В.С.Ярош

Решение особого вопроса теории чисел

Краткое введение

Математические доказательства могут быть индуктивными и дедуктивными.
Если доказательство берёт свое начало в недрах натуральных чисел и постепенно поднимается на уровень сложных математических форм, то такое доказательство является индуктивным . Оно всегда заканчивается формулами для вычисления доказуемого факта.
Если доказательство берёт своё начало в сложной математической форме и пытается выйти на формулу или на формулы для вычисления доказуемого факта , то такое доказательство называется дедуктивным. Ему очень трудно выйти на вычисляемый результат.
Индуктивное доказательство вырастает естественным путём , как ствол дерева вырастает из собственных корней.
Дедуктивное доказательство берёт своё начало в «веточке» математического древа , пытаясь выйти к его стволу , пробиваясь сквозь встречные потоки , поднимающиеся из корней древа.
Примером индуктивного доказательства может служить элементарное доказательство Последней теоремы Пьера Ферма (доказательство Ярош), берущее своё начало в недрах натуральных чисел и заканчивающееся формулами для вычисления бесконечных множеств корней уравнения Ферма..
Примером дедуктивного доказательства может служить доказательство Последней теоремы (доказательствоУайлса) , берущее своё начало в математических зарослях эллиптических кривых и не заканчивающееся формулами для вычисления доказуемого факта.

«Что такое математика? Математика содержит в себе черты волевой деятельности, умозрительного рассуждения и стремления к эстетическому совершенству. Её основные и взаимно противоположные элементы – логика и интуиция, анализ и конструкция, общность и конкретность… только совместное действие этих полярных начал и борьба за их синтез обеспечивают жизненность, полезность и высокую ценность математической науки»
Richard Courant and Herbert Robbins.

Моё решение проблемы Conjecture Beal основано на единстве интуиции, логики и геометрической интерпретации этих свойств разума.

Геометрическая интерпретация – в соответствии с Принципом всеобщей (геометрической) ковариантности, который подробно описан на страницах:

http://yvsevolod-26.narod.ru/index.html

http://yvsevolod-27.narod.ru/index.html

Формулы, которыми я пользуюсь, сконструированы мной в полном соответствии с упомянутыми выше полярными свойствами математической науки.

Курант и Роббинс полагают:

«Доказательство существования трансцендентных чисел ещё до Кантора было дано Ж.Лиувиллем. Оно даёт возможность на самом деле конструировать примеры таких чисел…Конструировать пример, вообще говоря, сложнее, чем доказывать существование.»

Мои формулы, приведенные ниже, позволяют конструировать, как бесконечные множества чисел натуральных и рациональных, так и бесконечные множества чисел иррациональных.

Имея в виду изложенное, перейдём к рассмотрению проблемы. Суть её состоит в следующем.

Conjecture Beal не включает в себя вопрос о количестве троек целых чисел (A.B,C) , которые имеют общие множители и которые удовлетворяют уравнениям:

$$A^x + B^y = C^z$$

$$A^p x + B^q y = C^r z$$

Вопрос о количествах троек (A,B,C) - это особый вопрос, решаемый вне Conjecture Beal.
Тем не менее я решаю этот вопрос на тот случай, если какой-то оппонент захочет поставить такой вопрос на пути признания моего доказательства справедливости Conjecture Beal.

Решение особого вопроса.

