On numbers and their representations

Introduction

This note is about numbers and their representations. Accordingly, we shall begin with a brief discussion of these concepts.

* * *

Quantity and number

Quantity is that concept which deals with the distinction between *less* and *more* , *a lot* and *a little* , etc. **Number** is an implementation of quantity, which arises out of our ability to view a quantity as being composed of discrete **units** . Units allow us to precisely measure and compare quantities, to ask and answer questions like "How much less?" and "How much more?" . The various amounts of units, arranged in order of size, form the sequence of **natural numbers** :

zero , *one* , *two* , *three* , *four* , and so forth .

Representing numbers, and positional systems

Generally speaking, in addition to concepts, humans need representations for concepts, to facilitate thought and communication. Representations for numbers range from natural language terms like the English 'thirty-four' and Hungarian 'harmincnégy' , to mathematical symbols like 34 and XXXIV .

In principle, however, the sequence of natural numbers goes on forever, or in any case well outstrips human mental capacity and memory. This precludes the possibility of using a different representation of each number, or at least implies that the representations of numbers should be arranged or related according to some simple pattern.

The finest system invented for this purpose is called a **positional system** , and the most well-known of these is called the **decimal (positional) system** . This is the number system in widespread use today, and some of its specimens are:

34 , 2 , 964 , and 8629 .

(History according to Wikipedia gives credit for the decimal system to two Indian mathematicians: Aryabhatta developed the positional system itself during the 5th century, and Brahmagupta invented the symbol for *zero* a century later.)

Layers of representation

While positional systems are simple to use —as any grade-schooler knows— , their conceptual underpinnings are relatively intricate. A positional system involves *two* distinct layers of representation: In the first layer, a number is "encoded" as a sequence of numbers; and in the second, that sequence is written down using conventional symbols, in a manner palatable for human consumption.

We often blur the distinction between these layers, just as we often blur the distinction between numbers and their conventional symbols —between the concept *two* and the symbol 2 , for example— . But we will be careful to make the distinction here, because we will be focussing primarily on the first step of the representation process, that is, the encoding of a number as a sequence of numbers. Though we will use symbols to discuss numbers in what follows —for there is no other way to communicate— , the reader should remember that our focus here is on *numbers* , not the symbols we use to write numbers down.

Encoding numbers

The basic ingredients of the encoding process are the **base** and the **digits** . The base is a natural number B (satisfying $2 \leq B$, but more on that later) , and the digits are the B natural numbers in the interval $[0..B)$. (By way of example, if the base is the number 9 , then the digits are the numbers 0 through 8 .)

A number is represented by a sequence of digits called a **digit string** . Digit strings are infinite, in the same way that the natural numbers are infinite; that is, there is a first position in the digit string, then a next position, and so on indefinitely. We shall freely refer to notions like **position** , **first position** , **next position** , **previous position** , and so forth, throughout the remainder of the document. The reader should bear in mind that notions like 'next' and 'previous' refer to the sequence ordering, and have absolutely nothing to do with the way the digit string is ultimately written down.

The value of a digit string

The final ingredient in the description of the encoding process is explaining how to compute the **value** of a digit string, that is, the number a digit string represents. To compute the value of a digit string, first the digits are **weighted** by multiplying them by successive powers of the base B : The first digit is multiplied by $B^0 = 1$; the second digit is multiplied by $B^1 = B$; the third digit is multiplied by B^2 ; and so forth. Finally, these weighted digits are added together.

Since digit strings are infinite, many digit strings do not have a well-defined value. In fact, this will be the case precisely when the string has infinitely many nonzero digits. Thus, in order to ensure that value is always defined, we restrict our attention to digit strings with finitely many nonzero digits.

Some examples

In order to look at some examples, we need to focus briefly on the second layer of representation, that is, conventions for representing digit strings in symbols. We will follow the standard conventions: Digit strings shall be written from right to left, with their digits written directly adjacent to one another. For the digits themselves, we use the common Hindu-Arabic numerals **0** through **9** . (We put numerals in boldface to distinguish them from the numbers they represent.) Finally, since a digit string has finitely many nonzero digits, we omit the infinite "tail" of zeroes.

Let's see an example. Let the base be the number 5 , so that the digits are the numbers 0 through 4 , and consider the digit string **2034** . According to the conventions given in the previous paragraph, this is a shorthand for the following sequence of numbers:

$$4 \quad , \quad 3 \quad , \quad 0 \quad , \quad 2 \quad , \quad 0 \quad , \quad 0 \quad , \quad 0 \quad , \quad \ldots \quad .$$

The value of the digit string **2034** is therefore:

$$4 \cdot 5^0 \; + \; 3 \cdot 5^1 \; + \; 0 \cdot 5^2 \; + \; 2 \cdot 5^3 \; + \; 0 \cdot 5^4 \; + \; 0 \cdot 5^5 \; + \; 0 \cdot 5^6 \; + \; \cdots \qquad ,$$

which is equal to the number 269 .

For further practice, the reader is invited to verify the following table, each entry of which gives a different representation of the number *thirty-four* :

base	digit string
2	**100010**
3	**1021**
4	**202**
5	**114**
6	**54**
7	**46**
8	**42**
9	**37**
10	**34**

.

And with these examples we conclude our discussion of concepts and terminology. We return our focus now to the "encoding" layer of representation; the conventions used in this section will not be used again.

* * *

A fantastic theorem

The remainder of this note is devoted to proving the following theorem:

> In each base, every natural number has a
> unique representation.

