А.Г. ПИСКУНОВ

ФОРМАЛИЗАЦИЯ ПАРАДИГМЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Аннотация

В работе даются определения таким основным понятиям ООП как класс, объект, наследование, инкапсуляция, полиморфизм через математические термины: множество, отношение, функция, абстрактный автомат. Строго различаются понятия тип (множество состояний абстрактного автомата) и класс (абстрактный автомат).

ВВЕДЕНИЕ

Идея пошаговой разработки программ с последовательным уточнением и устранением разного рода неопределенностей подымается практически во всех работах по программированию, начиная с учебников, таких как [1] , языков спецификаций [2] и заканчивая сложными современными технологиями [3] в которых используются автоматическое доказательство соответствия следующего шага предыдущему. Вслед за автором [4] будем рассматривать разработку программного обеспечения, как процесс преобразования текста с решением некоторой задачи из одного языка в другой язык с ее последовательной детализацией. При этом, в последовательности записи решений можно провести четкую границу между реализацией (т.е. записью решения на некотором (-ых) языке(-ах) программирования) и спецификацией (в самом крайнем случае это желание программиста сделать решить задачу).

Наличие удобных абстракций в языках спецификаций, позволяющую удобную ее запись в языке реализации должно существенно облегчать весь процесс разработки программного обеспечения. Например, если под пограммным обеспечением будет по-

ниматься приложение, использующее РСУБД, то тогда, в языках реализации будет использоваться один из промышленных SQL, а в языках спецификаций может использоваться реляционная алгебра как формальная математическая теория. С другой стороны, несмотря на признание важности как можно более ранней формализации в [1] и очевидное соображение, что абстракцией от пары данные + программы может быть только пара значения + функции (то есть, абстрактный автомат [5]), для абстрактных типов данных не было замечено простой формальной математической модели.

Например, определения из [6] , [7] мало похожи на математику. С другой стороны, в монографии [8] существенно используется понятие гибридного автомата. Однако это понятие избыточно переусложено, не дается определение наследования автоматов и, собственно, автомат считается моделью не класса, а объекта.

В работах [3] , [9] абстрактный тип, задаваемый схемами, очень близок к общепринятому понятию класса. Язык RSL поддерживает аналоги наследования классов - расширение схем-классов (extending), инкапсуляции - hiding и шаблонов - параметризованных схемы. Можно использовать объекты заданных

схем-классов, но, по видимому, невозможно при помощи схем описывать формальные параметров функции и передавать в функции объекты. Также в языке RSL отсутствует операция прямого произведения множеств, понимаемая как операция над данными, а не как средство задания нового типа. Это приводит к существенно более громоздкому описание схем РСУБД, чем при использовании реляционной алгебры или языка SQL (см. например, [10]).

Кроме того, смотри [11] , [12] , [13] , [14] , [15] .

Далее будет предпринята попытка определить основные понятия ООП через автомат Мура.

Обозначения

- либо общеприняты и с ними можно познакомиться, к примеру, в таком введениии в теорию множеств как [16] ;
- либо взяты из языка формальных спецификаций RSL [9] . Русско-язычные ресурсы по языку RSL: [17] , [18] ;
- либо это обозначение операции диагонального призведения (операции соединения) из набора элементарных операций над частичными функциями. см. [19] .

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И РЕЗУЛЬТАТЫ

Основные определения

Полувычислимая функция

(см. [19]) Функция f называется полувычислимой, если существует программа

- вырабатывающая на выходе значение f(x) если на ее вход значения значения x из области определения функции dom f ;
- никогда не останавливающаяся при подаче на вход значения не принадлежащего области определения f: x$\notin$dom f.

Автомат Мили

Пусть Q - множество состояний, X - множество входных сигналов, Y - множество выходных сигналов.

Тогда тройка (Q, $\delta : Q \times X \to Q$, $\lambda : Q \times X \to Y$) называется абстрактным автоматом Мили, где δ - функция переходов, λ - функция выходов

Автомат Мура

Пусть Q - множество состояний, X - множество

входных сигналов, Y - множество выходных сигналов.

