FRACTRAN Interpreter + FRACTRAN Programming

Vikram Ramanathan

GitHub Repo: fractran

1 Introduction

While I was learning about the Game of Life, I came across another one of Conway's ingenious creations, FRACTRAN [Con87]– a Turing-complete esoteric language. Intrigued by the mathematics behind FRACTRAN programming, I wanted to make my own FRACTRAN programs. However, since FRACTRAN is an esoteric language, there isn't widespread support for code of this nature. So, for me to able to play around with the elegant and powerful mathematics behind a FRACTRAN program, I first had to develop a FRACTRAN interpreter. I chose to write this interpreter in Python and here were the results.

2 Fundamentals of FRACTRAN

> A FRACTRAN program is defined as a finite list of fractions $\frac{p_1}{q_1}, \frac{p_2}{q_2}, ..., \frac{p_k}{q_k}$, where $p, q \in \mathbb{R}$, $q0$ and $k \in \mathbb{Z}$. Given a positive integer, n_0, the algorithm successively calculates n_{i+1} by multiplying n_i with the fraction that yields an integer. The algorithm halts if there is no such fraction.

Here's an implementation of an example given by Conway that demonstrates the afore mentioned algorithmic procedure. This particular example results in an infinite sequence so I have capped the output to the first 20 terms: Github: Fundamental Example

3 FRACTRAN Interpreter

This interpreter will be structured as follows – Input: FRACTRAN program (a sequence of fractions saved in a text file) with integer input, n_0 – Run-time: The python script will execute the FRACTRAN algorithm on the FRACTRAN program and n0. The prime factorization of the outputted number from the algorithm will be computed – Output: the last updated value of n will be outputted along with each of the exponents on the primes of its prime factorization

The implementation of this FRACTRAN Interpreter is given here: Github: FRACTRAN Interpreter

4 FRACTRAN Programming

4.1 Addition

In this section, we will be writing a fractran program that can add two positive integers.

> The fundamental theorem of arithmetic (a.k.a unique factorization theorem) states that 'every positive integer (except the number 1) can be represented in exactly one way apart from rearrangement as a product of one or more primes' (Hardy and Wright 1979, pp. 2-3).

The input to the Fractran program will be a Godel encoded number. The primes in the unique factorization of the inputted number will be the registers and the exponents of the primes will be the register states.

4.1.1 Walkthrough

One of the best ways to get to grips with an algorithm is to observe its step-by-step procedure using simple sample inputs. Here, we will add 2 and 9.

To add 2 and 9, we can choose any number a such that it can be factored as x^2 and y^9 where x and y are prime numbers.

For the purpose of explanation, let a = 17578125.

Here, a can be expressed as $3^2 * 5^9$

Therefore, the Factran program would simply be the fraction, 5/3. With every iteration, this multiplier would simply substitute the 3 in the unique factorization of 17578125 with a 5.

So, after one iteration, $3*3*5*5*5*5*5*5*5*5*5*5*5$ would become $3*5*5*5*5*5*5*5*5*5*5*5$. After the second iteration, we would have $5*5*5*5*5*5*5*5*5*5*5*5*5 = 511$.

The exponent that 5 is raised to is 11, which is the sum of the original two exponents we defined initially (2+9=11). Therefore, we have shown that this algorithm would be able to compute 2+9=11. Furthermore, it is apparent that the fraction in the Fractran program is reliant on x and y.

4.1.2 Example Program

add_prog.txt

```
IN 72
3/2
```

Here's the output:

```
[Fraction(3, 2)]
72
Factorization of IN n: [('2', 3), ('3', 2)]
n this iteration 72
nupd = [72, 108]
n this iteration 108
nupd = [72, 108, 162]
n this iteration 162
nupd = [72, 108, 162, 243]
n this iteration 243
All values of n: [72, 108, 162, 243]
Factorization of OUT n: [('3', 5)]
```

Here we see that we wised to perform the operation, 3+2. In order to do so, 72, which factors to 23 and 32 (as calculated and outputted by the program in line 3 above) was used as input to the Fractran program. The program itself was just one fraction, 3/2 (line 1). n after every iteration is appended to nupd and the final n is 243, which factors to 35. Thus, we successfully added the exponents in registers '2' and '3' and stored the added value in register '3'.

4.2 Multiplication

Now that we understand the basic mechanics as well as the output of a Fractran program, we can attempt to multiply two integers in Fractran.

4.2.1 Example Program

mult_prog.txt

```
IN 36
385/13,13/21,1/7,3/11,7/2,1/3
```

Since the output is quite lengthy, if you choose not run the program yourself, the verbose output is provided here: Github: Multiplication Output

Here, 2*2=4 was calculated where the register '2' stored 2, the register '3' stored 2 and the output, 4, was stored in register '5'.

Interesting! I was also able to use this fractran program to multiply 2 integers and add a 3rd integer to the product. Here's how I did it.

4.2.2 Example Multiply - Add Program

multadd_prog.txt

```
IN 337500
385/13,13/21,1/7,3/11,7/2,1/3
```

Once again, the verbose output can be found here: Github: Multiplication followed by Addition Output

Here, 2*3+5=11 was calculated where the register '2' stored 2, the register '3' stored 3 and the output, 11, was stored in register '5'.

4.3 Subtraction

Presented here is my solution for subtracting with Fractran. Note that, the input entered into and the output calculated by Fractran programs are restricted to positive integers so the difference of the two numbers must be positive for the Fractran algorithm to work.

4.3.1 Program

subtract_prog.txt

```
IN 497664
7/6,1/7
```

The verbose output from this program can be found here: Github: Subtraction Output

Here, 11-5=6 was calculated where the register '2' stored 11 and the register '3' stored 5. The final output of the operation was stored in register '2' and register '3' was destroyed.

4.4 Division

4.4.1 Program

div_prog.txt

```
IN 456192
91/66,11/13,1/33,85/11,57/119,17/19,11/17,1/3
```

The abridged output obtained,

```
Factorization of IN n: [('2', 9), ('3', 4), ('11', 1)]
.
.
.
Factorization of OUT n: [('5', 2), ('7', 1)]
```

Here, 9/4=2R1 was calculated where the register '2' stored 9, the register '3' stored 4, the register '5' stored the quotient, 2, and the register '7' stored the remainder, 1.

5 Conclusion

In this project, I developed a recursive Python FRACTRAN interpreter and four FRACTRAN programs that perform basic arithmetic operations. Generating verbose outputs really gave me insight into the concept of using the prime numbers in the unique factorization of a number as registers with their own unique, mutable

states. As several others have noted before me, the basic concept behind FRACTRAN is similar to a Minsky register machine [Min67]. However, it differs slightly from a Minsky register machine in that it cannot skip or jump to certain instructions. In the sequence of fractions, the next fraction in the sequence can be used as the multiplier if and only if the input does not produce an integer when multiplied by the fraction before.

It is truly fascinating that arithmetical operations on the input can be written simply as a program of fractions. I really enjoyed developing this interpreter and playing around with FRACTRAN.

References

[Min67] Marvin Lee Minsky. *Computation*. Prentice-Hall Englewood Cliffs, 1967.

[Con87] John H Conway. "Fractran: A simple universal programming language for arithmetic". In: *Open Problems in Communication and Computation*. Springer, 1987, pp. 4–26.