

Формализация парадигмы ООП: НАСЛЕДОВАНИЕ АБСТРАКТНЫХ АВТОМАТОВ

ПИСКУНОВ А.Г.

19 апреля 2010 г.

АННОТАЦИЯ

В работе термин наследование определяется на абстрактных автоматах Мили, что позволяет использовать термин автомат как синоним термина класс. Таким образом, такие основные понятия объектно - ориентированного программирования как абстрактный тип данных, класс, объект, процесс, наследование удается формально определить через прямые произведения, отношения на множествах, функции и абстрактные автоматы. Предлагаемая формализация понятий ООП существенно отличается от формализации Люка Карделли (Luca Cardelli [15]), которую можно считать общепринятой. Кроме того, представляется правильным, что хотя бы функциональные языки программирования должны иметь операцию диагонального произведения отображений (в [6] называемую соединением) из набора Черча, равно как операции проекции и композиции.

Содержание

1	ВВЕДЕНИЕ	3
2	ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И РЕЗУЛЬТАТЫ	4
2.1	Основные определения	4
2.2	Утверждение	5
2.3	Следствие	5
2.4	Утверждение	6
3	УСЛОВИЯ НАСЛЕДОВАНИЯ ФУНКЦИЙ	9
4	НАСЛЕДОВАНИЕ АБСТРАКТНЫХ АВТОМАТОВ МИЛИ	12
5	ПОСТРОЕНИЕ АВТОМАТА ПОТОМКА ПРИ ОДИНОЧНОМ НАСЛЕДОВАНИИ	13
5.1	Замечания по синтаксису	14
6	МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ	14
6.1	Замечания по синтаксису	15

1 ВВЕДЕНИЕ

Согласно Б.Мейеру

- под абстрактным типом данных будем понимать четверку - (тип; сигнатуры функций; аксиомы, связывающие функции; предусловия выполнения функций) (см. [4]);
- под классом будем понимать абстрактный тип данных (далее АД), поставляемый с возможно частичной реализацией (см. [3]), то есть, реализованный на некотором языке программирования.

Так как в определение класса неявно входит понятие языка программирования, его, более точно, надо формулировать в такой форме:

Под Λ - классом будем понимать АД, поставляемый с возможно частичной реализацией на языке программирования Λ .

Если в качестве языка реализации Λ взять язык формальных спецификаций, допускающих неявное описание функций (то есть с помощью сигнатур, аксиом и предусловий), то придется признать, что понятие АД неотлично от понятия Λ -класса. Например, в качестве такого языка можно использовать Z [10] (адаптированный под программистские потребности способ математической записи) или, более того, RSL [5] допускающий любой стиль записи программ от функционального до императивного. В [8] можно посмотреть пример алгебраического проектирования класса из метода разработки RAISE (см [13]). В этом методе, на начальных стадиях проектирования используется неявное описание функций АД, с последующим переходом к императивному стилю записи функций в таком подмножестве языка RSL, которое допускает автоматическую трансляцию RSL-кода в C++. Из чего следует эквивалентность понятий RSL-класса и C++-класса.

Таким образом, будем считать, что АД, равно как и класс, является синонимами пары (тип, множество функций). Далее в статье от пары (тип, множество функций) перейдем к тройке (множество состояний, функция состояний, функция выхода), то есть, к абстрактному автомату и зададим на множествах автоматов отношение наследования.

Добавление в статье условия наследования для функций выхода автомата Мили, позволяют обобщить определение полиморфизма введенное в [7] для автомата Мура очевидным образом.

Формализацию Люка Карделли можно посмотреть в [15] , <http://lucacardelli.name/>).

2 ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И РЕЗУЛЬТАТЫ

2.1 Основные определения

Полувывчислимая функция

(см. [6]) Функция f называется полувывчислимой, если существует программа:

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

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

Диагональное произведение отображений

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

Проекция

Отображение $\pi_{X_j} : X_1 \times \dots \times X_k \rightarrow X_j$ заданное как $\pi_{X_j}(x_1, \dots, x_k) = x_j$, 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] или [6] .