Тогда тройка (Q, δ : Q $\times$ X $\rightarrow$ Q, λ : Q $\rightarrow$ Y) называется абстрактным автоматом Мура, где δ - функция переходов, λ - функция выходов

Утверждение

Для каждого автомата Мили можно построить имитирующий его поведение автомат Мура.

Доказательство смотри, например, в конце раздела 2.2 [5] .

Диагональное произведение отображений

Пусть заданы отображения g_1 : X $\rightarrow$ Y_1 , ..., и g_k : X $\rightarrow$ Y_k. Тогда отображение (g_1 ,..., g_k): X $\rightarrow$ Y_1 $\times$ ... $\times$ Y_k, определенное условием (g_1 ,..., g_k)(x)= (g_1(x), ..., g_k(x)) для каждого x$\in$X, называется диагональным произведением отображений g_1, ..., g_k. Сами отображения g_1, ..., g_k называются компонентами диагонального произведения.

Проекция

Отображение $\pi_{Xj} : X_1 \times ... \times X_k \rightarrow X_j$ заданное

как π_{Xj} (x_1, ..., x_k) = x_j, где 1 ≤ j ≤ k и x_j∈X_j будет называться проекцией.

Обсуждение свойств диагонального произведения, композиции и проекции можно посмотреть в [16] .

Утверждение

Пусть T, X, Y некоторые множества, тогда каждое отображение f : T → X × Y можно представить как диагональное произведение композиций самой функции f и проекций множества значений на сомножители прямого произведения X × Y.

Доказательство. Пусть для f выполняется f(t) = (x, y), t∈T , x∈X, y∈Y. Тогда, по определению проекции, (x,y) = (π_X(x,y), π_Y(x, y)) = (π_X(f (t)),π_Y(f(t)). То есть, по определению, f есть диагональное произведение отображений π_X∘ f и π_Y∘ f.

Следствие

Любая функция f: T → Q × Y может быть выведена из двух функций f_s и f_o, причем множество значений первой, содержится в первом сомножителе - Q, то есть f_s: T → Q, а множество значений второй, во втором сомножителе - f_o : T → Y.

Обсуждение. В доказательстве леммы были использованы простейшие функции - проекция и элементарные операции над функциями - композиция и диагональное произведение (иначе соединение) см. раздел Частично рекурсивные функции в [19] . Следовательно, если функция f была частично рекурсивна, то функции - компоненты $f_s = \pi_Q \circ$ f и $f_o = \pi_Y \circ$ f тоже частично рекурсивны, а значит, полувычислимы. Таким образом, можно считать что функция f определяется (выводится из) через более простые функции компоненты. Полувычислимость функции будем понимать как синоним возможности написать эти функции на каком либо языке программирования.

Функции состояния и выхода

Далее, для некоторых множеств Q, X, Y (причем Y нельзя представить в виде декартового призведения Q с каким либо множеством) функцию f_s : Q $\times$ X $\rightarrow$ Q будем называть функциями состояния, а функции f_o : Q $\rightarrow$ Y функцией выхода.

Замечание

Похоже что, в работе [3] (см. стр 47) функции состояния обозначаются термином generator,

функции выхода - observer, а множество Q - типом интересов.

Утверждение

Пусть есть множество Q. Два любых набора функций:

- состояния: $\{ g_i : Q \times X_i \rightarrow Q \mid i \leq n \}$
- выхода: $\{ f_j : Q \rightarrow Y_j \mid j \leq m \}$

для некоторых множеств Q, X_i, $i \leq n$, Y_j, $j \leq m$ задают автомат Мура.

Доказательство. Пусть множество $N = \{1, ..., n\}$. Тогда зададим функцию переходов

$\delta : Q \times N \times X_1 \times ... \times X_n \rightarrow Q$ следующим образом:

δ (q, i, x_1, ... , x_n)is
 case i of
  1 → g_1(q, x_1),
  2 → g_2(q, x_2),
  ...
  n → g_n(q, x_n)
 end

И функцию выходов $\lambda : Q \rightarrow Y_1 \times ... \times Y_m$ следующим образом

λ (q) is (f_1(q),...,f_m(q))

Обозначив N $\times$ X_1 $\times$ ... $\times$ X_n через X, а Y_1 $\times$ ... $\times$ Y_m - через Y, можно окончательно убедиться что полученная тройка (Q, δ, λ) удовлетворяют определению автомата Мура по построению.

