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.

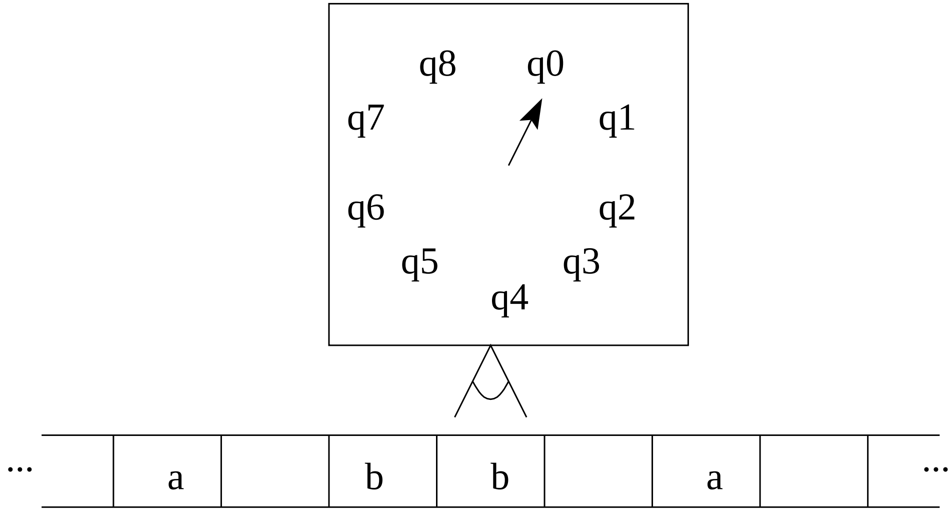
Modeling of Computing

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

Alan Mathison Turing (1912-1954)

A *Turing machine* has:

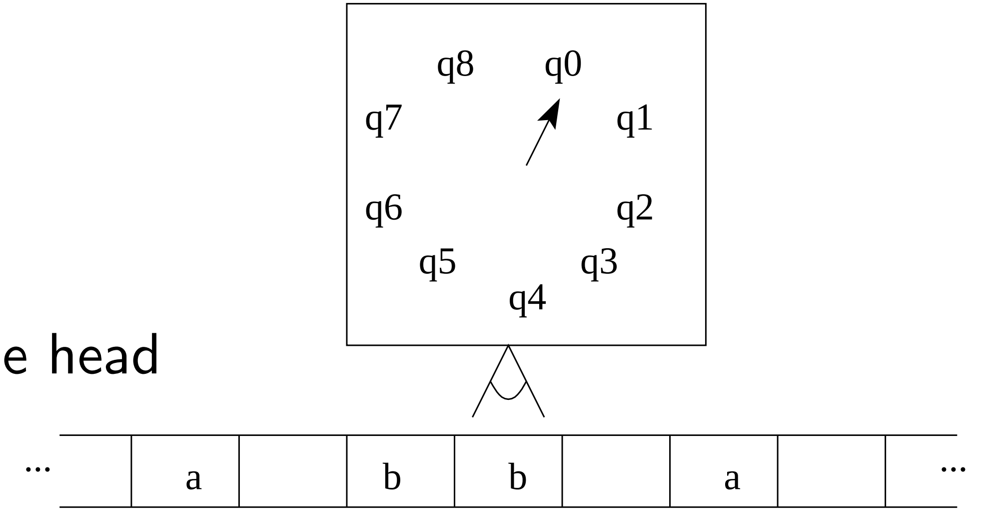
- a **processor** that can be in one of a finite number of states
- an **infinite tape** of symbols (from finite alphabet)
- a **head** that reads and writes the tape, one symbol at a time



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
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

