Lecture 2

Friday,
17 JAN 2014

- Develop what is meant by an algorithm

  fundamental model is the Turing machine
  (idealized computer which has a very simple set of instructions & an idealized unbounded memory)
  can be used to execute any algorithm whatsoever.
  many other models of computation are equivalent to this one

Fundamental Question: What resources are required to perform a given computational task?

1) Figure out which computational tasks are possible (give algorithms

2) Find limitations (i.e., lower bounds on the necessary resources)

Why study computer science?

1) we can reuse ideas developed here in order to develop the theory of computation

2) Much is already understood in the classical theory. It is good to know what resources are needed to solve certain tasks in order to see if there is a gap ~~between~~ what is possible classically & quantumly. For example, w/ factoring best known classical algorithm is exponential in the # of bits needed to represent the integer, while q. algorithm is polynomial.

3) need to learn how to think like a computer scientist.

Two models of computation:

1) Turing machine model (fundamental one)

2) circuit ~~complexity~~ model (more related to easiest q. model)

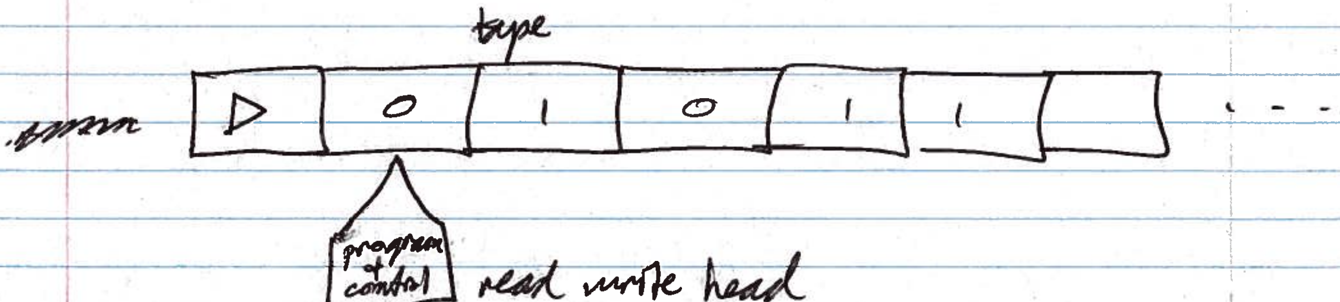
Background on Turing machine idea:

At the turn of the 1900s, Hilbert asked whether or not there existed an algorithm to solve all the problems of mathematics. He expected that there would be, but Church & Turing showed that there is not. (We will focus on more intuitive Turing's approach.)

Turing remarkably at age 23 in 1936

What does it mean to compute?

A Turing machine ~~M consists~~



components:

a) program – list of instructions

b) finite state control – like a processor

c) tape – like a memory

d) read-write tape-head – points to current position on

finite state control consists of
a finite set of states $q_1, \dots, q_m$
where m is some constant (should be large enough but can be a constant)

two special states q_s (starting state)
& q_h (halting state)

this is reached if the computation finishes.

Symbols on tape are chosen from an alphabet Γ - usually take
$\Gamma = \{\triangleright, b, 0, 1\}$

starting point

blank

Formally, a Turing machine M is a triple (Γ, Q, δ)

Γ is alphabet for tape
Q is the set of states for control register
δ is the transition function (program)

δ will consist of

1) read the symbol in the cell under the head
2) replace the current symbol on the tape w/ a new symbol.
3) change the state of the control register
4) move head to the right or left or don't move.

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, S, R\}$$

"if current state is $q \in Q$ + tape symbol is $\gamma \in \Gamma$, then next state is $q' \in Q$, tape symbol becomes $\gamma' \in \Gamma$, + move left, right, or stay."

Very simple example:

1: $\langle q_s, \triangleright, q_1, \triangleright, +1 \rangle$
2: $\langle q_1, 0, q_1, b, +1 \rangle$
3: $\langle q_1, 1, q_1, b, +1 \rangle$
4: $\langle q_1, b, q_2, b, -1 \rangle$
5: $\langle q_2, b, q_2, b, -1 \rangle$
6: $\langle q_2, \triangleright, q_3, \triangleright, +1 \rangle$
7: $\langle q_3, b, q_h, 1, 0 \rangle$

just outputs a constant function!

Another beginner example

Palindrome detector: detect if a string
is its mirror image.

can do many, many functions
(including everything that your computer can do)
We say that a function is computable
if there is a Turing machine that
will do so in a finite amount of time
(however no bound on efficiency).

<u>Church-Turing thesis:</u> Class of functions
computable by a Turing machine corresponds
exactly to the class of functions which we
would naturally regard as being computable
by an algorithm.

asserts an equivalence between rigorous mathematical
formalism & intuitive concept of an algorithm.

there is a lot of evidence in favor of
this, also we have not observed a violation
of this [Physical CT thesis: falsifiable Any physically
plausible computing device can be
simulated by a TM. Can nature be

this says nothing about efficiency

also q. computers compute the same class of functions.

Many variations of Turing machine model:
multiple tapes, 2-D tapes,

these changes don't change the class of computable functions essentially b/c one Turing machine can simulate another.

In fact, we can have a universal Turing machine. That is, Turing machines have three items, which uniquely identify it: (Γ, Q, δ)

~~1) initial configuration of tape~~

Since we can write this down, there is an encoding of every Turing machine as a sequence of zeros & ones

(i.e., we can write down programs as data)

leads to notion of general purpose electronic computers which can run arbitrary programs

If M is a Turing machine, then we denote its description by $\langle M \rangle$.

Turing observed that we can have a universal Turing machine

if M is a TM w/ desc. $\langle M \rangle$ then we feed input x & $\langle M \rangle$ to UTM & it outputs $M(x)$,

perhaps obvious today but somewhat counterintuitive on a first encounter,

b/c parameters for a given universal Turing machine are fixed, but those for the one to be simulated could be larger. The way around this is that we can use encodings.

The UTM can represent the other machine's state and transition table & follow the computation along step by step.

The universal Turing machine then leads to a resolution of Hilbert's question & ~~[illegible]~~ ~~[illegible] a critical part of~~ foreshadows an important topic in TCS

Church & Turing showed that the answer to Hilbert's question is "no." How?

Often, it is helpful for us to work w/ decision problems, i.e.,
(also known as languages)

Let $\{0,1\}^*$ denote the set of all binary strings. We can partition them as A_{yes} & A_{no} where

$$A_{yes} \cap A_{no} = \phi \quad \&$$

$$A_{yes} \cup A_{no} = \{0,1\}^*$$

goal is to decide if a given input is in A_{yes} or A_{no}. For example, input could be an encoding of an integer & task would be to decide primality

Then the halting problem is to compute the following function:

$$h(\langle M\rangle, x) = \begin{cases} 0 & \text{if TM } \langle M\rangle \text{ doesn't halt upon input of } x \\ 1 & \text{" " " halts " " "} \end{cases}$$

well posed mathematical question

Is there an algorithm to solve it?

As prep, consider the barber paradox

Suppose there is a town w/ one barber.
Every man in the town keeps clean shaven & does so by doing exactly one of two things:

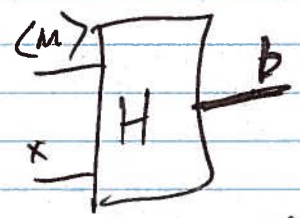
1) shaves himself
2) goes to the barber

What about the barber?

if he does 1) then he does 2),
" " " 2) " " 1).
contradiction, system cannot exist in the 1st place

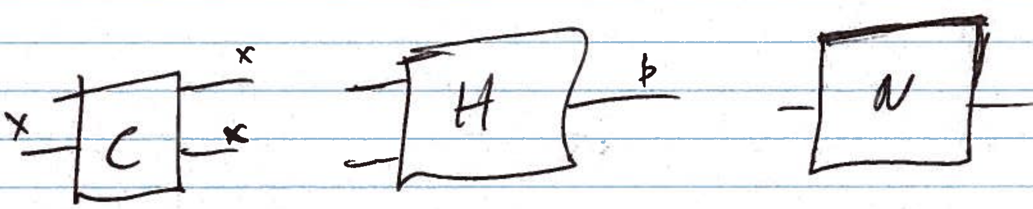
Suppose ∃ a machine H that perfectly solves the halting problem.

That is, given any desc. $\langle M \rangle$ of a Turing machine M & any input x it decides whether or not M halts on x.



Show that this cannot exist.

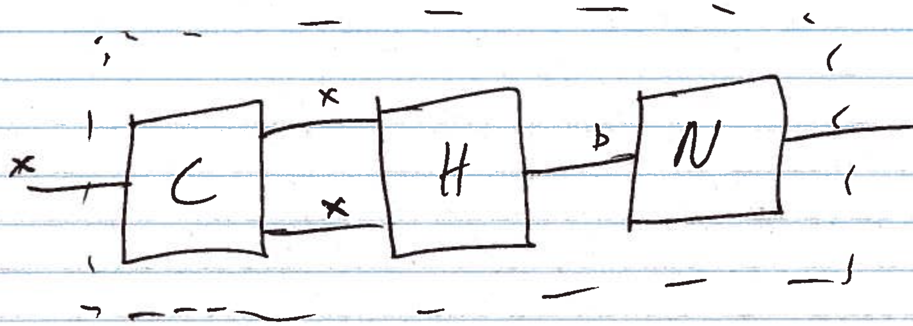
Construct a Turing machine Q consisting of three parts:



C just copies its input

H is as before

& N is the negator: if input is "not stuck", then it gets stuck
if input is "stuck", then it does not get stuck



Call the whole thing Q

it has a desc. <Q>

so we feed in <Q>

so H has to decide if Q gets stuck when a description of itself is fed in.

Suppose it outputs "not stuck!"
then this gets fed into N, which forces the machine to get stuck.
So H is wrong in this case,

Suppose it outputs "stuck",
then N does not get stuck &
again H is wrong

⇒ halting problem is undecidable

Two concepts important for us

1) can divide all problems into those that are computable & those that are not.

2) Reduction: We can show that other problems are hard by supposing that we have an algorithm for them & if we did, then we could solve the halting problem, I.e., there is a mapping of halting problem to another one

Complexity theory is a "scaled down" version ~~Another unde~~ of computability theory in which we add the quantifiers "polynomial time" in a # of places

Another undecidable problem:

Matrix mortality problem: Given a finite set of integer valued $n \times n$ matrices decide whether they can be multiplied together