Следствие

Любое множество Q с любым набором функций/подпрограмм, содержащие это множество Q в сигнатуре, задает автомат Мура.

Класс, тип

Для любого автомата (Q, δ, λ) множество его состояний Q будем называть типом. Термин класс будем считать синонимом термина автомат.

Для некоторого автомата A, через Q(A) будем обозначать его тип.

Вследствие утверждения классом можно называть также тройку (Q, { g_i : Q $\times$ X_i $\rightarrow$ Q | i $\leq$ n }, { f_i : Q $\rightarrow$ Y_i |j $\leq$ m })

Объект, процесс

Далее, для произвольного класса A = (Q, F_s, F_o)

- в сигнатуре функций, в частности, из множеств F_s и F_o, вместо типа класса Q(A), можно будет писать обозначение класса A;
- также позволяется запись a∈A, вместо a∈Q(A). Она означает, что хотя величина a является таким же кортежом как любой элемент Q(A), она не может использоваться ни в каких выражениях, кроме как в функциях с классом A (забегая вперед, с наследником класса A) в сигнатуре.

Любая величина a определенная как a∈A будет называться объектом. Если во множестве функций состояния F_s существует хотя бы одна функция с пустым вторым сомножителем, то есть вида g : A → A, то величина a будет называться процессом.

Заметим, что раз тип Q(A) и класс A - различные понятия, то можно говорить о объекте класса A и типа Q(A).

Замечание

Толкование объекта как кортеж очень близко к толкованию объекта как запись в работе [20] .

НАСЛЕДОВАНИЕ, ИНКАПСУЛЯЦИЯ, ПОЛИМОРФИЗМ

Ассоциативность

Отметим (см., например, [21] , стр. 146, раздел Естественное соединение) Операция соединения и, следовательно, декартового произведения - ассоциативны:

$(A \times B) \times C = A \times (B \times C) = A \times B \times C$

Ассоциативность декартового произведения означает, что при $X = A \times B$ функцию $f : X \to T$, можно рассматривать как функцию с сигнатурой $f: A \times B \to T$.

Лямбда выражения

Далее символ лямбда λ будет использоваться для записи лямбда выражений над функциями. Выражение

$\lambda\ val_1$: T_1,..., $val_n : T_n$:- expression(val_1, ..., val_n)

будет определять функцию с сигнатурой $T_1 \times .... \times T_n \to T$, где T - есть тип выражения expression(val_1, ..., val_n).

Например, выражение λ x: Int, y : Int :- x+y определяет функцию сложения двух целых чисел с

сигнатурой Int $\times$ Int $\rightarrow$ Int.

Наследование автоматов

Пусть есть автомат

B = (Q_2, { $g_{2,i} : Q_2 \times X_{2,i} \rightarrow Q_2$ | i $\leq n_2$ }, { $f_{2,i} : Q_2 \rightarrow Y_{2,i}$ |j $\leq m_2$ })

и автомат

A = (Q_1, { $g_{1,i} : Q_1 \times X_{1,i} \rightarrow Q_1$ | i $\leq n_1$ }, { $f_{1,i} : Q_1 \rightarrow Y_{1,i}$ |j $\leq m_1$ }). Будем говорить, что автомат B наследует автомат A (Класс B наследует класс A), если

- Q_2 есть декартовым произведением $Q_l \times Q_1 \times Q_r$, где Q_l и/или Q_r могут быть пустыми множествами;
- (условие наследования для функций состояния) существует частично определенное иньективное отображение $\tau : \{\text{i} \leq n_1\} \rightsquigarrow \{\text{i} \leq n_2\}$ такое что, для любого i $\leq n_1$ и любого $(q_l, q_1, q_r) \in Q_2$ функция $g_{2,\tau(i)}$ и функция $g_{1,i}$ связаны следующим соотношением:

  $g_{2,\tau(i)} = \lambda\,(q_l, q_1, q_r, \text{x}) : Q_2 \times X_{2,\tau(i)}$:- $(q_l, g_{1,i}(q_1, \text{x}), q_r)$,