The list of adjectives that can be used to describe this theorem is seemingly endless: fantastic, deep, important, fundamental, surprising, etc. Sadly, most proofs of this beautiful theorem are clumsy and ugly. The theorem deserves better, and we will try to do better here. The idea behind our proof comes from Chapter 5 of Netty van Gasteren's Ph.D. thesis, entitled *On the Shape of Mathematical Arguments* .

Proof

We prove the existence and uniqueness of representations separately. For existence, we will design an algorithm which computes a representation, given a number and a base. A simple counting argument will settle uniqueness.

Existence: Computing a representation

Given a natural number N and base B , we turn towards computing a representation of N in base B . We introduce a program variable d of type 'digit string' , and formulate our postcondition as the conjunction of:

(0) finitely many digits in d are nonzero

(1) every digit in d is a natural number

(2) the value of d equals N

(3) every digit in d is less than B .

Aiming to establish the postcondition with a loop, we investigate (0)–(3) with an eye towards forming an invariant. Conjunct (0) can be maintained by requiring that our loop body modifies finitely many digits of d . Conjunct (1) can be maintained by a similar design constraint. Conjunct (2) can be easily initialized by letting the first digit of d equal N , and letting all other digits equal 0 . Note that this initialization establishes (0) and (1) . Finally, conjunct (3) can be easily initialized by letting all digits of d equal 0 . Note that this initialization establishes (0) and (1) as well.

In summary, we choose (0) and (1) as conjuncts of the invariant, adopting the constraints that our loop body will modify finitely many digits of d , and that our program will never assign an unnatural value to a digit of d . We initialize our program by letting all digits of d after the first equal 0 , thereby establishing (0) and (1) .

Now, it would be nice to take (2) or (3) as a conjunct of the invariant as well, but we cannot take both, as their initialization requirements are in conflict for the first digit of d ! My preference is to take (2) as a conjunct of the invariant, so that the negation of (3) becomes the guard. (The opposite design decisions still lead to an algorithm, the design of which is left to the reader as an exercise.)

To summarize again: We initialize d by letting its first digit equal N , and by letting all other digits equal 0 , thereby establishing (0)–(2) . By design, (0) and (1) are invariant, and the negation of (3) is the guard. Thus, it remains to design a terminating loop body that maintains the invariance of (2) .

*

At this point, it is standard to let the termination requirement guide the design of the loop body, and afterwards refine that loop body so as to guarantee the invariance of (2) . We follow this approach here.

When the program variables consist of a handful of natural variables —in this case a sequence of them— , a simple choice for the variant function is the sum of those variables. Thus the question is whether d's **digit sum** (that is, the sum of its digits) is a suitable variant function. Upon reflection, it would appear to be: First of all, thanks to invariant (0) , the digit sum is defined, and second of all, thanks to invariant (1) , the digit sum is bounded from below by 0 . Thus we take d's digit sum as our variant function, and look for a suitable decrease of it.

Here we are in luck, as our guard, the negation of (3) , tells us that "some digit of d is at least B " . It is therefore sweetly reasonable to have our loop body decrement that digit by B . This decrement does not falsify (1) , thanks to the guard, and properly decreases the variant function, provided we have $1 \leq B$. (We need not adopt this as a constraint right now, as we will have to adopt a stronger constraint in just a moment.)

Finally, we turn to the invariance of (2) , ie maintaining the value of d . Our loop body decrements some digit by B , so if that digit is weighted by B^k , the value of d is decremented by $B \cdot B^k = B^{k+1}$. To compensate, we therefore have to increment d's value by B^{k+1} . Since the *next* digit is weighted by B^{k+1} , we can accomplish this by simply incrementing the next digit by 1 .

To summarize: The loop guard guarantees that some digit of d is at least B . The loop body decrements this digit by B , and increments the next digit by 1 , which leaves the value of d unchanged. (This operation is commonly known as a "carry" .) The digit sum —the variant function— is therefore decremented by $B - 1$, which is positive provided we have $2 \leq B$.

This concludes the design of our algorithm and its correctness proof. In the process, we learned that our fantastic theorem holds when the base is at least 2 . For the reader's convenience, we present an annotated version of the algorithm:

```
|[  con N,B : nat {2 ≤ B} ;   d : digit string ;

    " let the first digit of  d  equal  N ,  and let
      the rest of the digits equal  0 "

  ; { inv:  (0) ∧ (1) ∧ (2)   ||  vf:  d's  digit sum }

    do  ¬(3)  →  { some digit of  d  is at least  B  }

          " decrease that digit by  B ,  and increase
            the next digit by  1 "

          { (0) ∧ (1) ∧ (2) }

    od

    { (0) ∧ (1) ∧ (2) ∧ (3) }

]|          .
```

Uniqueness: A counting argument

Now that we have established the existence of a representation for every natural number, a simple counting argument will reveal the uniqueness of that representation.

Consider the B^k numbers in the interval $[0..B^k)$: The representations of these numbers have the property that all digits equal 0 after the first k digits. (Without this property, the digit string would have too large a value.) But there are only B^k digit strings with this property, because there are B choices for each of the first k digits. Since every number has a representation, and since distinct numbers cannot share representations, we conclude that every number in the interval $[0..B^k)$ has precisely one representation. Because k is arbitrary and $2 \leq B$, we therefore conclude that every number has precisely one representation.

* * *

An afterthought

I have now completed 100 JAWs , a fact which I can't help but be proud of. But then I remember that *one hundred* is only a number of distinction because it is a power of our base *ten* . For example, in base 13 , the same number would be written as the rather unimpressive **79** . (Because $7 \cdot 13^1 + 9 \cdot 13^0 = 100$.)

Santa Cruz, 15–30 November 2007

Jeremy Weissmann
jeremy@mathmeth.com