Формализация парадигмы ООП: НАСЛЕДОВАНИЕ АБСТРАКТНЫХ АВТОМАТОВ

ПИСКУНОВ А.Г.

16 октября 2007 г.

АННОТАЦИЯ

Статья посвящена обсуждению отношения наследования на множествах автоматов Мили и является продолжением работы [6] (ее полная и исправленная версия [7]), в которой

- понятие абстрактного типа данных (или, что тоже самое, понятие класса) рассматривалось как абстрактный автомат;
- были введены отношения наследования и полиморфизма на множествах автоматов Мура.

1 ВВЕДЕНИЕ

Согласно Б.Мейеру

- под абстрактным типом данных будем понимать четверку - (тип; сигнатуры функций; аксиомы, связывающие функции; предусловия выполнения функций) (см. [3]);
- под классом будем понимать абстрактный тип данных (далее АТД), поставляемый с возможно частичной реализацией (см. [2]), 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 АТД, поставляемый с возможно частичной реализацией на языке программирования Λ.

Если в качестве языка реализации Λ взять язык формальных спецификаций, допускающих неявное описание фукнций (то есть с помощью сигнатур, аксиом и предусловий), то придется признать, что понятие АТД неотличимо от понятия Λ-класса. Например, в качестве такого языка можно использовать Z [10] (адаптированный под программистские потребности способ математической записи) или, более того, RSL [4] допускающий любой стиль записи 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 АТД, 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 АТД, 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 перехода), то есть, к абстрактному автомату и зададим на множествах автоматов отношения наследования.

Добавленне в статье условия наследования для функций выхода автомата Мили, позволяют обобщить определение полиморфизма введенное в [6] для автомата Мура очевидным образом.

2 ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И РЕЗУЛЬТАТЫ

2.1 Основные определения

Полувычислимая функция

(см. [5]) 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, δ : Q $\times$ X $\rightarrow$ Q, λ : 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 , ..., 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] .

2.2 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 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.

2.3 Следствие

Любая функция h: T $\rightarrow$ Q $\times$ Y может быть выведена из двух 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. раздел Частично рекурсивные функции в [5]). 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 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, ..., n}. Тогда любое множество функций:
{ h_i : Q $\times$ X_i $\rightarrow$ Q $\times$ Y_i| i$\in$I } задает автомат Мили.

Доказательство. Сначала согласно 2.3 , постоим два множества фукций
G = { g_i : Q $\times$ X_i $\rightarrow$ Q | i$\in$I }
F = { f_i : Q $\times$ X_i $\rightarrow$ Y_i| i$\in$I }
Далее, по первому множеству функций зададим функцию переходов
δ : Q $\times$ I $\times$ X_1 $\times$ ... $\times$ X_n $\rightarrow$ Q следующим образом:
δ (q, i, x_1, ... , x_n) is
case i of
1 $\rightarrow$ g_1(q, x_1),
2 $\rightarrow$ g_2(q, x_2),
...
n $\rightarrow$ g_n(q, x_n)
end

И, по второму множеству зададим функцию выходов λ : Q $\times$ I $\times$ X_1 $\times$ ... $\times$ X_n $\rightarrow$ Y_1 $\times$ ... $\times$ Y_n следующим образом
λ (q, i, x_1, x_2, ..., x_n) is (f_1(q, x_1), f_2(q, x_2), ..., f_n(q, x_n))

Обозначив I $\times$ 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) удовлетворяет определению автомата Мили по построению.

Класс, тип

Для любого автомата Милли (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 2.4 классом можно называть также тройку (Q, $\{ g_i : Q \times X_i \rightarrow Q \mid i \leq n \}$, $\{ f_i : Q \rightarrow Y_i \mid j \leq m \}$) так как, при в построении автомата Мили не использовалось, что множества G и F имеют одинаковую мощность.

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.

Понятия объекта (процесса) бессмысленны при использовании аппликативного стиля записи программы. Возможно, имеет смысл, кортеж a определенный как a∈A называть объектом. Если во множестве функций состояния F_s существует хотя бы одна функция с пустым вторым сомножителем, то есть вида g : A → 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:

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

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

$g_2 = \lambda\ (q_l, q_1, q_r, x) : Q_2 \times X$:- $(q_l, g_1(q_1,x), q_r)$, где $x \in X$,

то есть, функция g_2 является диагональным произведение функций следующего вида

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

Наследование функций выхода Мили

Функция $f_2 : Q_2 \times \mathrm{X} \to \mathrm{Y}$ находится в отношении наследования с функцией $f_1 : Q_1 \times \mathrm{X} \to \mathrm{Y}$ если