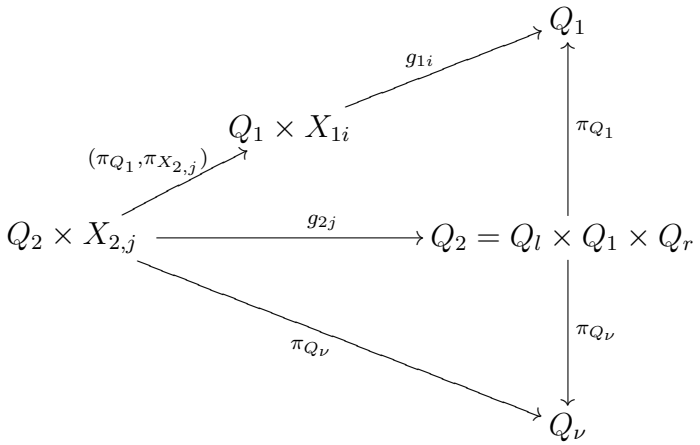

Рис. 1: Условие наследования функций состояния. $\nu \in \{l, r\}$, $X_{2,j} \stackrel{def}{=} X_{1,i}$, $Q_2 \stackrel{def}{=} Q_l \times Q_1 \times Q_r$

то есть, функция $g_{2,\tau(i)}$ является диагональным произведением

$g_{2,\tau(i)} = (\pi_{Q_l}, g_{1,i} \circ (\pi_{Q_1}, \pi_{X_{2,\tau(i)}}), \pi_{Q_r})$. Отметим, что $X_{2,\tau(i)} = X_{1,i}$ (см. рис. 1):

- (условие наследования для функций выхода) существует частично определенное иньективное отображение $\rho : \{\mathrm{i} \leq m_1\} \rightsquigarrow \{\mathrm{i} \leq m_2\}$, такое что $f_{2,\rho(i)} = f_{1,i} \circ \pi_{Q_1}$, $\mathrm{i} \leq m_1$ причем, $Y_{2,\rho(i)} = Y_{1,i}$ (см. рис. 2).

Автомат A будет называтся родителем, автомат B - потомком. Если считать, что B подкласс, A - надкласс, а функции состояния, для которых выпол-

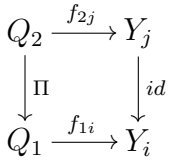

Рис. 2: Условие наследования для функций выхода

няется условие наследования (то есть, $g_{2,\tau(i)}$ и $g_{1,i}$) и функции выхода, для которых выполняется условие наследования (то есть, $f_{2,\rho(i)}$ и $f_{1,i}$) разделяются (sharing) между классами, то это определение вполне соответствует определению наследования в смысле [11] :

Inheritance is the sharing of attributes between a class and its subclass. Multiple inheritance occurs when a subclass inherits attributes from more than one class. Single inheritance occurs when a subclass inherits attributes from only one class.

Где под атрибутом может пониматься как переменная так и метод.

Пример множественного наследования

Рассмотрим пример множественного наследования в терминах языка C++:

```
class A        { public: int a; };
```

```
class B        { public: int a; };
class C : public A, public B {        } ;
```

И запишем его в терминах автоматов:

A = (A.Int, {a.set = λ (x,y): A.Int $\times$ Int :- y }, {a.get = λ x: A.Int :- x }).

Функции (и тип) класса будем указывать

- явно задавая функцию класса например, A.a.get;
- либо, в случае уникальности имени функции в каком либо тексте (спецификация или программе), просто задавая ее имя - a.get.

Причем давать имена set и get, вообще говоря, большой нужды нет, в связи с различными сигнатурами функций. Это сделано для дополнительного указания на аналоги операторов присвоения (set) и извлечения значения (get);

Далее, класс B = (B.Int, {a = λ (x,y): B.Int $\times$ Int :- y }, {a = λ x: B.Int :- x }) ;

и класс C = (A.Int $\times$ B.Int,

{ a_1 = λ (a,b,x): A.Int $\times$ B.Int $\times$ Int :- (A.a.set (a,x),b),

a_2 = λ (a,b,x): A.Int $\times$ B.Int $\times$ Int :- (a, B.a(b,x)) },

{ a_1 = A.a.get $\circ \pi_{A.Int}$, a_2 = B.a $\circ \pi_{B.Int}$ })

