Шилов Николай Вячеславович

ЗАМЕТКИ О ТРЁХ ПАРАДИГМАХ ПРОГРАММИРОВАНИЯ[1]

Аннотация

В статье обсуждаются парадигмы программирования и то, как разные парадигмы применяются для решения программистских задач. В частности, обсуждается, как решить одну трудную олимпиадную задачу по программированию с использованием трёх парадигм программирования: логического, функционального и императивного. Разработку эффективного императивного алгоритма решения этой задачи можно рассматривать как пример обращения аналогичной логической программы. Функциональный алгоритм при таком подходе представляет собой промежуточный вариант решения задачи, более эффективный, чем логический алгоритм, а эффективный императивный алгоритм фактически является ленивой мемоизацией этого функционального алгоритма.

Ключевые слова: парадигмы программирования, императивное программирование, функциональное программирование, логическое программирование, мемоизация, динамическое программирование.

1. ВВЕДЕНИЕ

1.1. НЕМНОГО КРИТИКИ И САМОКРИТИКИ

Поводом для этих заметок послужили две статьи [1] и [2] в журнале для старшеклассников и учителей «Потенциал». В этих статях (по мнению автора и редакции журнала) «в сжатой форме рассказывается про функциональный подход к описанию вычислительных процессов..., а также про применение этого подхода в информатике в функциональной парадигме программирования» с примерами функциональных программ на языке Haskell, в частности, к «одной из классических оптимизационных задач – задачи о ранце». Дело хорошее, тем более что на русском языке фактически нет научно-популярной литературы по функциональному программированию и другим парадигмам программирования, отличным от императивного.

Автору также приходится знакомить с понятием о разных парадигмах программирования (включая функциональное программирование) и старшеклассников, и студентов техникума, и студентов (разных курсов от первого – до магистратуры). Поэтому появилось желание по-своему рассказать о том, что такое парадигмы программирования, и показать, как разные парадигмы программирования могут быть использованы для решения сложной задачи олимпиадного уровня. В то же время предложенный в данной работе подход также может быть подвергнут критике, например за то, что не дает примеров программ на конкретном языке (в то время как статьи [1, 2] дают примеры программ на языке Haskell). Предварительный краткий вариант статьи был опубликован в виде доклада на IV Международной научно-практической конференции «Современные информационные технологии и ИТ-образование» [3].

1.2 ЧТО ТАКОЕ «ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ»?

Одна из «культовых» книг по философии и методологии науки – это диссертация Томаса Куну «Структура научных революций», защищенная около 40 лет

[1] Работа подготовлена при поддержке гранта РФФИ 08-01-00899а.

назад [4]. Согласно Т. Куну парадигма – это метод, подход к формулировке задач (проблем) и путей их решения. Само слово «парадигма» греческого происхождения и означает «пример», «образец», а в общефилософском смысле обозначает категорию, состоящую из сущностей с общими характеристиками. Первым, кто явно ввел в употребление понятие «парадигма программирования», был Роберт Флойд в лекции в 1978 г. по случаю присуждения ему самой престижной для ученых-программистов премии им. Тьюринга [5]. В своей лекции Р. Флойд ссылается только на известную диссертацию Т. Куна и трактует «парадигмы программирования» именно как разные способы постановки и способы решения программистских задач, но подчеркивает, что концептуально эти «разные способы» фиксируются обычно на уровне языков программирования. Поэтому, чтобы разобраться с парадигмами программирования, нам придется в некоторой степени разобраться в многообразии языков программирования.

Издательство O'REIL, специализирующееся на публикации литературы по языкам программирования, подготовило плакат[1] History of Programming Languages. Его полная версия имеет длину около 6 м, содержит сведения о хронологии и влиянии друг на друга 2500 языков программирования и отражает хронологию и взаимное влияние друг на друга наиболее значимых языков программирования. Хронология на этом плакате представлена осью времени (в верхней части плаката), влияние языков друг на друга – цветными линиями.

Заметим, однако, что этот плакат можно принять за основу «путеводителя» в мире языков программирования лишь на начальном этапе становления программирования и информационно-вычислительных технологий (первые 10–15 лет, начиная с 1950 г.). В это время компьютерных языков было немного, и почти все они[2] были языками императивного программирования[3]. Поэтому за основу классификации в этот период можно было принять хронологию появления и взаимного влияния языков.

Но для языков, появившихся позже, путеводитель в стиле плаката O'REILLY становится неприемлемым, так как в это время, наряду с языками последовательного императивного программирования, создаются языки параллельного программирования, языки декларативного программирования (прежде всего функционального); в середине 1970-х – начале 1980-х годов родилось несколько новых «программирований» – логическое и объектно-ориентированное в том числе. Для специалистов очевидно, что все перечисленные «программирования» – это (в соответствии с Т. Куном и Р. Флойдом) исторически сложившиеся парадигмы программирования, то есть разные способы ставить и решать задачи.

Три из перечисленных пяти парадигм мы конспективно обсудим в следующей части 2 на примере вычисления факториала. Для этого мы будем использовать соответствующий «псевдокод», то есть ориентированный на человека формат представления алгоритмов, а не код на каком-либо конкретном языке программирования. Хочется надеяться, что псевдокод будет понятен и неподготовленному читателю, но от читателя всё же требуется знакомство с некоторыми математическими понятиями, такими как целое число, функция, тождество, отношение, аксиомы и формальное доказательство. Цель этой части – подвести общий понятийный базис (без знакомства с формальной теорией), необходимый в дальнейшем обсуждении.

В части 3 мы обсудим, как решить одну сложную программистскую задачу (олимпиадного уровня) в терминах логи-

[1] См. http://www.oreilly.com/news/graphics/prog_lang_poster.pdf

[2] За, может быть, единственным исключением – LISP [6], который является функциональным языком программирования.

[3] Подробное объяснение понятия «императивное программирование» будет дано позже, а пока ограничимся пояснением, что оно означает «командный» стиль.

ческого программирования. При этом мы предполагаем, что читатель знаком с логической нотацией (кванторы, конъюнкция и дизъюнкция). В части 4 мы перейдём от логического решения к функциональному, проанализируем некоторые его недостатки и пути их преодоления (в частности, с использованием мемоизации функциональных программ). И наконец, в последней части 5 мы перейдём от функционального решения к императивному в терминах динамического программирования.

