ФОРМАЛИЗАЦИЯ ПАРАДИГМЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

ПИСКУНОВ А.Г.

13 октября 2007 г.

АННОТАЦИЯ

Документ представляет собой попытку формализовать основные понятия обьектно - ориентированного программирования. 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).

В работе существенную помощь оказали О.Головко и М.Николаев.

Никто не может изгнать нас из рая, который создал Кантор. (Д. Гильберт)

1 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 [5] , языков спецификаций [11] и заканчивая сложными современными технологиями [14] в которых используются автоматическое доказательство соответствия следующего шага предыдущему. Вслед за автором [8] 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 пониматься 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 [5] и очевидное соображение, что абстракцией от пары данные + программы может быть только пара значения + функции (то есть, абстрактный автомат [4]), для абстрактных типов данных не было замечено простой формальной математической модели.

Например, определения из [12] , [21] мало похожи на математику. С другой стороны, в монографии [7] 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 [14] , [13] 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 ([19]).

Кроме того, смотри [16] , [18] , [15] , [20] , [9] .

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 [1] ;
- либо взяты из языка формальных спецификаций RSL [13] . Русскоязычные ресурсы по языку RSL: [2] , [3] ;
- либо это обозначение операции диагонального призведения (операции соединения) из набора элементарных операций над частичными функциями. см. [6] .

Примеры будут приводиться в аппликативном (функциональном) стиле, то есть без использования переменнных и, естественно, ссылок на них.

2 ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И РЕЗУЛЬТАТЫ

2.1 Основные определения

Полувычислимая функция

(см. [6]) 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: $x \notin \mathrm{dom}\ f$.

Автомат Мили

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

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

2.2 Утверждение

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

Доказательство смотри, например, в конце раздела 2.2 [4] .

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 π_{Xj} : $X_1 \times ... \times X_k$ $\rightarrow$ X_j заданное как π_{Xj} (x_1, ..., x_k) = x_j, где 1 $\leq$ j $\leq$ k и $x_j \in X_j$ будет называться проекцией.

Обсуждение свойств диагонального произведения, композиции и проекции можно посмотреть в [1] .

2.3 Утверждение

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

Доказательство. Пусть для f выполняется f(t) = (x, y), t $\in$ T , x $\in$ X, y $\in$ Y. Тогда, по определению проекции, (x,y) = (π_X(x,y), π_Y(x, y)) = (π_X(f (t)),π_Y(f(t)) = ($\pi_X \circ$ f (t) ,$\pi_Y \circ$ f(t)) = ($\pi_X \circ$ f, $\pi_Y \circ$ f)(t) = f(t). То есть, f есть диагональное произведение отображений $\pi_X \circ$ f и $\pi_Y \circ$ f.

2.4 Следствие

Любая функция f: T $\rightarrow$ Q $\times$ 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 $\rightarrow$ Q, а множество значений второй, во втором сомножителе - f_o : T $\rightarrow$ 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 [6] . 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 × X → Q будем называть функциями состояния, а функции f_o : Q → Y функцией выхода.

Замечание

Похоже что, в работе [14] (см. стр 47, 95) функции состояния обозначаются термином generator, функции выхода - observer, а множество Q - типом интересов.

2.5 Утверждение

Пусть есть множество Q. Два любых набора функций:

- состояния: { g_i : Q × X_i → Q | i ≤ n }
- выхода: { f_j : Q → Y_j | j ≤ m }

для некоторых множеств Q, X_i, i ≤ n, Y_j, j ≤ m задают автомат Мура.

Доказательство. Пусть множество N = {1, ..., n }. Тогда зададим функцию переходов
δ : Q × N × X_1 × ... × X_n → Q следующим образом:
δ (q, i, x_1, ... , x_n) ≡
case i of
1 → g_1(q, x_1),
2 → g_2(q, x_2),

...
n $\rightarrow$ g_n(q, x_n)
end

И функцию выходов λ : Q $\rightarrow$ $Y_1 \times ... \times Y_m$ следующим образом
λ (q) $\equiv$ (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.

2.6 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 2.6 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 $\in$ A, вместо a $\in$ 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.

Любой кортеж a определенная как a $\in$ 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 $\rightarrow$ A, то кортеж a будет называться процессом.