И решение Conjecture Beal , и решение особого вопроса берут своё начало в недрах бесконечного множества натуральных чисел.
Здесь, как известно, существует кажущееся противоречие, которое заключается в кажущейся несовместимости двух утверждений теории чисел:

1. **Количество рациональных чисел (рациональных точек) всюду плотно на прямой.**
2. **Количество чисел иррациональных также всюду плотно на прямой.**

Мой алгоритм :

$$\{[(1+1)=2]\times 2^{n}\}\times N$$

$$\{[(1+2^{3})=3^{2}]\times (3^{3})^{n}\}\times N \qquad \textbf{(A)}$$

и мои формулы

$$a_{*}=[a_{o}{}^{2}\times D_{n}]^{1/n}$$

$$b_{*}=[b_{o}{}^{2}\times D_{n}]^{1/n}$$

$$c_{*}=[c_{o}{}^{2}\times D_{n}]^{1/n}$$

$$D_{n}=[a_{o}{}^{n-2}+b_{o}{}^{n-2}+c_{o}{}^{n-2}]^{1/n}/3$$

в которых :

$$a_{o}=v^{2}-u^{2}$$

$$b_{o}=2vu$$

$$c_{o}=v^{2}+u^{2}$$

примитивные тройки Пифагора, строящиеся из любой пары натуральных чисел различной чётности:

$$v > u \quad \textbf{(14)}$$

ДЕМОНСТРИРУЮТ ПРОЦЕСС ПРЕВРАЩЕНИЯ НАТУРАЛЬНЫХ ЧИСЕЛ В ЧИСЛА РАЦИОНАЛЬНЫЕ И ИРРАЦИОНАЛЬНЫЕ

Тройки иррациональных чисел $(a_{*,} b_{*}, c_{*})$ **, построенные из чисел натуральных и рациональных, формируют бесконечные множества решений базисных уравнений :**

$$a_{*}\ a_{*}^{\ n} + b_{*}^{\ n} = c_{*}^{\ n}$$

а тройки :

$$a = a_{*} \times S$$

$$b = b_{*} \times S$$

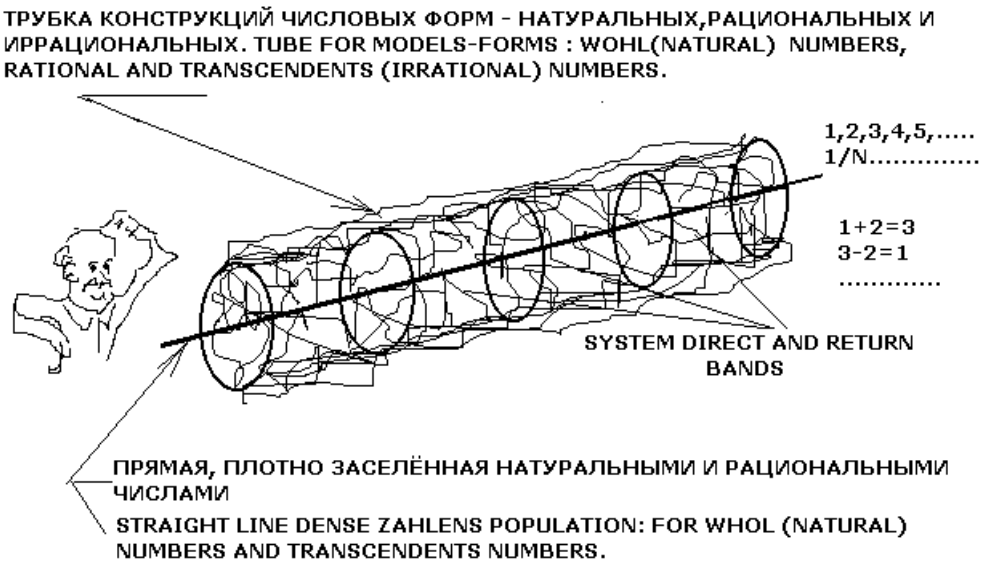
$$c = c_{*} \times S$$

где S любой множитель

составляют бесконечные множества решений любого уравнения:

$$a^{n} + b^{n} = c^{n}$$

Всё это многообразие чисел можно представить в виде геометрической модели.
Внутри такой модели пролегает прямая, плотно заселённая натуральными и рациональными числами.
Прямая находится внутри трубки, тело которой плотно заселено числами иррациональными.
Между прямой и трубкой пролегают нити прямых и обратных связей.



