

Why Mathematics in Computer Science?

What is a science?

Perhaps we should first ask about the 'science' part of Computer Science. What is science? This is a philosophical question which relates to the broader issue of how we gain knowledge about the world in which we exist.

The philosophical study of our sources of knowledge is known as *epistemology*. Since the laws of physics are supposed to be invariant over all time and space we could say loosely that physics espouses a Platonian view of knowledge in which the 'laws' are there and fixed and it is up to us to discover them: usually in some very pure mathematical form. We often call a set of rules (or in computer science an *algorithm*) which enables us to predict the outcome of some situation a model. So then we might characterise science as:

SCIENCE: The building of models which under clearly defined circumstances lead to accurate predictions regarding measurable quantities in the real world.

MODELS: Because the need is for *accurate* predictions of *measurable quantities* (Eg. Time, Energy, Data in MgBytes etc.) the 'MODEL' is invariably a *mathematical model*.

So what is mathematics?

MATHEMATICS: A language for the concise and accurate expression of relationships - usually (but not necessarily) quantifiable relationships.

CONCLUSION: Mathematics is an essential (indeed *the essential*) tool for a computer *scientist*. You can be an applications programmer or a systems manager without much knowledge of mathematics, but you *cannot* be a computer scientist.

So we ask what then is process of doing mathematics?

Key Concepts: Generalisation, Abstraction, Proof.

GENERALISATION: The extension of an idea or ideas to a wider class of instances.

If we work out a way to deal with (model) a particular kind of situation we may find the same method can be applied to a wider class of situations.

Example: We might have an algorithm (numerical recipe) for sorting integers. With a little thought we find we can use exactly the same method to sort real numbers, alphabetic strings, indeed any set of objects which has an associated ordering.

ABSTRACTION: This is the process of figuring out exactly what are the *common features* of many different instances. We look for common features which are important to the aspect we are considering.

Examples: The common feature required to describe a perfectly general **sort algorithm** are that we need to have a *set of objects* which has a *defined ordering relation* and given any two members of the set we need to be able to *determine which of the two comes first*.

Examples of abstractions in mathematics are: *graph, group, metric space* etc. All of which have very simple and intuitive definitions and vastly many different kinds of instances.

PROOF: *First approximation:* any argument which convinces a rational person that the conclusion follows from the premises.

Being computer scientists we should like something better than this. Can proof be automated? If we could automate proof on a computer (and we all agreed that the automation was valid) then the tedious process of ‘proving a theorem’ could be subjected to a program which constructed a step-by-step proof automatically. Then there would be very little need for mathematicians!

PROOF: *Second approximation:* any argument which proceeds step-by-step from premise to conclusion using agreed *logical* reasoning.

This certainly looks more promising but raises a number of questions. For example:

- What are the generally agreed logical reasoning processes?
- Can such processes be automated?

Anyone who can come with some sensible answers to these questions will deserve a bonus on their coursework. To better understand these questions let us consider some examples.

Example 1: Pythagoras’s theorem.

As an example we consider Pythagoras’s theorem. Everyone thinks they know all about this:

- *The square on the hypotenuse of a right angled triangle is equal to the sum of the squares on the two adjacent sides.*

