10b. Models of Computation

6.004x Computation Structures
Part 2 – Computer Architecture

Universality?

- Recall: We say a set of Boolean gates is universal if we can implement any Boolean function using only gates from that set.

- What problems can we solve with a von Neumann computer? (e.g., the Beta)
  - Everything that FSMs can solve?
  - Every problem?
  - Does it depend on the ISA?

- Needed: a mathematical model of computation
  - Prove what can be computed, what can't

Models of Computation

The roots of computer science stem from the evaluation of many alternative mathematical "models" of computation to determine the classes of computations each could represent.

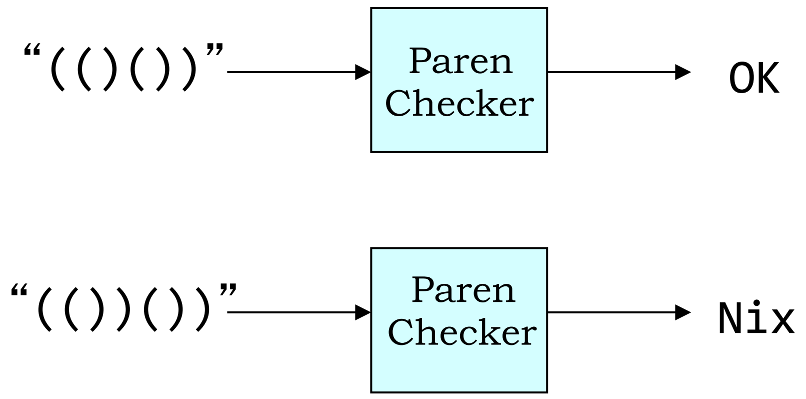
An elusive goal was to find a universal model, capable of representing *all* practical computations...

- switches
- gates
- combinational logic
- memories
- FSMs

Are FSMs the ultimate digital computing device?

FSM Limitations

Despite their usefulness and flexibility, there are common problems that cannot be solved by any FSM. For instance:



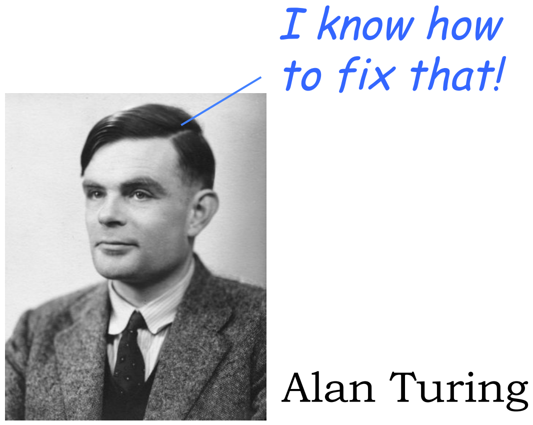
Well-formed Parentheses Checker:

Given any string of coded left & right parens, outputs 1 if it is balanced, else 0.

Simple, easy to describe.

Can this problem be solved using an FSM??? NO!

PROBLEM: Requires *arbitrarily* many states, depending on input. Must "COUNT" unmatched left parens. An FSM can only keep track of a finite number of unmatched parens: for every FSM, we can find a string it can't check.



Alan Turing

Turing Machines

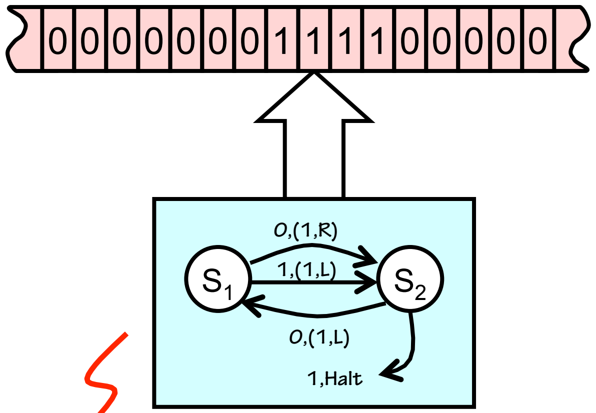
Alan Turing was one of a group of researchers studying alternative models of computation.

He proposed a conceptual model consisting of an FSM combined with an infinite digital tape that could be read and written at each step.