По построению видно, что удовлетворены все условия определения наследования:

- Тип автомата наследника Q_2, в данном случае, A.Int $\times$ B.Int содержит в качестве сомножителя тип любого из автоматов родителей Q_1, A.Int равно как и B.Int;

- Если в качестве родителя рассматривать класс A, то есть, под типом Q_1 понимать A.Int, а под набором функций состояния $\{g_{1,i}\}$ понимать множество {A.a.set}, то становится очевидно, что функциями A.a.set и a_1 находятся в отношении

  $a_1 = \lambda$ (q_1,q_r,x): A.Int $\times$ B.Int $\times$ Int :- (A.a.set (q_1,x),q_r),

  левый сомножитель Q_l пуст, правый Q_r = B.Int , $g_{2,1}$ в данном примере это a_1:

  $g_{2,1} = \lambda$ (q_l, q_1, q_r, x) : Q_2 $\times$ Int :- (q_l, $g_{1,1}$(q_1, x), q_r)

  Те же рассуждения годятся в случае, если в качестве родителя рассматривать класс B. Тогда пустым будет правый сомножитель Q_r;

- Условия соотношения функций выхода родителей и потомка выполняются по построению.

Таким образом, получено, что записанные автоматы A, B, C удовлетворяют определению наследования, причем автоматы A, B есть родителями, а автомат C - наследником. Еще раз отметим, проблемы множественного наследования от одного и того же класса,

```
class C : public A, public A {}
```

не представляет особых трудностей, если в качестве оператора разрешения области видимости :: языка C++ пользоваться полным или частичным путем из имен классов родителей, для указания функций заданного родителя. Естественно, в случае множественного наследования из одного и того же класса необходимо давать имя - синоним для каждого вхождения такого класса. Чтото вроде :

```
class C : public A, public A as B {} ;
```

Выражения над автоматами и инкапсуляция

Наследование автоматов, по сути, основано на декартовом произведении их типов и поэтому класс

```
class A        { public: int a; };
class C : public A, public A as B { float x; } ;
```

например, может записываться как C =

A $\times$ A as B $\times$ (Real,

$\{a_1$ = A.a, a = B.a, x = λ (a, b, x, val) : A.Int $\times$ B.Int $\times$ Real $\times$ Real :- (a,b,val)$\}$,

$\{a_1$ = A.a, a = B.a, x.get = λ (a, b, val) : A.Int $\times$ B.Int $\times$ Real :- val$\}$

)

Где функции состояния и выхода A.a и B.a определены согласно определения наследования в классе потомка C из соответствующих функций состояния и выхода A.a (класса родителя A) и B.a (класса родителя B). Необходимо еще раз отметить, что в случае однозначного именования, вообще говоря, незачем их специально именовать, Функция класса наследника по умолчанию получает имя от функции класса родителя. Далее, добавление функции для присвоения значения и извлечения значения для новой компоненты может происходить в сокращенной форме. По системе обозначений из [9] обозначить пару x, x.get можно , например, в следующем виде: x.get : Real $\leftrightarrow$ x. Общий вид - v : T $\leftrightarrow$ v, для некоторого типа T. Первое вхождение v обозначает функцию выхода v : Q $\rightarrow$ T, второе вхождение - функцию состояния v : Q $\times$ T $\rightarrow$ Q. Любая из функций может быть пропущена. Введенные обозначения позволяют сократить определение класса C до минимума:

C = A $\times$ A as B $\times$ (x : Real $\leftrightarrow$ x, $\{a_1$ = A.a$\}$, $\{a_1$ = A.a$\}$)

Инкапсуляция функций может выражаться через теоретико- множественную операцию разности множеств. Таким образом наследование

```
class A        { public: int a; };
class C : A, public A as B { public: float x; } ;
```

может быть записано как

C = A $\times$ A as B $\times$ (x : Real $\leftrightarrow$ x, $\{a_1$ = A.a$\}$, $\{a_1$ = A.a$\}$) $\setminus$ (, $\{$ $a_1\}$, $\{a_1$ $\}$)

Внутри анонимного класса (определенного слева от символа $\setminus$) можно пользоваться функциями a_1, а в самом классе C функций C.a_1 не существует.

