Формализация ООП: Типы, множества и классы

А.Г.Пискунов, С.М.Петренко

30 ноября 2011 г.

АННОТАЦИЯ

В работе собраны и, надеюсь, систематизированы вопросы связанные с определениями как таких терминов как тип, подтип и класс, уточнен принцип подстановки Лисков, обращено внимание на желательность неизменения домена класса при наследовании.

Содержание

1 ВВЕДЕНИЕ

Формализм понимается в смысле Клини [12, Формальные системы, метаматематика, стр.237] и является синонимом термина 'формальная система', или 'логическая система', или формальная теория. Термины, определенные формально, не должны допускать неоднозначной трактовки. Таким образом, определение термина 'тип' через ранее введенные формальные термины будет рассматриваться как шаг формализации, то есть, приближения описания ооп к формальной теории. Под ранее введенными формальными терминами будем понимать слова вроде 'элемент множества', 'множество', 'декартово произведение', 'кортеж', 'отношение', 'отображение', 'функция' и т.д. То есть, такие, которые были введены в работах по теории множеств и/или матлогике от признанных авторов (Бурбаки, Кантор, Гильберт, Клини, Черч, Манин, Калужнин, Мальцев, Колмогоров, Успенский, Тарский, Роджерс и т.д.). Формализация подразумевает исключение из использования не определенные у таких авторов термины, например, термин 'поведение'. Важность формального определения используемых терминов в такой дедуктивной теории как программирование представляется очевидной. В самом деле, после усилий перечисленных выше авторов было бы странно на вопрос из геометрии 'что такое треугольник?' получать ответ, аналогичный начальному определению Пирса в разделе [9, Типы в информатике] ([16, Types in Computer Science]):

> Система типов - это гибко управляемый синтаксический метод доказательства отсутствия в программе определенных видов поведения при помощи классификации выражений языка по типам вычисляемых ими значений.
>
> A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.

То есть, что то вроде

> Система треугольников это управляемый строителям метод измерения углов и доказательства отсутствия определенных видов отклонений от вертикали при возведении зданий

Тем не менее, определения типа нет у Пирса в [9], нет в фундаменгтальном труде Ахо, Сети и Ульмана [1]. У Ахо тип считается чем то очевидным и определяется на примере языков C и Pascal:

> И в Pascal, и в C типы делятся на базовые (фундаментальные) и создаваемые. Базовые типы - это атомарные типа без внутренней структуры, к которым в Pascal относятся, например, boolean, char, integer и real. Типы поддиапазонов, вроде 1..10 и перечисления вроде (violet, indigo, blue, green, yellow, orange, red) также могут рассматриваться как базовые. Pascal позволяет программисту создавать типы из базовых и других создаваемых типов, примерами чего могут служить массивы, записи и множества. Кроме того, в качестве создаваемых типов могут рассматриваться указатели и функции.

Все эти 'могут рассматриваться' (а могут и не рассматриваться) вызывает подозрение в отсутствии определения.

Даже такой формалист как Карделли в 'Понимание типов, абстракций данных и полиморфизма' [22] очень долго обходит этот вопрос:

> Вместо того, что бы спрашивать 'Что такое тип?' мы будем спрашивать почему типы нужны в языках программирования.
>
> Instead of asking the question What is a type? we ask why types are needed in programming languages.

И только ближе к середине статьи появляется раздел 'Типы - это множества значений'.

Таким образом, для выяснения вопроса придется приложить некоторые усилия.

2 ТИП - ЭТО МНОЖЕСТВО

2.1 Тип это множество значений

Поначалу кажется, что определение 'Тип - это множество' может быть принято в качестве основного предположения. Очень часто оно появляется явно, например, как в [5, Первичные выражения, значения и типы]

```
В ML тип есть множество значений
```

или неявно, как в [7, Типы данных, определяемых пользователем]

> Итак все функции, введенные в наших примерах, оперируют с объектами таких основных типов, как num, real и char

В самом деле, по видимому, слово 'оперировать' означает, что функции определены на некоторых основных типах и вырабатывают значения из этих же типов. Но функции задаются только на множестве допустимых значений и вырабатывают значения из множества допустимых значений.