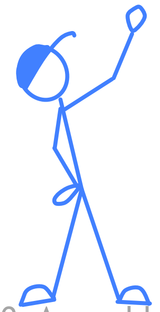
- encode input as symbols on tape
- FSM reads tape/writes symbols/ changes state until it halts
- Answer encoded on tape

Turing's model (like others of the time) solves the "FINITE" problem of FSMs.

Bounded tape configuration can be expressed as a (large!) integer



FSMs can be enumerated and given a (very large) integer index.

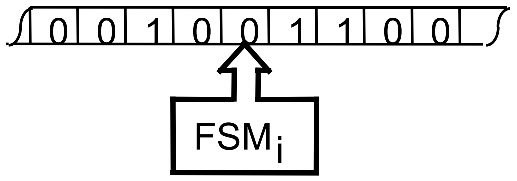


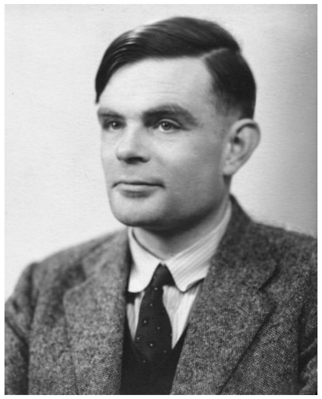
We can talk about TM 347 running on input 51, producing an answer of 42.
TMs as integer functions:

$$y = TM_I[x]$$

Other Models of Computation…

Turing Machines [Turing]





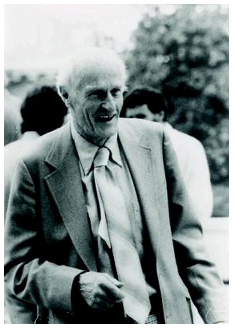
Alan Turing

Recursive Functions [Kleene]

$F(0,x) \equiv x$

$F(1+y,x) \equiv 1+F(x,y)$

```
(define (fact n)
  (... (fact (- n 1))
```



Stephen Kleene

Lambda calculus [Church, Curry, Rosser...]

Alonzo Church

$\lambda x.\lambda y.xxy$

```
(lambda(x)(lambda(y)(x (x y))))
```

Production Systems [Post, Markov]

Emile Post

$\alpha \rightarrow \beta$

```
IF pulse=0 THEN
   patient=dead
```

Computability

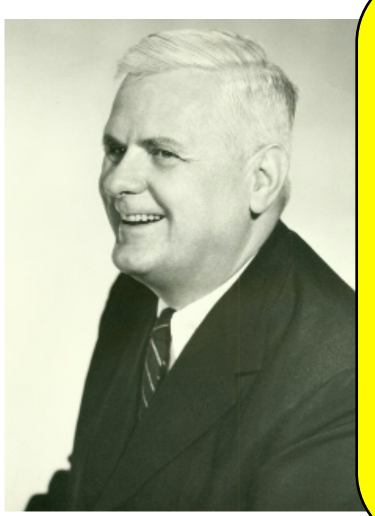
FACT: Each model studied is capable of computing exactly the same set of integer functions!

Proof Technique:

Constructions that translate between models

BIG IDEA:

Computability, independent of computation scheme chosen



Church's Thesis:

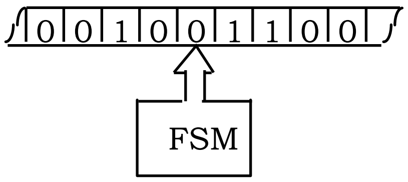
Every discrete function computable by ANY realizable machine is computable by some Turing machine.

$$f(x)\ \textit{computable} \Leftrightarrow \text{for some } k, \text{ all } x \quad f(x) = T_k[x]$$

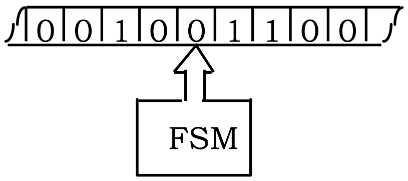
meanwhile…

Turing machines Galore!

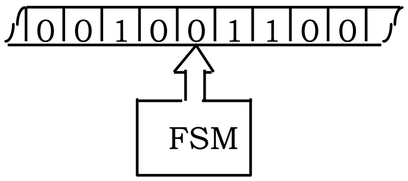
"special-purpose" Turing Machines….