2. ТАКОЙ РАЗНЫЙ ФАКТОРИАЛ

2.1 «ИМПЕРАТИВНЫЙ» ФАКТОРИАЛ

«Императивный» – от латинского «командный». Императивная постановка задачи: факториал натурального числа $N > 0$ – это произведение всех натуральных чисел от 1 до N. Императивный псевдокод состоит из команд (операторов над памятью). Например, для вычисления факториала можно предложить следующий низкоуровневый[1] императивный код:

```
1: var n, f : integer ;
2: input(n) ;
3: f: = 1 ;
4: if n = 1 then goto 8 ;
5: f : = f*n ;
6: n: = n-1 ;
7: goto 4 ;
8: output(f) ;
9: end.
```

Команды преобразуют значения в «ячейках памяти» вычислительной машины[2]. Запуск императивной программы соответствует размещению ее команд в памяти и последовательному исполнению команд, начиная с первой (команды с номером 1). Например, вычисление факториала числа 3 может быть изображено таблицей, в которой строки представляют последовательные моментальные состояния памяти некоторой «модельной» вычислительной машины; часть такой таблицы приведенана рис. 1[3].

Строчки этой таблицы надо понимать так. Строка, соответствующая первому моменту времени, описывает результат исполнения команды с номером 1 (var n, f : integer ;): она выделила две ячейки

Номер ячейки / Момент времени	1 зарезервирована для номера исполненной команды	2 зарезервирована для номера следующей команды		4	...	...	...
1	1	2	выделена для n	выделена для f	Свободная память		
2	2	3	3	выделена для f	Свободная память		
3	3	4	3	1	Свободная память		
4	4	5	3	1	Свободная память		
.....	.....	.....	.....	.....	.....		
12	4	8	1	6	Свободная память		
13	8	9	1	6	Свободная память		
14	9	Свободная память					

Рис. 1

[1] Мы используем код низкого уровня, чтобы сделать «прозрачнее» связь между операторами и памятью.
[2] Давайте считать, что любая ячейка способна хранить любое число в диапазоне от 0 до N!
[3] Память, занятая командами программы, не представлена в таблице.

памяти под переменные n и f (в нашем случае это ячейки 3 и 4), остальные ячейки оставила свободными и подготовила передачу управления команде со следующим по порядку номером 2. Второму моменту времени соответствует строка, которая описывает результат выполнения команды с номером 2 (input(n)): введенное значение (в нашем случае это 3) было занесено в ячейку, зарезервированную для n (ячейку 3), и подготовлена передача управления команде со следующим по порядку номером 3. В третий момент времени (см. соответствующую строку) в соответствии с командой с номером 3 (f: = 1) произошло присвоение значения 1 ячейке, зарезервированной для f (ячейке 4), и подготовлена передача управления команде со следующим по порядку номером 4. В четвёртый момент времени была исполнена команда с номером 4 (if n = 1 then goto 8), в результате чего значение, хранящееся в ячейке 3 (выделенной для n) было сравнено с 1. Так как значение было 3, то передачи управления команде с номером 8 не произошло, а была подготовлена передача управления команде со следующим по порядку номером 5. Подобным образом вычисления продолжаются до 13-го момента времени, когда после исполнения команды с номером 8 (output(f)) происходит вывод (на печать, экран и т. д.) значения 6, скопированного из ячейки, зарезервированной для f (ячейки 4), и подготовлена передача управления команде со следующим по порядку номером 9. В четырнадцатый (последний) момент работы программы была исполнена команда с номером 9 (end.), которая освободила всю память, включая ячейку 2, зарезервированную для номера следующей команды.

2.2 «ФУНКЦИОНАЛЬНЫЙ» ФАКТОРИАЛ

В функциональном программировании постановка задачи и решение задачи – определения функции, а вычисление – это цепочка равенств, где каждое следующее получается из предыдущего в результате однократного применения функции к аргументу (или аргументам). В частности, функциональная постановка задачи про факториал такова: факториал – это функция F, которая каждому натуральному числу сопоставляет натуральное число и удовлетворяет следующему тождеству: $F(x)$ = если $x = 1$ то 1 иначе $x*F(x-1)$. Функциональный псевдокод состоит из одного определения функции F, но включает два случая (или «клаузы[1]»):

$F : 1 = 1$;

$x = x* F(x - 1)$.

Приведённая программа означает, что если конкретное значение аргумента функции F есть 1, то значение функции F на этом аргументе равно 1; в противном случае значение функции F на этом аргументе равно значению аргумента, умноженному на значение функции F на декременте[2] аргумента. В более общем случае определение функции состоит из трех или более клауз или аргумент имеет более сложный вид, чем константа или переменная. Такое определение означает выбор в порядке перечисления первой клаузы, у которой выражение, стоящее слева от знака «=», может быть «отождествлено» со значениями аргумента функции[3]. Никаких ячеек памяти для хранения значений в функциональном программировании нет, но есть память (или «база клауз»), в которой хранится сама функциональная программа. Вызов функциональной программы состоит в применении одной из функций программы к конкретному значению аргумента. Вычисление вызванной функциональной программы – это цепочка равенств, соответствующих клаузам, начинающаяся с вызова одной из функций этой программы с фактическим значением аргумента и заканчивающаяся числом – значением этой функции на этом аргументе. Например, вычисление факториала числа 3:

[1] Предложение (*нем.*), высказывание.

[2] Декремент – это операция вычитания единицы.

[3] Далее мы приведём пример, как работает такой механизм.

$F(3)$ = (так как 3 не равно 1) = 3 * $F(3 - 1)$ = = 3 * $F(2)$ = (так как 2 не равно 1) = = 3 * (2 * $F(2 - 1)$) = 3 * (2 * $F(1)$) = = (так как 1 равно 1) = 3 * (2 * 1) = 3 * 2 = 6.

Только и всего, вместо длинной таблицы состояний памяти в императивном программировании.

2.3 «ЛОГИЧЕСКИЙ» ФАКТОРИАЛ

В логическом программировании постановка задачи есть аксиоматическое определение отношения между аргументами и результатами, а вычисление – это поиск доказательства. Например, логическое определение факториала может состоять из двух следующих аксиом, описывающих свойства отношения $P(m, n) \equiv$ «число m является факториалом числа n»:

$$P(1, 1) ;$$

$$P(m/n, n - 1) \Rightarrow \mathrm{P}(m, n).$$

Первая из этих аксиом означает, что «число 1 является факториалом числа 1», а вторая – что «если частное от деления нацело m на n является факториалом числа $(n - 1)$, то само число m является факториалом числа n». Функциональный псевдокод состоит из одного факта и одного правила[1]:

$$P\ (1, 1) ;$$

$$P(m, n) :: \mathrm{P}(m/n, n - 1).$$

Как и в функциональном программировании, никаких ячеек памяти для хранения значений в логическом программировании нет, но есть память (или «база знаний»), в которой хранится сама логическая программа. Вызов логической программы – это один из предикатов этой программы, в который подставлены фактические значения части аргументов, а значения некоторых аргументов предстоит определить так, что при их подстановке получится утверждение, доказуемое в аксиоматике, соответствующей этой логической программе. Унификация двух выражений в логическом программировании – это подбор значений переменных в этих выражениях, при котором оба выражения синтаксически совпадут. Вычисление по логической программе – это поиск «кратчайшего» доказательства с использованием унификации.