В Функциональном программировании на Haskel [4] утверждается то же самое:

> Типом называется множество значений, разделяющих некоторые свойства. Примеры типов, постоянно встречающихся программисту: целые числа, символы, булевы значения.

2.2 Типы это множества с дополнительными условиями

Карделли и Вагнер в [22, Types are Sets of Values], фокусируя внимание на системах типов (type systems), а не, собственно, типах как таковых, подчеркивает, со ссылкой на работу Д.Скотта [17], что системы типов образуют решетку и каждый тип является идеалом.

> There is a universe V of all values, containing simple values like integers, data structures like pairs, records and variants, and functions. This is a complete partial order, built using Scott's techniques, but in first approximation we can think of it as just a large set of all possible computable values. A type is a set of elements of V. Not all subsets of V are legal types: they must obey some technical properties. The subsets of V obeying such properties are called ideals. All the types found in programming languages are ideals in this sense, so we don't have to worry too much about subsets of V which are not ideals.

Hence, a type is an ideal, which is a set of values. Moreover, the set of all types (ideals) over V, when ordered by set inclusion, forms a lattice. The top of this lattice is the type Top (the set of all values, i.e. V itself). The bottom of the lattice is, essentially, the empty set (actually, it is the singleton set containing the least element of V).

The phrase having a type is then interpreted as membership in the appropriate set.

Есть множество V всех значений, содержащее такие простые значения, как целые, структуры данных, например, пары, записи, variant-типы и функции. Это законченный частичный порядок, полученный с помощью техники Скотта, но в первом приближении его можно рассматривать, как просто большое множество всех возможных вычисляемых типов.

Тип - это множество элементов V. Не все подмножества V являются корректными типами: они должны удовлетворять некоторым техническим требованиям. Подмножества V, удовлетворяющие таким требованиям, называются идеалами (ideals). Все типы в языках программирования в этом смысле являются идеальными, и нам не придется беспокоиться о подмножествах V, которые не являются идеалами.

Следовательно, тип - это идеал, который является множеством значений. Более того, множество всех типов (идеалов) в V, будучи упорядоченным посредством включения множеств, образует структуру. На вершине этой структуры находится тип Top (множество всех значений, то есть сама V). Основание структуры - это, в сущности, пустое множество (реально - одноэлементное множество, содержащее наименьший элемент V).

Выражение 'иметь тип', таким образом, можно интерпретировать в теоретико - множественном смысле - как принадлежность к соответству- ющему множеству.

2.3 Тип это множество термов языка программирования

Типы также часто трактуются как множество термов языка, а не значений, как на пример в упомянутой работе Пирса [9, Типы в информатике] или, например, в [21, Типы].

> We present a generalization of the ideal model for recursive polymorphic types. Types are defined as sets of terms instead of sets of elements of a semantic domain.

Мы представляем обобщение модели идеалов для рекурсивных типов. Типы определяются как множества термов, а не множества элементов сементического домена. То есть, не число 0 имеет целый тип, а строчка '0' в языке имеет целый тип.

2.4 Типы Харрисона

Но дальнейшее изучение вопроса заставляет отказаться от этой идеи. Связь между типами и множествами хотя оказывается не столь очевидна. Например, в [11, Типы] типы считаются средством удобным средством определения различных разновидностей данных. И трактуются как множества только в качестве аналогии.

> Введём для отношения 't имеет тип σ' обозначение $t : \sigma$. Подобная запись традиционно используется математиками при работе с функциональными пространствами, поскольку $f : \sigma \to \tau$ обозначает функцию f, отображающую множество σ во множество τ. Будем считать типы множествами, которые содержат соответствующие объекты, и трактовать $t : \sigma$ как $t \in \sigma$. Однако, несмотря на то, что мы предлагаем читателям также воспользоваться этой удобной аналогией, типизированное лямбда-исчисление будет в дальнейшем рассматриваться исключительно как формальная система, свободная от каких-либо интерпретаций.

