```
loop
/* step 1 */
  l := random_prime_generate();
  if theorem2.2(l,B) then
    /* here, l is 2ip+1 for small +ve integer i and prime p */
    /* find the d by setting degree of H_d(X) to 1 */
    for d in [-4, -8, -3, -7, -11, -19, -43, -67, -163] do
      if Legendre(d,l) == 1 then
        compute_q_s(l,d);
      endif
    endfor
  endif
endloop
```

BRIDGING THE GAP Between MATHEMATICS & COMPUTING

by C T Chong

```
generate_elliptic(l, d
  /* step 4 and
  j_0 = Table_
  if (j_0 ==
    a := 0;
    b :=
```

remembered, and tied in the 'field', without the diagram crutch. A reasonable number of the ⟨60⟩ do have easily-memorizable methods of tying, and hints on these are included in the individual knot notes in Section 6.4. A full description of such methods would necessarily take up much space, and even then might not be too clear. Far better that the reader should work out a method for one's-self from the hints and foolproof diagram. Of course, the existence or otherwise of such a method has a great impact upon the practical utility of an SB. As one might expect, a large number of the easily-memorizable methods are themselves symmetric; that is, one threads symmetrically with the free dark end in one hand, the free light end in the other, just like most people tie a REEF knot. In principle, symmetric tying methods should certainly be the easiest. But a fair number are asymmetric. To give an idea,

Mathematics has a history of several thousand years. By constrast, computing, at least in electronic form, goes back less than 70 years, to the publication of Gödel's paper (1931) on the incompleteness theorem for arithmetic. The work, in retrospect, may be considered to have laid the foundation for a modern theory of computation.

Mathematics has come a long way since its inception. Modern mathematics as we known it today has less than 200 years of history (here I am referring to the works of Euler, Gauss, Galois and their contemporaries). Yet mathematics has gone through several phases of what I would call soul-searching introspection during this period. And in each phase, developments were intimately linked with (the notion of) computing. Before discussing this, let us reflect briefly on the subject of computing.

THE ESSENCE OF COMPUTING

At the heart of any computation is the notion of an *algorithm*. Basic arithmetic, whether in the eyes of the Babylonians or an eight-year old child, is algorithmic. The simple rules of counting, which lead to addition and multiplication of numbers, are algorithmic. The geometry of Euclid, using a ruler and a compass to construct circles, triangles, arcs etc. are algorithmic. Galois' work on the theory of equations is nothing more than proving the non-existence of an algorithm for certain classes of equations.

An algorithm is a rule which governs a computation. When faithfully followed, it leads an input to an output, and yields the same result whether it is performed by a human being or a digital computer. Once the rules are understood, it does not require much intelligence for a person or a machine to implement them and arrive at correct answers for a given problem, even if the reasons behind introducing the rules are not well-understood. (As an example, we have all seen how students who have 'learned dy / dx' compute the derivatives of polynomial functions, without understanding, nor caring about, the meaning behind 'taking the derivative of a function'.)

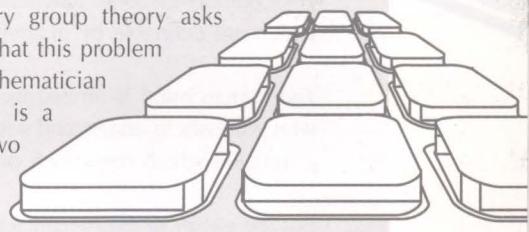
THE SECOND PHASE OF MATHEMATICAL DEVELOPMENT

Mathematics attempted to part ways with computing in the late 1800's. To my mind the major forces came from the works of Dedekind (on the foundations of analysis) and Hilbert (on invariant theory). Consider the following assertion: Every bounded set of real numbers has a limit point. Without the law of the excluded middle, it is an assertion that has to be taken almost by faith. But suppose one accepts the assertion, and wonders how a limit point may be found. In essence one is looking for an algorithm, starting with a bounded set, to compute its least upper bound. No algorithm is provided, nor hinted at by the assertion.

The "abstractization" of mathematics developed vigorously in the last century, in all forms and directions, culminating in the mathematics of the 1960's and 70's, when the Bourbaki philosophy held the centre stage, despite occassional protestations from 'recalcitrant mathematicians' like Brouwer (of intuitionism fame) and later Bishop. By and large, the philosophy remained intact until recent two-way interactions between mathematics and computing.

