International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | Impact Factor: 7.194 |

||Volume 8, Issue 4, April 2020||

Implementing Arbitrary Precision in Java Script Runtime

Pragathi N M, Mayura D Tapkire

B.E.Student, Department of Information Science and Engineering, National Institute of Engineering, Mysuru, India

Assistant Professor, Department of Information Science and Engineering, National Institute of Engineering, Mysuru, India

ABSTRACT: One of the well-known oddities of IEEE 754 (Floating point arithmetic standard) is 0.1 + 0.2 == 0.30000000000000004. This is because of how 64-bit floating point numbers are represented internally. While the difference might be very minute, applications in the domain of banking and astronomy cannot truly rely on such incorrect calculations. This problem can be solved using the arbitrary precision arithmetic. While arbitrary precision arithmetic is implemented in functional programming languages like Lisp, JavaScript does not natively support arbitrary precision arithmetic as of January 2020. It supports BigInt, however, implementing BigNum is currently in the proposal stage of the JavaScript Technical Committee tc39.

KEYWORDS: Arbitary precession arithmetic, domain of banking,astronomy

I. INTRODUCTION

To implement arbitrary precision arithmetic in JavaScript run-time engine such as Chrome's V8 engine and Mozilla' Spider-Monkey.The system allows developers to use arbitrary precision arithmetic in JavaScript runtime by making use of the Decimal object. It also supports functions like exponentials, logarithms, trigonometric and inverse-trigonometric operations. Some of the default configurations were assumed which were allowed to be tweaked.
For the known constants such as ln(10), PI, and E, the first 1000 digits of the constants were hard-coded for immediate use.

II. LITERATURE REVIEW

In most computer programs and computer environments, the accuracy of any calculation (including addition) is limited by the size of the computer word, that is, the largest number that can be stored in one of the processor registers. In mid-2002, the most common processor word size was 32 bits, which is the integer $2^{32} = 4\,294\,967\,296$. Therefore, general integer arithmetic on a 32-bit machine allows adding two 32-bit numbers to get 33 bits (one word plus one overflow bit), multiplying two 32-bit numbers to get 64 bits (Although the most common programming language, C, cannot directly access the top word and depends on the programmer to create a machine language function or to write a much slower function in C with a final overhead of about nine more multiplications) and part of a 64- Bit number by a 32-bit number that generates a 32-bit quotient and a 32-bit remainder / module.

Arithmetic with any precision consists of a number of algorithms, functions and data structures that have been specially developed for handling numbers of any size. These functions often modify standard techniques for paper and pencil arithmetic (e.g. long division) and apply them to numbers that are divided into word-sized fragments. An important difficulty in creating good arbitrary numerical accuracy is knowing where to stop a calculation. A simple example of this problem is shown by the binary extension 1/3, which is given by the non-deterministic binary decimal $0.0101010101\ldots_2$. As a result of the fact that exact numbers do not have termination binary fraction expansions, additional functionality must be incorporated into an arbitrary precision calculation system. This can be in the form of a failure test, or a configurable 'maximum precision' in which the calculation will always stop when it reaches a very small particular number.

III. EXISTING SYSTEM

Currently, both the most popular JavaScript engines provide something called a BigInt data type. BigInt is capable of holding arbitrary length of integers. It allows simple arithmetic operations such as addition, multiplication, comparison

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | Impact Factor: 7.194 |

||Volume 8, Issue 4, April 2020||

and subtraction. However, it has no support for non-integers. Fractions cannot be handled by BigInt and division is also not supported. An implementation of a floating-point system conforming to this standard may be realized entirely in software, entirely in hardware, or in any combination of software and hardware. For operations specified in the normative part of this standard, numerical results and exceptions are uniquely determined by the values of the input data, sequence of operations, and destination formats, all under user control.

IV. METHODOLOGY

4.1.Functional programming

Functional programming is an example of programming where we try to combine everything with a pure style of mathematical functions. This is a declared type of programming style. Its main focus is on "what can be solved" as opposed to an imperative style, where/ the main focus is on "how to solve". Use expressions instead of instructions. A representation is evaluated to produce a value while a command is executed to determine the variable.