3 ТИП - ЭТО НЕ МНОЖЕСТВО

Возражения против идеи 'тип это множество', по видимому появились еще в 1973 у Morris [20]. К сожалению, найти работу Морриса в свободном доступе не удалось. Geuvers [19] также начинает свое 'Введение в теорию типов' с утверждения 'Types are not sets'. На двух страницах объясняет почему типы немного похожи на множества (Types ate a bit like sets), но

на самом деле отличаются от них. После чего строит простую теорию типов аналогично Черчу [15].

В этих случаях оказалось, что тип мыслится только как элемент некоторой системы (точнее - иерархии) типов и не мыслится отдельно, сам по себе, как элемент множества не мыслится отдельно от множества. Кроме того, типы Рассела и Черча вполне удовлетворяют требованиям определения Пирса 1 . Эти системы типов были введены, для того, чтобы классифицировать выражения формального языка и, в случае правильного использования этого языка (в данном случае, теории Рассела или Черча), избегать появление нежелательного поведения текстов, написанных в рамках соответствующей теории. А именно, удалось избежать парадоксов, свойственных наивной теории Кантора. Подчеркнем, что проверку соответствия текста требованиям, накладываемым типом, можно делать при формальном просмотре текста, 'не вычисляя' значения каждого отдельного символа.

3.1 Типы Рассела

У Рассела - тип это некоторая численная характеристика, которую приписывают самим множествам, переменным и выражениям. Для того, чтобы выражение x isin Y можно было записать в формальной системе Рассела, необходимо, что бы выражение x имело тип на 1 меньше типа выражения Y. Таким образом элементы типа 1 - это некие 'индивидуальные сущности', а элементы типа 2 - это множества 'индивидуальных сущностей', элементы типа 3 - это множества множеств 'индивидуальнрых сущностей' и так далее.

3.2 Типы в типовом λ -исчислении у Барендрегта

Барендрегт в [2, Приложение А. Типовое лямбда-исчисление, стр.548] множество типов Typ индуктивно определяет следующим образом:

1. 0 isin Typ: 0 тип (основной тип, или тип индивидов);
2. σ , τ isin $Typ \Rightarrow (\sigma \rightarrow \tau)$ isin Typ : если σ, τ типы, то типом будет $(\sigma \rightarrow \tau)$.

Как видно, множество Typ строится совершенно формально из символов 0, $(0 \rightarrow 0)$, $((0 \rightarrow 0) \rightarrow 0)$ и так далее. Кроме формального построения множества типов, задается правило, по которому строится множество всех элементов заданного типа. Пусть X - любое множество. Тогда множество X_σ всех элементов типа σ строится индукцией по σ isin Typ следующим образом:

1. $X = X_0$;
2. $X_{\sigma\to\tau} = X_\tau^{X_\sigma}$ - множество всех функций из X_σ в X_τ.

Таким образом, совершенно аналогично Расселу (за исключением того, что номер типа заменяется на индекс из множества Typ) каждый тип σ является именем некоторого множества X_σ - множества всех элементов заданного типа.

3.3 Вариация системы Черча

Согласно изложению Дехтяренко [14, Программы как теории] система типов Черча определяется индуктивно на основе базовых типов. Но, в отличие от Барендрегта, кроме операции создания функции используется операции декартового произведения. Пусть D_A это множество всех элементов (домен типа) типа A, тогда

1. Если A и B - типы, то $A >< B$ это тип, представляющий декартово произведение соответствующих доменов $D_A >< D_B = D_{A\times B}$;
2. Если A и B - типы, то $A \to B$ это тип. Соответствующий домен - множество всех функций из D_A в D_B обозначаемое как $D_{A\to B}$.

Опять получаем некоторое имя - тип, которое определяет множество всех элементов данного типа (домен типа).

