FIT3014 Analysis and Design of Algorithms

Lecture 18
The Church-Turing Thesis

A./Prof. David Dowe
Clayton School of Information Technology
Monash University

Contents

- Turing Machines (TMs)
- An Addition Machine
- Equivalence
- Universality
- Nondeterministic Turing Machines (TMs)
- Church-Turing Thesis
- Ackermann's Function
- Busy Beaver Function
- Elusive Model Paradox (Scriven; Lewis & Shelby Richardson; ...)

Recommended Reading:
Dewdney *The Turing Omnibus* or *The (New) Turing Omnibus*
H. Lewis and C. Papadimitriou *Elements of the Theory of Computation,* 2nd ed. (1998), Prentice-Hall
Wikipedia

Abstract Complexity Theory

This part of the subject deals with large groupings of complexity classes of *problems* (or, functions) – in terms of time complexity of the fastest algorithms for solving them.

Roughly in order of complexity:

- **P**: polynomial-time problems
- **NP**: non-deterministic polynomial-time problems
- **NP-Complete**: non-deterministic polynomial-time problems which are "NP general"
- **NP-Hard**: problems which are "NP general" or harder
- **EXP**: exponential-time problems
- **Unsolvable, Noncomputable**: infinite-time problems

Turing Machines

Definition 1 *A Turing Machine M is a set of quintuples $\{Q_n\} = \{\langle q_i, q_j, s_k, s_l, H\rangle\}$ where*

- $q_i, q_j \in \{1, \ldots, m\}$ *(the machine states)*
- $s_k, s_l \in \{s_0, \ldots, s_r\}$ *(the symbols)*
- $H \in \{R, L\}$ *(tape head direction)*

such that no two quintuples have the same first and third elements.

The quintuples with $q_i = 1$ describe what happens in the START state of the machine.

Definition 2 *The size of a Turing machine M is the number of its distinct states, written $|M|$.*

This emphasises that Turing machines are abstract (idealised) objects — identifying them with sets of quintuples of numbers and symbols.

> Well, we also need a few other things like a read/write head and an infinitely long tape. These are common to all Turing machines; the set $\{Q_n\}$ "distinguishes" between machines.

Digression: Alternative definition of Turing Machine

We digress on this slide and present an alternative definition of Turing machine on this slide.

Definition 3 *A Turing Machine M is a set of quadruples $\{Q_n\} = \{\langle q_i, q_j, s_k, \{s_l, H\}\rangle\}$ where*

- $q_i, q_j \in \{1, \ldots, m\}$ *(the machine states)*
- $s_k, s_l \in \{s_0, \ldots, s_r\}$ *(the symbols)*
- $H \in \{R, L\}$ *(tape head direction)*

such that no two quadruples have the same first and third elements. The Turing machine in state q_i given input s_k either stays where it is and writes a symbol (s_l) or moves to the left or right without writing a symbol.

The quadruples with $q_i = 1$ describe what happens in the START state of the machine.

Definition 4 *The size of a Turing machine M is the number of its distinct states multiplied by the no. of distinct symbols - often called the state-symbol product.*

End of digression.

Turing Machines

We use the conventions:

- $s_0 = \sqcup$ = blank
- The tape is infinite to the right.
- In the START state, the head is located at the leftmost square.
- If there is no quintuple for *state* and *input* then HALT (variation: designate a halting state)

M operation:

---

1. $state \leftarrow 1$;
2. LOOP: $input \leftarrow$ READ TAPE;
3. IF $\exists Q_i$ WITH $(q_i = state)$ AND $(s_k = input)$ THEN
   (A) WRITE s_l ON TAPE;
   (B) MOVE HEAD H-WISE;
   (C) $state \leftarrow q_j$;

(D) GO LOOP;

ELSE HALT;

Turing Machines

Clearly, our Turing machines compute functions: the input is the symbol string on the tape at the start; the output is the string on the tape at the end; its operation is deterministic.

Of course, the machines may not halt.

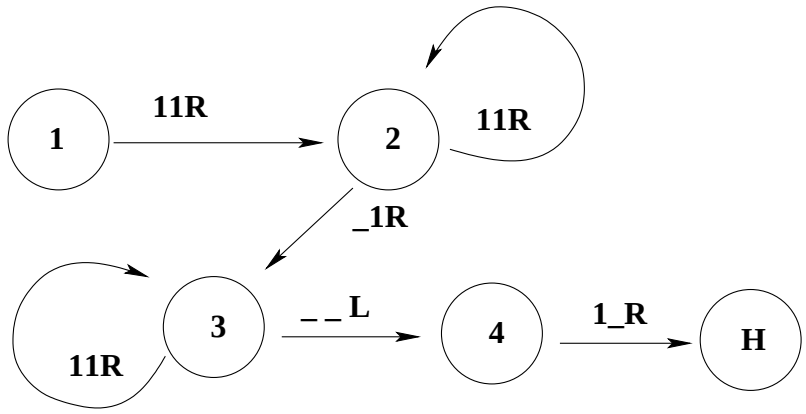
We can also talk of languages *accepted* or *recognised* by an M (or by a Turing machine, M). E.g.,

- An input is rejected iff
  - M attempts to read beyond the (left) end of the tape;
  - or, M reaches designated rejecting state (Choose one or both of these for your definition.)
- An input is accepted iff M halts in a non-rejecting state.

Example: An Addition Machine

This machine adds two unary numbers (both at least 1), terminated by blanks (and separated by a single blank).

unary: e.g., 4 is represented by "1111␣"



OR (recalling $< q_i, q_j, s_k, s_l, H >$):

$\{\langle 1, 2, 1, 1, R\rangle, \langle 2, 2, 1, 1, R\rangle, \langle 2, 3, \sqcup, 1, R\rangle, \langle 3, 3, 1, 1, R\rangle,$
$\langle 3, 4, \sqcup, \sqcup, L\rangle, \langle 4, 5, 1, \sqcup, R\rangle\}$

(This over-writes the blank (␣) in the middle with a 1 and removes a 1 from the right end of the second number.)

Definite Procedures (Algorithms)

The point: formalise the concept of definite procedure, algorithm, computation, effective process, effective calculation, ... — so that we can prove what properties it has

Basic idea is that a definite procedure is a sequence of operations which has the properties:

P1. It is finite.

P2. It is definite:

(a) It operates only on discrete, distinguishable states and symbols;

(b) Its operations are primitive

This is somewhat vague, whereas Turing computability is not.

Computability

Definition 5 (Computability)
A function f is computable iff there is a Turing machine which computes it.

Theorem 1 *There exist non-computable functions.*

Proof. See the Halting Problem. Q.E.D.

Second proof: $\aleph_0 < c$.

Definition 6 (Computability of Numbers)
A number x is computable iff there is a Turing machine which, given an index $i \in \mathbb{N}$ of the digital representation $\langle d_0, \ldots, d_i, \ldots \rangle$ of x, returns d_i.

Theorem 2 *All integers are computable.*

Proof. This is obvious. Q.E.D.

Theorem 3 *There exist non-computable numbers.*

Proof. See approx. Lecture 12. Q.E.D.

Second proof: Consider the decimal which is 0 followed by a decimal point followed by the concatenation of the output(s) from a non-computable function (as per Theorem 1).

Universality

As above, Turing machines can be described by strings listing their quintuples.

Definition 7 *A Universal Turing Machine (UTM) U is a Turing machine which*
$\forall M\ \exists D_M\ \forall x\ U(D_M x) = M(x)$.

(I.e., a UTM, U, can simulate any other Turing machine, M.)

Consequence: U takes input $\langle D_M, x \rangle$, where D_M describes a Turing machine M in the alphabet of U, and simulates its operation on input x.

Theorem 4 (Universality) *There is at least one UTM.*

See any text on computation theory for proof. The possibility of proof follows from the simplicity, definiteness and versatility of the definition of TMs.

Universality

Corollary 1 *There are infinitely many UTMs.*

This is trivial, given the theorem.

Exercise 1 *Demonstrate that your favourite programming language (C, C++, Java, Fortran, Python, perl, . . . Lisp, ...) is universal.*

How do you do that?

Equivalent Machines

We can try to generalise Turing machines in many ways.

Here is a simple generalisation: let the tape be 2-way infinite.

The set of 2-way computable functions is identical to the set of 1-way computable functions. To prove this (formally or informally), all we need do is show how to simulate any 2-way M' using a 1-way UTM M. Informally, we can simply describe a translation between tapes (where $d_1 \ldots d_m$ describes M'):

	−1	**0**	**1**	**···**		
···	⌴	**x1**	**···**	**xn**	⌴	**···**

0	**1**	**...**					**−1**	**−2**	
d1	**...**	**dm**	⌴	**x1**	**···**	**xn**	⌴		**...**

(Can you see another way of doing this?)

$\Rightarrow$ So, this is not a *real* generalisation

Equivalence

Other "ineffective" generalisations of Turing machines[a]:

- Multi-tape Turing machines
- Multi-headed Turing machines
- 2-dimensional tapes
- N-dimensional tapes
- Nondeterministic Turing machines

  (N.B.: **not** indeterministic machines! [b])

Ineffective here means: failing to extend the set of computable functions.

However, these variations may be very effective in *speeding up* some computations.

---

[a] ineffective in that they might possibly speed up some computations, but they do not extend the set of computable functions

[b] *nondeterministic* is taken to (effectively) mean "*parallel*", whereas *indeterministic* is taken to mean "*random*" or "*stochastic*"

Non-Deterministic Turing Machines (NDTMs)

Definition 8 *A nondeterministic TM (NDTM, NTM) is a Turing machine of Definition 1 (or Definition 3), with the difference that multiple distinct Q_i may share their first and third elements.*

In other words: an NDTM may have any finite number of distinct state transitions (and/or distinct outputs) given the same current state and input symbol.

How do you execute such multiplicities? In parallel, as a tree of multiple branches of execution. So,

- An NDTM accepts an input iff one of its parallel branches accepts it.
- You can describe an NDTM's output as an interleaving of n output tapes (where n is the number of branches).

Non-Deterministic Turing Machines (NTMs, NDTMs)

The last point already suggests how to simulate NDTMs with a basic TM; hence:

Theorem 5 *The set of functions computable with NDTMs is the same as the set of functions computable with TMs.*

Nondeterministic vs indeterministic
Some people think indeterministic machines are equivalent in this sense to TMs, because of this theorem. But indeterministic means some program step is (random or stochastic or) not determined, which is certainly not true of NDTMs.

- Indeterministic implies non-functional and so not a Turing machine
- Stochastic steps are not “primitive”

NDTMs are actually deterministic, parallel machines.

Church-Turing Thesis

Church-Turing Thesis: *A function is computable (effectively calcul[at]able, algorithmic, etc.) iff it is Turing computable (Turing, 1936-7).*

The following were all proved equivalent to Turing computability:

- General recursive functions (Kleene, 1936)
- λ-definability (Church, 1936)
- Finite combinatory processes (Post, 1936)

There seems to be a pattern there . . .

> Turing's computable functions [are] the result of a direct attempt to formulate mathematically the notion of effective calculability, while the other notions arose differently and were afterwards identified with effective calculability. (Kleene, 1952)

Church-Turing Thesis

Basic claim: Turing computability is all there is to computability.

Originally this was claimed not to be a thesis, but a *definition* of the intuitive concept of computability (e.g., Church — and also our Definition 5 above).

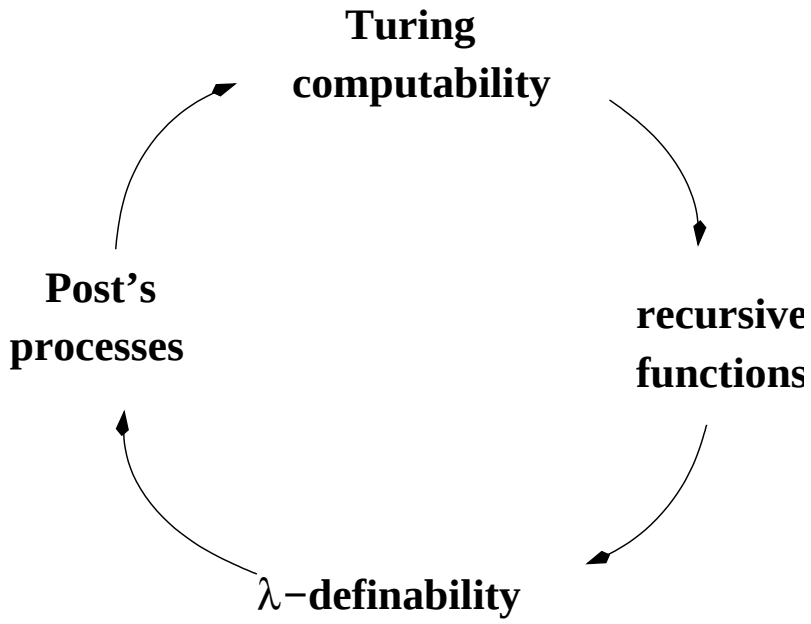
- This is a plausible view, now perhaps much less popular
- Consider: Science often progresses by replacing intuitive, vague ideas with precisely defined concepts
  E.g., heat (or temperature) vs. mean kinetic energy

Now, this is called a "Thesis", suggesting some kind of empirical content.

- There is some kind of evidence for it: all attempted definitions are proved equivalent.

Church-Turing Thesis

So:



Note: there is no interesting difference between the definition vs. thesis interpretation. The *real question* just takes alternative forms:

- Is the definition fruitful?
- Is the thesis true?

Church-Turing Thesis

How can we break out of the vicious circle of Turing computability?

Violations of the agreed, intuitive constraints on computation will do it . . .

Most examples I know of (which succeed in expanding the set of "computable" functions) somehow take advantage of infinity (and access to the non-computable) (rather than infinite memory or time [Turing machines already have those], but finite time/space uses of infinite time/space/complexity). Examples:

- My machine has a Halting function, which solves the Halting Problem in finite time for any machine and input.
- My machine has a Chaitin Ω function, which provides fixed-time arbitrary precision for the Halting probability.
- My machine has a function that returns the digits of the Cantor diagonal x_ω (Lecture 12 or thereabouts) to arbitrary precision in fixed time.
- My machine has the benefit of a non-computably connected 3-dimensional memory

These don't challenge - but rather illustrate - the thesis!

Church-Turing Thesis

How can we mount a challenge?

Ackermann's Function

How can we mount a challenge?

We can define arbitrarily complex functions; *ludicrously* complex functions. E.g., over the non-negative integer pairs:

Definition 9 (Ackermann's function)

$$\begin{aligned} \Psi(0,n) &= n+1 \\ \Psi(m+1,0) &= \Psi(m,1) \\ \Psi(m+1,n+1) &= \Psi(m,\Psi(m+1,n)) \end{aligned}$$

That doesn't look too bad; it's just nested recursion with the successor function! So, we can code it up so as to generate a table:

```
function ackermann (m,n)
  if (m=0) return (n+1);
  if (n-0) return ackermann(m-1,1)
     else return ackermann(m-1,ackermann(m,n-1));
```

Ackermann's function

Here are the first elements of the function, as computed by the code above:

Ψ	0	1	2	3	4	$\cdots$	n
0	1	2	3	4	5		
1	2	3	4	5	6		
2	3	5	7	9	11		
3	5	13	29	61	125		
4	13	"∞"					
:							
m							

where "∞" means: stack overflow!

For not so large m and n, $\Psi(m, n)$ is reported to require at least this many steps:

$$\underbrace{100^{100^{\cdots^{100}}}}_{n \text{ times}}$$

So, a googol = 10^{100} is puny[a], as is also
a googolplex = $10^{\text{a googol}} = 10^{10^{100}}$

[a] "google" is a misspelling (or something else).

Church-Turing Thesis

But surely mere *practical* non-computability doesn't challenge a thesis about *abstract, ideal* computability.

A real challenge has to produce a function which is **in principle** not Turing computable, and yet which somehow *ought* to be computable.

Let us make some more attempts:

- analogue computation [a],
- the (so-called) "*Busy Beaver*" function [b], and
- the elusive model paradox [c]

[a] some regard this as a genuine challenge to the Church-Turing thesis

[b] which I like and which is worth exploring

[c] originally noted in M. Scriven (1965), "An essential unpredictability in human behavior", B. B. Wolman and E. Nagel (eds.), in "Scientific Psychology: Principles and Approaches", pp411-425, Basic Books (Perseus Books). It was then partly resolved (without mention of Turing machines) in D. K. Lewis & J. Shelby-Richardson (1966), "Scriven on Human Unpredictability", Philosophical Studies: An International Journal for Philosophy in the Analytic Tradition, vol. 17, no. 5, Oct 1966. It was then independently re-discovered and (later) re-published in 2008 (in terms of Turing machines), when it was given the name "*elusive model paradox*".

Analogue Computation

Whereas digital computations are done by digital computers as we know them, *analogue* computations are done by physical devices.

(Analogy with [old] analogue cameras and [more modern] digital cameras.)

An analogue computation might be, e.g., to heat some water to a certain temperature, measure how much (how high) it expands, and then say that we are computing the output (height) as a function of input (temperature).

Not repeatable, (to me) unconvincing.
[But I don't know everything. What do you think?]

So, let's try at least one more attempt.

Busy Beaver Function

(Due to Rado.)

Given a TM M with a blank input tape (and fixed alphabet), define its "*productivity*" as:

$$\Sigma(M) =_{df} \begin{cases} \#1\text{s on tape} & \text{if halting in config } 111\cdots 1 \overset{\sqcup}{\triangle} \\ 0 & \text{otherwise} \end{cases}$$

What's the maximal productivity, $\Sigma(M)$, of an n-state Turing machine - over all TMs with n states?

> Any n-state M with maximal productivity is called a Busy Beaver.

Clearly, there's an answer to the question in principle $\forall n \geq 1$ since:

$$|\{M : M \text{ is an } n\text{−state TM}\}| < \aleph_0$$

So, for all n, let's call the answer to this question $\Sigma(n)$.

> In other words, there is no n-state TM with greater productivity than $\Sigma(n)$.

Busy Beaver Function

But what is $\Sigma(n)$? Contests have been run, proofs generated. So far, we have:

n	$\Sigma(n)$
1	1
2	4
3	6
4	13
5	≥ 4098
6	$\geq 1.2 \times 10^{865}$

The fifth is from an example machine; the others are via proofs.

So, not much is known about this function, except that it is another non-computable (function).

Busy Beaver Function

Theorem 6 $\Sigma(n)$ *is not computable.*

Proof by Reductio Ad Absurdum (or Reduction to the Absurd, or Contradiction)
Step one of reductio proofs: suppose $\Sigma(n)$ is computable. Then there is a TM which computes it; call it B.

> I.e., $B(n) = \Sigma(n)$.

We'll use the following additional machines for the proof:

- $I(n) = n + 1$ (increment)
- $D(n) = n + n = 2n$ (double)
- $W_n() = n$ (write n 1s)

Note that the W_n's have the property that: $|W_n| = n$

> Why? By construction: the first $(n - 1)$ states each write 1, move R, change to the next state; the n^{th} state writes 1 and halts.

Busy Beaver Function

Proof by Reductio (cont'd).
Consider the composite machine:
$IBD(n) = I(B(D(n)))$

- This machine computes $\Sigma(2n) + 1$
- It has some finite number of states; call that number n'

So, now we can construct the machine: $IBDW_{n'}$

- This computes $\Sigma(2n') + 1$ given a blank input
- By construction,
  $|IBDW_{n'}| = |IBD| + |W_{n'}| = n' + n' = 2n'$

So, this is a machine with $2n'$ states and "productivity" more than $\Sigma(2n')$, which is a contradiction. Hence, there is no such machine B. (That is, we can't calculate the *Busy Beaver* function for arbitrary n.) Q.E.D.

Elusive model paradox

Our last attempt[a] at challenging the Church-Turing thesis[b] will be the *elusive model* paradox

We have one TM, M_1, generating a sequence of numbers, $M_1(1), M_1(2), M_1(3), \ldots$.

We have another TM, M_2, which, having seen, $M_1(1), M_1(2), ..., M_1(i)$, tries to infer $M_1(i+1)$. At each stage, call this guess $M_2(i+1)$.

Because M_1 is a TM, M_2 should eventually lock in after some point and get all the remaining values right.

But, because M_2 is a TM, M_1 should (eventually) be able to guess what M_2 will guess and then make sure to set $M_1(i+1) = M_2(i+1) + 1$.

Because we can't have (after some point or any point) *both* $M_1(i+1) = M_2(i+1)$ *and* $M_1(i+1) = M_2(i+1) + 1$, this gives a contradiction.

Resolution?

---

[a] in this subject

[b] if this really is a challenge to the Church-Turing thesis

References

A. Church (1936). An unsolvable problem of elementary number theory. *Amer Jrn Math, 58,* 345-63.

A.K. Dewdney (1989). *The Turing Omnibus.* Computer Science Press.

D. L. Dowe (2008b). Minimum Message Length and statistically consistent invariant (objective?) Bayesian probabilistic inference - from (medical) "evidence". *Social Epistemology*, Vol. *22*, No. 4, pp433-460. {See p 455.}

S. Kleene (1936). General recursive functions of natural numbers. *Math Annalen, 112,* 727-742.

S. Kleene (1952). *Introduction to Metamathematics.* North-Holland.

D. K. Lewis & J. Shelby-Richardson (1966), "Scriven on Human Unpredictability", Philosophical Studies: An International Journal for Philosophy in the Analytic Tradition, vol. 17, no. 5, Oct 1966, pp69 - 74.

H. Lewis and C. Papadimitriou *Elements of the Theory of Computation*, 2nd ed. (1998), Prentice-Hall

E. Post (1936). Finite combinatory processes – formulation I. *Jrn Symbolic Logic, 1,* 103-5.

M. Scriven (1965), "An essential unpredictability in human behavior", B. B. Wolman and E. Nagel (eds.), in "Scientific Psychology: Principles and Approaches", pp411-425, Basic Books (Perseus Books).

A. M. Turing (1936-7). On computable numbers, with an application to the Entscheidungsproblem. *Proc London Math Soc, 42,* 230-65 and vol. *43*, 544-6.