Например, вычисление факториала числа 3:

1) для доказательства $P(m, 3)$ надо доказать $P((m/3), 3 - 1)$, так как аргументы $P(m, 3)$ невозможно унифицировать с аргументами факта $P(1, 1)$;

2) доказательство $P((m/3), 3 - 1)$ совпадает с доказательством $P((m/3), 2)$, так как (3–1) есть 2;

3) для доказательства $P((m/3), 2)$ надо доказать $P(((m/3)/2), 2 - 1)$, так как аргументы $P((m.3), 2)$ невозможно унифицировать с аргументами факта $P(1, 1)$;

4) доказательство $P(((m/3)/2), 2 - 1)$ совпадает с доказательством $P(((m/3)/2), 1)$, так как (2–1) есть 1;

5) так как аргументы $P(((m/3)/2), 1)$ возможно унифицировать с аргументами факта $P(1, 1)$ посредством подстановки 6 в качестве значения переменной m, то таким образом доказано $P(6, 3)$, то есть что «число 6 является факториалом числа 3», чем завершается вычисление примера.

2.4. ЗАЧЕМ НУЖНЫ РАЗНЫЕ ПАРАДИГМЫ

Итак, на примере вычисления факториала были показаны особенности трёх парадигм программирования: императивной, функциональной и логической. Теперь самое время разобраться, зачем нужны разные парадигмы программирования для эффективного решения программистских задач. Для этого мы рассмотрим одну задачу и покажем, как её можно решить (причём сначала логически, затем – функционально, а потом – императивно).

Формулировку задачи начнём с головоломки: есть 15 монет, среди которых одна – фальшивая, а остальные – настоящие; 13 настоящих монет и фальшивая монета внешне одинаковы, а одна настоящая монета помечена; все настоящие монеты имеют равный вес, а фальшивая –

[1] Правила соответствуют конструкции «если – то», а факты – это просто утверждения.

вес, отличный от веса настоящей монеты; спрашивается, как найти фальшивую монету за 3 взвешивания на чашечных весах?

Если условия головоломки понятны, то можно перейти к формулировке задачи как обобщения этой головоломки: как среди N монет (среди которых могут быть несколько помеченных настоящих) найти единственную фальшивую монету (другого веса) за K взвешиваний? Хочется предупредить, что это достаточно трудная программистская задача. Достаточно сослаться на то, что в 2000 г. эта задача была включена под номером «H» в конкурсное задание азиатского регионального тура ACM International Collegiate Programming Contest.

Начнём обсуждение нашей задачи с анализа того, какая информация доступна человеку после серии из нескольких (от 0 до K) взвешиваний монеток и как эту информацию представлять. Перед первым взвешиванием нам известна 1 настоящая монетка, а про все остальные $(N - 1)$ монет ничего не известно. Во время первого взвешивания на чаши весов было положено по $M \in [1..(N/2)]$ монет; во время этого первого взвешивания эти чашки весов или уравновесились, или одна оказалась легче, а другая – тяжелее; поэтому перед вторым взвешиванием мы имеем:

– в первом из этих случаев нам известно $V \geq 2$ настоящих монет[1], а про все остальные $U = (N - V)$ монет ничего не известно,

– во втором из случаев нам известно, что L монеток были на более лёгкой чаше весов, H монеток были на более тяжёлой чаше весов, $(2M - 1) \leq (L + H) \leq 2M$, $|L - H| \leq 1$, а про все остальные $V = (N - L - H)$ монет известно, что они настоящие[2].

Если попытаться обобщить эти наблюдения на ещё несколько взвешиваний, то можно предположить, что информация, известная после серии взвешиваний, может быть представлена четырьмя числами:

U – число монет, про которые пока ничего сказать нельзя,

L – число монет, про которые пока неизвестно, настоящие они или нет, но они участвовали во взвешиваниях и оказались на более лёгкой чаше весов,

H – число монет, про которые пока неизвестно, настоящие они или нет, но они участвовали во взвешиваниях и оказались на более тяжёлой чаше весов,

V – число монет, про которые известно, что они настоящие,

причём выполняются следующие естественные ограничения: $U + L + H + V = N$ и если $L + H > 0$, то можно считать что $U = 0$ (так как если фальшивая монетка уже участвовала во взвешиваниях, то все остальные, кроме $L + H$ монет, точно настоящие).

Итак, мы можем принять, что информация, доступная после серии взвешиваний, представима четвёркой чисел (U, L, H, V), удовлетворяющей двум ограничениям $U + L + H + V = N$, и если $L + H > 0$, то $U = 0$. Теперь можно обсудить, что такое «взвешивание» при таком представлении информации. Так как $U > 0$ или $L + H > 0$, то «взвешивание» в этих двух случаях следует рассмотреть отдельно.

В случае, когда $U > 0$, «взвешивание» состоит в том, что мы кладём u_1 из U монет на первую чашку весов, u_2 из U монет на вторую чашку весов и для выравнивания числа добавляем $v = |u_1 - u_2|$ настоящих монет из V на ту чашку весов, на которой было меньше монет[3]. Имеют место очевидные неравенства: $u_1 + u_2 \leq U$ и $v = |u_1 - u_2| \leq V$, означающие, что монеты мы можем брать только из числа имеющихся.

В случае, когда $L + H > 0$, «взвешивание» состоит в том, что мы кладём l_1 из L

[1] $V = 2M$, если помеченная настоящая монетка участвовала во взвешивании, иначе $V = (2M + 1)$.

[2] $L = H = M$, если помеченная настоящая монетка не участвовала во взвешивании, иначе $|L - H| = 1$ и $L + H = (2M - 1)$.

[3] Нам кажется очевидным, что во взвешивании должно участвовать равное число монет на первой и на второй чашке весов. Поэтому, например, если $u_1 = 5$, а $u_2 = 3$, то на вторую чашку весов надо добавить $v = 2$ настоящих монет; тогда на обеих чашках окажется по 5 монет.