3.4 Простая теория типов Черча

Прочтение у Барендрегта [2, замечание A.1.3] о том, что в литературе применяется обозначения, отличные от $(\sigma \to \tau)$, например, $(\sigma\tau)$ сильно облегчает чтение первоисточника [15].

> The class of type symbols is described by the rules that ι and o are each type symbols and that if α and β are type symbols then $(\alpha\beta)$ is type symbol: it is the least class of symbols which contains the symbols ι and o and is closed under operation of forming the symbol $(\alpha\beta)$ from the symbols α and β.
>
> Множество символов типа описывается правилом, что ι и o символы типа и, если α и β символы типа, то $(\alpha\beta)$ тоже символ типа: есть минимальное множество символов, которое содержит символы ι и o и замкнутое относительно операции формирования символа $(\alpha\beta)$ из символов α и β.

То есть, в явной форме записано, что типы это символы, которые вводятся для использования в качестве индексов возле переменных и констант.

> The type symbols enter our formal theory only as subscripts upon variables and constants
>
> Символы типа ввходят в нашу формальную теорию только как индексы под переменными и константами.

И, кроме того, каждый символ однозначно задает множество значений данного типа

1. o - тип пропозициональных высказываний (логический, по видимому);
2. ι - тип индивидуальных сущностей (individuals);
3. $(\alpha\beta)$ - тип функций из β в α.

3.5 Типы Стрейчи

Christopher Strachey в работе 'Фундаментальные концепции языков программировани' [27] утверждает, что тип определяет представление, ограничивает набор объектов, которые можно представлять и, в сложных языках программирования, используется для того, что бы проверять отсутствие грубых ошибок и правильно интерпретировать многозначные символы - операции, которые могут выполнять различные действия в зависимости от типа.

> The Type of an object determines its representation and constrains the range of abstract object it may be used to represent. Both the representation and the range may be implementation dependent.
>
> ...
>
> In more sophisticated programming languages, however, we use the type to tell us what sort of object we are dealing with (i.e., to restrict its range to one sort of object). We also expect the compiling system to check that we have not made silly mistakes (such as multiplying two labels) and to interpret correctly ambiguous symbols (such as +) which mean different things according to the types of their operands.

В силу того, что тип таки не должен определять представление (В самом деле, совершенно все равно используется BigEndian или LittleEndian в текущей реализации) и прикрывшись мнением Карделли

> Основная цель системы типов - обойти вопрос о представлениях и исключить ситуации, когда такие вопросы могут возникнуть.

получаем сухой остаток, мнения Стрейчи: 'тип определяет множество значений и фунции, определенные на этом множестве'.

3.6 Типы Кришнамурти

Заглянем напоследок в монографию Шрирама Кришнамурти [26]:

> A type is any property of a program that we can establish without executing the program. In particular, types capture the intuition above that we would like to predict a program's behavior without executing it.

Тип это свойство программы, которое можно установить без её выполнения.

> So any static prediction of behavior must necessarily be an approximation of what happens. People conventionally use the term type to refer not just to any approximation, but one that is an abstraction of the set of values.

То есть, любое статическое предсказание поведения обязано быть приблизительным. Обычно, под этим термином тип подразумевается не приблизительное описание, а нечто, что является множеством значений.

> We present a type system as a collection of rules, known formally as type judgments, which describe how to determine the type of an expression.

Мы рассматриваем систему типов как коллекцию правил, извесную как суждения о типах, которые описывают как определяется тип выражений.

> A type system for us is a collection of types, the corresponding judgments that ascribe types to expressions, and an algorithm for performing this ascription.

Система типов для нас это коллекция типов (в смысле идентификаторы), соответствующих им суждений, которые приписывают типы к выражениям и алгоритм для выполнения такого приписывания.

