Знания-Онтологии-Теории (ЗОНТ-15)

Семантика языка предикатного программирования

Шелехов В.И.

Институт Систем Информатики *СО РАН, пр. Лаврентьева, д. 6, г. Новосибирск, 630090, Россия.*

vshel@iis.nsk.su

Аннотация. Формализация описания языка программирования в виде онтологии является поверхностной и не оказывает заметного влияния на индустрию разработки языковых процессоров. Существенно сложнее формализация семантики исполнения языка программирования, т.е. формализация *операционной семантики*, лежащая в основе программного синтеза и дедуктивной верификации.

Истинный предикат $\beta(x, y)$ является *вычислимым* относительно x, если существует алгоритм, вычисляющий соответствующее значение y по записи предиката на языке исчисления предикатов. Разработана формальная операционная семантика вычислимого подмножества языка исчисления предикатов, используемого в качестве ядра при построении языка предикатного программирования. Рекурсивные предикаты определены через аппарат неподвижной точки. Определение операционной семантики проведено в рамках логики первого порядка.

Ключевые слова: формальная операционная семантика, дедуктивная верификация, программный синтез, тотальная корректность программы, онтология

1 Введение

Имеется много различных исследований по использованию языка логики в качестве языка программирования. Большая их часть связана с логическим программированием. В этом русле также значительное число работ по семантическому программированию [12]. Разработан ряд инструментов, генерирующих программы по логическим формулам, доказанным в системах компьютерного автоматического доказательства таких, как PVS [10] и Coq.

В данной работе вычислимая логическая формула на языке исчисления предикатов рассматривается как программа, которая может быть исполнена в стиле функционального или императивного программирования. Поскольку язык исчисления предикатов неудобен для проведения вычислений, вводится эквивалентный язык программирования P_0, используемый как минимальный полный алгоритмический базис в целях построения языка предикатного программирования P [8]. Для языка P_0 строится формальная операционная семантика, необходимая при разработке методов дедуктивной верификации и программного синтеза. Доказывается эквивалентность языка P_0 и соответствующего ему вычислимого подмножества языка исчисления предикатов.

2 Корректность программ-функций

Предикатное программирование ограничено классом *программ-функций*. Это простейший класс в системе *классификации программ*. Классификация ориентирована на разработку адекватной технологии программирования для каждого класса программ. Например, принципиально различаются классы программ для задач вычислительной математики и систем реального времени.

Соответственно, для этих классов следует разрабатывать разные методы и инструменты программирования.

Обычно рассматриваются классификации по назначению программ. Здесь же определяется классификация программ по их внутренней организации. Генеральная классификация определяет два класса программ: *программы-функции* и *реактивные системы*. Данные два класса составляют более 90% всех программ. Классификация реактивных систем, соответствующих автоматному программированию, рассматривается в работе [1]. Класс реактивных систем сложнее класса программ-функций в частности потому, что программа реактивной системы строится из частей, соответствующих программам-функциям.

Имеются другие, более сложные классы, например, языковые процессоры и операционные системы; они находятся на метауровне по отношению к первым двум классам. Языковые процессоры – это интерпретаторы программ, трансляторы, оптимизаторы, трансформаторы и т.д. Принципиально, что транслятор с одного языка на другой нельзя определить в виде программы-функции. Приведенная классификация не покрывает всего спектра программ и различных особенностей, например такой, как тесная интеграция программы с данными.

Программа принадлежит *классу программ-функций*, если она не взаимодействует с внешним окружением программы; точнее, если возможно перестроить программу таким образом, чтобы все операторы ввода данных находились в начале программы, а весь вывод собран в конце программы. Если подобная перестройка программы принципиально невозможна, ее следует определять в виде реактивной системы. Программа-функция должна всегда нормально завершаться с получением результата, поскольку бесконечно работающая и невзаимодействующая программа бесполезна. Программа определяет функцию, вычисляющую по набору входных данных (аргументов) некоторый набор результатов. Класс программ-функций, по меньшей мере, содержит программы для задач дискретной и вычислительной математики.

Программу-функцию U с набором аргументов x и набором результатов y будем записывать в виде $U(x\mathbf{:}\ y)$. *Спецификацией* программы-функции являются два предиката: *предусловие* $P(x)$ и *постусловие* $Q(x, y)$. Спецификацию программы будем записывать в виде $[P(x), Q(x, y)]$. Программу со спецификацией обычно принято записывать в виде тройки Хоара: $P(x)\ \{U(x\mathbf{:}\ y)\}\ Q(x, y)$.

Известные понятия тотальности и однозначности функции $f\colon x \to y$, где x и y – непересекающиеся наборы переменных, естественным образом переносятся на предикат $H(x, y)$, рассматриваемый как функция из x в y. Предикат $H(x, y)$ является *однозначным* в области X для набора переменных x, если истинно:

$$\forall x{\in}X\ \forall y_1,y_2.\ H(x, y_1)\ \&\ H(x, y_2) \Rightarrow y_1 = y_2\ .$$

Предикат $H(x, y)$ является *тотальным* в области X для набора переменных x, если:

$$\forall x{\in}X\ \exists y.\ H(x, y) \qquad .$$

Далее потребуется следующая лемма.

Лемма 1. Допустим, предикат $D(x, y)$ является однозначным в области X, а предикат $Z(x, y)$ – тотальный в области X. Пусть истинна формула $\forall x{\in}X.\ Z(x, y) \Rightarrow D(x, y)$. Тогда истинна следующая формула: $\forall x{\in}X.\ D(x, y) \Rightarrow Z(x, y)$. Как следствие, предикаты D и Z оказываются тождественными в области X.

Однозначность и тотальность спецификации $[P(x), Q(x, y)]$ определяются, соответственно, формулами:

$$P(x)\ \&\ Q(x, y_1)\ \&\ Q(x, y_2) \Rightarrow y_1 = y_2.$$
$$P(x) \Rightarrow \exists y.\ Q(x, y);$$

Программа должна соответствовать спецификации. Это требование формулируется в виде условия *частичной корректности*: если перед исполнением программы $U(x\mathbf{:}\ y)$ истинно предусловие $P(x)$, то в случае завершения программы должно быть истинно постусловие $Q(x, y)$ в момент ее завершения. Обязательным является также *условие завершения программы*: если перед исполнением программы $U(x\mathbf{:}\ y)$ истинно предусловие $P(x)$, то программа обязана завершаться. Объединение условий частичной корректности и завершения программы определяет условие *тотальной корректности*.

