Principles of Computation

David Hui

25th September 2016 –

Chapter 1

Of Linguistics

Let us begin by discussing the etymology of the noun ‘computation’. Initially we will break it down into the stem ‘compute’ and the suffix ‘-tion’. The suffix ‘tion’ is a modifier which converts a verb into a noun, namely ‘the noun of the action of the verb’. Its first use in English was during the Middle Ages. ‘–tion’ had been taken from Latin and popularised by Old French before finding its way across the Channel.

In our example, the infinitive of the verb is ‘to compute’. The ancestor of this word is the Latin infinitive ‘computare’, which has much the same meaning. But what does it mean to compute? The etymological tree offers more suggestions. ‘Computare’ itself was derived from two Old Latin words: ‘cum’ and ‘putare’ which translate to ‘thoroughly’ and ‘to value’. So computation is a form of valuing an object, an enumeration perhaps. Going further, we discover that ‘putare’ is derived from the Proto-Indo-European ‘*pew’, meaning ‘to purify’. Its onomatopoiec sound suggests that we are at the root of this word. It would seem that the meaning of ‘to purify’ is far removed from ‘to compute’!

On the contrary, the meaning of words changed as the Romans developed from savages to a civilisation. Confusingly, two different Roman subcultures derived two similar words from ‘*pew’: ‘purus’ and ‘putare’. These two words are the noun and verb of pure, respectively. Later, ‘purus’ too on other meanings, such as ‘to prune’, or ‘to trim’, in the sense that a trimmed plant was clean. This was homologically similar to the act of accounting, in which a complicated account was reduced to a number valuing the amount a citizen would owe the Emperor. It was then that ‘purus’ was amalgameted with the verb ‘putare’, which had meanwhile acquired a dialectical meaning. During logical arguments or discussions, false untruths were cleansed, and, according to the Romans, only the ‘pure’ truth remained.

This implies that an etymological reading of ‘computation’ is *the act of thoroughly simplifying a complex situation using logic to enumerate a solution’.*

During the latter stages of the 20th century, the word computer, originally meaning ‘one which computes’ took on an additional meaning. The modern day computer is an electronic device built to do computation. This implies that there is a misnomer of the term ‘Computer Science’ due to the double meaning! One possible explanation of the term is ‘the study of computation through experimentation’, and the other is ‘the study of an automated computer through experimentation’. Both definitions are sadly equally incomplete – the former does not include topics such as software design and the latter omits the practice of designing new algorithms. In addition, computation does not require the scientific method nor experimentation, as software design is more akin to engineering as algorithms are to mathematics. Studies is an umbrella word more fitting for perhaps this multidisciplanary field, so there is a case for calling the subject ‘Computational Studies’. However, as all academic disciplines are studies, postpending the word seems tautological. Instead, the subject should perhaps be named *Computation*, because of its brevity and ease of use. Similar to mathematics, ‘Pure Computation’ should refer to the design and proof of new algorithms, ‘Applied Computation’ to their implementation in software design, and ‘Applications of Computing’ to interface design.

It could be argued that the universe performs computation simply because its state is changing. The previous state is complex, and then the universe recognises this state and produces the next one according to logic from the laws of physics. This seems to suggest that a physical instantiaton is the only way in which computation can be achieved.

However, *representation* is a method of evading this time-and-space-consuming venture. It is a key feature of language, prevalent in words. A words is a method of representing a physical entity and, through discussion and conversation, more representations are produced, which may sometimes solve a problem. The computation being done through discussion is the ‘putare’ known to the Romans. Not all languages are ideal for computation, though. There is no objective way to define words, as every person and indeed computer would develop different personal opinions of words due to the difference in the objects they describe. These differences, though they may be small, may fester misunderstanding and an incorrect computation in the eyes of some. Thus a language containing universal definitions is needed. Thankfully, such a language exists – mathematics. Generally, problems are converted from physical entities into mathematical representations. Calculations are done on the input, returning an output which can then reconverted into physical entities. In most cases, the conversion is achieved by a human.

Finding a good algorithm is equally important to finding a good representation!

If it is possible to prove that the proceedure will always return an answer, then the proceedure is known as an algorithm. Proving an algorithm will output the correct answer given any input requires a four part proof.

1. Prove that there is a 1:1 mapping (bijection) between all possible physical input states and input representations.

2. Prove that the proceedure is an algorithm – that is, for all input representations, it always returns an output representation. This property is known as halting.

