим, что система типов данных может быть построена по другим принципам [8, 9]. Поэтому следует рассматривать семейство исчислений вычислимых предикатов с различными наборами базисных вычислимых предикатов.

Исчисление вычислимых предикатов является минимальным базисом для построения произвольного (чистого) языка функционального программирования. Конструкции языка определяются посредством иерархической системы обозначений через конструкции исчисления вычислимых предикатов. Таким способом построен язык предикатного программирования [10]. Отметим, что исчисление вычислимых предикатов не может быть базисом

для языков логического программирования, где вычисление реализуется с применением логического вывода и переборных механизмов.

Остается вопрос, каким образом исчисление вычислимых предикатов соотносится с классом императивных программ. Сформулируем следующий тезис. Для всякой императивной программы с предикатной спецификацией существует эквивалентная ей программа на некотором функциональном языке программирования, причем императивная программа может быть получена из функциональной применением некоторой последовательности оптимизаций.

## СПИСОК ЛИТЕРАТУРЫ


1. Шелехов В.И. Анализ общего понятия программы // Методы предикатного программирования. — Новосибирск, 2006. — Вып. 2. — С. 7–16.
2. Хоар К. О структурной организации данных // Структурное программирование. — М.: Мир, 1975. — С.98–197.
3. Tennent R.D. Semantics of Programming Languages. — Prentice Hall, 1991. — 236 p.
4. Tarski A. A Lattice-Theoretical Fixpoint Theorem and Its Applications // Pacific J. Math. — 1955. — Vol. 5. — P.285–309.
5. Abramsky S., Jung A. Domain theory // Handbook of Logic in Computer Science / Ed. by S. Abramsky, D. M. Gabbay, T. S. E. Maibaum. — Oxford University Press, 1994.— Vol. III.
6. Крицкий С.П. Трансляция языков программирования: синтаксис, семантика // Ростов-на-Дону: РГУ, 2005. — http://public.uic.rsu.ru/~skritski/scourses/Transl/Langs1.htm.
7. Burris S. N., Sankappanavar H. P. A Course in Universal Algebra. — Springer-Verlag, 1981. — 315 p.
8. Бажанов С.Е., Кутепов В.П., Шестаков Д.А. Язык функционального параллельного программирования и его реализация на кластерных системах // Программирование. — 2005. — № 5. — С. 18–51.
9. E.C.R.Hehner. A Practical Theory of Programming, second edition. — 2004. — http://www.cs.toronto.edu/~hehner/aPToP/
10. Шелехов В.И. Введение в предикатное программирование. — Новосибирск, 2002. — 82 с. — (Препр. / ИСИ СО РАН; № 100).
11. Лавров С.С. Программирование. Математические основы, средства, теория. — СПб: БХВ-Петербург, 2001. — 320 с.

**В.И. Шелехов**

# ИСЧИСЛЕНИЕ ВЫЧИСЛИМЫХ ПРЕДИКАТОВ

**Препринт**
**143**

Рукопись поступила в редакцию 22.02.07
Рецензент И.С. Ануреев
Редактор З. В. Скок

---

Подписано в печать 24.04.07
Формат бумаги 60 × 84 1/16 Объем 1.4 уч.-изд.л., 1.5 п.л.
Тираж 90 экз.

---

Центр оперативной печати «Оригинал 2»
г.Бердск, ул. Островского, 55, оф. 02, тел. (383) 214-45-35