Адекватная формализация условий корректности программы возможна лишь при использовании формальной операционной семантики языка программирования. *Операционную семантику* программы $U(x: y)$ определим в виде предиката:

$\mathcal{R}(U)(x, y) \cong$ для значения набора x исполнение программы U всегда завершается и существует исполнение программы, при котором результатом вычисления является значение набора y.

Данное определение исключает ситуацию, когда для некоторого значения набора x существуют два разных исполнения программы, одно из которых завершается, а другое – не завершается. В этом случае $\mathcal{R}(U)(x, y)$ будет ложным для любых y.

Однозначность и тотальность программы $U(x: y)$ для некоторого значения x определяются, соответственно, формулами:

$$\mathcal{R}(U)(x: y_1) \,\&\, \mathcal{R}(U)(x: y_2) \Rightarrow y_1 = y_2;$$
$$\exists y.\ \mathcal{R}(U)(x, y)\ .$$

Отметим, что тотальность программы для некоторого значения x есть в точности условие завершения программы для этого значения x. Программа называется *однозначной*, если она однозначна для всех значений аргументов. Далее ограничимся рассмотрением лишь однозначных программ.

Операционная семантика $\mathcal{R}(U)$ является эквивалентом программы U. Доказательство некоторого свойства программы $W(x, y)$ реализуется доказательством истинности формулы: $\mathcal{R}(U)(x, y) \Rightarrow W(x, y)$. Кроме того, эту формулу достаточно доказать при истинном предусловии. С учетом этого, условие частичной корректности программы $U(x: y)$ записывается в виде формулы:

$$\forall x,y.\ P(x) \,\&\, \mathcal{R}(U)(x, y) \Rightarrow Q(x, y)\ .$$

Условие завершения программы $U(x: y)$ при истинном предусловии представляется формулой:

$$\forall x.\ P(x) \Rightarrow \exists y.\ \mathcal{R}(U)(x, y)\ .$$

Формула для условия тотальной корректности программы получается объединением двух формул:

$$\forall x.\ P(x) \Rightarrow [\forall y.\ \mathcal{R}(U)(x, y) \Rightarrow Q(x, y)\ ] \,\&\, \exists y.\ \mathcal{R}(U)(x, y)\ . \qquad (1)$$

Лемма 2. Если программа $U(x: y)$ тотально корректна относительно спецификации $[P(x), Q(x, y)]$, то спецификация тотальна.

Теорема 1 тождества спецификации и программы. Рассмотрим программу $U(x: y)$ со спецификацией $[P(x), Q(x, y)]$. Допустим, программа является однозначной, а спецификация – тотальной. Предположим, программа выводима из спецификации, т. е.

$$P(x) \,\&\, Q(x, y) \Rightarrow \mathcal{R}(U)(x, y)\ . \qquad (2)$$

Тогда программа $U(x: y)$ является тотально корректной относительно спецификации.

Лемма 3. В условиях теоремы 1 истинна следующая формула:

$$P(x) \Rightarrow (\mathcal{R}(U)(x, y) \equiv Q(x, y))\ .$$

Следствие Леммы 3: в условиях Теоремы 1 спецификация $[P(x), Q(x, y)]$ является однозначной.

Лемма 4. Допустим, программа $U(x: y)$ является тотально корректной, а спецификация $[P(x), Q(x, y)]$ – однозначной. Тогда истинна формула (2).

Формула (2) может быть использована для доказательства тотальной корректности программ-функций. Однако она применима только для программ с однозначной спецификацией. В случае неоднозначной спецификации формула (2) становится недоказуемой, что вытекает из следствия Леммы 3.

Приведенные выше леммы и теорема доказаны в системе автоматического доказательства PVS: http://www.iis.nsk.su/persons/vshel/files/rules.zip.

3 Язык вычислимых предикатов P_0 и его операционная семантика

3.1 Вычислимые предикаты

Истинный предикат H(x, y) является *вычислимым* относительно x, если существует интерпретатор, вычисляющий соответствующее значение y по записи предиката на языке исчисления предикатов. Операционная семантика вычислимого подмножества исчисления предикатов определяется тождеством $\mathcal{R}(H) = H$. Программа интерпретатора будет сложна, поскольку язык исчисления предикатов не приспособлен для исполнения: любое вычисление сопряжено со сложным распознаванием внутри текста предиката H. Вполне естественно для целей вычисления использовать другой, более удобный, язык, так чтобы выполнение соответствующей программы U(x**:** y) на этом языке было эквивалентно исполнению предиката H(x, y) на языке исчисления предикатов, что гарантирует $\mathcal{R}(U) = H$.

Вычислимый предикат H(x, y) будем рассматривать вместе с эквивалентной программой U(x**:** y), называемой *предикатной программой*. Далее для предиката H(x, y) и предикатной программы U(x**:** y) будем использовать одно обозначение H(x**:** y).

Наша цель – определить минимальный полный *базис предикатных программ* в виде языка P_0, построить для него операционную семантику и далее расширить этот базис до удобного языка программирования P.

3.2 Язык P_0

Произвольная предикатная программа определяется следующей конструкцией:

<имя предиката>(<аргументы>**:** <результаты>) { <оператор> }

Аргументы и результаты – разные непересекающиеся наборы имен переменных. Набор аргументов может быть пустым.

Простейшим оператором, используемым в составе других операторов, является *вызов предиката* B(x**:** y), где B – имя предиката. Допускается также вызов предиката вида A(x**:** y), где A – имя переменной *предикатного типа*. Значение переменной A есть имя предиката.

Предположим, что x, y и z обозначают разные непересекающиеся наборы переменных. Набор x может быть пустым, наборы y и z не пусты. В составе набора переменных x может использоваться логическая переменная e со значениями **true** и **false**. Пусть B и C – имена предикатов, A и D – имена переменных предикатного типа. Операторами являются: *оператор суперпозиции* B(x**:** z); C(z**:** y), *параллельный оператор* B(x**:** y) || C(x**:** z), *условный оператор* **if (**e) B(x**:** y) **else** C(x**:** y), *вызов предиката* и *оператор каррирования*.

Ниже в таблице 1 представлен полный базис вычислимых предикатов и соответствующих им операторов:

Таблица 1. Вычислимые предикаты и их программная форма.