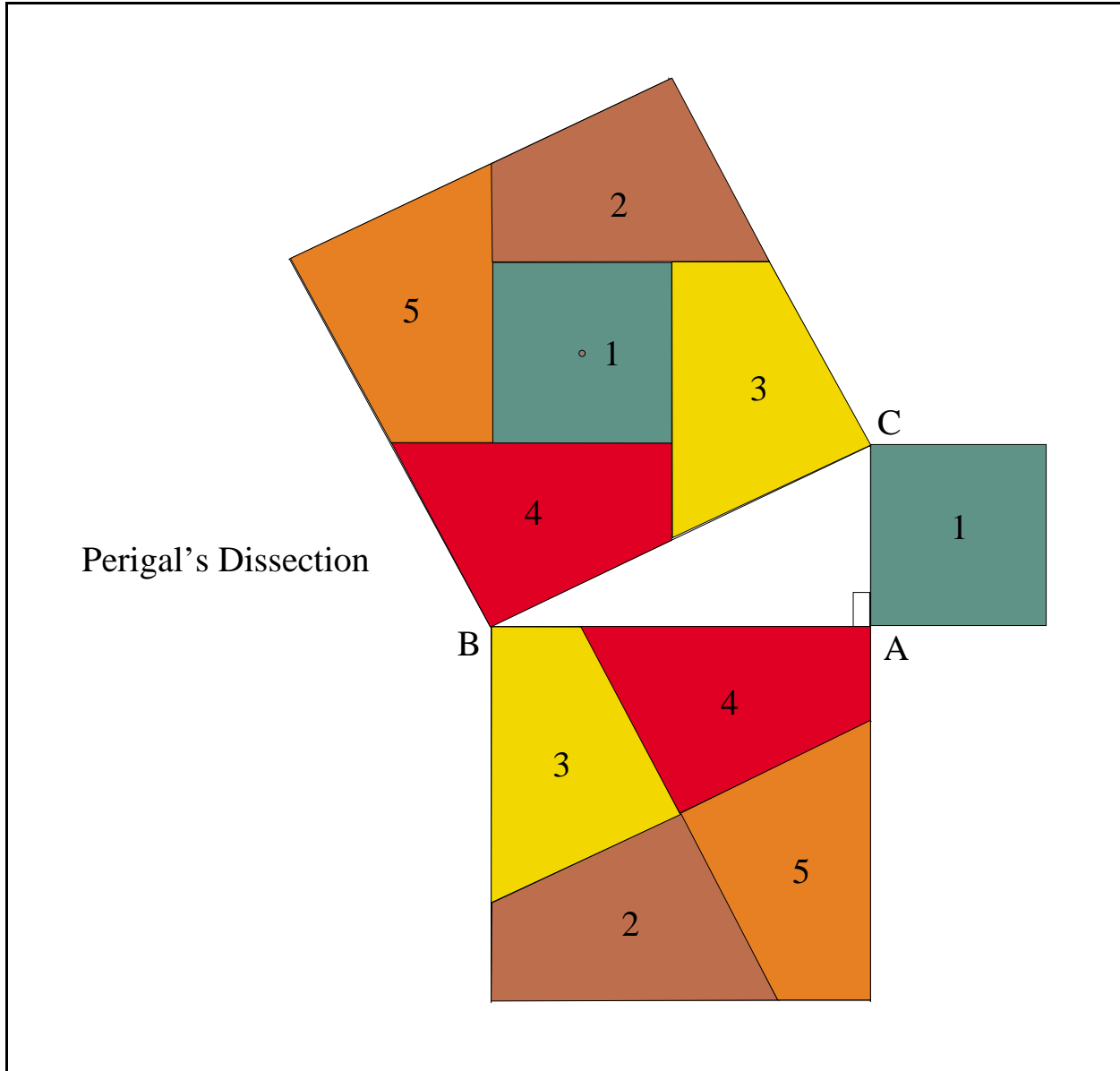


Figure 1 Perigal's dissection for Pythagoras's theorem.

We can see why this might be true by examining Figure 1.

- *Discussion:* Does Figure 1 constitute a proof? If not can we turn it into one? If so can the proof be automated?

Example 2: Warning.

Some dissection ‘proofs’ can be misleading. For example see Figure 2.