Фиг. 1

Аналогичную картину прямых и обратных связей мы наблюдаем и между бесконечным множеством уравнений, формируемых алгоритмом (А), и множеством уравнений типа :

$$1+2^3=3^2$$

$$2^5+7^2=3^4$$

$$7^3+13^2=2^9$$

$$2^7+17^3=71^2$$

$$3^5+11^4=122^2$$

$$17^7+76271^3=21063928^2$$

$$1414^3+2213459^2=65^7$$

$$43^8+96222^3=30042907^2$$

$$9262^3+15312283^2=113^7$$

$$33^8+1549034^2=15613^3$$

которые не имеют общих множителей.

Каждое из упомянутых множеств взаимосвязанных чисел образует свою собственную трубку, внутри которой пролегает прямая натуральных и рациональных чисел.
Между описанными здесь трубками числовых и геометрических многообразий существует сеть прямых и обратных связей.

Алгоритм конструирования сети этих связей известен только мне.

Я его опубликую после того, как Andrew Beal выплатит мне обещанный им приз за решение Conjecture Beal.
Решение Conjecture Beal принадлежит мне.
Вся его сущность заключена в простейшей конструкции алгоритма (А) .
Более подробную информацию об этом алгоритме читатель найдёт на сайте:
http://Conject.narod.ru/index.html

В заключение я хочу обратить внимание читателей на следующее очень важное обстоятельство :

ВСЕ ИСТИННЫЕ ФУНДАМЕНТАЛЬНЫЕ ЗАКОНОМЕРНОСТИ ПРЕДЕЛЬНО ПРОСТЫ И ЛАКОНИЧНЫ.

Конструкция моего фундаментального алгоритма (А) предельно проста и лаконична:

$$\{[(1+1)=2]\times 2^{n}\}\times N$$

$$\{[(1+2^{3})=3^{2}]\times (3^{3})^{n}\}\times N \qquad \textbf{(A)}$$

Эта конструкция генерирует бесконечное множество уравнений, подтверждающих справедливость Conjecture Beal.

Разве можно сравнить эту конструкцию с конструкциями эллиптических кривых A.Wiles ?
Или с конструкцией программы Peter Norvig , копию которой для наглядности я прилагаю :

Beal's Conjecture: A Search for Counterexamples

Beal's Conjecture is this:

There are no positive integers x,m,y,n,z,r satisfying the equation

$$x^m + y^n = z^r$$

where $m,n,r > 2$ and x,y,z are co-prime (that is, $gcd(x,y) = gcd(y,z) = gcd(x,z) = 1$).

There is a $75,000 prize for the first proof or disproof of the conjecture. The conjecture is obviously related to Fermat's Last Theorem, which was proved true by Andrew Wiles in 1994. A wide array of sophisticated mathematical techniques could be used in the attempt to prove the conjecture true (and the majority of mathematicians competent to judge seem to believe that it likely is true).

Less sophisticated mathematics and computer programming can be used to try to prove the conjecture false by finding a counterexample. This page documents my progress in this direction.

There are two things that make this non-trivial. First, we quickly get beyond the range of 32 or 64 bit integers, so any program will need a way of dealing with arbitrary precision integers. Second, we need to search a very large space: there are six variables, and the only constraint on them is that the sum must add up.

Suppose we wanted to search, all bases (x,y,z) up to 100,000 and all powers (m,n,r) up to 10. Then the first problem is that we will be manipulating integers up to 100,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000 (that is, 10^{50}). And we will be dealing with 1,000,000,000,000,000,000 (that is, 10^{18}) potential combinations of integers.

The "large integer" problem requires either a language that supports such integers directly, such as Lisp or Python, or a package that adds the functionality, such as the NTL package for C++.