H(x**:** y) ≅ ∃z. B(x**:** z) & C(z**:** y)	H(x**:** y) { B(x**:** z); C(z**:** y) }
H(x**:** y, z) ≅ B(x**:** y) & C(x**:** z)	H(x**:** y, z) { B(x**:** y) || C(x**:** z) }
H(x**:** y) ≅ (e ⇒ B(x**:** y)) & (¬e ⇒ C(x**:** y))	H(x**:** y) { **if** (e) B(x**:** y) **else** C(x**:** y) }
H(x**:** y) ≅ B(x˜**:** y)	H(x**:** y) { B(x˜**:** y) }
H(A, x**:** y) ≅ A(x**:** y)	H(A, x**:** y) { A(x**:** y) }
H(x**:** D) ≅ ∀y,z. D(y**:** z) ≡ B(x, y**:** z)	H(x**:** D) { D(y**:** z) {B(x, y**:** z)} }
H(A, x**:** D) ≅ ∀y,z. D(y**:** z) ≡ A(x, y**:** z)	H(A, x**:** D) { D(y**:** z) {A(x, y**:** z)} }

Набор x˜ составлен из набора переменных x с возможным добавлением имен предикатных программ.

Имеется два вида оператора каррирования: D(y: z) {B(x, y: z)} и D(y: z) {A(x, y: z)}. Результатом исполнения оператора каррирования является новый предикат D, получаемый фиксацией значения набора x.

Полная программа состоит из конечного набора предикатных программ. Исполнение полной программы начинается с некоторой предикатной программы, называемой *главной*. Любое имя предиката B, встречающееся в операторной части любой предикатной программы, должно быть определено одним из следующих способов:

- в составе полной программы имеется предикатная программа с именем B;
- предикат B является *базисным*;
- предикат B является *параметром* полной программы и определяется в другой полной программе.

Базисный предикат определяет одну из элементарных операций над числами или логическими значениями. Для базисного предиката нет предикатной программы – он считается предопределенным в языке P_0. Примеры базисных предикатов: +(a, b: c), <(a, b: e), =(a, b: e), где e – логическая переменная. Они соответствуют следующим отношениям: c = a + b, e = (a< b), e = (a = b). Базисный предикат =(a: b) соответствует отношению b = a. Базисный предикат является эквивалентом соответствующего оператора присваивания. Набор базисных предикатов языка P_0 может быть произвольным. Однако в данной работе мы ограничиваемся рассмотрением лишь однозначных базисных предикатов.

Для сопоставления с языками функционального программирования представим эквивалентную *функциональную форму* операторов. Оператор вызова предиката B(x: y) эквивалентен оператору присваивания y = B(x), где B(x) предстает как соответствующий вызов функции. Оператор суперпозиции B(x: z); C(z: y) может быть записан в виде y = C(B(x)). Параллельному оператору B(x: y) || C(x: z) соответствует пара соседних аргументов в составе некоторого другого вызова функции E(...B(x), C(x)...). Оператор каррирования D(y: z) {B(x, y: z)} может быть записан в виде оператора присваивания D = B(x), где B(x) – каррированая форма вызова функции.

3.3 Операционная семантика операторов

Наша цель – построить формальную операционную семантику языка P_0. При этом язык P_0 рассматривается независимо от его логической интерпретации, представленной в левой колонке Таблицы 1. Для формальной операционной семантики произвольного предиката H(x: y) используется обозначение $\mathcal{R}$(H).

Пусть A – переменная предикатного типа. В операционной семантике переменную A будем представлять структурой (A.name, A.pref) из двух полей: name – имя предиката и pref – *префикс каррирования*, определяющий набор значений, фиксированных применением оператора каррирования.

При подстановке имени программы C в качестве аргумента некоторого вызова предиката B(x˜: y) этот аргумент представляется структурным значением (C, ( )), где ( ) – обозначает пустой набор значений в качестве префикса каррирования.

Для произвольной предикатной программы H(x: y) { <оператор> } реализуется тождество $\mathcal{R}$(H)(x, y) ≡ ||<оператор>||, где ||<оператор>|| обозначает операционную семантику <оператора>. Определим операционную семантику различных видов операторов из Таблицы 1.

|| B(x: z); C(z: y) || ≅ ∃z. || B(x: z) || & || C(z: y) ||
|| B(x: y) || C(x: z) || ≅ || B(x: y) || & || C(x: z) ||
|| **if** (e) B(x: y) **else** C(x: y) || ≅ (e ⇒ || B(x: y) ||) & (¬e ⇒ || C(x: y) ||)
|| B(x˜: y) || ≅ $\mathcal{R}$(B)(||x˜||, y)
|| A(x: y) || ≅ || A.name(A.pref, x, y) ||
|| D(y: z) {B(x, y: z)} || ≅ D = (B, (x))
|| D(y: z) {A(x, y: z)} || ≅ D = (A.name, (A.pref, x)) .

Здесь (A.pref, x) обозначает новый префикс каррирования, полученный добавлением фиксированных значений набора x к префиксу A.pref; $\|x^{\sim}\|$ обозначает замену любого вхождения в $x^{\sim}$ имени предиката C на (C, ( )).

Поскольку для произвольного вызова предиката $\| B(x{:}\ y) \| \cong \mathcal{R}(B)(x, y)$, операционные семантики части операторов преобразуются к следующему виду:

$\| B(x{:}\ z);\ C(z{:}\ y) \| \equiv \exists z.\ \mathcal{R}(B)(x, z)\ \&\ \mathcal{R}(C)(z, y)$
$\| B(x{:}\ y)\ \|\ C(x{:}\ z) \| \equiv \mathcal{R}(B)(x, y)\ \&\ \mathcal{R}(C)(x, z)$
$\|$ **if** (e) B(x**:** y) **else** C(x**:** y) $\| \equiv (e \Rightarrow \mathcal{R}(B)(x, y))\ \&\ (\neg e \Rightarrow \mathcal{R}(C)(x, y))$
$\| A(x{:}\ y) \| \cong \mathcal{R}(A.name)(A.pref, x, y)$

Допустим, полная программа состоит из набора предикатных программ для предикатов $H_1, H_2, \ldots, H_k$. Рассмотрим систему логических тождеств:

$$\mathcal{R}(H_j)(x_j, y_j) \equiv \|<\text{оператор}>\|,\ j = 1,\ldots,k. \qquad (3)$$

Операционная семантика операторов в правой части тождеств выражается через $\mathcal{R}(H_1), \mathcal{R}(H_2), \ldots, \mathcal{R}(H_k)$, а также через операционные семантики базисных предикатов и предикатов, являющихся параметрами полной программы.