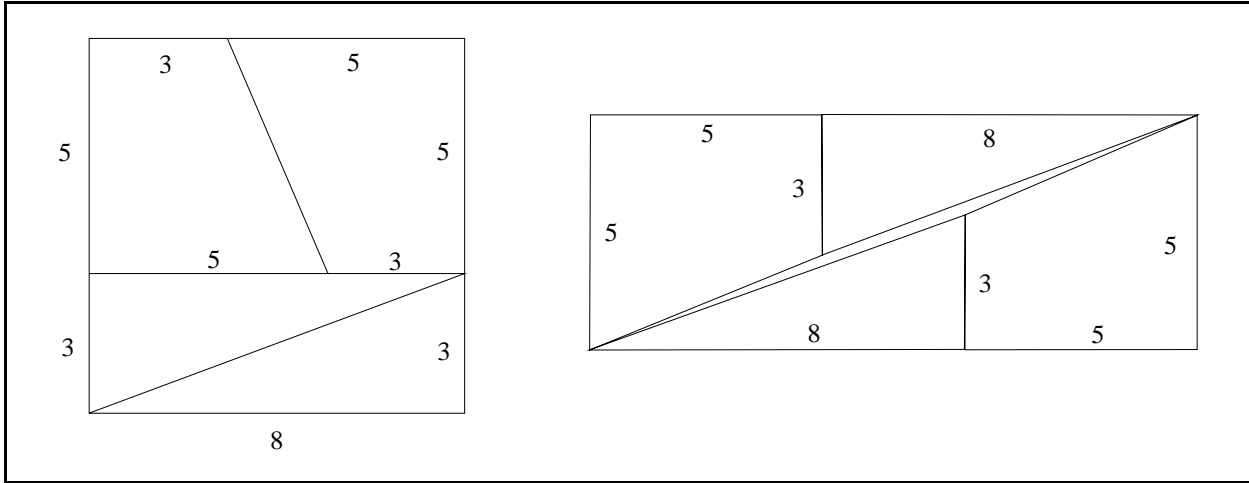


Figure 2 Some dissection ‘proofs’ are false.

In cardboard the parallelogram in the middle is scarcely discernable, in which case we might easily conclude that $8 \cdot 8 = 64 = 65 = 5 \cdot 13$.

Thus there is a sense in which diagrams such as Figure 1 should be viewed with suspicion when being examined as a proof.

Generalisation/Creativity

Pythagoras’s theorem says: *The square on the hypotenuse of a right angled triangle is equal to the sum of the squares on the other two sides.*

Do we now know everything there is to know about Pythagoras’s theorem?

- For example: if it is true for squares is it true for other geometrical shapes?

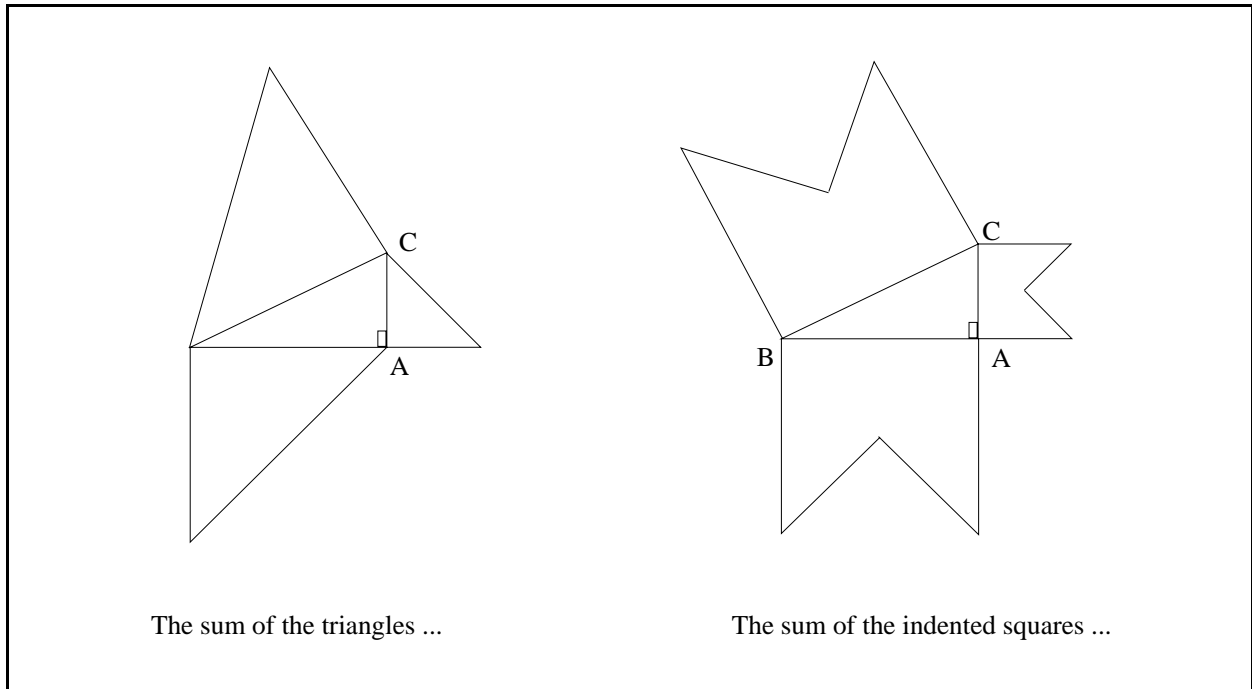


Figure 3 Other ways of putting it.