2.2 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 ее проекций на соответствующие 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, $(x, y) = (\pi_X(x, y), \pi_Y(x, y)) = (\pi_X(f(t)), \pi_Y(f(t))) = (\pi_X \circ f(t), \pi_Y \circ f(t)) = (\pi_X \circ f, \pi_Y \circ f)(t) = f(t)$. То есть, f есть диагональное произведение отображений $\pi_X \circ f$ и $\pi_Y \circ f$.

2.3 Следствие

Любая (полувычислимая) функция $h: T \rightarrow Q \times Y$ может быть получена из двух (полувычислимых) функций g и f , причем множество значений первой, содержится в первом сомножителе - Q , то есть $g: T \rightarrow Q$, а множество значений второй, во втором сомножителе - $f: T \rightarrow Y$.

Обсуждение. В доказательстве утверждения 2.2 были использованы простейшие функции - проекция и элементарные операции над функциями - композиция и диагональное произведение (иначе соединение, см. работу Манина [6, Частично рекурсивные функции]). 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 через (выводится из) более простые функции компоненты. Полувычислимость функции будем понимать как синоним возможности написать эти функции на каком либо языке программирования.

Замечание

В работе Роджерса [2, Пример: примитивнорекурсивные функции] для построения примитивнорекурсивных функций используется примитивная операция (iv) (тоже называемая композицией). Она заменяет операции композиции и диагонального произведения, используемые в работе Манина ([6, Частично рекурсивные функции]). Эта операция некоторым функциям

$f: Y_1 \times \dots \times Y_k \rightarrow Z$ и $g_1: T \rightarrow Y_1, \dots, g_k: T \rightarrow Y_k$ ставит в соответствие функцию $\lambda x: T \bullet f(g_1(x), \dots, g_k(x))$.

Как можно убедиться:

- в одном частном случае (при $k = 1$) композиция Роджерса совпадает с определением композиции Манина;
- в другом частном случае (когда f - тождественная функция, $f(x) = x$) она совпадает с определением диагонального произведения.

Здесь символ λ это не функция из определения автомата, а символ [лямбда выражения](#).

Функции состояния и выхода

Далее, для некоторых множеств Q, X, Y (причем Y нельзя представить в виде декартового произведения Q с каким либо множеством) функцию $g: Q \times X \rightarrow Q$ будем называть функциями состояния, а функции $f: Q \times X \rightarrow Y$ функцией выхода Мили (функция $f: Q \rightarrow Y$ будет называться функцией выхода Мура).

Замечание

Похоже что, в работе [12] (см. стр 47, 95) функции состояния обозначаются термином generator, функции выхода – observer, а множество Q – типом интересов.

2.4 Утверждение

Пусть есть некоторое множество Q , I обозначает множество индексов $\{1, \dots, n\}$. Тогда любое множество функций:

$\{ h_i : Q \times X_i \rightarrow Q \mid i \in I \}$ задает [автомат](#) Мили.

Доказательство. Сначала согласно 2.3 , построим два множества функций

$$G = \{ g_i : Q \times X_i \rightarrow Q \mid i \in I \}$$

$$F = \{ f_i : Q \times X_i \rightarrow Y_i \mid i \in I \}$$

Далее, по первому множеству функций зададим функцию переходов $\delta: Q \times I \times X_1 \times \dots \times X_n \rightarrow Q$ следующим образом:

$$\delta (q, i, x_1, \dots, x_n) \equiv$$

case i of
 $1 \rightarrow g_1(q, x_1),$
 $2 \rightarrow g_2(q, x_2),$
 \dots
 $n \rightarrow g_n(q, x_n)$
end

И, по второму множеству зададим функцию выходов $\lambda : Q \times I \times X_1 \times \dots \times X_n \rightarrow Y_1 \times \dots \times Y_n$ следующим образом

$$\lambda (q, i, x_1, x_2, \dots, x_n) \text{ is } (f_1(q, x_1), f_2(q, x_2), \dots, f_n(q, x_n))$$

Обозначив $I \times X_1 \times \dots \times X_n$ через X , а $Y_1 \times \dots \times Y_m$ – через Y , можно окончательно убедиться, что полученная тройка (Q, δ, λ) удовлетворяет определению автомата Мили по построению.