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

Замечание

Толкование объекта как кортеж очень близко к толкованию объекта как запись в работе [17] , в нашем случае роль меток значений играют функции состояния и выхода.

3 НАСЛЕДОВАНИЕ, ИНКАПСУЛЯЦИЯ, ПОЛИМОРФИЗМ

Ассоциативность

Отметим (см., например, [10] , 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 $f : X \rightarrow T$, можно рассматривать как функцию с сигнатурой $f: A \times B \rightarrow T$.

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 $T_1 \times .... \times T_n \rightarrow T$, 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 $Int \times Int \rightarrow Int$.

3.1 Наследование автоматов

Пусть есть автомат

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

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:

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

  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 $\rho : \{\text{i} \leq m_1\} \rightsquigarrow \{\text{i} \leq m_2$ }, такое что $f_{2,\rho(i)} = f_{1,i} \circ \pi_{Q_1}$, 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 выполняется 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 [16] :

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 (Наследование это разделение(общее использование) атрибутов между классом и его потомками. Множественное наследование встречается когда потомок наследует атбибуты более чем у одного класса. Простое наследование встречается когда потомок наследует атрибуты у одного класса).

где под атрибутом может пониматься как переменная так и метод.

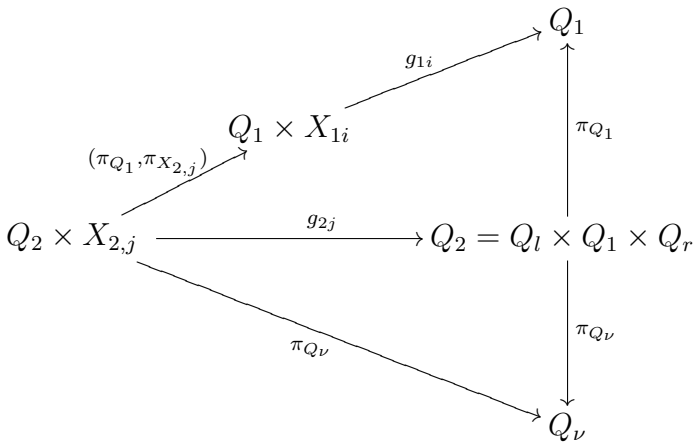

Рис. 1: Условие наследования функций состояния. $\nu \in \{l, r\},\ X_{2,j} \stackrel{def}{=} X_{1,i}$, $Q_2 \stackrel{def}{=} Q_l \times Q_1 \times Q_r$

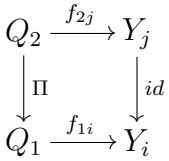

Рис. 2: Условие наследования для функций выхода

Далее, условия наследования для фукнций состояния и функций выхода для каждой пары автомата родителя A с типом Q_1 и автомата потомка B с типом Q_2 позволят задать операторы наследования функций состояния (ς похожа на S, θ похожа на O):

- для любой g функции состояния из A через $\varsigma_{B:A}$(g) обозначим (π_{Q_l}, g$\circ$ (π_{Q_1}, π_X), π_{Q_r});
- для любой f функции выхода выхода из A - $\theta_{B:A}$ (f) = f$\circ$ π_{Q_1}.

Если понятно о каких автоматах идет речь, то один или оба индекса B:A будем опускать.

3.2 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),

$a_2 = \lambda$ (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 Q_2 автомата наследника C, в данном случае равен A.Int $\times$ B.Int и содержит в качестве сомножителя Q_1 тип любого из автоматов родителей как A.Int, так и B.Int;

- Если в качестве родителя рассматривать класс A, то есть, под типом Q_1 понимать A.Int, а под набором функций состояния $\{g_{1,1}\}$ 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, и функцией $g_{2,1}$ из определения в данном примере является функция a_1:

