CS 758/858: Algorithms

http://www.cs.unh.edu/~ruml/cs758

Turing Machines

What is ‘information processing’?

Take some input, process it, render some output.

Would like an abstract model for this, independent of realization.

No homunculi! ‘Process’ steps must be clear and unambiguous.

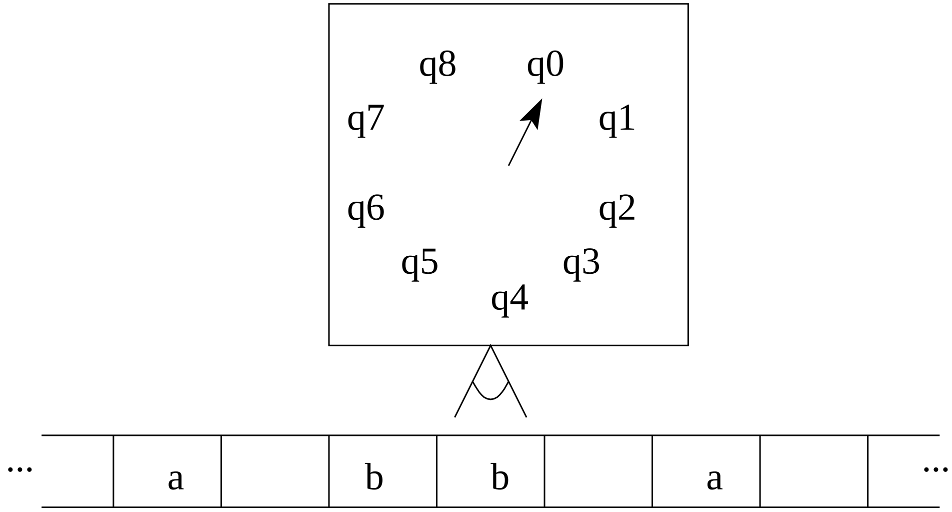
Modeling of Computing

- finite-state machine: regular langauges
- pushdown automaton: context-free languages
- Turing machine: computable languages

Alan Mathison Turing (1912-1954)

A *Turing machine* has:

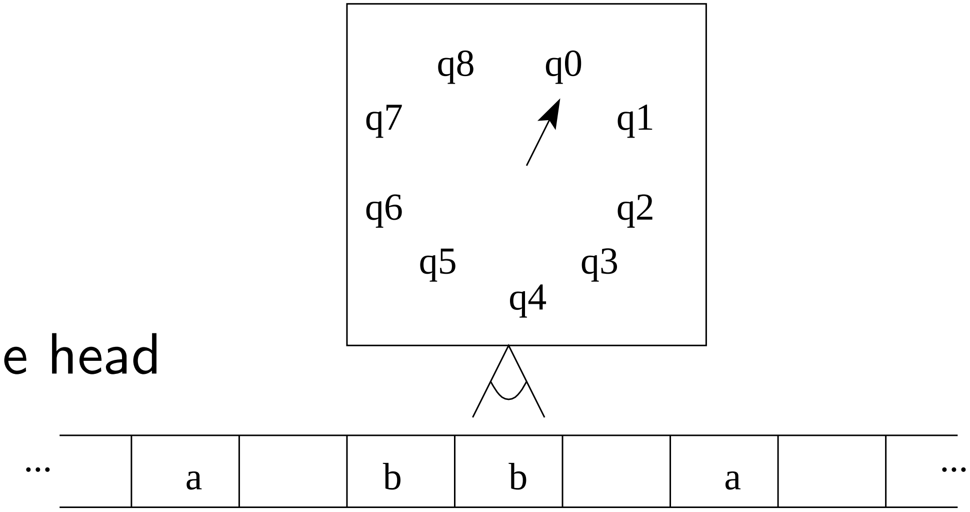
- a **processor** that can be in one of a finite number of states
- an **infinite tape** of symbols (from finite alphabet)
- a **head** that reads and writes the tape, one symbol at a time



The set up

A *Turing machine* has:

- a **processor** that can be in one of a finite number of states
- an **infinite tape** of symbols (from finite alphabet)
- a **head** that reads and writes the tape, one symbol at a time



The processor looks at

1. the symbol under the head
2. its current state

and then

3. writes a symbol (could be same as old)
4. moves the head left, right, or stays still
5. puts itself in a next state (could be same as old)

In summary

A Turing machine is:

1. a finite alphabet of possible tape symbols (including $\square$)
2. an infinite tape of symbols (initially $\square$, except for input)
3. a starting head position
4. a finite set of possible processor states
5. a starting processor state
6. a set of 'final' processor states that are 'accept' or 'reject'
7. a set of transition rules for the processor

One of the first (and still most popular) abstract models of computation.

Extensions

- tape infinite in only one direction
- multiple tapes at once
- multiple heads at once
- 2-D "tape"

All polytime related!

Church-Turing Thesis

Any 'effective computing procedure' can be represented as a Turing machine.

Other models

equivalent to Turing machines (compute time may vary):

- Post rewriting systems (grammars)
- recursive functions
- λ calculus
- parallel computers
- cellular automata
- certain artificial neural networks (most are weaker)
- quantum computers

There must be something substantive about this!

Universal machines

Can represent Turing machine as a table

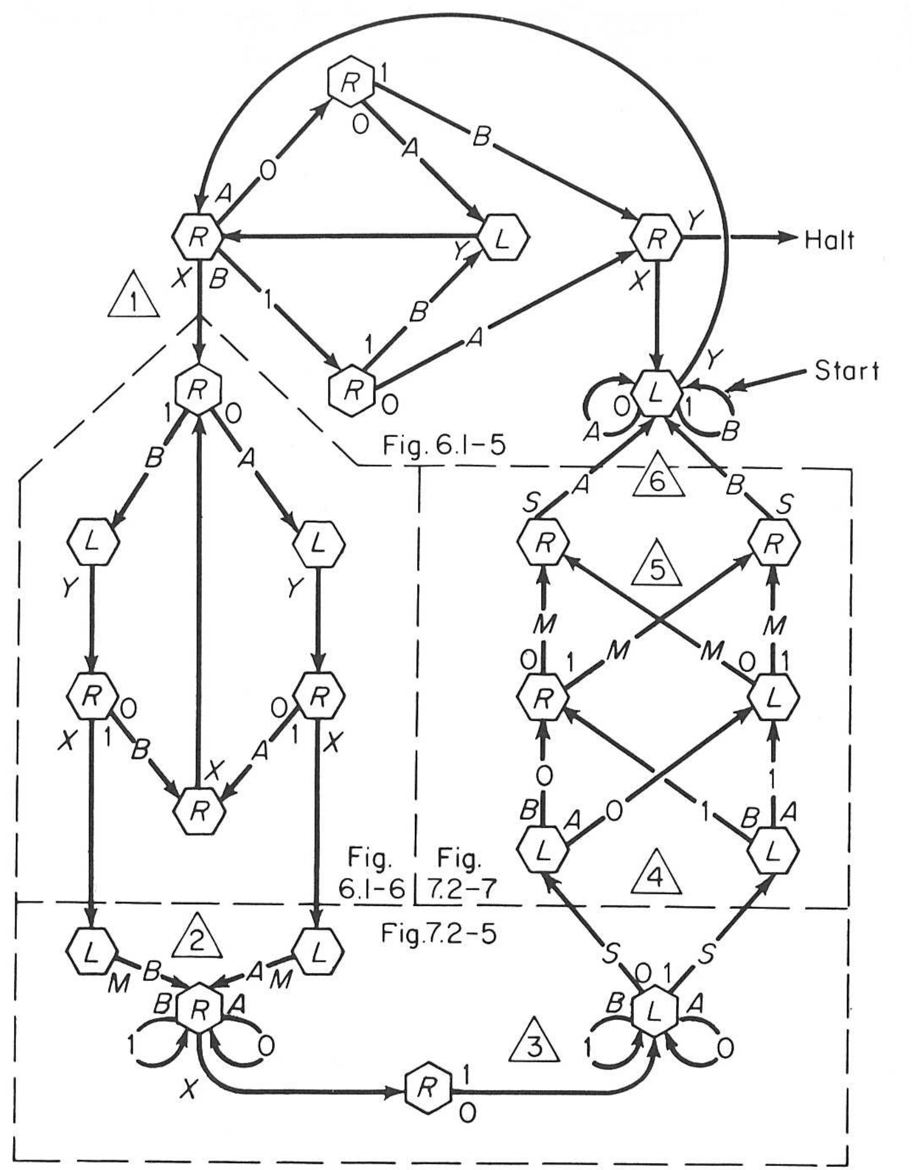
$$\begin{array}{c} \text{state, symbol} \rightarrow \text{symbol, action, state} \\ \text{state, symbol} \rightarrow \text{symbol, action, state} \\ \vdots \end{array}$$

Can write the table on an input tape

Universal machine: input is machine and machine's input

'Stored program' computation

Minsky's universal machine



Undecidability

Turing Machine Languages

M **accepts** L = M **recognizes** L

- M enters accepting state (as opposed to reject state or not halting)
- $\Rightarrow$ L is Turing-recognizable
- 'recursively-enumerable' languages

M **decides** L

- M always eventually halts (either accepting or rejecting)
- $\Rightarrow$ L is Turing-decidable
- 'recursive' languages, more restricted

Software Verification: The Halting/Accepting Problem

$L_H = \{\langle M, w \rangle : M$ is a TM and M accepts $w\}$

Software Verification: The Halting/Accepting Problem

$L_H = \{\langle M, w\rangle : M$ is a TM and M accepts $w\}$

deciding L_H: halt with Y or N
accepting L_H: halt with Y or either halt with N or run forever

Any universal machine can accept L_H.
But can a machine decide it?

Proof (1/2)

$L_H = \{\langle M, w\rangle : M$ is a TM and M accepts $w\}$

Assume $\exists M_H$ that decides L_H.
So, $M_H(\langle M, w\rangle) \mapsto$ accept iff M accepts w, reject otherwise

Reminder:

deciding L_H**:** halt with Y or N
accepting L_H**:** halt with Y or either halt with N or run forever

$L_H = \{\langle M, w\rangle : M$ is a TM and M accepts $w\}$

Assume $\exists M_H$ that decides L_H.
So, $M_H(\langle M, w\rangle) \mapsto$ accept iff M accepts w, reject otherwise

Simplification 1: $M_{SH}(\langle M\rangle) \mapsto$ accept iff M accepts M,
reject otherwise
Simplification 2: $M_{ISH}(\langle M\rangle) \mapsto$ reject iff M accepts M,
accept otherwise

Can such a machine M_{ISH} exist?
It must exist if M_H can exist!

Reminder:

deciding L_H**:** halt with Y or N
accepting L_H**:** halt with Y or either halt with N or run forever

Proof (2/2)

$L_H = \{\langle M, w\rangle : M$ is a TM and M accepts $w\}$

Assume $\exists M_H$ that decides L_H.
$M_{ISH}(\langle M\rangle) \mapsto$ reject iff M accepts M, accept otherwise

Reminder:

deciding L_H**:** halt with Y or N
accepting L_H**:** halt with Y or either halt with N or run forever

Proof (2/2)

$L_H = \{\langle M, w \rangle : M \text{ is a TM and } M \text{ accepts } w\}$

Assume $\exists M_H$ that decides L_H.
$M_{ISH}(\langle M \rangle) \mapsto$ reject iff M accepts M, accept otherwise

run M_{ISH} on itself!

$M_{ISH}(\langle M_{ISH} \rangle) \mapsto$ reject iff M_{ISH} accepts M_{ISH},
otherwise accept (M_{ISH} rejects M_{ISH})

Reminder:

deciding L_H**:** halt with Y or N
accepting L_H**:** halt with Y or either halt with N or run forever

Proof (2/2)

$L_H = \{\langle M, w\rangle : M$ is a TM and M accepts $w\}$

Assume $\exists M_H$ that decides L_H.
$M_{ISH}(\langle M\rangle) \mapsto$ reject iff M accepts M, accept otherwise

run M_{ISH} on itself!

$M_{ISH}(\langle M_{ISH}\rangle) \mapsto$ reject iff M_{ISH} accepts M_{ISH},
otherwise accept (M_{ISH} rejects M_{ISH})

Contradiction! M_{ISH} and therefore M_H cannot exist.
L_H is undecidable.

Reminder:

deciding L_H**:** halt with Y or N
accepting L_H**:** halt with Y or either halt with N or run forever

Summary

$L_H = \{\langle M, w\rangle : M$ is a TM and M accepts $w\}$

Assume $\exists M_H$ that decides L_H.
M_H accepts $\langle M, w\rangle$ iff M accepts w

Summary

$L_H = \{\langle M, w \rangle : M$ is a TM and M accepts $w\}$

Assume $\exists M_H$ that decides L_H.
M_H accepts $\langle M, w \rangle$ iff M accepts w
M_{SH} accepts $\langle M \rangle$ iff M accepts M

$L_H = \{\langle M, w\rangle : M$ is a TM and M accepts $w\}$

Assume $\exists M_H$ that decides L_H.
M_H accepts $\langle M, w\rangle$ iff M accepts w
M_{SH} accepts $\langle M\rangle$ iff M accepts M
M_{ISH} rejects $\langle M\rangle$ iff M accepts M

Summary

$L_H = \{\langle M, w\rangle : M$ is a TM and M accepts $w\}$

Assume $\exists M_H$ that decides L_H.

M_H accepts $\langle M, w\rangle$ iff M accepts w

M_{SH} accepts $\langle M\rangle$ iff M accepts M

M_{ISH} rejects $\langle M\rangle$ iff M accepts M

M_{ISH} rejects $\langle M_{ISH}\rangle$ iff M_{ISH} accepts M_{ISH} — Yikes!

Summary

$L_H = \{\langle M, w\rangle : M$ is a TM and M accepts $w\}$

Assume $\exists M_H$ that decides L_H.
M_H accepts $\langle M, w\rangle$ iff M accepts w
M_{SH} accepts $\langle M\rangle$ iff M accepts M
M_{ISH} rejects $\langle M\rangle$ iff M accepts M
M_{ISH} rejects $\langle M_{ISH}\rangle$ iff M_{ISH} accepts M_{ISH} — Yikes!

No Turing machine can tell if another halts.

By Church-Turing, no algorithm for the halting problem exists.

There are problems for which no algorithm can exist.

Break

- asst 12
- asst 13

Rice's Theorem

The function computed by a Turing machine is the mapping from its input (string of symbols initially on the tape) to its output (string of symbols on its tape when it halts)

Theorem: Any non-trivial property of the function computed by a Turing machine is undecidable.

Therefore, we cannot decide anything 'non-trivial' about the function computed by a Turing machine.

Henry Gordon Rice, Professor of Math at UNH in the 1950s!

Proof Sketch

Example: does a given TM compute the `add 1` function?

Assume machine *isAdd1()* can decide whether or not its input is a Turing machine that computes the `add 1` function.

Now, given M and input x, we can decide if $M(x)$ halts:

- Make a temporary machine $T(i) = \{M(x); \text{return } i + 1\}$
- Now, test if T satisfies the *isAdd1* property: $\mathit{isAdd1}(T)$

Can now decide the halting problem:

- If $M(x)$ halted, then $\mathit{isAdd1}(T)$ says "Yes" because $T(i)$ computed $i + 1$
- If $M(x)$ never halts, then $T(i)$ never halts and $\mathit{isAdd1}(T)$ must say "No"

So *IsAdd1()* cannot exist.

Summary

Turing machines

- model what we mean by computation, independent of hardware
- are not something you want to program much yourself
- seem to be able to express any algorithm
- provide an example of stored-program interpretation
- illustrate limits on what can be computed
- provide the foundation for computational complexity

Coping with NP-Completeness

- find tractable special case
- run only on small inputs
- heuristic optimal algorithm that's usually fast
- heuristic non-optimal algorithm that's always fast
  - if bounded suboptimality: 'approximation algorithm'

EOLQs

For example:

- What's still confusing?
- What question didn't you get to ask today?
- What would you like to hear more about?

Please write down your most pressing question about algorithms and put it in the box on your way out.
Thanks!