Класс, тип

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

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

Вследствие утверждения 2.4 классом можно называть также тройку $(Q, \{ g_i : Q \times X_i \rightarrow Q \mid i \in I \}, \{ f_j : Q \times X_j \rightarrow Y_j \mid j \in J \})$ где I, J множества индексов.

Объект, процесс

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

- в сигнатуре функции f , в частности, из множеств $f \in F_s$ или $f \in F_o$, вместо типа класса $Q(A)$, можно будет писать обозначение класса A . Например, не $f : Q \times X \rightarrow Y$, а $f : A \times X \rightarrow Y$;
- также допускается запись $a \in A$, вместо $a \in Q(A)$. Она означает, что хотя величина a является таким же кортежем 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.

Понятия объекта (процесса) бессмысленны при использовании аппликативного стиля записи программы. Возможно, имеет смысл, кортеж a определенный как $a \in A$ называть 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 можно называть процессом.

При императивном стиле записи, для абстрактного автомата A объектом (процессом) будет называться переменная, объявленная этим автоматом.

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

Ассоциативность декартового произведения

Отметим (см., например, [9], стр. 146, раздел "Естественное соединение")

Операция соединения и, следовательно, декартового произведения – ассоциативны, то есть

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

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

$$\lambda \text{ val}_1 : T_1, \dots, \text{val}_n : T_n \bullet \text{expression}(\text{val}_1, \dots, \text{val}_n)$$

будет определять функцию с сигнатурой $T_1 \times \dots \times T_n \rightarrow T$, где T - есть тип выражения $\text{expression}(\text{val}_1, \dots, \text{val}_n)$.

Например, выражение $\lambda x : \text{Int}, y : \text{Int} \bullet x + y$ определяет функцию сложения двух целых чисел с сигнатурой $\text{Int} \times \text{Int} \rightarrow \text{Int}$.

3 УСЛОВИЯ НАСЛЕДОВАНИЯ ФУНКЦИЙ

Пусть есть множества Q_l, Q_1, Q_r, Q_2 , связанные соотношением $Q_2 = Q_l \times Q_1 \times Q_r$. Пусть $q_1 \in Q_1, q_l \in Q_l, q_r \in Q_r$.

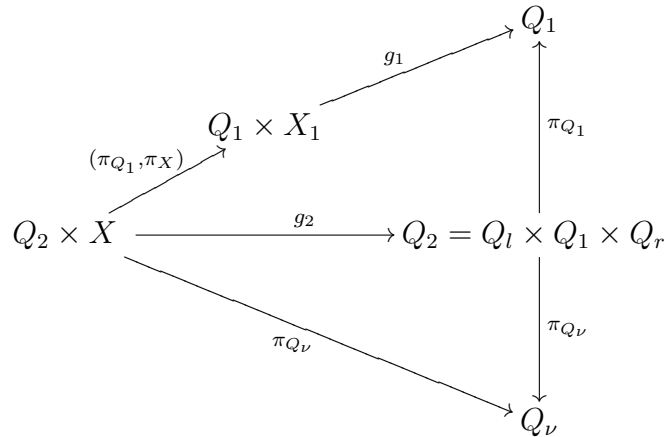
Наследование функций состояния

Функция $g_2 : Q_2 \times X \rightarrow Q_2$ находится в отношении наследования с функцией $g_1 : Q_1 \times X \rightarrow Q_1$ если

$$g_2 = \lambda (q_l, q_1, q_r, x) : Q_2 \times X \bullet (q_l, g_1(q_1, x), q_r), \text{ где } x \in X,$$