монет на первую чашку весов, l_2 из L монет на вторую чашку весов, h_1 из H монет на первую чашку весов, h_2 из H монет на вторую чашку весов и для выравнивания числа добавляем $v = |l_1 + h_1 - l_2 - h_2|$ настоящих монет из V на ту чашку весов, на которой было меньше монет. Имеют место очевидные неравенства: $l_1 + l_2 \le L$, $h_1 + h_2 \le H$ и $v = |l_1 + h_1 - l_2 - h_2| \le V$, означающие, что монеты мы можем брать только из числа имеющихся.

3. ЛОГИКА ПОИСКА МОНЕТКИ

В соответствии с логической парадигмой программирования, нам надо ввести отношение $P(U, L, H, V, M)$, означающее, что среди $U + L + H + V$ монет (где U, L, H и V имеют смысл, описанный выше в разделе 2.4) за M взвешиваний можно найти единственную фальшивую монету, и аксиоматизировать его. Наша аксиоматизация $A \times P$ будет состоять из четырёх аксиом.

Первая аксиома описывает случай, когда сразу можно сказать, что фальшивая монетка найдена:

$$U + L + H = 1 \Rightarrow P(U, L, H, V, M).$$

Словами эту аксиому можно прочитать так: если осталась только одна монета, о которой мы не знаем, что она настоящая, то она и есть фальшивая. Мы, однако, будем использовать чуть более общую аксиому:

$$U + L + H \le 1 \Rightarrow P(U, L, H, V, M).$$

Формально говоря, эта аксиома допускает исключительную ситуацию, когда $U = L = H = 0$, которая означает, что если единственную фальшивую монетку надо найти в пустом множестве монет или множестве, состоящем из одной монеты, то задача поиска уже решена[1].

Вторая аксиома формализует правило, что если $L + H > 0$, то можно считать что $U = 0$: $L + H > 0$ & $P(0, L, H, V + U, M) \Rightarrow P(U, L, H, V, M)$. Словами её можно прочитать так: если есть монетки, которые были на лёгкой или тяжёлой чашке весов, то все остальные монетки настоящие (причем, для того чтобы сделать этот вывод, не требуется дополнительно[2] взвешивать монетки). Эта аксиома носит вспомогательный характер и позволяет формализовать переход от M взвешиваний к $(M - 1)$-ому взвешиванию[3] в виде двух простых аксиом для $P(U, 0, 0, V, M)$ и $P(0, L, H, V, M)$, вместо одной, но сложной аксиомы для $P(U, L, H, V, M)$.

Эти две аксиомы формализуют следующее простое наблюдение: в наборе монет можно найти единственную фальшивую монетку за M взвешиваний тогда и только тогда, когда есть такое взвешивание, что при любом его исходе:

A. чаши уравновешены,

B. первая легче второй,

C. первая тяжелее второй –

фальшивую монетку можно найти за $(M - 1)$ взвешивание.

Третья аксиома описывает переход от M взвешиваний к $(M - 1)$ в случае, когда $U > 0$, но $L = H = 0$:

$$\exists\, u_1, u_2\ (0 < u_1 + u_2 \le U\ \&\ |u_1 - u_2| \le V\ \&$$
$$\text{a. } P(U - u_1 - u_2, 0, 0, V + u_1 + u_2, M - 1)\ \&$$
$$\text{b. } P(0, u_1, u_2, V + (U - u_1 - u_2), M - 1)\ \&$$
$$\text{c. } P(0, u_2, u_1, V + (U - u_1 - u_2), M - 1)\ ) \Rightarrow$$
$$\Rightarrow \mathrm{P(U,\ 0,\ 0,\ V,\ M)}.$$

Словами её можно прочитать так: если существует такой набор монет для первой и второй чашек весов ($\exists\ u_1, u_2$), что:

– эти монеты взяты из имеющихся «неопределённых» монет ($0 < u_1 + u_2 \le U$),

– имеется достаточно настоящих монет для взвешивания равных количеств на обеих чашках ($|u_1 - u_2| \le V$),

– при любом исходе взвешивания ($A = a$, $B = b$ и $C = c$) достаточно $(M - 1)$ взвешиваний,

то достаточно M взвешиваний для того чтобы найти фальшивую монетку ($P(U, 0, 0, V, M)$). Соответствие исходов

---

[1] Конечно же, задача найти единственную фальшивую монетку в множестве монет, которое может быть пусто, звучит несколько странно. Но формализуется задача вполне естественно: если множество монет не пусто, то найти в нём единственную фальшивую монетку. Да и аналогия у нас имеется: ведь принято считать, что 0! = 1.

[2] Обратите внимание, что значение M одно и то же в левой и правой частях аксиомы.

[3] То есть от $P(U, L, H, V, M)$ к $P(..., ..., ..., ..., M - 1)$.

A, B и C условиям a, b и c в аксиоме (на наш взгляд) является очевидным.

Четвёртая аксиома описывает переход от M взвешиваний к $(M-1)$ в случае, когда $U=0$, но $L+H>0$:

$\exists\, l_1, l_2, h_1, h_2\ (0 < l_1 + l_2 + h_1 + h_2 \,\&\, l_1 + l_2 \le L \,\&$
$h_1 + h_2 \le H \,\&\, |l_1 + h_1 - l_1 - l_2| \le V \,\&$
a. $P(0, L - l_1 - l_2, H - h_1 - h_2, V + l_1 + l_2 +$
$h_1 + h_2, M - 1)\ \&$
b. $P(0, l_1, h_2, V + (L - l_1) + (H - h_2), M - 1)\ \&$
c. $P(0, l_2, h_1, V + (L - l_2) + (H - h_1), M - 1)) \Rightarrow$
$\Rightarrow P(0, L, H, V, M)$.

Словами её можно прочитать так: если существует такой набор монет для первой и второй чашек весов ($\exists\ l_1, l_2, h_1, h_2$), что:

– эти монеты взяты из имеющихся «лёгких» и «тяжёлых» монет
$(0 < l_1 + l_2 + h_1 + h_2, l_1 + l_2 \le L$ и $h_1 + h_2 \le H)$,

– имеется достаточно настоящих монет для взвешивания равных количеств на обеих чашках ($|l_1 + h_1 - l_1 - l_2| \le V$),

– при любом исходе ($A = a$, $B = b$ и $C = c$) достаточно $(M-1)$ взвешиваний,

то, чтобы найти фальшивую монету, достаточно M взвешиваний.