Полиморфизм

Как видно из обсуждения наследования и инкапсуляции - эти два термина имеют отношения к конструированию автоматов. В случае полиморфизма ситуация другая, так как он для конструирования классов не используется.

Замечание

Как отмечается в монографии [8] :

"Традиционный"полиморфизм заключается в том, что вместо декларированного объекта определенного класса фактически мо-

гут использоваться экземпляры любых производных классов от заданного. Полиморфизм "по интерфейсу"означает, что можно заменить локальный объект на любой другой, в котором существуют внешние переменные, соответствующие по идентификатору и типу значения внешним переменным заменяемого объекта.

Или то же в работе [11]

Given classes C and D where D is a subclass of C, then an instance of class D (d) may be used as an instance of class C. This is referred to as (subtyping) polymorphism.

Ограничимся "традиционным"полиморфизмом.

Полиморфизм функций автомата родителя и автомата наследника

Пусть есть автомат родитель A и автомат наследник B. В обоих автоматах есть функция с одинаковым именем f. Для определенности, функцию из автомата потомка обозначим как f_B. Будем говорить, что функции f и f_B находятся в отношении полиморфизма, если

- В случае функций состояния ($f_B : \mathrm{Q(B)} \times \mathrm{X} \to \mathrm{Q(B)}$

и f: Q(A) $\times$ X $\rightarrow$ Q(A)) не выполняется условие наследования функций состояния (см. рис. 1):

$f_B \neq (\pi_{Q_l}, \text{f}\circ (\pi_{Q(A)}, \pi_X), \pi_{Q_r})$.

- В случае функций выхода (f_B : Q(B) $\rightarrow$ Y и f: Q(A) $\rightarrow$ X) не выполняется условие наследования функций выхода (см. рис. 2):

  $f_B \neq \text{f}\circ \pi_{Q(A)}$.

Полиморфизм функций автоматов наследников

Пусть автомат A является родителем автоматов B и C. Пусть во всех трех классах существует функция с одним именем f (как и раньше обозначения f_B, f_C нужны для различения этих функций). Функции f_B и f_C находятся в отношении полиморфизма относительно класса A, если

f_B и f находятся в отношении полиморфизма и f_C и f находятся в отношении полиморфизма.

Полиморфизм множеств функций автоматов-наследников

Если у классов B и C наследующих класс A, найдутся имена функций F = $\{ f_1, ..., f_n \}$, находящихся в отношении полиморфизма относительно класса

A, то классы B и C находятся в отношении полиморфизма относительно класса A и множества имен функций F.

Отношение полиморфизма относительно A и множества F позволяет записывать шаблон некоторой функции f общей для всех наследников класса A, если в f используются только функции с именами из множества F.

Рассмотрим следующий пример. В примере через Real : Int $\rightarrow$ Real обозначена функция для преобразования целого числа в плавающее:

- P = (Bool, {x = λ (b, v) : Bool $\times$ Int :- if v = 0 then false else true end },)
- I = P $\times$ (Int, {x = λ (b, i, v) : Q(I) $\times$ Int :- (b, v) },)
- R = P $\times$ (Real, {x = λ (b, r, v) : Q(R) $\times$ Int :- (b, Real(v)) },)
- и задана некоторая функция y которой в качество области определения вместо типа появляется класс P:

  f : P $\times$ Int $\rightarrow$ P f(p, v) is P.x (p, v)

Такое определение функции означает, что в этом примере задано 3 функции f:

- f : Q(P)× Int → Q(P) f(p, v) is P.x (p, v)
- f : Q(I)× Int → Q(I) f(p, v) is I.x (p, v)
- f : Q(R)× Int → Q(R) f(p, v) is R.x (p, v)

В каждой из которых используется функция x соответствующего типа для соответствующего автомата. И, естественно, запись f(p, 1), для p∈Q(P) и для p∈Q(I) означает применение различных функций. Таким образом, разделение понятия тип и класс позволяет согласовать полиморфизм и проверку типов.

В примере классы P, I, R находятся в отношении полиморфизма относительно класса P и множества { x }.