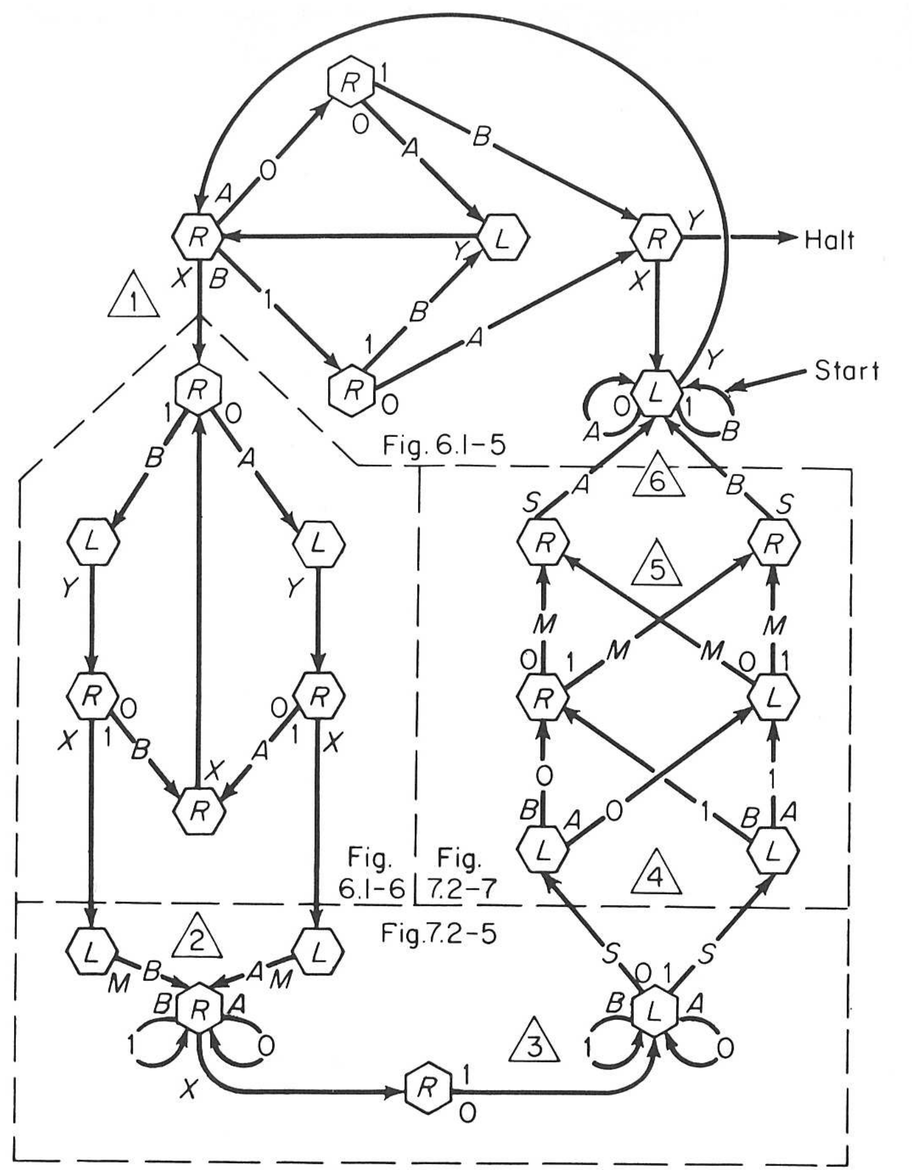
$$\begin{array}{c} \text{state, symbol} \rightarrow \text{symbol, action, state} \\ \text{state, symbol} \rightarrow \text{symbol, action, state} \\ \vdots \end{array}$$

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

The halting problem

H: given M and its input i, does M halt on i?

The halting problem

H: given M and its input i, does M halt on i?

deciding H: output Y or N
accepting H: halting $(= Y)$ or computing forever $(= N)$

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

A simpler problem

H: given M and its input i, does M halt on i?

SH: given M, does M halt on its own encoding?

But can a machine decide this simpler problem?

Reminder:

deciding H: output Y or N
accepting H: halting $(= Y)$ or computing forever $(= N)$

A simpler problem

H: given M and its input i, does M halt on i?

SH: given M, does M halt on its own encoding?

ISH: given M, does M **not** halt on its own encoding?

'Can a machine decide SH?' is fundamentally the same as
'Can a machine decide ISH?' which is easier than
'Can a machine accept ISH?'

Reminder:

deciding H: output Y or N
accepting H: halting $(= Y)$ or computing forever $(= N)$

A paradox

H: given M and its input i, does M halt on i?

SH: given M, does M halt on its own encoding?

ISH: given M, does M **not** halt on its own encoding?

Let's assume we have a machine S that accepts ISH.

What happens when S is given itself as input? Does it halt?

Reminder:

deciding H: output Y or N
accepting H: halting $(= Y)$ or computing forever $(= N)$

A paradox

H: given M and its input i, does M halt on i?

SH: given M, does M halt on its own encoding?

ISH: given M, does M **not** halt on its own encoding?

Let's assume we have a machine S that accepts ISH.

What happens when S is given itself as input? Does it halt?

If S halts on S, the definition of ISH means S doesn't halt on S.
If S doesn't halt on S, that means that S does halt on S.

Reminder:

deciding H: output Y or N
accepting H: halting $(= Y)$ or computing forever $(= N)$

A paradox

H: given M and its input i, does M halt on i?

SH: given M, does M halt on its own encoding?

ISH: given M, does M **not** halt on its own encoding?

Let's assume we have a machine S that accepts ISH.

What happens when S is given itself as input? Does it halt?

If S halts on S, the definition of ISH means S doesn't halt on S.
If S doesn't halt on S, that means that S does halt on S.

Contradiction!

Reminder:

deciding H: output Y or N
accepting H: halting $(= Y)$ or computing forever $(= N)$

Implication: undecidability

Assuming we have a machine S that accepts ISH leads to a contradiction.

So no such S can exist.

ISH is 'not Turing-acceptable.'
Thus certainly not decidable.

Implication: undecidability

Assuming we have a machine S that accepts ISH leads to a contradiction.

So no such S can exist.

ISH is 'not Turing-acceptable.'
Thus certainly not decidable.

SH is undecidable. (Otherwise we could decide ISH.)

Implication: undecidability

Assuming we have a machine S that accepts ISH leads to a contradiction.

So no such S can exist.

ISH is 'not Turing-acceptable.'
Thus certainly not decidable.

SH is undecidable. (Otherwise we could decide ISH.)

H is harder and thus certainly undecidable.

Implication: undecidability

Assuming we have a machine S that accepts ISH leads to a contradiction.

So no such S can exist.

ISH is 'not Turing-acceptable.'
Thus certainly not decidable.

SH is undecidable. (Otherwise we could decide ISH.)

H is harder and thus certainly undecidable.

No Turing machine can compute H.

By Church-Turing, no procedure for H exists in any medium.

There are problems for which no algorithm can exist.

Break

- asst 12
- asst 13
- 'swarm' algorithms: metaheuristics or robots?

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!