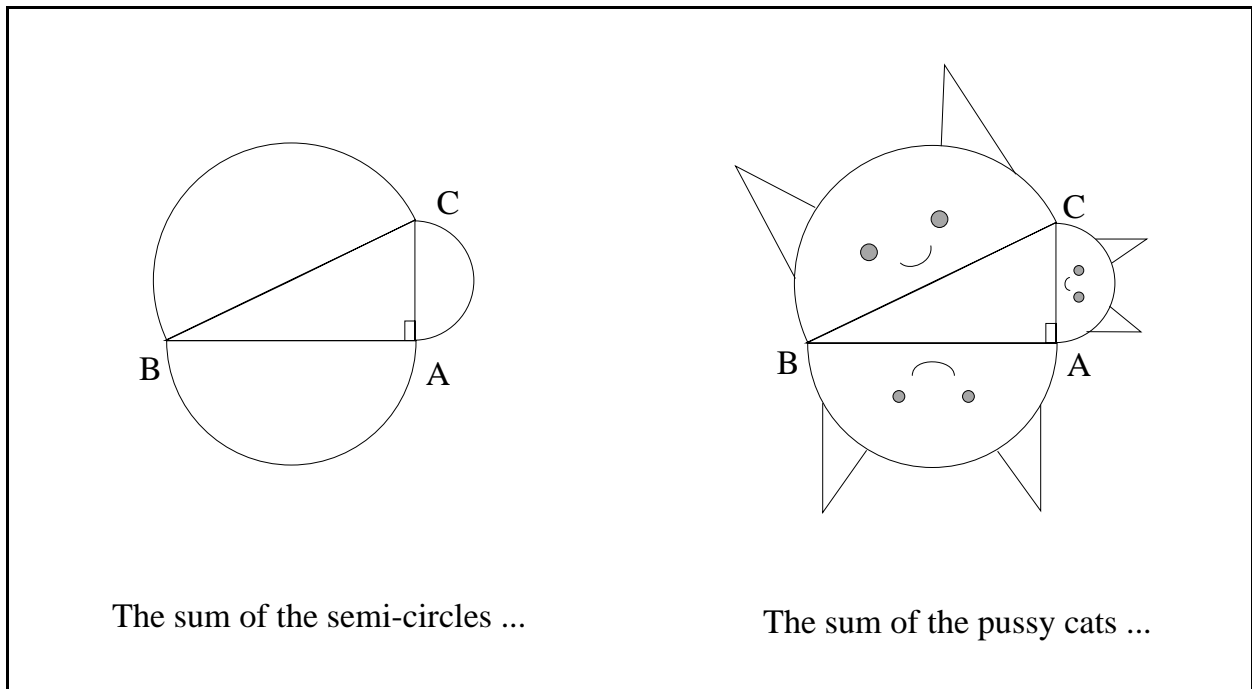


Figure 4 The pussy cat theorem.

Antonia J. Jones 14 October 2001

All the propositions suggested by these figures are obviously true if Pythagoras's theorem is true.

Thus we can see that generalization is a useful conceptual tool: we take a proposition and try to see if the idea can be applied to a wider (more general) set of circumstances.

- If the proposition does not generalize in a particular direction this shows us a logical boundary and suggests a *necessary condition* that must be fulfilled before the proposition can be true.
- If the proposition does generalize then we have discovered a new insight which helps us to understand that the proposition has more power than we originally thought.

Example 3: Instant Insanity.

Instant insanity (crazy cubes) is a simple puzzle using four coloured cubes. Each face of each cube is coloured either red, white (sometimes yellow), blue, or green. The object of the puzzle is to arrange all four cubes in a line so that each of the four long sides has four different colours on it.