4 ТИП ЭТО НЕ КЛАСС

Есть еще один вариант. Тип - это класс. Такого мнения придерживается, например, Страуструп: 'класс - это определенный пользователем тип'. Тогда выделение подтипа (subtyping) должно совпадать с наследованием (inheritance). Это утверждение пытается опровергать Роберт Мартин с помощью известного принципа подстановки Лисков [25, The Liskov Substitution Principle]. Класс оказывается отличен от типа и с помощью наследования можно добиться образования подклассов, которые ведут себя не так, как хотелось бы вели себя типы.

Карделли и Абади в своей Теории Объектов [23], частичный перевод см [6], сразу отделяют тип от класса и, не уточняя, что же такое тип, с разных сторон рассматривают влияние на язык свойств

1. наследование-это-выделение_подтипа ;
2. наследование-это-не_выделение_подтипа.

В конце концов типом у них называется набор типов полей и функций и тип называется спецификацией объекта.

> In general, elements of $InstanceTypeOf(cell)$ have in common only the type signature of the attributes specified in *cell*; this is sometimes called the *objectprotocol* for cells.
>
> Вообще говоря, общими для элементов типа $InstanceTypeOf(cell)$ являются только сигнатуры атрибутов, указанных в класса *cell*. Это множество всех сигнатур иногда называют протоколом объектов класса cell.

4.1 Типы Мейера

Идея о спецификации развивается в курсе Бертрана Мейера [3, Основы объектно-ориентированного программирования]. Понятие спецификации абстрактного типа данных определяется в следующих разделах:

- типы;
- функции;
- аксиомы;
- предусловия.

> Тип - это совокупность объектов, характеризуемая функциями, аксиомами и предусловиями. Не будет большой ошибкой рассматривать пока тип как множество объектов в математическом смысле слова "множество тип STACK как множество всех возможных стеков, тип INTEGER как множество всех целых чисел и т.д.

> Класс это абстрактный тип данных, поставляемый, возможно с частичной реализацией.

5 ИТАК ТИП ЭТО ...

Посколько классом стало все, даже целые числа с операциями сложения и умножения, то абстрактным типом данных можно называть спецификацию любого класса. Более того, как показывается в [18] любые, самые простые типы вроде множества натуральных чисел можно описывать в спецификацией в смысле Мейера. То есть, слово абстрактный - теряет смысл, который ранее отличал нечто новое, проектируемое программистом от встроенных типов вроде целых.

Проделаем незначительное изменение определения 'Тип - это совокупность объектов, характеризуемая фукнциями, аксиомами и предусловиями' и сформулируем его в форме тип - это список функций, аксиом и предусловий:

Тип это

- обозначение множества значений и его свойства - то есть, домен типа и аксимы, которым он удовлетворяет;
- сигнатура функций, определенных на домене типа;
- аксиомы, которым удовлетворяют функции;

Поскольку, сама сигнатура функции и объявление типа мало чем отличается от аксиомы (в самом деле, объявление функции можно рассматривать как аксиому о существовании такой функции), то неформально можно считать, что тип - это множество аксиом.

Далее, рассмотрим не спецификацию стека, а нечто более простое (точнее, более, сложное) - аксиоматику Пеано, которая является основой для определения кольца целых и поля вещественных чисел. Оказывается, что на деле, спецификация кольца целых и поля вещественных чисел, совершенно не отличаются от спецификации абстрактных типов данных по Мейеру.

Поэтому, по определению Мейера их реализацию в языках программирования можно считать классами, а типом, то есть, спецификаций будем называть соответствующую аксиоматику, задающую кольцо целых или поле вещественных чисел. Например, для кольца целых и класса целых основой является аксиоматика Пеано. Записанная на языке формальных спецификаций RSL, она полностью удовлетворяет объяснениям из курса Мейера [3]. В данных примерах нет предусловий, но синтаксис, скажем, того же языка RSL, позволяет их дописывать. См. работу [8].