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

  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 {} ;
```

3.3 Выражения над автоматами и инкапсуляция

Наследование автоматов, по сути, основано на декартовом произведении их типов и, поэтому для наследования можно использовать символ прямого произведения, переопределив операцию прямого произведения на автоматах. Тогда

класс

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

например, может записываться как C =

A × A as B × (Real,

{C.A.a = ς (A.a) as a, C.B.a = ς (B.a) as a_1, x = λ (a, b, c, val) : A.Int × B.Int × Real × Real :- (a,b,val)},

{C.A.a = θ (A.a) as a, C.B.a = θ(B.a) as a_1, x.get = λ (a, b, val) : A.Int × B.Int × Real :- val}

)

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, описывать функции присвоение значиния и извлечения, таких как x можно в сокращенной форме. По системе обозначений из [13] (раздел Variant Definition) обозначить пару x (она же x.put, x.get) можно, например, в следующем виде: x : Real ↔ x. Естестенно было бы использовать это обозначение в месте предназначенном для описания типа. Общий вид - f : T ↔ f, для некоторого типа T и автомата с множеством состояний Q. Первое вхождение f (именно f: T) обозначает функцию выхода f : Q → T, второе вхождение (T↔f) - функцию состояния f : Q × T → Q. Кроме того, можно по умолчанию применять операторы наследования ς и θ к функциям из классов родителей.

Введенные обозначения позволяют сократить определение класса C до минимума:

C = A × A as B × (x : Real ↔ x, { B.a as a_1}, { B.a as a_1})

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; } ;
```

в котором запрещается доступ к функциям из класса A, может быть записано как

C = A × A as B × (x : Real ↔ x, { B.a as a_1 }, { B.a as a_1 }) \ (, { a} , {a })

Внутри анонимного класса (определенного слева от символа \ ) можно пользоваться функциями a, a в самом классе C функций a (полное имя C.A.a) не существует.

3.4 Полиморфизм

Рассмотрим определения полиморфизма. Как отмечается, например, в монографии [7] :

"Традиционный"полиморфизм заключается в том, что вместо декларированного объекта определенного класса фактически могут 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 определение полиморфизма в работе [16] :

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. (Пусть даны классы C и D, причем D есть наследником класса C, тогда экземпляр класса D (d) может быть использован как экземпляр класса C. Это называется полиморфизмом.)

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. Будем говорить, что функции A.f и B.f находятся в отношении полиморфизма, если

- В случае функций состояния (B.f : Q(B) × X → Q(B) и A.f: Q(A) × X → Q(A)) не выполняется условие наследования функций состояния (см. рис. 1):

  B.f ≠ (π_{Q_l}, A.f∘ ($\pi_{Q(A)}$,π_X), π_{Q_r}) .
- В случае функций выхода (B.f : Q(B) → Y и A.f: Q(A) → X) не выполняется условие наследования функций выхода (см. рис. 2):

  B.f ≠ f∘ $\pi_{Q(A)}$.

Полиморфизм функций автоматов наследников

Пусть автомат A является родителем автоматов B и C. Пусть во всех трех классах существует функция с одним именем f. Функции B.f и C.f находятся в отношении полиморфизма относительно класса A, если

B.f и A.f находятся в отношении полиморфизма и B.f и A.f находятся в отношении полиморфизма.

Полиморфизм множеств функций автоматов-наследников

Если у классов B и C, наследующих класс A, найдутся функции 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 функций F.

Для использования полиморфизма введем отображение M, ставящий в соответствие некоторому объекту o (т.е. кортежу из множества состояний некоторого класса) его класс A, M(o) = A ⇔ o ∈ A.

Рассмотрим пример. Так как выбран аппликативный стиль записи, то для демонстрации того, как для "декларированного объекта определенного класса фактически могут использоваться экземпляры производных классов"нельзя использовать привычный пример массива ссылок на объекты класса родителя. Придется рассмотреть пример, где в качестве параметра функции передаются объекты класса родителя

В примере через Real : Int → Real обозначена функция для преобразования целого числа в плавающее, а в автоматах заданы только функции состояния и не заданы функции выхода:

- P = (Bool, {x = λ (b, v) : Bool $\times$ Int :- if v = 0 then false else true end }, $\emptyset$)
- I = P $\times$ (Int, {x = λ (b, i, v) : Q(I) $\times$ Int :- (b, v) }, $\emptyset$)
- R = P $\times$ (Real, {x = λ (b, r, v) : Q(R) $\times$ Int :- (b, Real(v)) }, $\emptyset$)
- пусть явно задана некоторая функция у которой в качестве области определения вместо типа появляется класс P и используется функция x :Bool $\times$ Int $\rightarrow$ Bool из класса P:

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

  (Данное определение функции читается так: задана функция f с сигнатурой f : P $\times$ Int $\rightarrow$ P для которой запись f (p, v) для любых p $\in$ P и v $\in$ Int определяется выражением после символа $\equiv$ (есть по определению), в нашем случае - M(p).x(p, v)). Тогда, запись f(s, 1), для некоторого кортежа s $\in$ P будет означать P.x(s,1), а для кортежа s $\in$ I означать I.x(s,1), а для кортежа s $\in$ R - R.x(s,1). То есть, вместо декларированного кортежа - параметра класса P фактически использовались кортежи классов P, I и R. Отметим что в данном примере, классы P, I, R находятся в отношении полиморфизма относительно класса P и множества из функции {x}.