В этой аксиоме, наверное, надо прокомментировать только то, почему исходы взвешиваний A, B и C соответствуют условиям a, b и c в аксиоме. Дело обстоит очень просто: если фальшивая монета легче и уже участвовала во взвешиваниях, то она не может при очередном взвешивании оказаться ни на уравновешенных чашках весов, ни на более тяжёлой чаше; аналогично, если фальшивая монета тяжелее и уже участвовала во взвешиваниях, то она не может при очередном взвешивании оказаться ни на уравновешенных чашках весов, ни на более лёгкой чаше.

Аксиоматизация $A \times P$ построена. Она является надёжной и полной в следующем смысле.

Утверждение 1. Для любого набора неотрицательных целых чисел (U, L, H, V, M) следующие утверждения эквивалентны:

1. В аксиоматической системе $A \times P$ можно доказать свойство $P(U, L, H, V, M)$.

2. Существует метод поиска единственной фальшивой монеты среди $N = U + L + H + V$ монет путём взвешивания их на чашечных весах не более M раз (где U, L, H, и V имеют смысл, описанный в разделе 2.4).

Доказательство (набросок). По построению из (2) следует (1). Следование (2) из (1) доказывается индукцией по $S = U + L + H$. Базис индукции – это случай $S \le 1$ – является тривиальным в силу первой аксиомы. Индукционная гипотеза состоит в предположении, что утверждение верно для всех $S \le K$ (где K – некоторое натуральное число). Шаг индукции – переход от $S \le K$ к $S = (K+1)$ – основан на следующем простом аргументе: для любой пятёрки натуральных чисел (U, L, H, V, M), если $(U + L + H) > 1$, то в аксиоматике $A \times P$ есть аксиома, в которой заключение импликации унифицируется с $P(U, L, H, V, M)$, а все наборы значений (U', L', H', V', M'), которые могут возникнуть в посылке этой импликации под квантором, удовлетворяют неравенству $(U' + L' + H') < (U + L + H)$; поэтому, по предположению индукции, $P(U', L', H', M-1)$ влечёт существование метода поиска фальшивой монеты за $(M-1)$ взвешиваний. ▪

Вот, собственно, и все: наша задача сформулирована в логической парадигме в виде аксиоматизации предиката P. И, соответственно, практически готова логическая программа: осталось аксиомы «превратить» в факты и правила.

Первая аксиома превращается в четыре факта

$P(0, 0, 0, V, M)$, $P(1, 0, 0, V, M)$,
$P(0, 1, 0, V, M)$ и $P(0, 0, 1, M)$.

Вторая аксиома превращается в правило

$P(U, L, H, V, M) :: L + H > 0\ \&$
$P(0, L, H, V + U, M)$.

Третья аксиома трансформируется в правило

$P(U, 0, 0, V, M) :: \exists\, u_1, u_2\colon 0 < u_1 + u_2 \le U\ (|u_1 - u_2| \le V\ \&\ P(U - u_1 - u_2, 0, 0, V + u_1 + u_2, M - 1)$
$\&\ P(0, u_1, u_2, V + (U - u_1 - u_2), M - 1)$
$\&\ P(0, u_2, u_1, V + (U - u_1 - u_2), M - 1))$,

а четвёртая – в правило

$P(0, L, H, V, M) :: \exists\, l_1, l_2 : l_1 + l_2 \le L; h_1, h_2:$

$h_1 + h_2 \le H$
$(\ 0 < l_1 + l_2 + h_1 + h_2\ \&\ |l_1 + h_1 - l_1 - l_2| \le V\ \&$
$P(0, L - l_1 - l_2, H - h_1 - h_2, V + l_1 + l_2 + h_1 + h_2, M - 1)\ \&$
$P(0, l_1, h_2, V + (L - l_1) + (H - h_2), M - 1)\ \&$
$P(0, l_2, h_1, V + (L - l_2) + (H - h_1), M - 1)\ )$.

Заметим, что перед нами действительно программа, вернее алгоритм, записанный на логическом псевдокоде: его можно выполнить для любого набора входных данных (U, L, H, V, M), так как справа во всех правилах используются только ограниченные кванторы существования. Если эту логическую программу занести в базу знаний машины логического вывода, то машина сможет доказать запрос $P(14, 0, 0, 1, 3)$, соответствующий нашей головоломке, и решить другие запросы, например, $P(39, 0, 0, 1, ?)$, «требующий» найти число взвешиваний, которых достаточно для того, чтобы найти единственную фальшивую монетку среди 39 монет с использованием дополнительной помеченной настоящей монетки.

4. ФУНКЦИОНАЛЬНЫЙ ПОДХОД К ЧИСЛУ ВЗВЕШИВАНИЙ

Описанный логический алгоритм хорош простотой дизайна, но он, однако, имеет и свои существенные недостатки. В частности, нет никакой уверенности, что в ответ на запрос $P(39, 0, 0, 1, ?)$ машина логического вывода «выдаст» ответ $P(39, 0, 0, 1, \mathbf{4})$, означающий, что четырёх взвешиваний достаточно, чтобы найти единственную фальшивую монетку среди 39 монет с использованием дополнительной помеченной настоящей монетки; не исключено, что на этот запрос машина выдаст очевидный ответ $P(39, 0, 0, 1, \mathbf{39})$, ведь, в силу утверждения 1, очевидно, что в $A\times P$ можно доказать $P(39, 0, 0, 1, \mathbf{39})$ (то есть что единственную фальшивую монету можно найти среди 39 монет за 39 взвешиваний). В логическом программировании поправить дело можно, но достаточно сложно, а вот в функциональном программировании эта проблема решается достаточно просто.

Утверждение 2.

Пусть M: $\mathbf{N}\times\mathbf{N}\times\mathbf{N}\times\mathbf{N} \to \mathbf{N}$ – функция[1], такая что для любого набора из U, L, H и V монет (где U, L, H и V имеют смысл, описанный в разделе 2.4) равна минимальному числу взвешиваний монет на чашечных весах, которого достаточно для идентификации единственной фальшивой монеты среди $N = U + L + H + V$ монет. Тогда для этой функции имеют место следующие равенства (см. рис. 2).