3. Prove that the algorithm always outputs the desired representation.

4. Prove that there is a bijection between output representations and all possible physical output states.

In addition, it is useful to establish the runtime of the algorithm. Usually, only bounds on the runtime can be established. The validity of these results, too, will need to be proved.

As an example and a first proof, we will now show that it is possible to achieve universal computation by means of representation using cardinal numbers, albeit with some assumptions.

We cannot prove the existence of a bijection between all possible input states in the universe and ordinal numbers, simply because it is impossible to enumerate the former condition. This we will assume that this fact is possible. Similarly, we cannot prove the existence of a bijection between the universe and all output states.

We will choose to represent ordinal numbers in a binary format, because of the stablity of electronic representation. It is easier to determine the state of a bistable variable than the state of a multivariate variable. As the cardinal numbers are a representation of counting, it is only necessary to show that there is a consistent method of counting and that there is a null term, in other words, a binary representation satisfies the Peano Axioms.

The null term is 0, and the rules for counting are: if the rightmost symbol is 0, change it to 1. If it is 1, change it to 0 and recursively go left until there are no more digits, whereupon a 1 is prepended to the existing list.

Thus a reformulation of the statement to prove is $m \longrightarrow n$, where m and n are binary strings of 0s and 1s of arbitrary length. This can be achieved by running induction twice.

We can observe that if the length of n is 1, then any output string of arbitrary length can be generated simply by concatenating many binary strings of length 1. Thus initially, we will prove that it is possible to generate all possible binary strings of length 1 from all possible binary strings.

The problem is thus put into a conjoined format into proving $m \longrightarrow 0 \wedge m \longrightarrow 1$. Thus completes our first induction and begins the second.

When m is of length 1, we can see immediately that two operations are required: $0 \longrightarrow 0$ and $1 \longrightarrow 0$. Similarly, operations are required for $0 \longrightarrow 1$ and $1 \longrightarrow 1$. Let us call the operation when the input is equivalent to the output CARRY and the other NOT.

When m is of arbitrary length, let us assume that we can compute it into a binary string of length 1. Now let us take a binary string of length m+1. Let us substitute the first m digits of the string into a binary string of length 1 by our assumption. Thus the following eight relations are generated. Let us now show that it is possible to satisfy these eight conditions by exhaustion.

$$\begin{array}{ll}
00 \longrightarrow 0 & 00 \longrightarrow 1 \\
01 \longrightarrow 0 & 01 \longrightarrow 1 \\
10 \longrightarrow 0 & 10 \longrightarrow 1 \\
11 \longrightarrow 0 & 11 \longrightarrow 1
\end{array}$$

Let us now introduce a new operation called OR that takes two inputs and return one output. It follows the following rule:

$$\begin{gathered}0 \text{ OR } 0 \longrightarrow 0\\ 0 \text{ OR } 1 \longrightarrow 1\\ 1 \text{ OR } 0 \longrightarrow 1\\ 1 \text{ OR } 1 \longrightarrow 1\end{gathered}$$

As we can see, the following combinations of rules satisfy the eight conditions thus.

$$\begin{array}{cc}0 \text{ OR } 0 \longrightarrow 0 & \text{NOT } (0 \text{ OR } 0) \longrightarrow 1\\ \text{NOT } (0 \text{ OR } 1) \longrightarrow 0 & 0 \text{ OR } 1 \longrightarrow 1\\ \text{NOT } (1 \text{ OR } 0) \longrightarrow 0 & 1 \text{ OR } 0 \longrightarrow 1\\ \text{NOT } (1 \text{ OR } 1) \longrightarrow 0 & 1 \text{ OR } 1 \longrightarrow 1\end{array}$$

Thus by exhaustion and induction twice, we have a proof. ■

It is quite interesting to note that only one of two binary operations in Boolean arithmetic is needed for completion. In this instance, OR is chosen rather than AND because of its exitation energy, perhaps making it thermodynamically efficient. The fact becomes less surprising given De Morgan's Laws, which show how one binary operation can be expressed in terms of the other.

Note that for this proof we do not prove correctness because it is not defined for this proof. Each problem has its own criterion for correctness, which usually correspond to a subset of the above transformations. We have only proved that computation is possible for all problems that can be represented in cardinal numbers.