3.4 Некоторые сведения из теории решеток [2 – 4]

Обозначение $(D, \sqsubseteq)$ определяет непустое множество D с отношением порядка $\sqsubseteq$. $(D, \sqsubseteq)$ – *частично упорядоченное множество*, если $\sqsubseteq$ рефлексивно: $\forall a \in D.\ a \sqsubseteq a$, антисимметрично: $a \sqsubseteq b\ \&\ b \sqsubseteq a \Rightarrow a = b$ и транзитивно: $a \sqsubseteq b\ \&\ b \sqsubseteq c \Rightarrow a \sqsubseteq c$. Для произвольного подмножества $S \subseteq D$ *наименьшим* (*наибольшим*) элементом S называется $a \in S$, такой что $a \sqsubseteq x$ $(x \sqsubseteq a)$ для всех $x \in S$. *Верхней гранью* S называется $a \in D$, такой что $x \sqsubseteq a$ для всех $x \in S$. *Нижней гранью* S называется $a \in D$, такой что $a \sqsubseteq x$ для всех $x \in S$.

Частично упорядоченное множество $(D, \sqsubseteq)$ называется *нижней* (*верхней*) *полурешёткой*, если для любого его подмножества (в том числе и пустого) существует наименьшая верхняя (наибольшая нижняя) грань. Частично упорядоченное множество называется *полной решёткой*, если оно является верхней и нижней полурешеткой.

Далее будем считать, что $(D, \sqsubseteq, \perp)$ – полная решетка с наименьшим элементом, т. е. $\forall a \in D.\ \perp \sqsubseteq a$. Пусть $f: D \to D$ – произвольная тотальная функция на D. Функция f называется *монотонной*, если $\forall a,b \in D.\ a \sqsubseteq b \Rightarrow f(a) \sqsubseteq f(b)$. Последовательность $\{a_m\}_{m\geq 0}$ является *возрастающей цепью*, если $a_0 \sqsubseteq a_1 \sqsubseteq \ldots \sqsubseteq a_m \sqsubseteq \ldots$. Для натурального n и $x \in D$ определим: $f_0(x) = x,\ f_{n+1}(x) = f_n(f(x))$.

Лемма 5. $\{f_n(\perp)\}_{n\geq 0}$ – для монотонной функции f определяет возрастающую цепь: $\perp \sqsubseteq f(\perp) \sqsubseteq f_2(\perp) \sqsubseteq \ldots \sqsubseteq f_n(\perp) \sqsubseteq \ldots$.

Для наименьшей верхней грани цепи будем использовать обозначение: $\cup_{m\geq 0} a_m$. Тотальная функция $f: D \to D$ называется *непрерывной*, если для любой возрастающей цепи $\{a_m\}_{m\geq 0}$ выполняется равенство $f(\cup_{m\geq 0} a_m) = \cup_{m\geq 0} f(a_m)$.

Лемма 6. Непрерывная функция является монотонной.

Неподвижной точкой функции f называется решение уравнения $x = f(x)$.

Теорема 2 Клини. Пусть f – непрерывная функция. Тогда $\cup_{n\geq 0}\{f_n(\perp)\}$ является наименьшей неподвижной точкой f.

3.5 Семантика рекурсивных предикатов

Для предикатов B и C отношение dependent(B, C) обозначает *непосредственную зависимость* B от C, если в программе предиката B имеется вызов предиката C. Отношение def(B, C) обозначает *зависимость* B от C, если существуют предикаты X_1, …, X_m ($m > 1$) такие, что $B = X_1$, $C = X_m$ и истинно dependent(X_i, X_{i+1}) для $i = 1, \ldots, m - 1$.

Предикат B является *рекурсивным*, если истинно отношение def(B, B). Для рекурсивного предиката B *рекурсивным кольцом* называется набор предикатов rec(B) = {C | def(B, C) & def(C, B)}. Очевидно, что если $C \in$ rec(B), то rec(B) = rec(C).

Возможны более сложные формы рекурсии, опосредованные через подстановку предиката C аргументом вызова B(x˜: y) в случае, когда C зависит от B. Более экзотическая форма рекурсии может быть реализована через оператор каррирования. В данной работе запрещаются сложные формы рекурсии, поскольку для построения реальных алгоритмов в них нет необходимости. Действуют следующие ограничения. Если предикат C является аргументом вызова B(x˜: y), то B и C не могут находиться в одном рекурсивном кольце. В рекурсивном кольце не может быть предикат, определяемый через оператор каррирования (две последние строки Таблицы 1).

Пусть имеется рекурсивное кольцо предикатов $H_1, H_2, \ldots, H_n$. Систему логических тождеств (3) представим для операционных семантик предикатов рекурсивного кольца в следующем виде:

$$\mathcal{R}(H_j)(x_j, y_j) \equiv K_j(\mathcal{R}(H_1), \mathcal{R}(H_2), \ldots, \mathcal{R}(H_n))(x_j, y_j),\ j = 1,\ldots,n.$$

Здесь функция K_j определяет ||<оператор>||, соответствующий программе предиката H_j, в зависимости от операционных семантик предикатов рекурсивного кольца. Будем считать, что наборы $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$ составлены из различных переменных. Перепишем систему в векторном виде:

$$G = K(G)\ , \tag{4}$$

где $G = (\mathcal{R}(H_1), \mathcal{R}(H_2), \ldots, \mathcal{R}(H_n))$ – вектор семантик предикатов рекурсивного кольца; $K = (K_1, K_2, \ldots, K_n)$ – вектор функций, применяемых к семантикам предикатов. Предикат G определен на кортеже наборов переменных $(x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$. Множество всех различных кортежей обозначим через T_G. Предикат $\mathcal{R}(H_j)$ есть j-я компонента (проекция) предиката G, т. е. $\mathcal{R}(H_j) = G_j$ = pr(j, G).

Будем считать, что каждый из типов переменных, аргументов и результатов предикатных программ, рассматриваемый с отношением включения "$\subseteq$", являются полной решеткой. Отметим, что примитивные типы «целый», «вещественный» и другие, а также предикатные типы в представлении (A.name, A.pref) являются полными решетками.

Множество кортежей T_G с отношением включения "$\subseteq$" является полной решеткой [2; 3]. Минимальным элементом решетки является пустое множество кортежей $\varnothing$. Для предикатов H и L на множестве T_G определим отношение:

$$H \subseteq L \ \cong \{ u \in T_G \mid H(u)\} \ \subseteq \ \{ u \in T_G \mid L(u)\}.$$

Множество предикатов на T_G является полной решеткой. Минимальным элементом решетки является тождественно ложный предикат F. Рассмотрим разбиение на компоненты: $H = (H_1, H_2, \ldots, H_n)$ и $L = (L_1, L_2, \ldots, L_n)$. Для каждой из компонент отношение $\subseteq$ определяется аналогичным образом; оно также является полной решеткой. Истинно соотношение: $H \subseteq L \equiv \forall i{=}1..n.\ H_i \subseteq L_i$.