то есть, функция g_2 является диагональным произведением функций следующего вида

$$g_2 = (\pi_{Q_l}, g_1 \circ (\pi_{Q_1}, \pi_X), \pi_{Q_r})$$



Наследование функций состояния, $\nu \in \{l, r\}$

Наследование функций выхода Мили

Функция $f_2 : Q_2 \times X \rightarrow Y$ находится в отношении наследования с функцией $f_1 : Q_1 \times X \rightarrow Y$ если

$$f_2 = \lambda (q_l, q_1, q_r, x) : Q_2 \times X \bullet f_1(q_1, x), \text{ где } x \in X,$$

то есть, функция f_2 является композицией функций следующего вида $f_2 = f_1 \circ (\pi_{Q_1}, \pi_X)$

$$\begin{array}{ccc}
 Q_2 \times X & \xrightarrow{(\pi_{Q_1}, \pi_X)} & Q_1 \times X \\
 & \searrow f_2 & \swarrow f_1 \\
 & Y &
 \end{array}$$

Наследование функций выхода Мили

Наследование функций выхода Мура

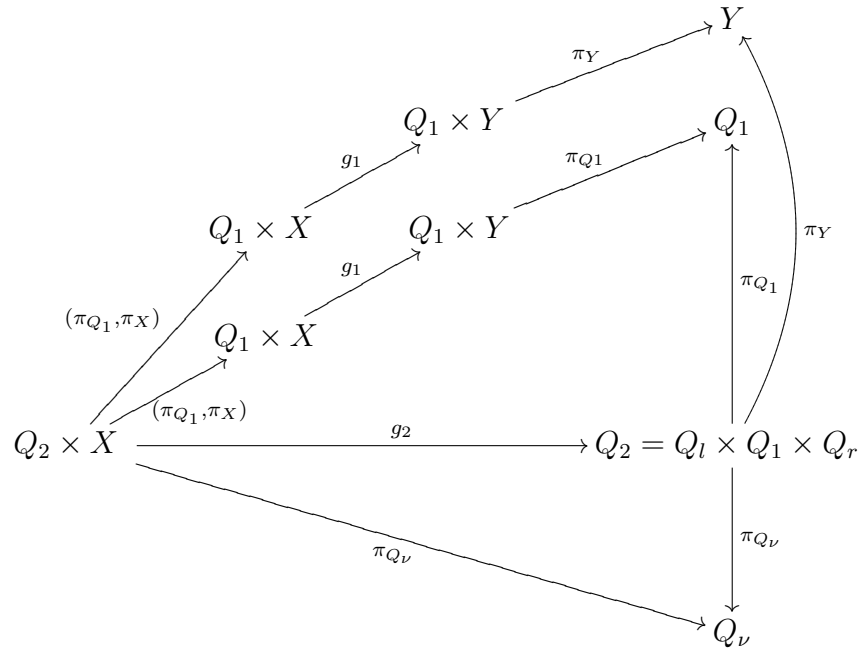
Функция $f_2 : Q_2 \rightarrow Y$ находится в отношении наследования с функцией $f_1 : Q_1 \rightarrow Y$ если $f_2 = \lambda (q_l, q_1, q_r) : Q_2 \bullet f_1(q_1)$, то есть, функция f_2 является композицией функций следующего вида $f_2 = f_1 \circ \pi_{Q_1}$

$$\begin{array}{ccc}
 Q_2 & \xrightarrow{\pi_{Q_1}} & Q_1 \\
 & \searrow f_2 & \swarrow f_1 \\
 & Y &
 \end{array}$$

Наследование функций выхода Мура

Наследование функций общего вида

Функция $g_2 : Q_2 \times X \rightarrow Q_2 \times Y$ находится в отношении наследования с функцией $g_1 : Q_1 \times X \rightarrow Q_1 \times Y$ если $g_2 = \lambda (q_l, q_1, q_r, x) : Q_2 \times X \bullet (q_l, \pi_{Q_1}(g_1(q_1, x)), q_r, \pi_Y(g_1(q_1, x)))$, где $x \in X$, то есть, функция g_2 является диагональным произведением функций следующего вида $g_2 = (\pi_{Q_l}, \pi_{Q_1} \circ g_1 \circ (\pi_{Q_1}, \pi_X), \pi_{Q_r}, \pi_Y \circ g_1 \circ (\pi_{Q_1}, \pi_X))$