If we start to think about ways to solve this puzzle the obvious way is to test every possible arrangement and find one of these that meets the criterion. There are some obvious symmetries that we can factor out.

- The order of the cubes in the line makes no difference. There are four ways to choose the first cube, three ways to choose the second cube, etc. So there are $4 \cdot 3 \cdot 2 \cdot 1 = 24$ ways to arrange the cubes in a line. This reduces the number of cases we have to look at by a factor of 24.
- Rotating the whole line of cubes about the long axes by 90 degrees changes nothing. There are four possible orientations. So this is a further reduction.

If we consider the possible rotations for each cube there are three axes and four positions about each axis which gives $4^3 = 256$ positions. There are four cubes so without the second simplification (above) there are $4^3 \cdot 4^3 \cdot 4^3 \cdot 4^3 = 256^4 = 4,294,967,296 \approx 4.3 \times 10^9$ ways of arranging them.

By hand, not very feasible! By computer, if it takes 10^{-6} sec to check one configuration then the total time would be about 4,300 secs, i.e. about 71 mins.

Is there a better way?

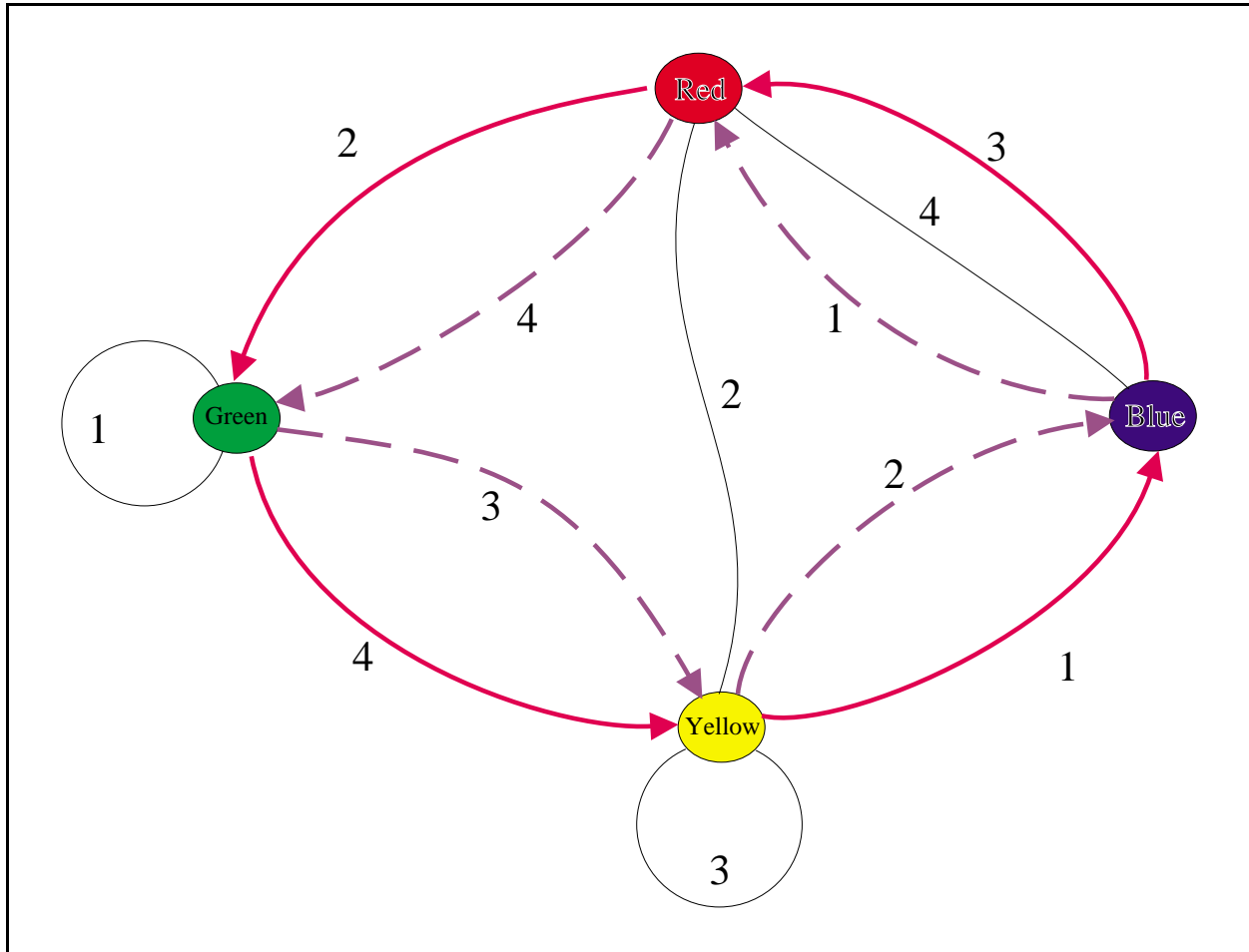


Figure 5 Graph theoretic interpretation of instant insanity.

Suppose we label each cube, say 1, 2, 3, 4. Next we create a graph with the four colours as nodes. We pick up the first cube, choose a face (say yellow) and look to see what is the colour of the opposite face (say blue). We then join the yellow node to the blue node with an arrowed arc and label this arc 1 (because it was the *first* cube). We do this for every pair of faces on every cube, see Figure 5.

We see that if we can find two disjoint (no arcs in common) circuits on this graph which use each arc label (1, 2, 3, 4) exactly once then we can immediately construct a solution.

Moreover, if we find all such pairs of disjoint arcs we can find all possible solutions!

- The point of this example is that, whereas we can write a computer program to solve this problem by exhaustive search (which if it was a much larger problem

Antonia J. Jones 14 October 2001

would quickly become infeasible), the graph theoretic solution requires genuine creative insight - the program hasn't been written which could solve a whole variety of essentially different problems like this.

So clearly the mathematical way of thinking:

- The idea of 'abstraction';
- 'What constitutes a proof';
- 'How do we construct a proof';
- 'Can we automate the process of proof construction';
- 'Can we generalize some proposition so as to better understand it';

all bear a strong connection to what we can and can't do with computers.

Computational complexity.

Whereas in mathematics we are concerned with ideas, their definitions and their logical consequences and how to construct proofs, in the physical sciences there is a slightly different idea of proof.