Доказательство (набросок). Эти равенства имеют место в силу соответствующих аксиом системы $A\times P$. Первые два из этих равенств не нуждаются в дополнительных объяснениях. А вот третье и четвёртое равенства, наверное, следует прокомментировать. В этих равенствах в правой части «1 +» соответствует переходу от $(M - 1)$ к M для числа взвешиваний в соответствующих аксиомах $A\times P$, «min» заменяет кванторы существования, так как нас интересует наискорейший поиск монетки, а «max» заменяет конъюнкцию трёх усло-

1. если $U + L + H \le 1$, то $M(U, L, H, V) = 0$;
2. если $L + H > 0$, то $M(U, L, H, V) = M(0, L, H, V + U)$;
3. $M(U, 0, 0, V) = 1 + \min_{0< u_1 + u_2 \le U,\ |u_1 - u_2| \le V} \max$
   $\{M(U - u_1 - u_2, 0, 0, V + u_1 + u_2), M(0, u_1, u_2, V + U - u_1 - u_2), M(0, u_2, u_1, V + U - u_1 - u_2)\}$;
4. $M(0, L, H, V) = 1 + \min_{l_1 + l_2 \le L,\ h_1 + h_2 \le H,\ 0 < l_1 + l_2 + h_1 + h_2,\ |l_1 + h_1 - l_1 - l_2| \le V} \max$
   $\{M(0, L - l_1 - l_2, H - h_1 - h_2, V + l_1 + l_2 + h_1 + h_2), M(0, l_1, h_2, V + (L - l_1) + (H - h_2)),$
   $M(0, l_2, h_1, V + (L - l_2) + (H - h_1))\})$.

Рис. 2

[1] Здесь $\mathbf{N}$ – множество натуральных чисел, а не число монет N.

вий, поскольку необходимо учесть число оставшихся взвешиваний при любом исходе данного взвешивания. ▪

Утверждение 3. Существует единственная функция M: $\mathbf{N}\times\mathbf{N}\times\mathbf{N}\times\mathbf{N} \to \mathbf{N}$, удовлетворяющая равенствам 1 – 4 из утверждения 2.

Доказательство (набросок). Существование такой функции следует из утверждения 2. Единственность (то есть равенство функции из утверждения 2) доказывается так же, как и утверждение 1 – индукцией по $S = U + L + H$. ▪

Таким образом, на основании утверждения 3, функциональная постановка задачи о необходимом и достаточном числе M взвешиваний для поиска единственной фальшивой монетки среди $U + L + H + V$ монет[1] готова: M – это функция M: $\mathbf{N}\times\mathbf{N}\times\mathbf{N}\times\mathbf{N} \to \mathbf{N}$, удовлетворяющая равенствам 1–4 из утверждения 2. Соответствующий функциональный псевдокод, представляющий алгоритм решения задачи, состоит из следующих клауз (см. рис. 3).

Этот псевдокод отличается от определения функции M в постановке задачи тем, что первое условное равенство «$M(U, L, H, V) = 0$, если $U + L + H = 1$» расписано в виде четырёх клауз, а второе условное равенство «$M(U, L, H, V) = M(0, L, H, V + U)$, если $L + H > 0$» заменено на клаузу «$(U, L, H, V) = M(0, L, H, V + U)$», которая поставлена в конец списка. Первое условное равенство заменено на клаузы $(1, 0, 0, V) = 0$, $(0, 1, 0, V) = 0$, $(0, 0, 1, V) = 0$ и $(0, 0, 0, V) = 0$. Помещение клаузы «$(U, L, H, V) = M(0, L, H, V + U)$» в конец списка объясняется тем, что в случае, когда определение функции состоит из нескольких клауз, такое определение означает выбор в порядке перечисления первой клаузы, у которой выражение, стоящее слева от знака «=», может быть «отождествлено» со значениями фактических аргументов функции, а так как левые части первых шести клауз имеют специальный вид $(1, 0, 0, V)$, $(0, 1, 0, V)$, $(0, 0, 1, V)$, $(U, 0, 0, V)$ и $(0, L, H, V)$, а левая часть последней – самый общий вид (U, L, H, V), то последняя клауза как раз будет «работать» только тогда, когда $L + V > 0$.

В окончании этой части давайте посмотрим, как работает описанный функциональный алгоритм (на примере вычисления необходимого и достаточного числа взвешиваний для поиска единственной фальшивой монеты среди четырёх с использованием одной дополнительной настоящей монеты)[2] (см. рис. 4).

Можно заметить наличие дублирующих вызовов функций: дважды встречают-

$$
\begin{aligned}
M\colon\ & (1, 0, 0, V) = 0;\\
& (0, 1, 0, V) = 0;\\
& (0, 0, 1, V) = 0;\\
& (0, 0, 0, V) = 0;\\
& (U, 0, 0, V) = 1 + \min_{0 < u_1 + u_2 \le U,\ |u_1 - u_2| \le V} \max\{M(U - u_1 - u_2, 0, 0, V + u_1 + u_2),\\
& \qquad\qquad M(0, u_1, u_2, V + U - u_1 - u_2),\\
& \qquad\qquad M(0, u_2, u_1, V + U - u_1 - u_2)\};\\
& (0, L, H, V) = 1 + \min_{l_1 + l_2 \le L,\ h_1 + h_2 \le H,\ 0 < l_1 + l_2 + h_1 + h_2,\ |l_1 + h_1 - l_1 - l_2| \le V}\\
& \qquad\qquad \max\{M(0, L - l_1 - l_2, H - h_1 - h_2, V + l_1 + l_2 + h_1 + h_2),\\
& \qquad\qquad M(0, l_1, h_2, V + (L - l_1) + (H - h_2)),\\
& \qquad\qquad M(0, l_2, h_1, V + (L - l_2) + (H - h_1))\};\\
& (U, L, H, V) = M(0, L, H, V + U).
\end{aligned}
$$

Рис. 3

[1] U, L, H и V имеют тот же смысл, что и выше.

[2] В этом примере достаточно много рутинного, поэтому в нём сразу выполнены некоторые арифметические вычисления и не указано, какая применяется клауза.

$$M(4, 0, 0, 1) = 1 + \min_{0 < u_1 + u_2 \le 4,\ |u_1 - u_2| \le 1} \max\{M(4 - u_1 - u_2, 0, 0, 1 + u_1 + u_2), M(0, u_1, u_2, 5 - u_1 - u_2), M(0, u_2, u_1, 5 - u_1 - u_2)\} =$$
$$= 1 + \min_{(u_1, u_2) = (0,1), (1,0), (1,1), (1,2), (2,1), (2,2)} \max\{M(4 - u_1 - u_2, 0, 0, 1 + u_1 + u_2), M(0, u_1, u_2, 5 - u_1 - u_2), M(0, u_2, u_1, 5 - u_1 - u_2)\} =$$
$$= 1 + \min\{ \max\{M(3, 0, 0, 2), M(0, 0, 1, 4), M(0, 1, 0, 4)\},$$
$$\max\{M(3, 0, 0, 2), M(0, 1, 0, 4), M(0, 0, 1, 4)\}, \max\{M(2, 0, 0, 3), M(0, 1, 1, 3), M(0, 1, 1, 3)\},$$
$$\max\{M(1, 0, 0, 4), M(0, 1, 2, 2), M(0, 2, 1, 2)\}, \max\{M(1, 0, 0, 4), M(0, 2, 1, 2), M(0, 1, 2, 2)\},$$
$$\max\{M(0, 0, 0, 5), M(0, 2, 2, 1), M(0, 2, 2, 1)\} \}.$$