Наследование функций общего вида, $\nu \in \{l, r\}$

Замечание

В определении наследования функций общего вида использовались разные функции проекции $\pi_{Q_1} : Q_1 \times X \rightarrow Q_1$ и $\pi_{Q_1} : Q_1 \times Y \rightarrow Q_1$

Замечание 2

Обсуждение наследования функций общего вида не являлось целью статьи в связи с Утверждением 2.2 . Определение было приведено, чтобы показать возможность обобщения наследования двоек произвольного вида: $(Q, \{ f_i : Q \times X_i \rightarrow Q \times Y_i \mid i \leq n \})$.

Видимая и истинная сигнатуры функций

Пусть объект q и функция f принадлежат автомату A . В силу сложившейся программистской традиции будем записывать выражение

$f(q, x)$ как $q.f(x)$

сигнатуру $f: Q \times X \rightarrow Q \times Y$ назовем истинной. Сигнатуру той же функции, записанную в сокращенной форме $f: X \rightarrow Y$ будем называть видимой сигнатурой.

4 НАСЛЕДОВАНИЕ АБСТРАКТНЫХ АВТОМАТОВ МИЛИ

Пусть есть автомат $B =$

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

и автомат $A =$

$(Q_1, \{ g_{1,i} : Q_1 \times X_{1,i} \rightarrow Q_1 \mid i \leq n_1 \}, \{ f_{1,j} : Q_1 \rightarrow Y_{1,j} \mid j \leq m_1 \})$.

Будем говорить, что автомат B находится в отношении наследования с автоматом A (наследует автомат A) (Класс B наследует класс A), если

- Q_2 есть декартово произведение $Q_l \times Q_1 \times Q_r$, где Q_l и/или Q_r могут быть пустыми множествами;
- существует частично определенное инъективное отображение $\tau : \{i \mid i \leq n_1\} \rightsquigarrow \{i \mid i \leq n_2\}$ такое что, для любого $i \leq n_1$ функции состояния $g_{2,\tau(i)}$ и $g_{1,i}$ находятся в отношении наследования;
- существует частично определенное инъективное отображение $\rho : \{i \mid i \leq m_1\} \rightsquigarrow \{i \mid i \leq m_2\}$, такое что функции выхода $f_{2,\rho(i)}$ и $f_{1,i}$, $i \leq m_1$ находятся в отношении наследования.

Замечание

Отображение $f : X \rightarrow Y$ называется инъективным, если из $f(x_1) = f(x_2)$ следует $x_1 = x_2$.

Автомат A будет называться родителем, автомат B - потомком.

Далее, условия наследования для функций 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 (ζ похожа на S , θ похожа на O):

- для любой g функции состояния из A через $\varsigma_{B:A}(g)$ обозначим функцию $(\pi_{Q_l}, g \circ (\pi_{Q_1}, \pi_X), \pi_{Q_r})$;
- для любой g функции общего вида из A через $\varsigma_{B:A}(g)$ функцию $(\pi_{Q_l}, \pi_{Q_1} \circ g \circ (\pi_{Q_1}, \pi_X), \pi_{Q_r}, \pi_Y \circ g \circ (\pi_{Q_1}, \pi_X))$;
- для любой f функции выхода из A аналогично - $\theta_{B:A}(f) = f \circ (\pi_{Q_1}, \pi_X)$ (функция выхода Мили),
 $\theta_{B:A}(f) = f \circ \pi_{Q_1}$ (функция выхода Мура).