Рассмотрим цепь предикатов $\{G^m\}_{m\geq 0}$, определяемую следующим образом:

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

Естественно ожидать, что предел последовательности $\{G^m\}_{m\geq 0}$ (если он существует) даст нам неподвижную точку – решение системы G = K(G).

Лемма 7. Пусть $\{H^m\}_{m\geq 0}$ – возрастающая цепь предикатов на T_G. Тогда для наименьшей верхней грани цепи справедливо равенство

$$\cup_{m\geq 0} H^m = (\cup_{m\geq 0} H_1^m, \ldots, \cup_{m\geq 0} H_n^m)\ .$$

Лемма 8. Пусть $\{H^m\}_{m\geq 0}$ – возрастающая цепь предикатов и $H^{\sim} = \cup_{m\geq 0} H^m$. Пусть $u \in H^{\sim}$. Тогда $\exists k\ \forall m \geq k.\ u \in H^m$.

Доказательство от противного. Допустим, что для всех k набор u не принадлежит вектор-графику H^k. Определим предикат $H^{\#}$, совпадающий с $H^{\sim}$ всюду, за исключением значения u, который ему не принадлежит. Предикат $H^{\#}$ является верхней гранью последовательности $\{H^m\}_{m\geq 0}$, причем $H^{\#} \subseteq H^{\sim}$ и $H^{\#} \neq H^{\sim}$, а тогда грань $H^{\sim}$ не является минимальной, что приводит к противоречию. □

Замечание. Лемма верна и для возрастающей цепи компонент предикатов $\{H_j^m\}_{m\geq 0}$.

Лемма 9. Вектор-функция V, составленная из непрерывных функций V_i $(i = 1, ..., n)$, является непрерывной.

Доказательство. Пусть $\{H^m\}_{m\geq 0}$ – возрастающая цепь предикатов. В соответствии с Леммой 7 имеет место цепочка равенств:

$$\cup_{m\geq 0} V(H^m) = \cup_{m\geq 0}(V_1(H^m), ..., V_n(H^m)) = (\cup_{m\geq 0} V_1(H^m), ..., \cup_{m\geq 0} V_n(H^m)) =$$
$$= (V_1(\cup_{m\geq 0} H^m), ..., V_n(\cup_{m\geq 0} H^m)) = V(\cup_{m\geq 0} H^m).\ \square$$

Представим проекции вектор-функции K на компонентные функции для оператора суперпозиции, параллельного и условного операторов.

$$K_S(P, Q)(x, y) \equiv \exists z.\ P(x, z)\ \&\ Q(z, y)$$
$$K_P(P, Q)(x, y) \equiv P(x, y)\ \&\ Q(x, z)$$
$$K_C(P, Q)(x, y) \equiv (e \Rightarrow P(x, y))\ \&\ (\neg e \Rightarrow Q(x, y))\ .$$

Лемма 10. Функции K_S, K_P и K_C для оператора суперпозиции, параллельного оператора и условного оператора, непрерывны относительно предикатов-семантик P и Q.

Доказательство. Докажем непрерывность функции $K_S(P, Q)$ для оператора суперпозиции. Пусть $\{H^m\}_{m\geq 0}$ – возрастающая цепь предикатов. Здесь H вырождается в набор (P, Q). Необходимо доказать, что $K_S(\cup_{m\geq 0}(P, Q)^m) = \cup_{m\geq 0} K_S((P, Q)^m)$. В соответствии с Леммой 7 $\cup_{m\geq 0}(P, Q)^m = (\cup_{m\geq 0} P^m, \cup_{m\geq 0} Q^m)$. Следовательно, необходимо доказать равенство $K_S(\cup_{m\geq 0} P^m, \cup_{m\geq 0} Q^m) = \cup_{m\geq 0} K_S(P^m, Q^m)$ при условии, что $\{P^m\}_{m\geq 0}$ и $\{Q^m\}_{m\geq 0}$ – возрастающие цепи предикатов, или в развернутом виде:

$$\{(x, y) \mid \exists z.\ (x, z) \in \cup_{m\geq 0} P^m\ \&\ (z, y) \in \cup_{m\geq 0} Q^m\} =$$
$$\cup_{m\geq 0}\{(x, y) \mid \exists z.\ (x, z) \in P^m\ \&\ (z, y) \in Q^m\}.$$

Пусть (x, y) принадлежит множеству в левой части. Тогда для некоторого z_0 истинны $(x, z_0) \in \cup_{m\geq 0} P^m$ и $(z_0, y) \in \cup_{m\geq 0} Q^m$. В соответствии с Леммой 8 существует такое k, что $(x, z_0) \in P^m$ и $(z_0, y) \in Q^m$ при $m \geq k$. Далее, очевидно, (x, y) принадлежит правой части равенства.

Нетрудно показать, что $\{(x, y) \mid \exists z.\ (x, z) \in P^m\ \&\ (z, y) \in Q^m\}_{m\geq 0}$ – возрастающая цепь предикатов. Допустим, (x, y) принадлежит наименьшей верхней грани этой цепи, т. е. правой части равенства. Принадлежность (x, y) множеству в правой части доказывается применением Леммы 8.

Доказательство непрерывности функций K_P и K_C проводится аналогично. □

Функция $K_C(P, Q)$ для условного оператора **if** (e) $B(x\mathbf{:}\ y)$ **else** $C(x\mathbf{:}\ y)$ не является непрерывной относительно вхождения переменной e [1]. Поэтому вхождение e не может быть источником рекурсии. **Ограничение**: допустим, значение переменной e является результатом вызова $E(...\mathbf{:}\ e...)$. Предикат, программа которого есть условный оператор, и предикат E не могут входить в одно и то же рекурсивное кольцо.

Допустим, система (4) определений кольца предикатов не использует сложные формы рекурсии и удовлетворяет отмеченному ограничению. Из Лемм 9 и 10 следует, что вектор-функция K в системе уравнений $G = K(G)$ для кольца предикатов является непрерывной. В соответствии с теоремой Клини о неподвижной точке решение системы $G = K(G)$ есть $G = \cup_{m\geq 0} G^m$, где $G^0 = F$, $G^{m+1} = K(G^m)$, $m \geq 0$. $\cup_{m\geq 0} G^m$ – наименьшая неподвижная точка вектор-функции K.

---

[1] Не обеспечивается даже монотонность для $e \Rightarrow P(x, y)$.