5.1 Аксиоматика Пеано

```
---- File:peano.rsl


--
--      операция сравнения '=' и ключевое слово 'is'
--      имеют различный смысл, но в этом примере смысл практически совпадает.
--      если в схеме появляются "глобальные переменные" - множество
--      всех глобальных переменных наз. состояние - то операция = и
--      is будут иметь различный смысл. is означает равно для всех состояний
--

scheme PEANO =
    class
        type                        -- завели домен типа N
```

```
            N
        value
            zero : N,               -- завели значение zero
                                    -- (это не переменная, нельзя присваивать)
            succ: N -> N            -- функция для построения значений домена
        axiom
          [first_is_zero]         --
                                    -- квантор всеобщности
            all n : N :-            -- n - это обозначение

                ~(succ(n) is zero)  -- для каждого n результат succ(n)
            ,                       -- не есть zero
          [linear_order]
            all  n1, n2 : N :-
                (succ(n1) is succ(n2)) => (n1 is n2)
            ,
          [induction]
            all p : N -> Bool :-    -- для любого предиката
                                    --
                    (p(zero) /\ (all n : N :- p(n) => p(succ(n))))
                         =>
                             (all n : N :- p(n))
    end

--
-- ~     отрицание
-- :-    выполняется
-- =>    влечет
-- /\    логическое и
-- \/    логическое или
-- exist квантор существования
--
--

---- End Of File:peano.rsl
```

5.2 Описание фукнций сложения и умножения

```
---- File:nat.rsl


PEANO                           --  подключение схемы PEANO

scheme NAT =
```

```
    extend PEANO with
    class
        value
          plus : N >< N -> N    -- вводим функции (операции)  сложить
          ,
          mult : N >< N -> N    -- и умножить
        axiom
          [plus_zero]
            all n : N :-
            plus(n, zero) is n
            ,
          [plus_succ]
            all n1, n2 : N :-
            plus(n1, succ(n2)) is succ(plus(n1, n2))
            ,
          [mult_zero]
            all n : N :-
            mult(n, zero) is zero
            ,
          [mult_succ]
            all n1, n2 : N :-
            mult(n1, succ(n2)) is plus(mult(n1, n2), n1)

    end

---- End Of File:nat.rsl
```

6 СИСТЕМЫ ТИПОВ И ВЫДЕЛЕНИЕ ПОДТИПА (SUBTYPING)

В заключение надо вспомнить, что на деле используются системы типов, а не просто множество типов. Домены типов, рассматриваемые как элементы, образуют структуру решетки, и, по видимому, должны удовлетворять некоторым дополнительным свойствам. К таким свойствам относится свойство множества быть идеалом, см. раздел 2.2 где упоминалось мнение Карделли.

Если считать, что тип (спецификация) - это множество аксиом, то выделение подтипа определяется легко. К множеству аксиом добавляется новая аксиома и, в случае, если новый списко аксиом оказывается не противоречивым можно считать, что выделен подтип. Проведенный с этой точки зрения анализ примера Роберта Мартина [25] выполненный

с этой точки зрения в работе [10] позволил уточнить форму принципа подстановки Лисков, который может звучать так:

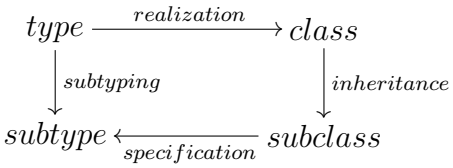
> Пусть T и S некоторые классы. Любая программа P, использующая обращения к переменной t: T, продолжает удовлетворять своей спецификации при присвоении t значения любой переменной s: S, с возможностью обратного присваивания нового значения из t в s после выполнения P, тогда и только тогда, когда спецификация класса (тип) S является подтипом спецификации класса (типа)T.

где, под присвоением значения (как указано в [23], [6]) понимается присвоение адреса записи с объектом в другую переменную или передача адреса записи через механизм фактических - формальных параметров в подпрограмму. Никакого преобразования значений из тип в тип не выполняется.