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

5 ПОСТРОЕНИЕ АВТОМАТА ПОТОМКА ПРИ ОДИНОЧНОМ НАСЛЕДОВАНИИ

Одиночное наследование (single inheritance) (см. [14]) встречается когда класс потомок наследует атрибуты у одного класса. Определим следующую процедуру для построения автомата потомка $B = \{ Q_2, G_2, F_2 \}$ по автомату - родителю $A = \{ Q_1, G_1, F_1 \}$:

- зададим, возможно пустой, сомножитель Q_r . Если сомножитель Q_r не пуст, то в силу программисткой традиции, будем перечислять типы его компонент справа от автомата A ;
- Для любой g – функции состояния автомата A потребуем существование ее образа $\varsigma_{B:A}(g)$ во множестве функций состояния автомата B :
 $G_2 \supseteq \{ B.A.g \mid B.A.g : Q_2 \times X_g \rightarrow Q_2 \bullet B.A.g = \varsigma_{B:A}(g) \wedge g \in G_1 \}$
- Аналогично, для каждой f – функции выхода автомата A потребуем существование ее образа $\theta_{B:A}(f)$ во множество функций выхода:
 $F_2 \supseteq \{ B.A.f \mid B.A.f : Q_2 \times X_f \rightarrow Y_f \bullet B.A.f = \theta_{B:A}(f) \wedge f \in F_1 \}$.

Если для некоторой функции, обозначенной как $B.A.h$, в множествах G_2 или F_2 нет функции, обозначенной как h , с такой же сигнатурой, то в обозначении $B.A.h$ можно опускать обозначения автоматов $B.A$. В противном случае будем в явной форме давать синоним, как это принято в SQL. Например, $\theta_{B:A}(f)$ as f_1 .

Построенный таким образом автомат B будет находится в отношении наследования с автоматом A по построению. Отметим, что сама процедура позволяет строить только частные случаи пар автоматов находящихся в отношении наследования.

5.1 Замечания по синтаксису

Для доступа к атрибуту x класса некоторого класса, то есть, в наших терминах к соответствующему элементу из некоторого кортежа используются функция состояния $x: Q \times T \rightarrow Q$ и функция выхода $x: Q \rightarrow T$. Записывать эти функции можно по системе обозначений из [11] (раздел Variant Definition) в виде $x: T \leftrightarrow x$. Первое вхождение x (а именно $x: T$) обозначает функцию выхода $x: Q \rightarrow T$, второе вхождение ($T \leftrightarrow x$) - функцию состояния $x: Q \times T \rightarrow Q$. В таком случае нет необходимости упоминание этих функций во множествах функций состояния и выхода. Остальные функции автомата достаточно задавать с указанием видимой сигнатуры.

Рассмотрим пример класса в терминах языка C++:

```
class A { public: int a; public void g (int b) { a*=b;} };
```

Запись этого класса могла бы быть, например, такой

$$A = (a: \text{Int} \leftrightarrow a, \{ g = \lambda b: \text{Int} \bullet a(\text{this}, a(\text{this}) * b) \}, \emptyset);$$

причем λ выражение, определяющее функцию g , с указанием истинной сигнатуры выглядит следующим образом

$$g = \lambda \text{this}: \text{Int}, b: \text{Int} \bullet a(\text{this}, a(\text{this}) * b)$$

Оператор наследование из следующего примера:

```
class B : A { public: float x;}
```

мог бы выглядеть как то так: $B = A \times (x:\text{Float} \leftrightarrow x, \emptyset, \emptyset)$

6 МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ

Пусть есть автоматы $A = (Q(A), G_A, F_A)$, $B = (Q(B), G_B, F_B)$, C . Причем автоматы C и A находятся в отношении наследования, и автоматы C и B находятся в отношении наследования. Тогда автоматы C и A , B находятся в отношении множественного наследования, если

- множества состояния $Q(A)$ и $Q(B)$ являются прямыми сомножителями множества $Q(C)$;
- для любой пары $g_A \in G_A$ и $g_B \in G_B$ выполняется $\varsigma_{C:A}(g_A) \neq \varsigma_{C:B}(g_B)$;
- для любой пары $f_A \in F_A$ и $f_B \in F_B$ выполняется $\theta_{C:A}(f_A) \neq \theta_{C:B}(f_B)$.

По видимому, второе и третье условие выполняются являются следствием первого, но их явное выписывание представляется полезным.