Данная система имеет непустое решение лишь при наличии в рекурсивном кольце предиката, определяемого через условный оператор **if** (e) B(x**:** y) **else** C(x**:** y), причем лишь один из предикатов, B или C, может принадлежать рекурсивному кольцу.

3.6 Семантика полной программы

Систему уравнений (4) для операционных семантик предикатов рекурсивного кольца перепишем в следующем виде:

$$G = K(\mathcal{R}(H_1), \mathcal{R}(H_2), \dots, \mathcal{R}(H_n), \mathcal{R}(E_1), \mathcal{R}(E_2), \dots, \mathcal{R}(E_s)) \quad . \tag{5}$$

Здесь $E_1, E_2, \dots, E_s$ $(s > 0)$ – предикаты, используемые в программах рекурсивного кольца, но не принадлежащие ему.

Допустим, полная программа содержит несколько рекурсивных колец предикатов. Пусть семантика предикатов кольца α определяется системой (5). Кольцо α *зависит* от кольца β, если один из предикатов $E_1, E_2, \dots, E_s$ принадлежит рекурсивному кольцу β. Рекурсивные кольца образуют граф. Вершинами являются кольца, а дугами – отношения зависимости колец. Допустим, имеется цикл в графе колец. Нетрудно доказать, что предикаты разных колец, находящихся на цикле, должны принадлежать одному кольцу. Следовательно, набор рекурсивных колец полной программы представляется в виде *дерева рекурсивных колец*.

Полная программа может зависеть от параметров – предикатов, определенных в других полных программах, которые, в свою очередь, могут зависеть от других параметром. Будем считать, что зависимость множества полных программ от параметров имеет структуру *дерева зависимостей* – не допускаются циклы для связей по параметрам.

Наконец, необходимо определить операционную семантику для всех базисных предикатов. Например, для базисного предиката +(a, b**:** c) операционная семантика $\mathcal{R}$(+) задается тождеством: $\mathcal{R}(+)(a, b, c) \cong c = a + b$. Для любого базисного предиката B должно реализоваться тождество: $\mathcal{R}(B) = B$.

Итак, операционная семантика полностью определена для произвольных программ языка P_0. Следующая задача: для произвольного предиката H доказать тождество $\mathcal{R}(H) = H$, где $\mathcal{R}(H)$ определяется для программы предиката H, а вхождение H справа понимается в смысле предиката, определенного в левой колонке Таблицы 1.

Лемма 11. Пусть A – переменная предикатного типа. Тогда истинно A(x**:** y) ≡ A.name(A.pref, x**:** y). Здесь A(x**:** y) – предикат, а не оператор вызова предиката.

Доказательство. Из операционной семантики следует тождество || A(x**:** y) || ≡ A.name(A.pref, x, y) в случае $\mathcal{R}$(A.name) = A.name, но не || A(x**:** y) || ≡ A(x**:** y). Доказательство проводится интерпретацией определения A = (A.name, A.pref) для предикатов левой части Таблицы 1.

Имеются три возможных источника значения переменной A: вызов вида B(x̃**:** y) и два вида оператора каррирования.

Пусть значением A является предикат C, подставляемый аргументом в вызове вида B(x̃**:** y). Тогда A = (C, ( )) или A.name = C и A.pref – пустой префикс. Реализуется цепочка тождеств: A(x**:** y) ≡ C(x**:** y) ≡ A.name(x**:** y) ≡ A.name(A.pref, x**:** y).

Пусть A является результатом предиката: H(z**:** A) ≅ ∀y,z. (A(x**:** y) ≡ B(z, x**:** y)). Тогда A = (B, (z)), т.е. A.name = B и A.pref = z. Реализуется цепочка тождеств:
A(x**:** y) ≡ B(z, x**:** y) ≡ A.name(A.pref, x**:** y).

Пусть A является результатом предиката: H(z, V**:** A) ≅ ∀y,z. (A(x**:** y) ≡ V(z, x**:** y)). Тогда A(x**:** y) ≡ V(z, x**:** y) ≡ (V.name, V.pref)(z, x**:** y) ≡ V.name(V.pref, z, x**:** y). Истинность последнего тождества обеспечивается применением индуктивного предположения, поскольку префикс V.pref короче A.pref. Полагаем A = (V.name, (V.pref, z)), что реализует доказательство A(x**:** y) ≡ A.name(A.pref, x**:** y).

Следствие Леммы 11. (B, x)(y: z) ≡ B(x, y: z). Это аналог операции каррирования в языках функционального программирования.

Лемма 12. Допустим, для предикатов B и C истинны утверждения $\mathcal{R}(B) = B$ и $\mathcal{R}(C) = C$, для переменной A предикатного типа истинно $\mathcal{R}(A.name) = A.name$. Тогда для всех видов операторов Таблицы 1 истинно тождество $\mathcal{R}(H) = H$.

Доказательство. Для оператора суперпозиции в программе H(x: y) { B(x: z); C(z: y) } реализуется следующая цепочка тождеств:

$$\mathcal{R}(H)(x, y) \equiv \| B(x: z); C(z: y) \| \cong \exists z. \| B(x: z) \| \,\&\, \| C(z: y) \| \equiv$$
$$\exists z. \mathcal{R}(B)(x, z) \,\&\, \mathcal{R}(C)(z, y) \equiv \exists z. B(x, z) \,\&\, C(z, y) \equiv \exists z. B(x: z) \,\&\, C(z: y) \cong H(x: y).$$

Доказательство для параллельного и условного операторов реализуется аналогично.

Для вызова A(x: y) ввиду Леммы 11 истинна цепочка тождеств:

$$\mathcal{R}(H)(A, x, y) \equiv \| A(x: y) \| \cong \mathcal{R}(A.name)(A.pref, x, y) \equiv A.name(A.pref, x: y) \equiv$$
$$\equiv A(x: y) \equiv H(A, x: y).$$

Для оператора каррирования D(y: z) {B(x, y: z)} при использовании Следствия Леммы 11 реализуется цепочка тождеств:

$$\mathcal{R}(H)(x, D) \equiv \| D(y: z) \{B(x, y: z)\} \| \cong D = (B, (x)) \equiv \forall y,z. (D(y: z) \equiv (B, (x))(y: z)) \equiv$$
$$\forall y,z. (D(y: z) \equiv B(x, y: z)) \cong H(x: D).$$