The "many combinations" problems requires a clever search algorithm. As a start, we can cut the space in half (without loss of solutions) by only considering combinations with x less than or equal to y. This helps a little, getting us down to $0.5 * 10^{18}$ combinations. The next step is to eliminate the need to consider combinations of z, r. The idea is that we can precompute all z^r combinations and store them in a hash table. Then when we get a $x^m + y^n$ sum, we just look it up in the table, rather than enumerating all z, r pairs. This gets us down to $0.5 * 10^{12}$ combinations, at the cost of having to store 10^6 large integers. Both storage and projected computation times are now within range of what I can expect to handle on my home PC.

Now its time to implement the program. I chose Python as the implementation language because it happened to already be installed on the machine I had (I also had Lisp installed, but it was an evaluation copy that was limited in the amount of memory allowed, so I couldn't use it for this problem). Python is the slowest of the choices in terms of execution speed, but it allowed me to program a solution in only a few hours.

The Python program is very straightforward: enumerate all *x,y,m,n* combinations, and for each one check if the sum of the exponents is in the table. Initial testing on small search spaces was rewarding: it took only 3.3 minutes to verify that there are no solutions with *x,y,m,n* less than 100. I was then ready to scale up, but first I made some very minor improvements to the algorithm: instead of looking up the sum in the table with `table.get(x**m + y**n)`, I moved each part of the calculation out of as many loops as possible, and pre-computed as much as possible. In other words, I changed from the original:

```
def beal(max_base, max_power):
    bases, powers, table = initial_data(max_base, max_power)
    for x in bases:
        for y in bases:
            if y > x or gcd(x,y) > 1: continue
            for m in powers:
                for n in powers:
                    sum = x**m + y**n
                    r = table.get(sum)
                    if r: report(x, m, y, n, nth_root(sum, r), r)
```

to the optimized version running about 2.8 times faster:

```
def beal(max_base, max_power):
    bases, powers, table, pow = initial_data(max_base, max_power)
    for x in bases:
        powx = pow[x]
        for y in bases:
            if y > x or gcd(x,y) > 1: continue
            powy = pow[y]
            for m in powers:
                xm = powx[m]
                for n in powers:
                    sum = xm + powy[n]
                    r = table.get(sum)
                    if r: report(x, m, y, n, nth_root(sum, r), r)
```

Results

Alas, I have found no counterexamples yet. But I can tell you some places where you shouldn't look for them (unless you think there's an error in my program). Running my program on a 400 MHz PC using Python 1.5 I found that:

```
beal(   100, 100)  took    3.3 mins  (   9K elements in table)
beal(   100,1000)  took   19.3 hours (  92K elements in table)
beal(  1000, 100)  took    6.2 hours (  95K elements in table)
beal( 10000,  30)  took   52   hours ( 278K elements in table)
beal( 10000, 100)  took  933   hours ( 974K elements in table)
beal( 50000,  10)  took  109   hours ( 399K elements in table)
beal(100000,  10)  took  445   hours ( 798K elements in table)
beal(250000,   7)  took 1323   hours (1249K elements in table)
```

To put it another way, in the following table a red cell indicates there is no solution involving a b^p for any value of b and p less than or equal to the given number. A yellow cell with a "?" means we don't know yet. Run times are given within some cells in minutes (m), hours (h) or days (d).

	p=7	p=10	p=30	p=100	p=1000
b=100	-	-	-	3m	19h
b=1,000	-	-	-	6h	?
b=10,000	-	-	2d	39d	?
b=100,000	-	18d	?	?	?
b=250,000	55d	?	?	?	?

How do we know the program is correct? We don't for sure, but removing "`or gcd(x,y) > 1`" prints some non-co-prime solutions that can be verified by hand, suggesting that we're doing something right.

Комбинаторика этой программы бессмысленна.

Бессмысленность программы состоит в том, что её автор произвольно берёт числа из тела трубок числовых комбинаций.

Я имею в виду числовые трубки, изображённые выше на Фиг.1.

Можно брать бесконечное множество числовых комбинаций из этих трубок, строить статистические таблицы и никогда не найти закономерность, связывающую такие комбинации в простую конструкцию (А).