Условным термином восстановление спецификации будем называть создание спецификаций (чтото вроде реверс инжиниринга) для некотрого уже реализованного класса. Критерий подстановки показывает условия когда выполнение последовательность действий:

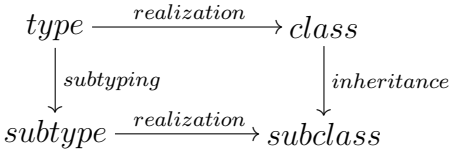
- реализация типа;
- наследование;
- восстановление спецификации.

дает результат такой же, как создание подспецификации (выделение подтипа). Что означает естественное в теории категорий ([13, стр.37]) требование о коммутативности стрелок:



выделение подтипов и наследование

или



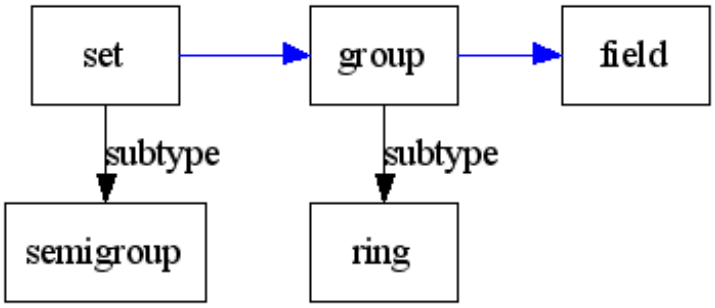
выделение подтипов и наследование 2

Во втором случае записано, что результат последовательности действий выделение типа с последующей реализацией эквивалентен результату последовательности реализации надтипа с последующим наследованием. Так же в [10] удалось выяснить, что

- при выделении подтипа желательно неизменение домена типа, и, значит, чтобы спецификация подкласса оставалось подтипом спецификации типа, при наследовании желательно неизменение домена класса;
- в силу не совместимости системы аксиом, группа целых не является подтипом полугруппы натуральных, равно как и полугруппа натуральных не является подтипом группы целых.

В заключение можно привести первую, в принципе, возможную систему типов и их реализаций:

- натуральные множества (операция дай_следующий) - его реализация, по видимому, близка к *Sequence* из *Oracle* [24];
- полугруппа натуральных - реализация, чтото вроде $System.UInt32$ из $.NETFrameworktype$, если удалить умножение и деление;
- группа целых;
- кольцо целых - $System.Int32$ из $.NETFrameworktype$;
- поле рациональных - $System.Decimal$ $.NETFrameworktype$. И, по аналогии с парой полугруппа натуральных - группа целых, рассмотрим пару кольцо целых - поле рациональных. В поле рациональных единица представима в виде суммы двух одинаковых положительных величин, в то время как из аксиоматики кольца следует невозможность этого утверждения.



где, черная стрелка означает отношение быть подтипом (subtyping), синяя - нет. Наследование в сторону увеличения домена выбрано специально, ибо, по видимому, оно сохраняется исторический порядок появления классов.

Коротко взаимосвязь типов и классов можно выразить следующими утверждениями:

- иерархия типов определяется законами логики из заданных аксиом и не может быть изменена;
- иерархия классов определятся произволом программиста, и может записываться в программе, чуть ли не в произвольном порядке;
- соответствие иерархий выполняется тогда, когда для каждой пары подкласс - надкласс спецификация подкласса является подтипом спецификации надкласса;
- проверка типов в компиляторе гарантирует работу программы только тогда, когда иерархия типов соответствует иерархии классов;
- при использовании формальных методов разработки: дизайн по контракту, венский метод разработки, метод спецификаций RAISE, Z метод, и т.д., по видимому, создаются иерархии классов, соответствующие иерархиями типов.

Предметный указатель

ССЫЛКИ

[1] Ахо, Сети, Ульман. Компиляторы. Принципы, технологии, инструменты. Вильямс, Москва-Петербург-Киев, 2003.

[2] Х.Барендрегт. Лямбда - исчисление. Его синтаксис и семантика. (Гос.изд.физ.мат.лит., Москва, 1960).

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

[4] Алексей Бешенов. Функциональное программирование на Haskell: Часть 2.Основные типы и классы. http://www.ibm.com/developerworks/ru/library/l-haskell2/index.html.