Замечание

> В тоже время в классе I (равно как и R), кроме его собственной функции x: Q(I) × Int → Q(I), по определению наследования существует функция I.P.x : Q(I) × Bool → Q(I), полученная из родительского автомата I.P.x = λ (b,i,v) : Q(I) :- (P.x(b,v),i).

ЛИТЕРАТУРА

[1] А. Ахо, Д. Хопкрофт и Д. Ульман, *Структуры данных и алгоритмы* (Издательский дом Вильямс, Москва, 2003).

[2] J. Bowen, *Formal Specification and Documentation using Z: A Case Study Approach* (International Thomson Computer Press, 1996), http://www.jpbowen.com/pub/zbook.pdf.

[3] The RAISE Method Group, *The RAISE Development Method* (Prentice Hall International Limited, London, 1995), ftp://ftp.iist.unu.edu/pub/RAISE/method_book/book.zip.

[4] Г. В. Костин, *Процесс разработки ПО и ЯП* (Русский BugTraq, 2003), http://bugtraq.ru/library/programming/languages.html.

[5] Под ред. М. А. Арбиба, *Алгебраическая теория автоматов, языков и полугрупп* (Статистика, Москва, 1975).

[6] D. J. Armstrong, *The Quarks Object-Oriented Development* COMMUNICATIONS OF THE ACM **49** (2), 123 (2006).

[7] Wikipedia, *Object-oriented programming* (2006), http://en.wikipedia.org/wiki/Object-oriented_programming.

[8] Ю. Б. Колесов, *Объектно-ориентированное моделирование сложных динамических систем* (СПбГПУ, Санкт-Петербург, 2004), http://www.exponenta.ru/educat/systemat/kolesov/index.asp.

[9] H. D. Vang, C. George, T. Janowski and R. Moore, *Specification Case Studies in RAISE* (Springer, London, 2002), ftp://ftp.iist.unu.edu/pub/RAISE/case_studies/case_studies_book.zip.

[10] A. G. Piskunov and V. L. Ilyuhin, *Использование языков формальных спецификаций при проектировании реляционных баз данных*, http://users.iptelecom.net.ua/ãgp1/ru/gsau.html.

[11] J. Shield, *Towards an Object-Oriented Refinement Calculus*, PhD Thesis (The University Of Queensland, Australia, 2001), http://www.itee.uq.edu.au/s̃se/SVRC-TRS/tr98-20.ps.gz.

[12] M. Abadi, L. Cardelli and R. Viswanathan, *An Interpretation of Objects and Object Types*, *in Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, St. Petersburg Beach, Florida, USA, 1996* (ACM Press New York, NY, USA, 1996), pp. 396-409, http://citeseer.ist.psu.edu/abadi96interpretation.html.

[13] G. P. Smith, *An Object-Oriented Approach to Formal Specification*, PhD Thesis (The University Of Queensland, Australia, 1992), http://www.itee.uq.edu.au/s̃mith/papers/thesis.pdf.

[14] T. S. von Werthenstein, *Object-Oriented Programming in Explicit Mathematics: Towards the Mathematics of Objects*, PhD Thesis (University of Bern, 2001), http://citeseer.ist.psu.edu/451899.html.

[15] С. Зыков, *Современные языки программирования и .NET. Основы объектно-ориентированного подхода. Курс лекций* (Изд-во МИФИ (ГУ), Москва, 2003), http://www.ict.edu.ru/ft/005130/index.html.

[16] Р. А. Александрян и Э. А. Мирзаханян, *Общая топология. Учебное пособие для вузов* (Высшая школа, Москва, 1979).

[17] Кафедра системного программирования МГУ, *Методы Формальной Спецификации Программ*, http://www.ispras.ru/ RedVerst/RedVerst/Lectures and training courses/MSU course Formal specification of software/RMain.html.

[18] Е. А. Кузьменкова и А. К. Петренко, *Формальная спецификация программ на языке RSL*, http://www.ergeal.ru/archive/cs/rsl/index.htm.

[19] Ю. И. Манин, *Вычислимое и невычислимое* (Советское радио, Москва, 1980).

[20] L. Cardelli, *A Semantics of Multiple Inheritance* Information and Computation **76**, 138 (1988).

[21] К.Дейт, *Введение в системы баз данных*, 6е изд. (Диалектика, Киев, 1998).