4.2 SYSTEM DESIGN

The objective was fulfilled by defining the working Decimal object. The implementation of the function in various edge cases were handled by referring to the IEEE 754 standards as well as the functioning of the runtime engine as specified in the specs laid down by the tc39 of JavaScript.
Some of the defaults includes:

- ROUND_UP
- ROUND_DOWN
- ROUND_CEIL
- ROUND_FLOOR
- ROUND_HALF_UP
- ROUND_HALF_DOWN
- ROUND_HALF_EVEN
- ROUND_HALF_CEIL
- ROUND_HALF_FLOOR

Some of the functions considered for the implementation were:

- Trigonometric: sine, cos, tan and their multiplicative inverses
- Inverse Trigonometric: asin, acos, atan and their multiplicative inverses
- Logarithmic and exponential
- Roots
- Precision, round, ceil, floor, abs

There were some read-only functions defined, and there were other static functions defined to be used not by the objects, but by the class. For example, in native JavaScript, there is Math.PI, and Math.E.

4.3 SYSTEM IMPLEMENTATION

The implementation of the Decimal object was started with laying down the necessary components of the object.
That includes the configuration parameters as well as the proper way of handling the numerical values.
One of the primitive ways of accomplishing the arbitrary precision for basic calculation is the lazy evaluation method. The numerator and the denominator is often stored separately. While performing the operations, the numerator and denominator of two or more numbers are handled separately. That was the initial step taken in order to achieve the arbitrary precision. Whenever the value was required, it was evaluated at that time, by computing the division, with the limit of precision as configured.

However, this does not work well, when there are more complex operations involved. Taylor series operations for complex functions is tempting, but that also bring along the loss of precision. One of the ways to handle this was to set the flags of the operations: such as logarithmic, trigonometric, as well as inverse trigonometric.
The easier functions such as ceil, floor, round, as well as abs were calculated by using the naïve definitions of the functions. For the moderate ones like multiplication and division, the initial method of repeated addition, and subtraction respectively were handled for the integers. However, this is not the most optimal method to implement in order to achieve the final result when there are floating point numbers involved.
Some of the default configurations were assumed which were allowed to be tweaked.

For the known constants such as ln(10), PI, and E, the first 1000 digits of the constants were hard-coded for immediate use. Parsing of the numbers as string was handled character by character basis, looking for a floating point or an exponent, and handling the cases separately in the constructor.

V. TESTING

Various test cases were handled by the unit testing library called JestJS. For each function, setting the configuration, the tests were performed to check for the value of the evaluation.
Several criteria, such as significand, exponential, floating parts were considered, and the values of the functions were tested for even the NaN conditions.

Expected	Expression	Operation
NaN	Asin(40)	inverse trigonometric
1.5707963267948966192	Asin(1)	inverse trigonometric
1	Cos(0)	trigonometric
2	Cube root(8)	root
-4.5306548960834927777	Cube root(-93)	Root
5	Abs(-5)	Absolute
0.2	0.1 + 0.2	Addition
18446744073709551616	2 power 64	Exponent
4.605170185988091368	Ln (100)	Logarithmic

VI. EXPERIMENTAL RESULTS

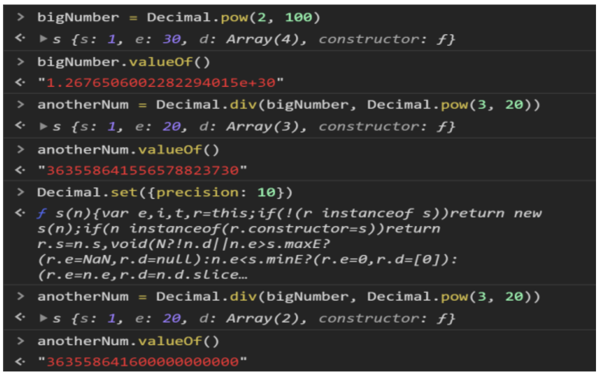

Figure 1. The decimal object performing exponential operation.

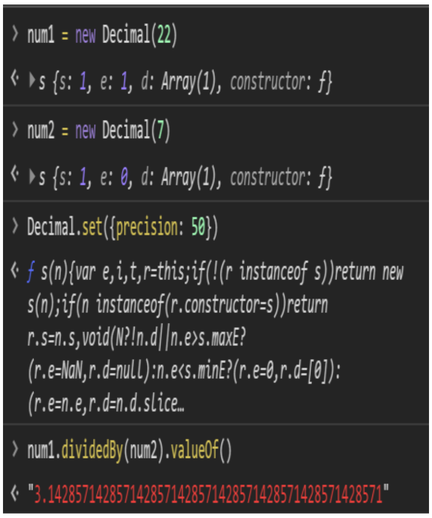

Figure 2. Decimal object performing division operation

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | Impact Factor: 7.194 |

||Volume 8, Issue 4, April 2020||

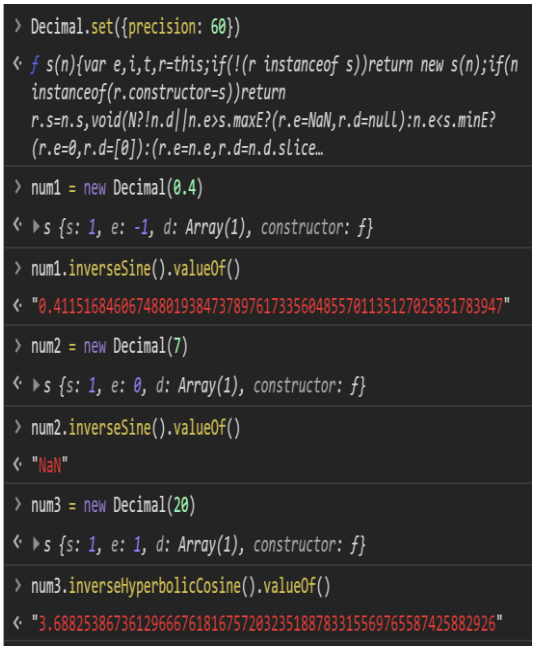

Figure 3. Decimal object performing inverse trigonometric operations.

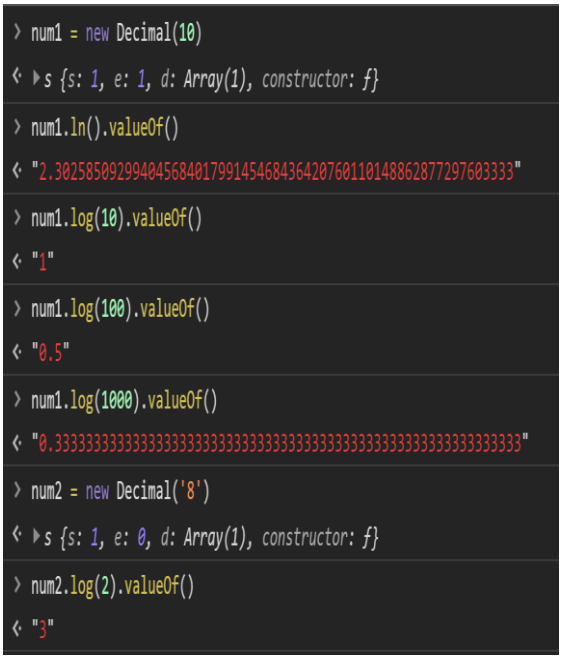

Figure 4. Performing logarithmic operations on Decimal objects.

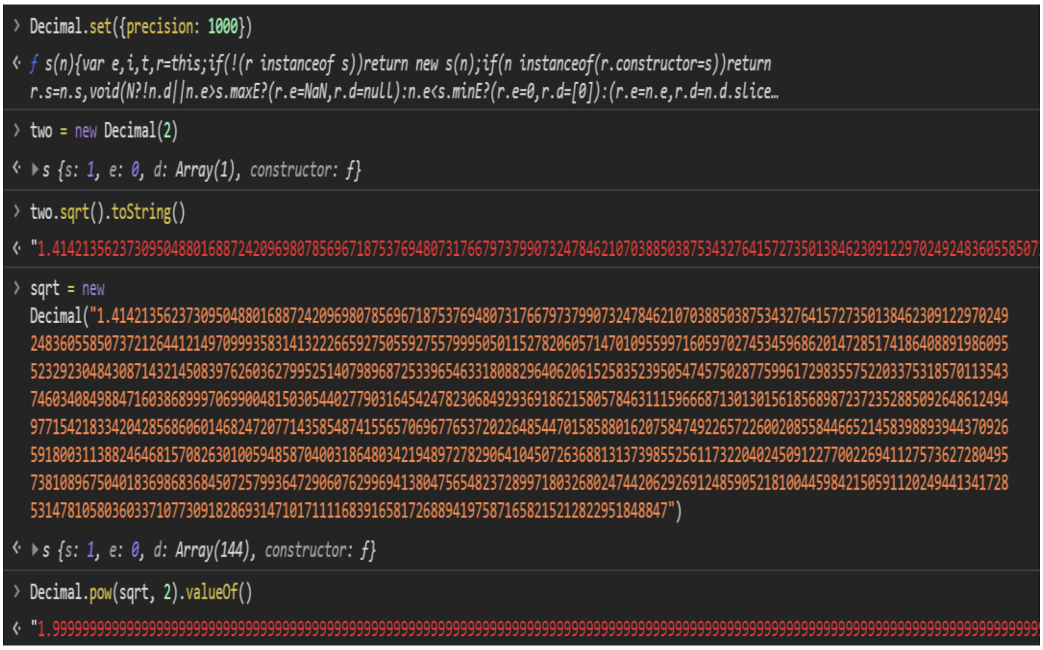

Figure 5. Performing square root operations on Decimal objects.

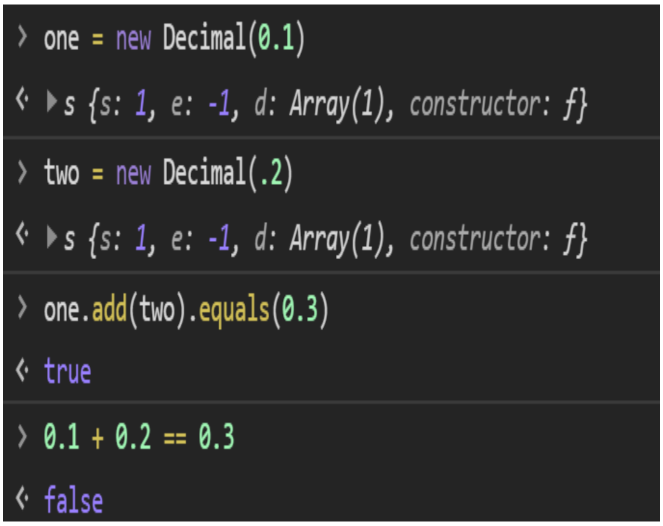

Figure 6. Overcoming the IEEE 754 shortcomings.

VII. ADVANTAGES, DISADVANTAGES AND LIMITATIONS

7.1 Advantages

The advantages includes:

- Greater precision of arithmetic operations.
- Overcoming the downfalls of IEEE 754.
- Functional style of the implementation means there are no side effects since there is no mutability.
- Allows developers to set precision arbitrarily as required.
- Provides various functions for mathematical operation.
- Ability to set the configuration and change at will.

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | Impact Factor: 7.194 |

||Volume 8, Issue 4, April 2020||

7.2 Disadvantages and limitations

- The theoretical limitation of the concept is the memory available in the machine.
- This is not a native implementation and therefore it is not the most memory efficient solution.
- Operators are not overloaded in order to not rely on the functions.

VIII. CONCLUSION

In conclusion, now there exists a Decimal object which supports arbitrary precision arithmetic in JavaScript runtime engine. It also supports some of the complex functions which were previously not available in the run time.

REFERENCES

1. https://mathworld.wolfram.com/ArbitraryPrecision.html Uznanski, Dan. "Arbitrary Precision." From *MathWorld*--A Wolfram Web Resource, createdby EricW. Weisstein
2. https://ieeexplore.ieee.org/document/4610935 IEEE Std 754-2019 (Revision of IEEE 754-2008)
3. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt , by MDN contributors
4. https://github.com/tc39/proposal-decimal
5. https://www.geeksforgeeks.org/functional-programming-paradigm/
6. https://www.sciencedirect.com/science/article/pii/S1567832604000748, The Journal of Logic and Algebraic Programming, Volume 64, Issue 1, July 2005
7. https://people.eecs.berkeley.edu/~fateman/papers/decimalfloat.pdf Arbitrary-precision Decimal Floating-Point Numbers: More than You Expected, Less than You Asked For? Richard J. Fateman
8. https://old.cescg.org/CESCG-2015/papers/Langer-Arbitrary-Precision_Arithmetics_on_the_GPU.pdf Arbitrary-Precision Arithmetics on the GPU Bernhard Langer Supervised by: Thomas Auzinger
9. https://ieeexplore.ieee.org/document/7891894 (Volume: 66 , Issue:12 , Dec. 1 2017)
10. https://www.researchgate.net/publication/222592446_Arbitrary_precision_real_arithmetic_Design_and_algorithms Arbitrary precision real arithmetic: Design and algorithms, 10.1016/j.jlap.2004.07.003, Journal of Logic and Algebraic Programming
11. D. Goldberg**What every computer scientist should know about floating point arithmetic** ACM Computing Surveys, 23 (1) (1991), pp. 5-47, Available from: <http://docs.sun.com/htmlcoll/coll.648.2/iso-8859-1/NUMCOMPGD/ncg_goldberg.html>.