Рис. 4

ся $M(3, 0, 0, 2)$, $M(0, 1, 1, 3)$, $M(0, 1, 2, 2)$, $M(0, 2, 1, 2)$ и $M(0, 2, 2, 1)$; единственный «уникальный» вызов функции – это $M(2, 0, 0, 3)$. Возможный вариант оптимизации состоит в том, чтобы вычислять только новые[1] вызовы функции M, сохранять их значения в памяти (в виде списка или хеш-таблицы), а значения повторных[2] вызовов функции M уже брать из памяти. Эта техника хорошо известна в функциональном программировании как мемоизация (memoization) [7].

5. КАТЕГОРИЧЕСКИЙ ИМПЕРАТИВ: БЫСТРО НАЙТИ!

5.1. НА ВСЕ СЛУЧАИ ЖИЗНИ

Можно предположить, что эффективность мемоизации можно повысить, если выполнять ее в ленивом режиме: вместо вычисления всякого нового вызова функции M и сохранения его значения, можно сохранять сами вызовы (не вычисляя их) и вычислить их только в конце. Однако ленивая мемоизация возможна только тогда, когда можно без исполнения вызова функции (то есть статически) предсказать множество всех вызовов, которые могут возникнуть при выполнении этого вызова. К счастью, в нашем случае дело обстоит именно так в соответствии со следующим утверждением.

Утверждение 4.

Пусть M: $\mathbf{N}\times\mathbf{N}\times\mathbf{N}\times\mathbf{N} \to \mathbf{N}$ – функция, определённая функциональным алгоритмом, описанным в части 4. Тогда для любых неотрицательных целых чисел U, L, H, и V множество вызовов функции M, которые возникают при вычислениях $M(U, L, H, V)$, содержится в следующем множестве[3] (см. рис. 5).

Доказательство можно провести индукцией по $S = U + L + H$. ▪

$$\mathrm{FCM}(U, L, H, V) =$$
- $= \{M(0, L, H, V') : V' \le V + U\} \cup \{M(0, L', H', V') :$
  $(L' < L$ и $H' \le H)$ или $(L' \le L$ и $H' < H)$, $V' \le V + U + L + H - L' - H'\}$,
  если $U > 0$ и $L + H > 0$;
- $= \{M(0, L', H', V') :$
  $(L' < L$ и $H' \le H)$ или $(L' \le L$ и $H' < H)$, $V' \le V + U + L + H - L' - H'\}$,
  если $U = 0$ и $L + H > 0$;
- $= \{M(U', 0, 0, V') :$
  $U' < U$, $V' \le V + U - U'\} \cup$
  $\{M(0, L', H', V') : U \ge L' + H', V' \le V + U - L' - H')\}$, если $U > 0$ и $L + H = 0$.

Рис. 5

[1] То есть для тех значений аргументов (фактических параметров), для которых ранее функция не вычислялась.

[2] То есть для тех значений аргументов (фактических параметров), для которых функция уже была вычислена ранее.

[3] FCM – аббревиатура от «Function Calls in M».

Но в таком случае мы естественно получаем следующую императивную постановку задачи о поиске фальшивой монетки: для заданного $N \geq 0$ заполнить четырёхмерную таблицу $T[0..N, 0..N, 0..N, 0..N]$ по следующему правилу

$T[U, L, H, V]$ = минимальное число взвешиваний, позволяющее найти единственную фальшивую монету среди

$$S = U + L + H + V \leq N \text{ монет,}$$

где U, L, H и V имеют смысл, описанный в разделе 2.4.

Неформально говоря, императивная постановка задачи предлагает найти и занести в таблицу T значения функции $M(U, L, H, V)$ «на все случаи жизни», то есть для всевозможных значений аргументов, которые удовлетворяют неравенству $U + L + H + V \leq N$.

Императивное решение – последовательное заполнение таблицы в порядке возрастания значения суммы

$$S = U + L + H + V,$$

причём для каждого значения S этой суммы сначала надо вычислить все значения $M(0, L, H, V)$, а потом – все значения $M(U, 0, 0, V)$ в соответствии с правилами, уже знакомыми нам из функционального решения задачи (см. рис. 6).

Соответствующий императивный алгоритм может выглядеть, например, так (см. листинг 1).

Вот, собственно, и всё: императивное решение готово. Вывод, который можно сделать прямо сейчас – это вывод о пользе разных «программирований»: к эффективному императивному алгоритму мы пришли через анализ неэффективного функционального алгоритма, который был получен из логического алгоритма.

5.2 ЧТО ДЕЛАТЬ?

Вернее, что делать читателю дальше? Во-первых, можно попробовать запрограммировать все три алгоритма на соответствующих алгоритмических языках и сравнить их эффективность. Но можно попробовать решить и реализовать следующую более сложную задачу: написать программу с тремя входными данными

U – количество монет, среди которых надо найти единственную фальшивую,

V – дополнительное количество помеченных настоящих монет (возможно $V = 0$),

M – разрешённое количество взвешиваний,

результатом которой является или *impossible* или другая выполнимая диалоговая программа ALPHA (на том же самом языке), реализующая стратегию определения (не более чем) за M взвешиваний единственной фальшивой монеты среди U монет с использованием дополнительных V помеченных настоящих монет. Исходная программа должна выводить *impossible* тогда и только тогда, когда такой стратегии не существует. Программа ALPHA осуществляет стратегию поиска в следующем смысле:

Все $(U + V)$ монет имеют разные номера от 1 до $(U + V)$, все помеченные настоящие монеты имеют номера от 1 до V. Диалог с ALPHA начинается с выбора пользователем номера фальшивой монеты (от $(V + 1)$ до $(V + U)$ включительно) и её веса по отношению к настоящим мо-

- $T(1, 0, 0, V) = T(0, 1, 0, V) = T(0, 0, 1, V) = T(0, 0, 0, V) = 0$ для всех $V[0..N]$;
- $T(U, 0, 0, V) = 1 + \min_{0 < u_1 + u_2 \leq U, |u_1 - u_2| \leq V} \max\{T(U - u_1 - u_2, 0, 0, V + u_1 + u_2),$
  $T(0, u_1, u_2, V + U - u_1 - u_2), T(0, u_2, u_1, V + U - u_1 - u_2)\}$;
- $T(0, L, H, V) = 1 + \min_{l_1 + l_2 \leq L,\ h_1 + h_2 \leq H,\ 0 < l_1 + l_2 + h_1 + h_2,\ |l_1 + h_1 - l_1 - l_2| \leq V}$
  $\max\{T(0, L - l_1 - l_2, H - h_1 - h_2, V + l_1 + l_2 + h_1 + h_2), T(0, l_1, h_2, V + (L - l_1) + (H - h2)),$
  $T(0, l_2, h_1, V + (L - l_2) + (H - h_1))\}$;
- $T(U, L, H, V) = T(0, L, H, V + U)$.