Замечание

В тоже время в классе I (равно как и в классе R), кроме его собственной функции I.x: Q(I) $\times$ Int $\rightarrow$ Q(I), по определению наследования существует функция I.P.x = $\varsigma_{I:P}$ (P.x(b,v),i) полученная из родительского автомата P.

Предметный указатель

Содержание

ССЫЛКИ

[1] Г. Кантор и др. Теория множеств. http://users.iptelecom.net.ua/~tchingiz/articles/setTeory.pdf. 3, 5

[2] Кафедра системного программирования МГУ. Методы Формальной Спецификации Программ. http://www.ispras.ru/~RedVerst/RedVerst/Lecturesandtrainingcourses/MSUcourseFormalspecificationofsoftware/RMain.html. 3

[3] А.К.Петренко Е.А. Кузьменкова. Формальная спецификация программ на языке rsl. http://www.ergeal.ru/archive/cs/rsl/index.htm. 3

[4] Под редакцией М.А. Арбиба. Алгебраическая теория автоматов, языков и полугрупп, 1998. Москва, Статистика, 1975. 2, 4

[5] А.Ахо, Д.Хопкрофт, Д. Ульман. Структуры данных и алгоритмы, 2003. Издательский дом Вильямс. 2

[6] Манин Ю.И. Вычислимое и невычислимое, 1980. http://agp1.nm.ru/arts/manin2.html. 3, 4, 6

[7] Колесов Ю.Б. Объектно-ориентированное моделирование сложных динамических систем, 2004. http://www.exponenta.ru/educat/systemat/kolesov/index.asp. 2, 14

[8] Костин Г.В. Процесс разработки ПО и ЯП, 2003. http://bugtraq.ru/library/programming/languages.html. 2

[9] Зыков С. Современные языки программирования и .NET. Основы объектно - ориентированного подхода, 2003. http://www.ict.edu.ru/ft/005130//index.html. 3

[10] К.Дейт. Введение в системы баз данных, 1998. 6-е издание. 8

[11] Jonathan Bowen. Formal specification and documentation using z: A case study approach, 2003. http://www.jpbowen.com/pub/zbook.pdf. 2

[12] Deborah J. Armstrong. The Quarks Object-Oriented Development, 2006. COMMUNICATIONS OF THE ACM, Vol 49, No 2. 2

[13] Chris George. Introduction to RAISE, 2002. http://users.iptelecom.net.ua/~agp1/arts/RAISE4.pdf. 3, 13

[14] Chris George. Introduction to RAISE. UNU-IIST report No. 249, 2002. http://users.iptelecom.net.ua/~agp1/arts/report249.pdf. 2, 3, 6

[15] Graeme Paul Smith. An Object-Oriented Approach to Formal Specification, 1992. http://www.itee.uq.edu.au/~smith/papers/thesis.pdf. 3

[16] Jamie Shield. Towards an Object-Oriented Refinement Calculus, 2001. Thesis. 3, 9, 14

[17] Luca Cardelli. A Semantics of Multiple Inheritance, 1988. Information and Computation 76, 138-164,1988. 8

[18] Martin Abadi, Luca Cardelli, Ramesh Viswanathan. An Interpretation of Objects and Object Types. 3

[19] A.G. Piskunov and V.L. Ilyuhin. The usage of formal specification languages for the design of rdbms. http://users.iptelecom.net.ua/~agp1/ru/gsau.html. 3

[20] Thomas Studer von Werthenstein. Object-Oriented Programming in Explicit Mathematics: Towards the Mathematics of Objects, 2001. Bern, 3.April 2001. 3

[21] Wikipedia. Object-oriented programming, 2006. http://en.wikipedia.org/wiki/Object-oriented_programming. 2