Computability I - Computable Numbers

CS605 - Math & Theory of CS

T.J. Naughton

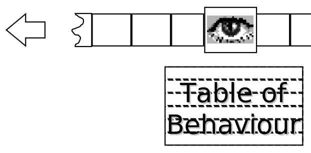
Maynooth University, Ireland

Turing, 1936

- A. M. Turing,
  "On computable numbers, with an application to the *Entscheidungsproblem*",
  Proceedings of the London Mathematical Society, 1936

Turing, "On computable numbers..."

- Turing begins his definition of computation by describing a computer* who has a number of configurations (states of mind), a tape divided into squares each capable of bearing a symbol, and the ability scan one square at a time.



* which was a person ('one who computes') in the language of the 1930s.

Turing, "On computable numbers..."

- The person's actions are totally determined by their current state of mind and the currently scanned symbol
  - erase/write one symbol
  - scan left/right one square
  - change configuration (state)
- Although these actions are restricted in form, they represent a set of atomic elements from which all mathematical operations can be composed.

Turing, "On computable numbers..."

- All possible actions that the person may make are given by a *'table of behaviour'*.
- Each table of behaviour then effectively defines a new person in Turing's system.

State i	*Read*	*Write*	*Move*	*State i+1*

Turing, "On computable numbers..."

Some restrictions:

- Finite no. of states of mind so there is no confusion between states (in fact, it has been proved that two states is sufficient)
- Finite no. of symbols (which still permits an infinite number of possibilities) How?
- Unlimited tape length (although there will be a finite number of non-blank symbols on it at any particular time)

Turing, "On computable numbers..."

- An important observation by Turing in his defence of imposing a finite number of states of mind was that complicated states could be simulated by *writing more symbols on the tape*.

Q Have you ever encountered that before?

Turing, "On computable numbers..."

- Computing is normally done by hand by
  - writing certain symbols on 2-D paper, one at a time, and applying operations to them.
- Turing justified the generality of his computer by showing that they
  - use 1-D paper to write out certain symbols one at a time, use the atomic operations from which all mathematical operations are composed, and have enough paper to represent even the most complicated state of mind.

Turing, "On computable numbers..."

At this stage

- we will not describe the table of behaviour in any more detail
- an appreciation of its content is sufficient

State i	*Read*	*Write*	*Move*	*State i+1*

Turing's machine

- So the person follows the table of behaviour, scanning squares, reading and writing symbols and changing their state of mind.
- Later, Turing states without clarification that we "may now construct a machine to do the work of this computer..."

  This claim could demand much attention in an AI context.

Turing's machine

- So how exactly is the human computer's 'state of mind' realised physically?
- Turing uses the example of how a person might take a break in the middle of their work, and before going away and forgetting all about it they write down a list of unambiguous instructions describing the current state of the work and how it is to be continued.

Turing's machine

- If the person works in a particularly unfocussed manner, they may never do more than one step at each sitting, and so

  the state of progress at any stage in the computation will be fully described by the list of instructions and the symbols on the tape.

Turing's machine

- The combination of tape symbols and instructions is called the "state formula" or global state or configuration.
- We know that the next global state may be derived from the current one,

  so we must presume that the transition may be expressed by the axioms of some functional calculus (there are rules for deriving one global state from another).

Turing's machine

- If this presumption holds, we can build a machine to write down the successive global states, and so calculate/compute the required number!
- This was the Turing machine.

Turing's machine

- Turing machines (TMs) have another important property.
- Since computing was reduced to the mechanical process of looking up entries in a table, TMs have the ability to inspect the table of behaviour of another machine, and emulate it.

Turing's machine

- Practically, this would be achieved by encoding the other machine's table on the tape along with the input.
- The ability of TMs to assume another's table of behaviour will be taken to its logical conclusion a little later.

Turing, defining computability

- Turing was now ready to give his definition of computability,

  however he cleverly rephrases Hilbert's original question...
- Recall how Gödel's *Incompleteness theorem* began with the idea that statements *about* numbers could be coded *as* numbers?

Turing, defining computability

- Well, Turing used this to refresh Hilbert's question *about proofs*.
- Turing's machine is essentially one which executes *statements about numbers,*

  but he was able to describe it in terms of a machine that computes *the value of numbers*.

Turing, defining computability

- Turing maintained that the underlying problem addressed by Gödel’s *Incompleteness theorem* was how we might specify the infinite in finite terms,

  for example...

Turing, defining computability

- How might we specify the infinite sequence of digits in a real number, such as π=3.14159...?
- What does it mean to say that there is a definite method for calculating such a number?
- Can we define all of these ‘calculable numbers’?

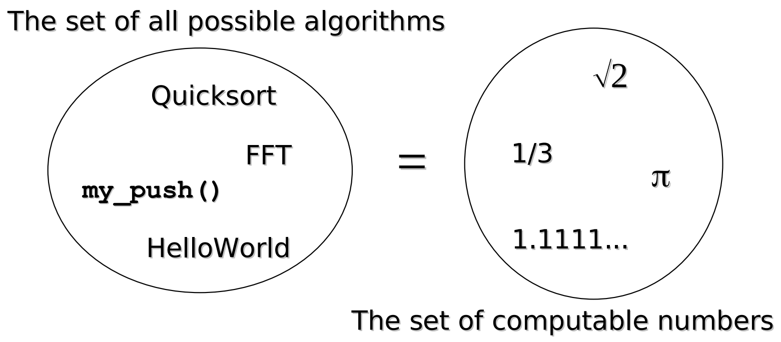
Turing, defining computability

- The *computable numbers* (according to Turing) form a class defined as those <u>infinite</u> real numbers that can be printed by a TM starting with a blank tape.
- It can be proved that π and $\sqrt{2}$ are computable numbers, as well as every real number (irrational or otherwise) defined by the normal equations of mathematics.

Turing, defining computability

- Turing equates the limits of what statements can do with numbers with the limits of writing out numbers by TM.
- He hypothesises that the set of all things that instructions (algorithms) can do with numbers can be equated with the set of numbers that TMs can write out.

Turing, defining computability


The set of all possible algorithms
Quicksort
FFT
my_push()
HelloWorld
=
√2
1/3
π
1.1111...
The set of computable numbers

Turing, defining computability

- This was the basis for Turing’s formal definition of computability

  and it has never since been challenged.

Turing, defining computability

In summary then

- Each table of behaviour defines a new TM.
- A number is *computable* if one of the TMs computes it.
- A process/calculation is computable if it maps to one of the computable numbers under some global encoding scheme.

Turing, defining uncomputability

- Armed with this definition of computability, however, we can prove that there are some numbers which are not of the computable class; that some *uncomputable numbers* exist.

- How?

CS605 Computability I