Для оператора каррирования D(y: z) {A(x, y: z) ввиду Леммы 11 истинна цепочка тождеств:

$$\mathcal{R}(H)(A, x, D) \equiv \| D(y: z) \{A(x, y: z)\} \| \cong D = (A.name, (A.pref, x)) \equiv$$
$$\forall y,z. (D(y: z) \equiv (A.name, (A.pref, x))(y: z)) \equiv$$
$$\forall y,z. (D(y: z) \equiv A.name(A.pref, x, y: z)) \equiv \forall y,z. (D(y: z) \equiv A(x, y: z)) \cong$$
$$H(A, x: D). \square$$

Лемма 13. Допустим, в составе полной программы имеются предикатные программы $H_1, H_2,\ldots, H_n$. В этих программах встречаются вызовы других программ $E_1, E_2,..., E_s$, $s > 0$. Допустим, истинны тождества $\mathcal{R}(E_j) = E_j$, $j = 1,...,s$. Предположим также, что для любой переменной A предикатного типа, встречающейся в программах $H_1, H_2,\ldots, H_n$, истинно утверждение $\mathcal{R}(A.name) = A.name$. Тогда истинны тождества $\mathcal{R}(H_j) = H_j$, $j = 1,...,n$.

Доказательство. Рассмотрим случай, когда среди $H_1, H_2,\ldots, H_n$ нет рекурсивных предикатов. Доказательство проводится индукцией по n с использованием Леммы 12.

Далее рассмотрим случай, когда $H_1, H_2,\ldots, H_n$ есть в точности кольцо рекурсивных предикатов. Систему тождеств для $H_1, H_2,\ldots, H_n$, определяемых в левой части Таблицы 1, запишем в векторном виде:

$$H = K(H_1, H_2,\ldots, H_n, E_1, E_2,..., E_s) \ .$$

С учетом $\mathcal{R}(E_j) = E_j$ и $\mathcal{R}(A.name) = A.name$, а также отсутствия операторов каррирования в рекурсивных программах, вектор-функция K оказывается той же самой, что и в системе для операционных семантик предикатов $H_1, H_2,\ldots, H_n$:

$$G = K(\mathcal{R}(H_1), \mathcal{R}(H_2), \ldots, \mathcal{R}(H_n), E_1, E_2, \ldots, E_s) \ ,$$

полученной из (5) заменой $\mathcal{R}(E_j)$ на E_j. Поскольку системы тождеств идентичны, то их решения будут совпадать, т.е. истинны $\mathcal{R}(H_j) = H_j$, $j = 1,...,n$.

Наконец, рассмотрим общий случай, когда $H_1, H_2,\ldots, H_n$ включает несколько рекурсивных колец. Набор этих колец образует дерево колец. Рекурсивное кольцо, являющееся листом, соответствует предыдущему случаю. Далее применяется индукция по n. □

Теорема 3. Для произвольной предикатной программы H(x: y) на языке P_0 истинно тождество $\mathcal{R}(H) = H$. Предполагается, что данное тождество истинно для всех базисных предикатов.

Доказательство. Допустим, программа $H(x{:}\ y)$ принадлежит полной программе Π. Обозначим через $SUBS(\Pi)$ набор предикатов, являющихся значениями переменных предикатного типа, встречающихся в программе Π. Для набора предикатов M из Π обозначим через $\Pi[M]$ минимальную полную программу, являющуюся частью Π и содержащую программы предикатов набора M. Пусть $\Pi_0 = \Pi$, $\Pi_{j+1} = \Pi_j[SUBS(\Pi_j)]$, $j = 0, 1, 2, \ldots$. Докажем, что при некотором k программа $\Pi_k \neq \varnothing$, а $\Pi_{k+1} = \varnothing$. Допустим обратное, т. е. существует такое k, что $\Pi_k \neq \varnothing$ и $\Pi_{k+1} = \Pi_k$. Пусть Π_k построено на базе предикатов $C_1, C_2,\ldots, C_m$ ($m > 0$), являющихся значениями переменных предикатного типа $D_1, D_2, .., D_r$. Предположим, программа Π_k содержит вызовы $D_1(...), D_2(...), .., D_r(...)$. Если один из вызовов $D_j(...)$ отсутствует в Π_k, переменную D_j можно будет исключить из рассмотрения при построении программы Π_{k+1}.

Допустим, вызов $D_1(...)$ встречается в программе $\Pi_k[\{C_1\}]$. Тогда, при исполнении программы C_1 произойдет вызов $D_1(...)$, который снова запустит C_1. Получим сложную форму рекурсии, которая запрещена. Следовательно, вызов $D_1(...)$ не может зависеть от C_1. Пусть $D_1(...)$ зависит от C_2. Пусть $D_2(...)$ зависит от C_1. В этом случае C_1 вызовет $D_2(...)$, затем C_2, далее $D_1(...)$ и C_1. Следовательно, вызов $D_2(...)$ не может зависеть от C_1 и C_2; пусть он зависит от C_3. Продолжая анализ для C_3 и последующих предикатов, мы неизбежно получим для полного списка $C_1, C_2,\ldots, C_m$ рекурсивную цепочку зависимостей со сложной формой рекурсии. В итоге приходим к противоречию.

Следовательно, $\Pi_{k+1} = \varnothing$ и Π_k не содержит переменных предикатного типа. К набору предикатных программ, входящих в программу Π_k, применим Лемму 13. Набор программ, входящих в Π_k, зависит лишь от параметров полной программы Π_k и базисных предикатов. Потребуем для них выполнение тождества $\mathcal{R}(H) = H$, что по Лемме 13 обеспечивает выполнение тождества $\mathcal{R}(H) = H$ для всех программ Π_k. В частности, тождество $\mathcal{R}(H) = H$ реализуется для предикатов $C_1, C_2,\ldots, C_m$ ($m > 0$), являющихся значениями переменных предикатного типа. Далее по индукции Лемму 13 можно применить к программам $\Pi_{k-1}, \Pi_{k-2},\ldots,\Pi_0$. В итоге, доказано, что тождество $\mathcal{R}(H) = H$ выполняется для всех программ исходной программы Π при условии, что тождество $\mathcal{R}(H) = H$ выполняется для параметров Π и базисных предикатов.

Докажем истинность $\mathcal{R}(H) = H$ для предикатов – параметров полной программы Π. Рассмотрим дерево зависимостей по параметрам. Листья этого дерева не содержат параметров. Далее применяется индукция по числу полных программ в дереве зависимостей. □

Теорема 4. Произвольная предикатная программа $H(x{:}\ y)$ на языке P_0 является однозначной. Напоминаем, что требование однозначности введено для всех базисных предикатов.

