Л.В Яблокова[1], Д.Е. Яблоков[2]

ОБОБЩЁННАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА РАСЧЁТА МАТЕМАТИЧЕСКОЙ КОНСТАНТЫ В ТЕРМИНАХ АЛГЕБРАИЧЕСКИХ АБСТРАКЦИЙ

([1] Самарский университет, [2] ООО «ИнтеллектСофт»)

Попытки рассмотрения обобщённых численных алгоритмов с позиций абстрактной алгебры предпринимались уже неоднократно [1, 2]. Примеры таких реализаций [1], по сути, являются фундаментом для понимания, применения и развития обобщённого программирования. Основной акцент в них делается на том, что любой алгоритм, в зависимости от специфики его работы, накладывает определённый набор абстрактных требований на свои аргументы. Процесс осмысления этих требований можно представить в виде нескольких правил, задающих последовательность взаимодополняющих шагов, приводящих к получению качественной обобщённой реализации. Во-первых, очень важно правильно выбрать алгоритм, возвращающий корректный, ожидаемый результат за приемлемое время. Затем необходимо подобрать подходящую алгебраическую структуру [2] (иерархию структур), в терминах которой, будут выполняться требуемые вычисления и на её основе сформировать наиболее эффективный, с наименьшим количеством элементарных операций, вычислительный базис [2]. На третьем, заключительном шаге, нужно произвести моделирование в системе типов используемого инструмента кодирования и предусмотреть возможность обобщения моделей на семейства типов, объединённых по критерию соответствия выбранным алгебраическим абстракциям.

В статье будет рассмотрен алгоритм вычисления математической константы π [3], основанный на понятии арифметико-геометрического среднего (Рис. 1).

```
1  algorithm pi(eps)
2        a = 1        b = rec(sqrt(2))
3        t = rec(4)   x = 1
4        while ((a - b) > eps)
5              y = half(a + b)
6              b = sqrt(a * b)
7              t = t - x * sqr(y - a)
8              x = twice(x)
9              a = y
10       end
11       pi = sqr(a + b) / 4 * t
12 end
```

Рис. 1. Псевдокод алгоритма расчёта $\boldsymbol{\pi}$ на основе арифметико-геометрического среднего

Табл. 1. Характеристики работы алгоритма

Порог точности	Время (сек.)	Количество итераций	Начало расхождения в мантиссе (позиция)
10^{-3}	0.0000338	2	8
10^{-6}	0.0000534	3	11
10^{-9}	0.0000622	3	19
10^{-12}	0.0000623	4	24
10^{-15}	0.0000629	4	41
10^{-18}	0.0000812	4	41
10^{-21}	0.0002181	5	49
10^{-24}	0.0047874	5	49

Хотя в описании алгоритма содержатся дорогостоящие операции с высокой степенью сложности, такие как вычисление квадратного корня (Рис. 2), получаемая высокая точность результата за малое количество повторений основного цикла (Рис. 1, строки 4-10) компенсирует этот недостаток. Корректное завершение алгоритма расчёта, т.е. условие при котором происходит выход из тела цикла (Рис. 1, строка 4), гарантируется следующим: $\frac{a+b}{2} > \sqrt{ab} \Rightarrow (a-b)^2 > 0$, при $a > 0$, $b > 0$ и $\mathtt{eps}\ \epsilon(0;1)$.

```
1 algorithm sqrt(x, eps)
2       r = x / 2
3       do
4             t = r
5             r = (t + x / t)/2
6       while (abs(t - r) > eps)
7       sqrt = r
8 end
```

Рис. 2. Псевдокод алгоритма вычисления квадратного корня по методу Ньютона

Чтобы двигаться дальше, следует удостовериться в том, что хотя, на первый взгляд кажется очевидным, что все переменные (Рис. 1, 2) должны иметь тип double, сами алгоритмы могут работать и с множеством других подобных типов, объединённых в семейства (виды [2]). Главным условием принадлежности к виду является поддержка типом определённого набора синтаксических конструкций. Сгруппировать эти конструкции можно по некоторым семантическим критериям, относящим их к одной или нескольким алгебраическим абстракциям. Именно на этом уровне должна проходить формализация синтаксической и семантической общности для типов данных, имеющих общий синтаксис и, возможно, различную семантику. Поэтому необходимо определить область операторов, т.е. установить какие именно элементарные операции должны поддерживаться. Это: сложение, вычитание, умножение, деление и сравнение по критерию "больше". Для операций +, –, *, / без труда можно найти аналогию среди уже известных алгебраических

структур. Понятие поля сочетает в себе свойства аддитивной и мультипликативной абелевых групп, т.е. содержит все четыре упомянутые операции, а также операции аддитивной и мультипликативной инверсий и соответствующие нейтральные элементы. Кроме того по определению в этой структуре заложены ассоциативность, коммутативность и дистрибутивность умножения и сложения, что полностью удовлетворяет требованиям к арифметической части. Возможность сравнения с помощью операции > подразумевает, что поддерживающий её объект, относится к типу, моделирующему концепцию упорядоченной структуры [4]. Для этого, в силу определений, должно быть реализовано транзитивное, нерефлексивное и ассимметричное, однородное отношение строгого порядка (Рис. 3).

```
1  public interface IStrictlyOrdered<T>
2  {
3          bool Lt(T arg1, T arg2);
4  };
```