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

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

Наследование функций выхода Мура

Функция $f_2 : Q_2 \to \mathrm{Y}$ находится в отношении наследования с функцией $f_1 : Q_1 \to \mathrm{Y}$ если

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

то есть, функция g_2 является композицией функций следующего вида $f_2 = f_1 \circ \pi_{Q_1}$

Наследование функций общего вида

Функция $g_2 : Q_2 \times \mathrm{X} \to Q_2 \times \mathrm{Y}$ находится в отношении наследования с функцией $g_1 : Q_1 \times \mathrm{X} \to Q_1 \times \mathrm{Y}$ если $g_2 =$

$\lambda\ (q_l, q_1, q_r, \mathrm{x}) : Q_2 \times \mathrm{X}$:- $(q_l, \pi_{Q_1}(g_1(q_1,\mathrm{x})), q_r, \pi_Y(g_1(q_1,\mathrm{x})))$,

где $\mathrm{x} \in \mathrm{X}$, то есть, функция g_2 является диагональным произведением функций следующего вида

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

Замечание

В определении наследования функций общего вида использовались разные функции проекции $\pi_{Q_1} : Q_2 \times \mathrm{X} \to Q_1$ и $\pi_{Q_1} : Q_1 \times \mathrm{Y} \to Q_1$

Замечание 2

Обсуждение наследования функций общего вида не являлось целью статьи в связи с Утверждением 2.2 . Определение было приведено, чтобы показать существование общего метода.

Видимая и истинная сигнатуры функций

Пусть объект q и функция f принадлежат автомату A в силу сложившейся программисткой традиции записывать выражение f(q, x) как q.f(x)

сигнатуру f: Q × X → Q × Y назовем истинной. Сигнатуру той же функции записанную в сокращенной форме f: X → Y будем называть видимой сигнатурой.

4 НАСЛЕДОВАНИЕ АБСТРАКТНЫХ АВТОМАТОВ МИЛИ

Пусть есть автомат B = $(Q_2, \{ g_{2,i} : Q_2 \times X_{2,i} \rightarrow Q_2 \mid \mathrm{i} \leq n_2 \}, \{ f_{2,i} : Q_2 \rightarrow Y_{2,i} \mid \mathrm{j} \leq m_2 \})$
и автомат A = $(Q_1, \{ g_{1,i} : Q_1 \times X_{1,i} \rightarrow Q_1 \mid \mathrm{i} \leq n_1 \}, \{ f_{1,i} : Q_1 \rightarrow Y_{1,i} \mid \mathrm{j} \leq m_1 \})$.

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 иньективное отображение $\tau : \{\mathrm{i} \leq n_1\} \rightsquigarrow \{\mathrm{i} \leq n_2 \}$ такое что, для любого $\mathrm{i} \leq n_1$ функции состояния $g_{2,\tau(i)}$ и $g_{1,i}$ находятся в отношении наследования;
- существует частично определенное иньективное отображение $\rho : \{\mathrm{i} \leq m_1\} \rightsquigarrow \{\mathrm{i} \leq m_2 \}$, такое что функции выхода $f_{2,\rho(i)}$ и $f_{1,i}$, $\mathrm{i} \leq m_1$ находятся в отношении наследования.

Автомат A будет называтся 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 функций состояния (ς похожа на S, θ похожа на O):

- для любой g функции состояния из A через $\varsigma_{B:A}$(g) обозначим функцию (π_{Q_l}, $g\circ$ (π_{Q_1}, π_X), π_{Q_r});
- для любой g функции общего вида из A через $\varsigma_{B:A}(g)$ функцию (π_{Q_l}, $\pi_{Q_1}\circ$ $g\circ$ (π_{Q_1}, π_X), π_{Q_r}, $\pi_Y\circ$ $g\circ$ (π_{Q_1}, π_X));
- для любой f функции выхода выхода из A аналогично - $\theta_{B:A}$ $(f) = f\circ$ (π_{Q_1}, π_X) (функция выхода Мили), $\theta_{B:A}$ $(f) = f\circ$ π_{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. В силу программисткой традиции, перечислив типы его компонет справа от автомата A;
- Для каждой g функции состояния автомата A во множество функций состояния автомата B добавим ее образ $\varsigma_{B:A}$(g): $G_2 \supseteq \{B.A.g | B.A.g$:$Q_2 \times X_g \to Q_2$:- B.A.g = $\varsigma_{B:A}$(g) /\ g$\in G_1$ }

- Аналогично, для каждой f функции выхода автомата A во множество функций выхода добавим ее образ $\theta_{B:A}$ (f): $F_2 \supseteq \{B.A.f | B.A.f$:$Q_2 \times X_f \to Y_f$:- B.A.f = $\theta_{B:A}$ (f) /\ f$\in F_1$ }.

Если, для некоторой функии обозначенной как B.A.h в множествах G_2 или F_2 нет второй функции обозначенной как h с такой же сигнатурой, то в обозначении B.A.h можно опускать обозначения автоматов B.A или в явной форме давать синоним, как это принято в SQL. Например, $\theta_{B:A}$ (f) as f1.

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, то есть, в наших терминах, к элементу кортежа некоторого типа T использоватся функция состояния x: Q × T → Q и функция выхода x: Q → T. Записывать эти функции можно по системе обозначений из [11] (раздел Variant Definition) в виде - x : T ↔ x. Первое вхождение x (a именно x: T) обозначает функцию выхода x : Q → T, второе вхождение (T↔x) - функцию состояния x : Q × T → Q. Остальные функции автомата необходимо явно 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: Int ↔a, { g = λ: b: Int :- a(this, a(this) * b) }, ∅);

причем, λ выражение в полной форме должно выглядеть следующим образом

λ: this: Int, b: Int :- a(this, a(this) * b)

Оператор наследование из следующего примера

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

мог бы выглядеть както так:
B = A × (x:Float ↔x, ∅, ∅)

6 МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ

Пусть есть автоматы $A = (\mathrm{Q}(A), G_A, F_A)$, $B = (\mathrm{Q}(B), G_B, F_B)$, C. Причем автоматы C и A находятся в отношении наследования, и автоматы C и B находятся в отношении наследования. Тогда автоматы C и A, C 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 $g_B \in G_B$ $\varsigma_{C:A}(g_A) \neq \varsigma_{C:B}(g_B)$;
- для любой пары $f_A \in F_A$ и $f_B \in F_B$ $\theta_{C:A}(f_A) \neq \theta_{C:B}(f_B)$

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

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: Int ↔a, ∅ , ∅)
B = (a: Int ↔a, ∅ , ∅)
C = A × B

Записть a ((x,y), z) для некоторых x, y, z :Int будет считаться применением функции, наследованной из класса A (как первого встреченного слева) то есть, являться сокращением записью C.A.a ((x,y), z). Для обращения ко второй надо либо явно ее задать C.B.a ((x,y), z) или обьявить синоним. Например, в такой форме

C = A × B (a as a1)

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

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

должен быть текст

A = (a: Int ↔a, ∅ , ∅)

C = A × A as B (a as a1)

Предметный указатель

Содержание

ССЫЛКИ

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

[2] Бертран Мейер. Основы объектно-ориентированного программирования. Статические структуры: классы. http://www.intuit.ru/department/se/oopbases/7/. 1

[3] Бертран Мейер. Основы объектно-ориентированного программирования. Абстрактные типы данных (АТД). http://www.intuit.ru/department/se/oopbases/6/10.html. 1

[4] А.К.Петренко Е.А. Кузьменкова. Формальная спецификация программ на языке rsl. http://www.ergeal.ru/archive/cs/rsl/index.htm. 2

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

[6] А.Г. Пискунов. Формализация парадигмы обьектно-ориентированного программирования, 2007. http://www.realcoding.net/article/view/4570. 1, 2

[7] А.Г. Пискунов. Формализация парадигмы обьектно-ориентированного программирования: критика определения Гради Буча, 2007. http://i.com.ua/~agp1/ru/oopFormalizm.html. 1

[8] А.Г. Пискунов. The RAISE Method Group: АЛГЕБРАИЧЕСКОЕ ПРОЕКТИРОВАНИЕ КЛАССА, 2007. http://www.realcoding.net/article/view/4538. 2

[9] К.Дейт. Введение в системы баз данных, 1998. 6-е издание. 7

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

[11] Chris George. Introduction to RAISE, 2002. http://users.iptelecom.net.ua/~agp1/arts/RAISE4.pdf. 11

[12] Chris George. Introduction to RAISE. UNU-IIST report No. 249, 2002. http://users.iptelecom.net.ua/~agp1/arts/report249.pdf. 5

[13] The RAISE Method Group. The RAISE Development Method, 1999. http://users.iptelecom.net.ua/~agp1/arts/book.pdf. 2

[14] Jamie Shield. Towards an Object-Oriented Refinement Calculus, 2001. Thesis. 10