Рис. 6

Листинг 1

```
const n = 100 ;
var u, l, h, v, s, l1, l2, h1, h2, u1, u2 : integer ;
var t : integer array of [0..n, 0..n, 0..n, 0..n] ;
for v = 0 to n do
   begin t(1, 0, 0, v): = 0 ; t (0, 1, 0, v): = 0 ; t(0, 0, 1, v): = 0 ; t(0, 0, 0, v) = 0 end ;
for s = 2 to n do
begin
   for l = 0 to s do
      for h = 0 to (s – l) do
         t(0, l, h, s – l – h):= 1 + min_{l1 + l2 ≤ l, h1 + h2 ≤ h, 0 < l1 + l2 + h1 + h2, |l1 + h1 – l1 – l2| ≤ (s – l – h)
            max{t(0, l – l1 – l2, h – h1 – h2, s – l – h + l1 + l2 + h1 + h2),
               t(0, l1, h2, s – l1 – h2), t(0, l2, h1, s – l2 – h1)} ;
   for u = 0 to s do
      t(u, 0, 0, s – u):= 1 + min_{0 < u1 + u2 ≤ u, |u1 – u2| ≤ (s – u)
         max{t(u – u1 – u2, 0, 0, s – u + u1 + u2),
            t(0, u1, u2, s – u1 – u2), t(0, u2, u1, s – u1 – u2)} ;
   for u = 0 to s do
      for l = 0 to (s – u) do
         for h = 0 to (s – u – l) do
            t(u, l, h, v):= t(0, l, h, s – l – h)
end.
```

нетам (легче или тяжелее). Далее диалог состоит из последовательности раундов, количество которых не может превышать M. В каждом раунде программа ALPHA выводит два непересекающихся подмножества номеров монет для помещения их на первую и вторую чаши весов и запрос к пользователю о результате взвешивания. Пользователь, в свою очередь, отвечает, какая чашка легче, в соответствии со своим начальным выбором номера фальшивой монеты и ее относительного веса. Сессия заканчивается выводом ALPHA номера фальшивой монеты.

Так как задача состоит в том, чтобы написать программу, которая порождает другую программу, то можно первую программу назвать метапрограммой, а саму задачу – задачей о метапрограмме. Императивный алгоритм решения задачи о метапрограмме можно найти в [8] и [9].

Но есть еще одна интересная головоломка о монетах, алгоритмическое решение обобщения которой автору пока не известно[1]. Вот она: есть 40 монет, среди которых три – фальшивые, а остальные – настоящие, все монеты внешне одинаковы; все настоящие монеты имеют равный вес, а фальшивые – легче настоящих монет; спрашивается, как найти 18 настоящих монет за 3 взвешивания на чашечных весах? Если условия этой головоломки понятны, то можно перейти к формулировке программисткой задачи как обобщения этой головоломки: как среди N монет найти V настоящих за K взвешиваний, если известно, что среди этих монет есть ровно L лёгких фальшивых монет[2], а все настоящие монеты имеют равный вес. Условно её можно назвать задачей о том, как найти деньги (например, на исследования). Императивные, функциональные и логические алгоритмы решения этой задачи присылай-

[1] То есть автор знает, как решить конкретную головоломку, но не знает алгоритма, решающего обобщение этой головоломки.

[2] Известно именно число L лёгких монет, а не сами монеты.

Литература

1. *Душкин Р.В.* Функциональный подход в программировании. Потенциал, 2009, № 8. С. 47–55.

2. *Душкин Р.В.* Задача о ранце. Потенциал, 2009, № 9. С. 48–55.

3. *Шилов Н.В.* Заметки о преподавании парадигм программирования. IV Международная научно-практическая конференция «Современные информационные технологии и ИТ-образование». М.: ИНТУИТ.РУ, 2009. С. 318–325.

4. *Кун Т.* Структура научных революций. Издательство АСТ, 2003.

5. *Флойд Р.* О парадигмах программирования. В кн.: Лекции лауреатов премии Тьюринга. М: Мир, 1993.

6. *McCarthy J.* Recursive Functions of Symbolic Expressions and Their Computation by Machine. Communications of ACM. 1960. Vol. 3 (4).

7. *Астапов Д.* Рекурсия + Мемоизация = Динамическое Программирование. Практика функционального программирования, № 3, 2009, С. 17–33 / http://fprog.ru/2009/issue3/.

8. *Ии К., Шилов Н. В., Бодин Е. В.* О программных логиках – просто. Системная Информатика. Вып. 8, Новосибирск: Наука, 2002. С. 206–249.

9. *Shilov, Yi.* How to find a coin: propositional program logics made easy. Current Trends in Theoretical Computer Science, World Scientific. Vol. 2, 2004. P.181–214.

Abstract

This paper is about programming paradigms, how different paradigms can fertilize each other. In particular the paper discusses a challenging contest programming problem and solves it in three programming paradigms: logic, functional and imperative. It can be considered as a case study of algorithm inversion, since we start with logic algorithm, that answers the question «Is balancing M times sufficient for detecting a single fake in a set of coins?», and finishes with imperative algorithm, that effectively computes the minimal number of balancing that is sufficient for detection the fake. Functional paradigm is used for developing an intermediate functional algorithm that also computes the minimal number of balancing, but inefficiently, while the efficient imperative algorithm is a «lazy memoization» of the functional one.

Keywords: programming paradigms, imperative programming, functional programming, logical programming, memoization, dynamic programming.

Шилов Николай Вячеславович, кандидат физ.-мат. наук, старший научный сотрудник Института систем информатики имени А.П. Ершова СО РАН

shilov@iis.nsk.su