HOW TO DO/LEARN MATHEMATICS

We do not pretend to have the complete answer to the question. But we wish to make an observation: *The process of mathematical reasoning is primarily algorithmic.* Mathematical abstraction removes the algorithmic component from the process, although not always successfully. A simple problem in elementary group theory asks the student to show that the intersection of two groups is a group. The student complains that this problem cannot be done because he has not been given the two groups to work with. To the mathematician this shows the student's incapability to reason logically (abstractly). To the student it is a question of algorithm. It is likely that when presented with the detailed description of two particular groups (preferably finite and of small order), he will be able to show that the intersection is a group. However, choosing two arbitrary groups to start the problem is quite a different matter, since no algorithm has been given to guide the student on how the choice is to be made. The process of choosing two groups is an abstraction, and is an application of the Axiom of Choice on the category of groups. It is a nontrivial action, and requires the student to give up the certainty of an algorithmic approach for the uncertainty of abstraction. The ability to do this is often associated with what one calls *mathematical maturity*. Indeed this is an uncertain process because if the student has enough courage to demand the teacher to show how the groups are to be chosen, the reply would probably be, " It doesn't matter how the two groups are to be chosen, just choose any two." This reply is not satisfactory to the student, because the student is demanding an algorithm. The teacher refuses to give one, because there is none.



Algorithms quite often stay after mathematical abstractization, albeit in a different form. The concept of a probability space, in its mathematical form, is an abstract concept. Generating a probability space, however, is algorithmic, but at a different level. First of all, natural numbers provide the starting point. A logical necessity, and entirely algorithmic in reasoning (at

least from the modern perspective), leads one from natural numbers to the notion of infinite ordinals and cardinals. Building on this, one takes countable unions and intersections, starting from the open sets and closed sets, and iterates through the process of transfinite induction on countable ordinals, and arrives at the collection of Borel sets. This is a computational process in a larger universe (the universe of sets). While it is of a higher level than that for the solution of quadratic equations, it is algorithmic nonetheless. Strip away the ordinals from the inductive process, and the idea of a probability space gains a higher perception of abstraction, but loses its algorithmic origin. A good intuition is abandoned along the way.

In doing mathematics one looks for algorithms. To solve a general problem, one divides it into special cases. Small facts are found (often expressed through lemmas and propositions) to establish each special case, leading to the proof of the general result. Informally this is an algorithmic approach. It is also how mathematics is explained to others. The proof of the classification theorem for finite simple groups is an illustrative example. The classification theorem states that every finite simple group is either a cyclic group of prime order, an alternating group A_n, $n \geq 5$, a group of Lie type, or a sporadic simple group (of various sizes). Arranging the proof of the theorem into different cases, each case consisting of subcases, which are in turn split into sub-subcases, provides the working mind with a framework with which to attack and understand the problem. Each sub-subcase is attempted with some special features in mind, and later generalized.

DEFINABILITY AS GENERALIZED ALGORITHMS

Fundamental results in the theory of computation tells us that most functions on natural numbers are not algorithmic (recursive, or computable). A compromise, and very common in mathematical practice, is definability. Let me explain.

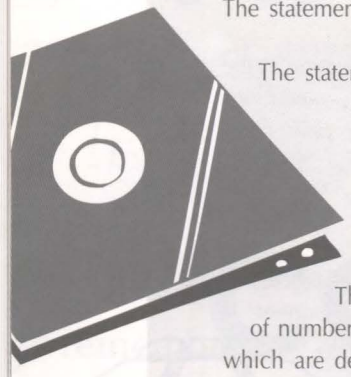
A 13 year old has no problem seeing that $2^2 > 0$. This statement is algorithmic. It is computational. It takes him some time to see that 'the square of every nonzero number is positive'. In mathematical form, this statement reads

$$\forall x \, [\, x \neq 0 \rightarrow x^2 > 0 \,].$$

The observant reader will see that there is a 'for all' quantifier ($\forall$) at the beginning of the statement. It is clearly a statement more complicated than $2^2 > 0$ which is quantifier free. Now the statement 'the set A of numbers is unbounded' has an even more complicated structure:

$$\forall x \exists y [y > x \ \& \ y \in A].$$

The statement starts with two quantifiers $\forall\exists$, which is more complicated than a plain $\forall$ in the example above.



The statement 'limit of the sequence x_n is 0' assumes the form

$$\forall \varepsilon \exists n \forall m \geq n [\, |x_m| < \varepsilon \,].$$

Here we see $\forall\exists\forall$, a form obviously more complex than what we have seen earlier. Forgetting that it took us some time to appreciate this, we often wonder why students 'can't understand limits'.