When physicists try to make predictions they are following one of the basic principles of science:

- Postulate the basic laws - they are supposed to hold for all time and in all places.
- Calculate the consequences.
- Perform experiments or observations to verify the predictions. Successful verification does not constitute a 'proof' of the law but failure to verify might constitute a disproof (if all the loopholes in the logic have been plugged).

- The *advantage* of having such laws available is that because they are supposed invariant over time and space one may be able to make predictions for circumstances that have never been observed before - we call this extrapolation.
- The *disadvantage* is that sometimes the calculations directly from the laws (or 'first principles' as it is often called) may be so complicated or take so long as to be impractical.

The barrier which often presents itself is one of *computational complexity*. As a simple example consider the protein folding problem. A big protein has thousands of constituent atoms and we might know its atomic structure exactly. The biological action of the protein is what we would like to predict. Now if you were to hold the protein by both ends and let go it would collapse into something which on the right scale would look like a tangled ball of wool. The biological action of the protein is largely determined by what is left on the outside of the ball of wool. So the problem is simple: we know the effects of atomic bonds, we know the structure so let's just plug all this into

Antonia J. Jones 14 October 2001

a computer program and compute the folded structure. That sounds good, but except for fairly small molecules it can't be done - the program takes too long to run. But things are even worse than this!

Indeed even without the Heisenberg uncertainty principle (which says you cannot measure both the position and momentum of a particle with an arbitrary degree of precision) the universe *a la* Newton really contained the seeds of its own destruction. A *chaotic* process is one which for which a slight variation of the initial conditions can have a huge effect on the long term behaviour of the system.

- Even if you know all the initial conditions of some quite simple chaotic process exactly (which actually you can't) then the amount of computation required to predict a long way into the future with the fastest computer one could imagine would still require a time greater than the estimated life expectancy of the universe.

This is the first lesson of chaos. An example is the weather - where we know all the laws and can measure to our heart's content but we cannot even predict reliably several days into the future, let alone several months.

Thus the purely logical or mathematical question of what can and cannot be computed in a reasonable time has profound implications for all of science and not just computer science.