Обобщение формулировки множественного наследования с двух родителей на произвольное число не представляется затруднительным.

6.1 Замечания по синтаксису

Рассмотрим пример множественного наследования в терминах языка C++:

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

который в наших терминах записывается следующим образом

$$A = (a: \text{Int} \leftrightarrow a, \emptyset, \emptyset)$$

$$B = (a: \text{Int} \leftrightarrow a, \emptyset, \emptyset)$$

$$C = A \times B$$

Выражение $a((x,y), z)$ для некоторых $x, y, z : \text{Int}$ будет считаться применением функции состояния $a: (\text{Int} \times \text{Int}) \times \text{Int} \rightarrow (\text{Int} \times \text{Int})$, наследованной из класса A (так как это первая встреченная слева функция, среди функций с одинаковой сигнатурой). Таким образом, будем считать, что она является сокращением полной записи $C.A.a((x,y), z)$. Для обращения ко второй функции надо либо явно ее задать $C.B.a((x,y), z)$ или объявить синоним. Например, в такой форме

$$C = A \times B (a \text{ as } a1)$$

Также, представляется целесообразным вводить синонимы для классов. Тогда аналогом C++ текста

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

МОЖЕТ БЫТЬ ТЕКСТ

$A = (a: \text{Int} \leftrightarrow a, \emptyset, \emptyset)$

$C = A \times A \text{ as } B (a \text{ as } a1).$

Предметный указатель

Автомат , 12
Класс, 12
ассоциативность, 8
автомат, 4, 7
автомат , 6, 12
диагональное произведение отображений,
4
функция состояния , 6
функция выхода, 6
инъективное отображение, 12
класс, 7
лямбда выражение, 8
лямбда выражения., 6
объект, 8
процесс, 8
тип, 7
тип, 8

ССЫЛКИ

- [1] Г. Кантор и др. Теория множеств. <http://agp1.hx0.ru/articles/setTheory.pdf>.
- [2] Х.Роджерс. Теория рекурсивных функций и эффективная вычислимость. (Мир, Москва, 1972).
- [3] Бертран Мейер. Основы объектно-ориентированного программирования. Статические структуры: классы. <http://www.intuit.ru/department/se/oopbases/7/>.
- [4] Бертран Мейер. Основы объектно-ориентированного программирования. Абстрактные типы данных (АТД). <http://www.intuit.ru/department/se/oopbases/6/10.html>.
- [5] А.К.Петренко Е.А. Кузьменкова. Формальная спецификация программ на языке rsl. <http://www.ergeal.ru/archive/cs/rsl/index.htm>.
- [6] Манин Ю.И. Вычислимое и невычислимое, 1980. <http://agp.hx0.ru/arts/manin2.djvu>.
- [7] А.Г. Пискунов. Формализация парадигмы объектно-ориентированного программирования, 2007. <http://www.realcoding.net/article/view/4570>.
- [8] А.Г. Пискунов. The RAISE Method Group: АЛГЕБРАИЧЕСКОЕ ПРОЕКТИРОВАНИЕ КЛАССА, 2007. <http://www.realcoding.net/article/view/4538>.
- [9] К.Дейт. Введение в системы баз данных, 1998. 6-е издание.
- [10] Jonathan Bowen. Formal Specification and Documentation using Z: A Case Study Approach, 2003. <http://www.jpbowen.com/pub/zbook.pdf>.
- [11] Chris George. Introduction to RAISE, 2002. <http://agp1.hx0.ru/arts/RAISE4.pdf>.
- [12] Chris George. Introduction to RAISE. UNU-IIST report No. 249, 2002. <http://agp1.hx0.ru/arts/report249.pdf>.

- [13] The RAISE Method Group. The RAISE Development Method, 1999.
<http://agp1.hx0.ru/arts/book.pdf>.
- [14] Jamie Shield. Towards an Object-Oriented Refinement Calculus, 2001.
Thesis.
- [15] Luca Cardelli. A Semantics of Multiple Inheritance, 1988. Information
and Computation 76, 138-164,1988.