Доказательство проводится аналогично доказательству Теоремы 3. Доказывается однозначность операторов Таблицы 1. Доказательство однозначности предикатов рекурсивного кольца реализуется от противного с использованием Леммы 8. □

3.7 Случай неоднозначных предикатов

Операционная семантика определена лишь для однозначных предикатов. Проблема в том, что для оператора суперпозиции $B(x{:}\ z); C(z{:}\ y)$ в случае неоднозначного предиката $B(x{:}\ z)$ не гарантируется завершение оператора суперпозиции. Допустим, оператор суперпозиции реализуется для некоторого z_1, т.е. истинно $B(x{:}\ z_1)\ \&\ C(z_1{:}\ y)$. Допустим, исполнение предиката $B(x{:}\ z)$ может также завершиться другим значением z_2, т.е. истинно $B(x{:}\ z_2)$. При этом не исключена возможность, что предикат $C(z_2{:}\ u)$ не определен для z_2, т.е. будет ложным для любых результатов u. В данном случае в соответствии с определением операционной семантики предикат $\mathcal{R}$ должен быть ложным для оператора суперпозиции на значениях (x, y), поскольку исполнение не всегда завершается. Проблему можно разрешить введением в Таблице 1 следующего предиката для оператора суперпозиции:

$$H(x{:}\ y) \cong \exists z.\ (B(x{:}\ z)\ \&\ C(z{:}\ y))\ \&\ \forall z.\ (B(x{:}\ z) \Rightarrow \exists u.\ C(z{:}\ u))\ .$$

Разумеется, остается открытым вопрос о полезности программ-функций, которые могут завершаться и не завершаться для разных исполнений при тех же аргументах.

4 Обзор работ

Формализованное описание языка программирования в виде онтологии встречается в ряде работ, например, в виде совместного онтологического описания [5] пяти языков стандарта IEC 61131-3 для программируемых логических контроллеров и для проецирования на единое внутреннее представление семейства реализуемых языков [6]. Однако применение онтологического аппарата для описания языков пока не оказало заметного влияния на индустрию разработки языковых процессоров. Формальная операционная семантика является существенно более сложной и глубокой в сравнении с онтологическими формализациями, поскольку семантика исполнения программы определяет главную часть языка программирования. Формальная семантика является обязательным базисом для дедуктивной верификации и программного синтеза.

Наиболее близким подходом к предикатному программированию является *предикативное* программирование [7] Э. Хехнера, в котором программа синтезируется как результат последовательных уточнений (refinements) по спецификации программы, представленной логической формулой. Уточнения реализуются применением общеизвестных правил логического вывода. Автор не утруждает себя построением формальной семантики, а просто полагает конструкции введенного им простого императивного языка эквивалентами логических формул. У предикатного программирования много общего с *семантическим* программированием [12], которое, однако, позиционируется в сторону логического программирования.

Разработка формальной семантики языка предикатного программирования проводилась ранее [9, 11]. Также как и предыдущая версия ядра языка предикатного программирования, язык P_0 является полным алгоритмическим базисом. При этом язык P_0 существенно проще. В частности, в нем нет типов данных. Как следствие, возможны бестиповые языки предикатного программирования. Операционная семантика языка P_0 построена более простым способом, ближе к классическому, без определения интерпретатора предикатных программ. Соответственно, все доказательства стали короче и проще.

5 Заключение

Для языка P_0, являющегося вычислимым подмножеством языка исчисления предикатов и используемого при построении языка предикатного программирования P [8], разработана формальная операционная семантика и доказана эквивалентность языка P_0 и соответствующего подмножества языка исчисления предикатов, т.е. доказано тождество $\mathcal{R}(H) = H$.

Язык P_0 ограничен однозначными предикатами. Разработка операционной семантики для неоднозначных предикатов входит в планы дальнейших работ. Рекурсивные предикаты определены с использованием аппарата неподвижной точки. В связи с этим запрещена рекурсия через предикаты, являющиеся аргументами или результатами других предикатов. Источником рекурсии является условный оператор **if** (e) B(x: y) **else** C(x: y), где лишь один из предикатов, B или C, может принадлежать рекурсивному кольцу; при этом запрещена рекурсия по e. В рекурсивное кольцо могут также входить другие предикаты, используемые в операторах суперпозиции и параллельных операторах.

Литература

[1] Шелехов В.И. Оптимизация автоматных программ методом трансформации требований. *Тр. 2-й межд. конф. «Инструменты и методы анализа программ»*. Кострома, КГТУ. 2014. С. 175-183. http://persons.iis.nsk.su/files/persons/pages/req_k.pdf

[2] Крицкий С. П. Трансляция языков программирования: синтаксис, семантика, перевод: Учеб. пособие. Р.-на-Дон.: РГУ, 2005. URL: http://public.uic.rsu.ru/~skritski/scourses/Transl/Langs1.htm.

[3] Гретцер Г. Общая теория решеток: Пер. с англ. 1982. 456 с.

[4] Клини С. К. Введение в метаматематику: Пер. с англ. М.: 1957. 526 с.

[5] Будаговский Д. А., Дубинин В. Н. Онтология языков программирования стандарта IEC 61131-3, *Сб. статей XVIII Международной научно-методической конференции "Университетское образование"*. - Пенза, 2014. С. 164-166.

[6] Князева М.А., Тимченко В.А. Модель онтологии проекций языков программирования высокого уровня на единое представление программ. *Тр. 11-й конф. по искусственному интеллекту КИИ-08*. Москва, 2008. Т. 1. С. 124 - 132.

[7] Hehner E.C.R. Predicative programming, parts I and II. *Communications of the ACM* 27(2). 1984. P. 134-151.

[8] *Карнаухов Н.С., Першин Д.Ю., Шелехов В.И*. Язык предикатного программирования Р. *Препр. ИСИ СО РАН*; N 153. Новосибирск, 2010. 42с.
http://persons.iis.nsk.su/files/persons/pages/plang12.pdf

[9] Shelekhov V. The language of calculus of computable predicates as a minimal kernel for functional languages. *BULLETIN of the Novosibirsk Computing Center. Series: Computer Science. IIS Special Issue.* 2009. 29(2009). P.107-117.

[10] PVS Specification and Verification System. *SRI International*. http://pvs.csl.sri.com/

[11] Шелехов В.И. Предикатное программирование. Учебное пособие. Новосибирск: НГУ, 2009. 109с.

[12] Гончаров С.С., Ершов Ю.Л., Свириденко Д.И. Методологические аспекты семантического программирования. *Научное знание: логика, понятия, структура*. Новосибирск, 1987. С.154-184.