Рис. 3. Пример кода модели концепции строго упорядоченной структуры

Синтаксис языка C#, начиная с версии 3.0, позволяет добавлять методы расширения [5], т.е. внешние статические методы без необходимости создания производного или изменения сигнатуры целевого типа. Эту особенность можно использовать для реализации дополнительных операций, присутствующих в алгоритмах (Рис. 1, 2) и расширяющих синтаксис и семантику «упорядоченного поля» (Рис. 4, 5).

Модели алгебраических абстракций, выраженные обобщёнными интерфейсами, формируют иерархию наследования (Рис. 5), определяющую базовый контракт для классов-реализаций, т.е. тот функциональный минимум, который достаточен при проведении вычислений. Любой конкретный класс, наследующий одному, всем или нескольким узлам иерархии, может быть применён в качестве параметра `concept` в том или ином методе (Рис. 4-6), требования к аргументам которого, совпадают с унаследованными синтаксическими особенностями.

```
1  static bool Et<C, T>(this C concept, T arg1, T arg2) where C : IStrictlyOrdered<T>
2  {
3          return (!concept.Lt(arg1, arg2) && !concept.Lt(arg2, arg1));
4  }
5  // ...
6  static bool Gt<C, T>(this C concept, T arg1, T arg2) where C : IStrictlyOrdered<T>
7  {
8          return concept.Lt(arg2, arg1);
9  }
10 // ...
```

Рис. 4. Пример кода методов расширения для некоторых операций отношения

```
1  static T Sub<C, T>(this C concept, T arg1, T arg2)
2        where C : IAdditiveAbelianGroup<T>
3  {
4        return concept.Add(arg1, concept.Neg(arg2));
5  }
6  static T Abs<C, T>(this C concept, T arg)
7        where C : IStrictlyOrdered<T>, IAdditiveGroup<T>
8  {
9        return concept.Lt(arg, concept.Zero) ? concept.Neg(arg) : arg;
10 }
11 static T Twice<C, T>(this C concept, T arg) where C : IAdditiveAbelianGroup<T>
12 {
13       return concept.Add(arg, arg);
14 }
15 static T Half<C, T>(this C concept, T arg) : where C : IField<T>
16 {
17       return concept.Div(arg, concept.Twice(concept.One));
18 }
   static T Sqr<C, T>(this C concept, T arg) where C : IMultiplicativeAbelianGroup<T>
   {
         return concept.Mul(arg, arg);
   }
```

Рис. 5. Пример кода методов расширения
для дополнительных математических действий

```
1  public static T PI<C, T>(this C concept, T threshold)
2        where C : IField<T>, IStrictlyOrdered<T>
3  {
4        var a = concept.One;
5        var _2 = concept.Twice(a);
6        var _4 = concept.Twice(_2);
7        var b = concept.Rec(concept.Sqrt(_2, threshold));
8        var t = concept.Rec(_4);
9        var x = a;
10
11       while (concept.Gt(concept.Abs(concept.Sub(a, b)), threshold))
12       {
13             var y = concept.Div(concept.Add(a, b), _2);
14             b = concept.Sqrt(concept.Mul(a, b), threshold);
15             t = concept.Sub(t, concept.Mul(x, concept.Sqr(concept.Sub(a, y))));
16             x = concept.Twice(x);
17             a = y;
18       }
19
20       return concept.Div(concept.Sqr(concept.Add(a, b)), concept.Mul(_4, t));
21 }
```

Рис. 6. Фрагмент кода реализации алгоритма расчёта **π**

Код алгоритма (Рис. 6) является обобщённым, т.к. может работать с различными типами данных во множестве ситуаций, при условии соответствия установленным в универсальных объявлениях ограничениям (Рис. 6, строка 2). Так же код алгоритма представлен в терминах алгебраических структур, что сужает понятийный аппарат до терминологической базы универсальной алгебры. Всё это, по мнению авторов, чрезвычайно полезно, т.к. ключевые

особенности, используемых в обобщённом программировании абстракций, могут быть получены из удобной алгебраической таксономии, проясняющей свойства и взаимосвязи между составляющими.

Литература

1. Stepanov, A. From mathematics to generic programming / A. Stepanov, E. Rose. – Boston: Addison–Wesley, 2015. – 293 p.
2. Stepanov A. Elements of Programming / A. Stepanov, P. McJones. – Boston: Addison–Wesley, 2011. – 272 p.
3. Brent, R.P. Fast multiple-precision evaluation of elementary functions // Journal of the Association of Computing Machinery. 1976, 23(11). – С. 713–735.
4. Цветов, В.П. О двойственных упорядоченных полугруппах бинарных отношений / В. П. Цветов // Перспективные информационные технологии, 2017. – С. 299-303.
5. URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods