MM Research Preprints, 212–219
No. 18, Dec. 1999. Beijing

Development of An Object-Oriented Number System [1)]

Hong Yang
Beijing Municipal Computing Center
hyang@mmrc.iss.ac.cn

Zhuo-Jun Liu and Dong-Dai Lin
Institute of Systems Science
Academia Sinica
{zliu,ddlin}@mmrc.iss.ac.cn

Abstract. With the development of computer algebra and its applications, the computer algebra system attracts more and more attention in scientific research and education fields. Obviously, a number system that manipulates multi-precision numbers is a necessary and important part for any computer algebra system. In this paper, we will discuss the implementation and technique details of our object-oriented number system. First we give out object-oriented analysis and design for the number system, and then some important algorithms and their implementation are discussed. A new representation for multi-precision integers is also presented in this paper.

1. Introduction

With the development of computer algebra and its applications, the computer algebra system has attracted more and more attention in scientific research and education fields. Unlike numerical computing system, computer algebra system are expected to manipulate mathematical expressions symbolically and handle numbers exactly so that no errors arise in their operations. It's well known that the range of numbers that can be represented and manipulated by the computer's hardware is limited by the computer word size. A number system that enables exact arithmetic on arbitrary (large) integers is called multi-precision number system. It is obviously a necessary and important part for any computer algebra system, and is a prerequisite for the implementation of the whole computer algebra system.

In this paper, we will present such a multi-precision number system. Our system can be used either as a separate multi-precision system or as a basic module for a computer algebra system. In fact, it has been used as a basic module of ELIMINO system, which is a computer-mathematics research system based on Wu's method. The main objectives of the number system are that:

- Computations with the number system should run at optimal speed.
- The number system is expected to provide plentiful functions for the development of a whole computer algebra system.
- The number system itself should be easy to use, that is, it should have a friendly interface.

[1)] This work was supported in part by the 973 project

In order to make the number system to be extended, especially to be easily integrated into a whole computer algebra system, we adopt the object-oriented technology in its development.

In the next section, we will give an outline of the analysis and design for the number system. In the third section of this paper, the internal representation and some important algorithms for multi-precision integers will be discussed.

2. Analysis and design of the number system

Numbers are the most fundamental objects to be manipulated in the implementation of any computer algebra system. According to the view of object-oriented technique, the number system can be considered as a collection of number objects. In this section, we will discuss the object-oriented analysis and design for the number system and thus we can get a developing outline of the number system.

2.1. The architecture of the number system

As a separate system or as a basic module of ELIMINO system, the number system is expected to be able to represent and manipulate several types of numbers such as multi-precision integers, fractions and floating-point numbers. Therefore the number system should contain at least three main classes TBigInt, TFraction and TFloatNum that correspond to the above types. Since the numbers of different types always have some common properties, we have also defined a class TNumber as the base class of the three main classes. Thus the common attributions and methods can be shared among these classes. This also means that the object of any number type can be treated as an object of type TNumber, and its exact type needn't be cared about. In addition, arithmetic operations can work on mixed kinds (types) of numbers since the system knows how to convert to each other among them automatically. Such a TNumber class can bring us much convenience, e.g. it allows us to build a class of polynomial with TNumber coefficients, or a class of matrices with TNumber entries. Such classes are of course much more generic. The following Fig. 1 shows the structure relationship among these classes.

Fig. 1: Architecture of The Number System

2.2. Attribution analysis for the number system

Attributions are data elements used to describe an instance of an object or classification structure. They clarify what is meant by the name of an object, by adding more detail about the abstraction being modeled.

As a base class, a TNumber object has an internal pointer **"ptr"** of type TNumber * which can point to any derived object of TNumber and thus allows TNumber objects to represent many different subtypes of numbers.

TBigInt is the most fundamental and important data type of the number system. According to the representation of multi-precision integers, two main attributions of TBigInt are **"numBlock"** and **"len"**. **"numBlock"** is a pointer to the memory space in which the data of a multi-precision integer is stored. **"len"** records the length of the memory space allocated to the integer along with its sign. More detailed information about the internal representation for multi-precision integers will be discussed in Section 3.1.

On the basis of TBigInt, the class TFraction and the class TFloatNum are defined very straightforward. TFraction has two attributions **"nu"** and **"de"** of type TBigInt * to represent the numerator and the denominator, respectively, and TFloatNum has two attributions **"mant"** and **"exp"** of type TBigInt * and type int to represent the mantissa part and the exponent (base 10) part of a floating-point number, respectively. It's noticed that the value of the floating-point number is $mant * 10^{exp}$. The attribution **"len"** is used to indicate the number of digits carried in the mantissa part (i.e. the accuracy of a floating-point number).

From the above discussion, we can see that both class TFraction and class TFloatNum are designed to be constructed by the class TBigInt, so the implementation of the class TBigInt is a basis of the development of the whole number system and it is what we focus on in this paper.

2.3. Method analysis for the number system

A method is the processing to be performed upon receipt of a message. The method analysis for the class TBigInt means the analysis of the related operations on multi-precision arithmetic. A number system that mainly deals with multi-precision integers must include at least the following operations:

- Input and output subroutines of a multi-precision integer.
- The arithmetic operations, essentially addition, subtraction, multiplication, division and exponentiation, etc.
- Left and right shifts of a multi-precision integer by small integers.

The above operations are all basic and necessary operations. They have a very close relationship with the internal representation for integers and should be developed first in the number system. On the basis of the implementation of these operations, we can further accomplish modular arithmetic as well as other mathematical operations when needed. These operations may include:

- The greatest common divisors (GCD) of multi-precision integers.
- Integer factorization and its related operations.
- Some useful mathematical functions, such as sqrt(square root), log2(base 2 logarithm), integer factorial, etc.

- The related operations on modular arithmetic.

The method analysis of the class TFraction and the class TFloatNum is similar to that of the class TBigInt, therefore we will not give out more discussion about it.

3. Implementation of the number system

In this section, we will first discuss the internal representation for multi-precision integers, and then some important algorithms and their implementation are presented. Please refer [**?**] to see how the number system works.

3.1. Internal representation for multi-precision integers

It is well known that the integer that can be represented by the computer's hardware is limited. We shall call an integer that can not be represented by the computer's hardware (i.e. can not be stored in one computer word) **a multi-precision integer**, which is the essence of multi-precision arithmetic. A multi-precision integer N is constructed by a linear list $(a_0, a_1, ..., a_{l-1})$. The value of the multi-precision integer is

$$N = s\sum_{i=0}^{l-1} a_i B^i$$

In the above formula, B is the **base** of the representation. So a multi-precision integer can also be considered as a base B integer. We shall call a non-negative integer less than B **a single-precision integer**. Here, $a_i (i = 0, 1...l-1)$ are all single-precision integers, which we shall call the B**-digit** of N. s is either 1 or -1, corresponding to the sign of N. l is called the B**-length** of N.

In principle, the base B may be any positive integer up to the upper limit for integers allowable for the computer's hardware. But from the consideration of implementation, the usual choice for B is to choose the square root of the computer word size as the base. This choice will help to easily implement the multi-precision arithmetic because all the arithmetic operations between B-digits can be performed in one computer word. We assume that the working platform on which our number system runs is a 32-bit machine, so we choose $B = 2^{16}$ as the base of our number system.

As being known to all, there are three conventional representation methods to implement the linear list data structure for multi-precision integers. The three methods contain **the linked list representation, the dynamic array representation** and **the fixed-length array representation**, among which the first two representations are used in most computer algebra systems.

Each of the three representation methods has both advantages and disadvantages. The linked list representation, which uses each node to represent each single-precision digit (B-digit), involves at least two disadvantages: 1. a considerable amount of memory space is required for the pointers; and 2. the access time for each digit is much higher than the access time required in the array representation.

Relative to the linked list representation, the dynamic array representation uses less storage and less access time. However this method requires more sophisticated storage management algorithm.

The fixed-length array representation, which means every multi-precision integer is allocated an array of length L (L is a pre-pecified constant), can overcome the difficulty of storage management, but it will waste a considerable amount of memory space in storing the high-order zero digits.

Considering all the disadvantages mentioned above and the implementation of the storage management module in ELIMINO system[**?**], we adopt another representation in our number system, which is a reasonable compromise and we call it **the block representation**. Let n_b is a pre-defined positive integer by the system's storage management module. We use some blocks to represent the multi-precision integer N. Each block is a fixed-length array that contains n_b single-precision digits and is allocated from the storage management module in ELIMINO system. There are $T = [(l-1)/n_b] + 1$ blocks in all to represent the integer. The block representation is shown as Fig. 2.

Fig. 2: Block Representation

In fact, the idea of the block representation is to combine the linked list representation with the fixed-length array representation. The block representation is an effort that tries to increase the number system's efficiency by taking a balance among the storage management, the access time and the cost of the memory space.

The following table compares the dynamic array representation with the block representaion by testing the computing time of the integer mulitiplication (classical algorithm) on ALPHA workstation $600^{5/333}$. In the table, n means the number of decimal digits of the two multiplicands, and len means their B-length. The times are given in microseconds.

len	n	Array Repr.	Block Repr.
313	1506	83330	66664
496	2386	233324	199992
626	3011	299988	283322
695	3345	383318	366652
772	3717	466648	449982
991	4772	566644	733304
3126	15052	5249790	7433036
6251	30103	21765796	29682146

From the above table, we can see that the block representation shows more superior than the dynamic array representation when the multiplicands are not so large. The reason is that the allocation time for a fixed-length block from the storage management module in ELIMINO system is less than that for a dynamic array from the operation system directly.

Obviously, the main disadvantage of the block representation is that the code complexity of the algorithms for basic arithmetic operations is much higher than that of the array representation.

3.2. Implementation of some algorithms in the number system

Many algorithms for multi-precision arithmetic and rational arithmetic have been implemented in our number system. It is obvious that we cannot discuss all the implemented algorithms in the space available here. In this section, we only give out the implementation of the input/output algorithms in which we adopt a strategy to get a substantial saving in the processing time, and the implementation of the FFT-based (Fast Fourier Transformation) multiplication algorithm, which is a different implementation from the usual way.

3.2..1. Input and output algorithms

The input and output subroutines are basic and important operations in the number system. They are used to provide interfaces for the system. The I/O problem is actually the conversion between decimal and base B representation. That is, the input subroutine is to convert an integer from decimal to base B representation, while the output subroutine does the reverse work.

From the consideration of efficiency, a slight modification to the input algorithm can be achieved by converting from decimal first to base 10^k ($k = \lfloor log_{10}B \rfloor$) representation and then computing the base B representation from there. A similar modification can also be performed on the output algorithm. Compared with the direct conversion between decimal and base B, the modified algorithms can save a lot of computing time.

3.2..2. An implementation of the FFT-based multiplication algorithm

For the multiplication of two multi-precision integers, the most frequently used algorithms are classical algorithm ($O(n^2)$), Karatsuba algorithm
($O(n^{log_2 3})$) and FFT-based algorithm ($O(\ n * ln(n) * ln(ln(n))\ )$), among which the FFT-based algorithm is the fastest algorithm in theory and will be discussed in detail in this section.

It is noticed that the FFT-based algorithm implemented in many computer algebra systems is usually the algorithm suggested by A.Schönhage and V.Strassen[**?**] which is a thoroughly symbolic method. The trade-off point for the algorithm (i.e. the length of the multiplicands when the FFT-based method becomes faster than the classical method) is nearly 20000 decimal digits [**?**]. The fact shows that the algorithm is of theoretical interest but has not a practical significance for computer algebra system. In our number system, a hybrid implementation of floating-point operations and symbolic computations for the FFT-based algorithm has been realized. Therefore, this implementation is different from A.Schönhage and V.Strassen's symbolic implementation. It is shown from our experiments that we have deduced the trade-off point by making use of floating-point operations in the FFT-based algorithm and thus making the algorithm more practical.

The main idea of the FFT-based algorithm comes from the fact that the multiplication of two multi-precision integers can be considered as the convolution product of two vectors.

Suppose $u = (u_0, u_1, ..., u_{n-1})$ and $v = (v_0, v_1, ..., v_{n-1})$ are two vectors corresponding to two multi-precision integers. $w = (w_0, w_1, ..., w_{n-1})$ is the convolution product of u and v: $w = u \otimes v$. That is, $w_i = \sum_{j=0}^{n-1}(u_j * v_{i-j}),\ i = 0, 1, ..., n-1$.

According to the property of Fourier transformation, we can get a FFT-based algorithm for determining the convolution product w. The algorithm can be briefly described as follows:

Step 1. Calculate the Fourier transformation of u and v: $\mathcal{F}(u) = \tilde{u} = (\tilde{u}_0, \tilde{u}_1, \ldots, \tilde{u}_{n-1})$ and $\mathcal{F}(v) = \tilde{v} = (\tilde{v}_0, \tilde{v}_1, \ldots, \tilde{v}_{n-1})$.

Step 2. Calculate $\tilde{w} = (\tilde{w}_0, \tilde{w}_1, \ldots, \tilde{w}_{n-1})$ by $\tilde{u}$ and $\tilde{v}$: $\tilde{w}_i = \tilde{u}_i * \tilde{v}_i,\ i = 0, 1, \ldots, n-1$.

Step 3. Apply the inverse Fourier transformation to calculate w : $w = \mathcal{F}^{-1}(\tilde{w})$.

When the FFT-based algorithm is used to evaluate the multiplication of two multi-precision integers, only an adjusting step for w needs to be added:

Step 4. Perform the following steps on w: First, propagate the carries from left to right by floating-point operations. Second, transform each floating-point digit to integer digit. Then w is the multiplication of u and v.

It should be noticed that the floating-point operations are required in each step of the above algorithm.

We must point out that our implementation of the FFT-based algorithm has one fundamental limitation: the round-off errors existed in the floating-point operations. But we must notice a fact that the result of the algorithm remains exact as long as the precision of the floating-point operations is high enough. [**?**] gives out a detailed discussion about the precision of the floating-point operations which can get enough information. It is proved by our experiments that using the 64-bit (i.e. double type in C++ language) floating-point operations is enough for the multiplication of integers with less than 1024 B-digits (i.e. about 5000 decimal digits). In fact, theoretical disscusion in [**?**] shows that using the 128-bit floating-point operations is enough for the multiplication of integers with less than $1024 * 4^8$ B-digits. Note that such integer will cost 120 Megabytes in space memory. So we can say that using the 128-bit floating-point operations is sufficient for almost all the multiplication of two multi-precision integers that we would meet.

Fig. 3: Comparison of Two Multiplication Algorithms

Fig. 3 compares both multiplication algorithms on ALPHA workstation $600^{5/333}$. The two multiplicands are generated randomly with n decimal digits. It follows from the figure that the trade-off point for the FFT-based algorithm implemented in our number system is up to roughly 500 decimal digits.

4. Remarks

In this paper, we have discussed the implementation of our object-oriented number system. Some technique details are presented. We notice that there are still a lot of futher work to do with our number system, including the optimization of related algorithms, the implementation of some new methods and so on. In our succeeding work, we will try to make the number system to be an efficient and robust basis for the development of computer algebra system by continuously increasing its efficiency and perfecting its function.

References

[1] Dongdai Lin, Jun Liu and Zhuojun Liu. Mathematical Research Software: ELIMINO. Proceedings of the 3rd ASCM, 107-114, Aug. 6-8, 1998. Lanzhou.

[2] Franz Winkler. Polynomial Algorithms in Computer Algebra. Linz, Austria, 1996.

[3] von zur Gathen, J. and Gerhard, J., Modern Computer Algebra, Cambridge Press, 1999.

[4] D.E.Knuth. The Art of Computer Programming, Volume 2: Seminumerical Algorithms (second edition). Addison-Wesley, 1981.

[5] K.O.Geddes, S.R.Czapor and G.Labahn. Algorithms for Computer Algebra. Kluwer Academic Publishers, 1992.

[6] Henri Cohen. A Course in Computational Algebraic Number Theory. Springer-Verlag Berlin Heidelberg, 1993.

[7] Hans Riesel, Prime Numbers and Computer Methods for Factorization. Birkhauser Boston, Inc. 1985.

[8] Zhuojun Liu and Wenda Wu. STAR: A Small Tool for Algebraic Research. Proceedings of ASCM96, Kobe of Japan, 1996.

[9] B.Buchberger, G.E.Collins, R.Loos and R.Albrecht. Computer Algebra Symbolic and Algebraic Computation (second edition). Springer-Verlag/Wien, 1983.