The theory of computation (recursion theory) is a subject which investigates, among other things, the complexity of number-theoretic functions. It attempts to provide classifications of functions according to their complexity, and those which are definable by quantifiers ($\forall$, $\exists$, $\forall\exists$, $\exists\forall$, $\forall\exists\forall$, $\exists\forall\exists$, ...), and beyond.

The human mind is limited in its ability to understand complex functions or statements. I have not met a mathematician who professes to understand a function which requires 23 alternations of quantifiers. Personally I have difficulty understanding a function which requires 5 or 6 alternations of quantifiers to define.

But the power of mathematics is in transforming possibly very complicated functions into more manageable forms. Taylor's theorem is a nice example: Every such function f has a Taylor series expansion

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(0)x^n}{n!}.$$

No matter how complicated the function is, it always has a $\forall\exists\forall$ expression, modulo the infinite set of values $f^{(n)}(0)$ (we may use these values as an oracle to compute f). Such reduction of complexity is common in mathematical practice, often expressed in terms of theorems. The Weierstrass approximation theorem, which states that every continuous function is the limit of polynomials, is another example. Polynomials are computable. The notion of limit is definable via a $\forall\exists\forall$ statement. Together they provide an insight into the nature of continuous functions, via definability. Again the classification of finite simple groups is another example.

Of course it is not true that the complexity of a mathematical statement is necessarily commensurate with the difficulty of its proof. Fermat's last theorem assumes the form of $\forall$ ..., yet there is no elementary proof at this point.

A THESIS

Our thesis is that mathematical practice is inherently (generalized) algorithmic. When effective algorithms are not available, definability is the next best thing. To reduce definability further to the algorithmic level, approximations in the form of limits are used. Hence a highly complicated function is transformed into a computable function by means of Taylor's theorem. The subject of numerical analysis may be viewed as that which maps a given function to a computable one through the introduction of analytic methods.

In terms of the number of alternating quantifiers used, the complexity of mathematical statements are often low. When higher complexity is required, for example in a complicated proof, one resorts to definability with oracles. This is how mathematics is handled, because of the limitation of the human brain.

The informal approach or understanding of the nature of mathematics is formalizable with the use of logic. Stephen Simpson initiated the study of *reverse mathematics*. In this subject, there is a hierarchy of what one calls subsystems of analysis. These systems are of increasing proof-theoretic strength. Various results in algebra and analysis have been shown to be equivalent to certain subsystems. For example, the assertion that every bounded set of reals has a limit point is known to be equivalent to the system where 'every arithmetically definable set exists'. Roughly this says that in such a system, if there is a property definable by a statement with a finite alternation of quantifiers, then there is actually a set with the given property. In naive set theory, this is a basic assumption. But in applications of mathematics, especially when computation is involved, such an assumption is often modified to mean 'definable set existence', or even 'computable set existence', with the help of approximations. Thus set existence is ultimately a major concern of mathematical research.

The bridge that links mathematics and computing is therefore definablility. At the algorithmic level, the most natural example, other than computability theory itself, is in my opinion discrete dynamical systems. Take complex dynamics as a model. The notion of iteration of complex valued polynomials can easily be described via flow charts. Fatou sets are recursively enumerable (or computably enumerable) in the sense of computation theory. This means that there is a computation machine which for a given input, outputs the answer 'yes' if the input is an element of the Fatou set, and outputs nothing (perhaps because the machine never halts) otherwise. Julia sets are just those inputs for which the machine never halts. Once such a perspective is adopted, it is not difficult to see why it is difficult to describe the geometry of Julia sets. As Blum, Shub and Smale have pointed out (1989), most Julia sets are not computable, i.e. for most cases, there is no algorithm to decide if a given complex number belongs to a particular Julia set. Computer graphics will therefore always be an approximation for geometric representation of objects in complex dynamics, not because of the limitation of hardware or software, but because of the uncomputability of such objects.

By crossing the bridge, mathematics returns to the very intuition which propelled its growth: the attempt to solve problems algorithmically and, failing this, to understand why it is impossible to do so. At this level, there is no gap between mathematics and computing.

Text of the Presidential Address delivered at the 28th AGM of Singapore Mathematical Society on 10 February 1996.

Chong Chi Tat is Professor of Mathematics at National University of Singapore. He is also the Head of the Department of Information Systems and Computer Science, as well as Vice Dean of the Faculty of Science.