[5] Роберт Харпер, перевод Ковтун, Романенко. Введение в Стандартный ML, 1996.

[6] Люка Карделли Мартин Абади. Классы и типы в языках, основанных на классах, 2010. http://agp.hx0.ru/oop/TvsC.pdf.

[7] А.Филд, П.Харрисон. Функциональное программирование, 1993. Мир.

[8] А.Г. Пискунов. The RAISE Method Group: Алгебраическое проектирование класса, 2007. http://www.realcoding.net/article/view/4538.

[9] Бенджамин Пирс. Типы в языках программирования, 2010. http://newstar.rinet.ru/~goga/tapl/tapl-toc.html.

[10] А.Г. Пискунов. Об одном примере нарушения принципа подстановки Лисков, 2010. http://www.realcoding.net/articles/ob-odnom-primere-narusheniya-printsipa-podstanovki-liskov.html.

[11] Д.Харрисон, перевод Алекс Отт и et al. Введение в Функциональное программирование, 1997. Кембридж.

[12] С.К.Клини. Математическая логика, 1973. Мир, 1973.

[13] Р.Голдблатт. Топосы. Категорный анализ логики, 1983. Москва, Мир, http://agp.hx0.ru/arts/Goldblatt.Toposy.Kategornyj_analiz_logiki.Mir.1983.djvu.

[14] Дехтяренко. Декларативное программирование, 2003. http://www.softcraft.ru/paradigm/dp/index.shtml.

[15] Alonzo Church. A Formulation of the Simple Theory of Types. (The Journal of Symbolic Logic, Vol. 5, No. 2. (Jun., 1940), pp. 56-68.), http://links.jstor.org/sici?sici=0022-4812%28194006%295%3A2%3C56%3AAFOTST%3E2.0.CO%3B2-Q.

[16] Benjamin Pierce. Types and Programming Languages, 2002. The MIT Press.

[17] D.Scott. Data types as lattices, 1976. SIAM Journal of Computing, Vol 5, No 3, September 1976, pp.523-587.

[18] The RAISE Method Group. The RAISE Development Method, 1999. ftp://www.iist.unu.edu/pub/RAISE/method_book/book.zip.

[19] Herman Geuvers. Introduction to Type Theory, 2008. Technical University Eindhoven, The Netherlands, 2008, http://www.cs.ru.nl/~herman/PUBS/IntroTT.pdf.

[20] J. H. Morris. Types are not sets, 2008. Conference Record of the ACM Symposium on Principles, of Programming Languages, pp. 120-124, Boston October 1973.

[21] Jerom Vouillon, Paul-Andre Mellis. Semantic Types: A Fresh Look at the Ideal Model for Types. www.pps.jussieu.fr/~vouillon/publi/cbv.ps.gz.

[22] Luca Cardelli, Peter Wegner. On Understanding Types, Data Abstraction, and Polymorphism, 1985. Computing Surveys, Vol 17 n. 4, pp 471-522, December 1985, http://lucacardelli.name/Papers/OnUnderstanding.A4.pdf.

[23] Luca Cardelli Martin Abadi. A theory of objects, 1996. Springer, http://books.google.com.ua/books?id=4xT3LgCPP5UC.

[24] Oracle. Oracle Documentation. http://download.oracle.com/docs/cd/B19306_01/appdev.102/b31695/intro.htm#sthref71.

[25] Robert C.Martin. The Liskov Substitution Principle, 2003. C++ Report, March 1996, http://www.objectmentor.com/resources/articles/lsp.pdf.

[26] Shiram Krishnamurthi. Proframming Languages:Application and Interpretation, 2003. http://www.cs.brown.edu/~sk/Publications/Books/ProgLangs/PDF/plai-2006-01-15.pdf.

[27] Christopher Strachey. Fundamental concepts in programming languages, 2000. http://www.itu.dk/courses/BPRD/E2009/fundamental-1967.pdf.