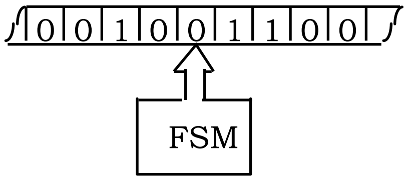
Factorization



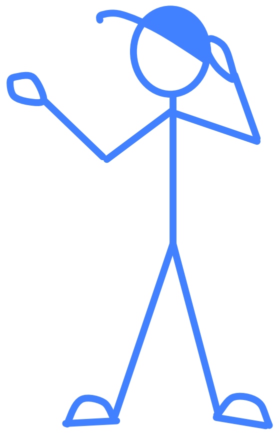
Primality Test



Multiplication



Sorting

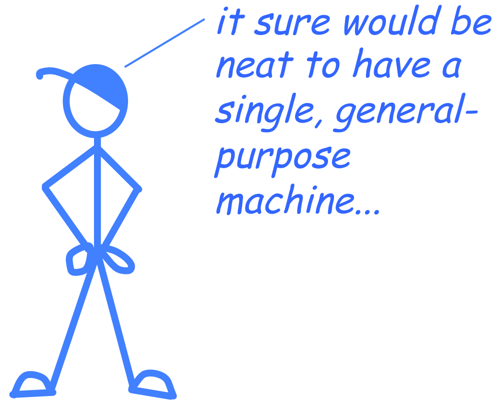


Is there an alternative to infinitely many ad-hoc Turing Machines?

The Universal Function

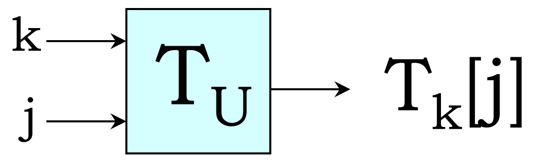
Here's an interesting function to explore: the Universal function, U, defined by

$$\boxed{U(k, j) = T_k[j]}$$


it sure would be neat to have a single, general-purpose machine...

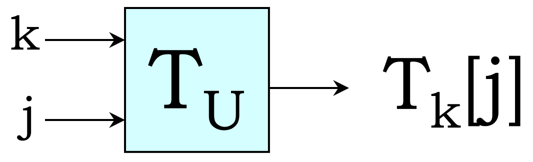
Could this be computable???

SURPRISE! U is computable by a Turing Machine:



In fact, there are infinitely many such machines. Each is capable of performing *any* computation that can be performed by *any* TM!

Universality



What's going on here?

k encodes a "program" – a description of some arbitrary machine.

j encodes the input data to be used.

T_U *interprets* the program, emulating its processing of the data!

KEY IDEA: Interpretation.
Manipulate *coded representations* of computing machines, rather than the machines themselves.

Turing Universality

The *Universal Turing Machine* is the paradigm for modern general-purpose computers!

Basic threshold test: Is your computer *Turing Universal* ?

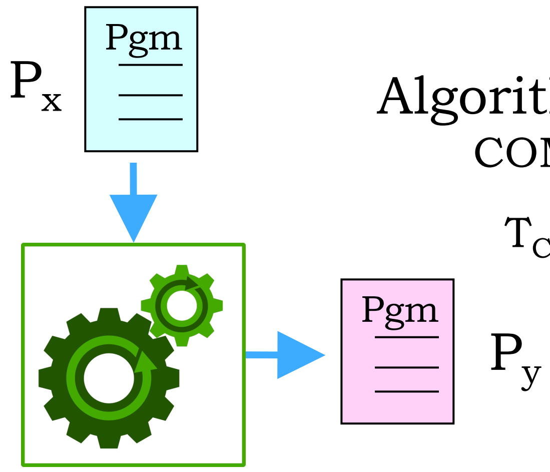
- If so, it can emulate every other Turing machine!
- Thus, your computer can compute any computable function

To show your computer is Universal: demonstrate that it can emulate some known UTM.

- Actually given finite memory, can only emulate UTMs + inputs up to a certain size
- This is not a high bar: conditional branches (BEQ) and some simple arithmetic (SUB) are enough.

Coded Algorithms: Key to CS

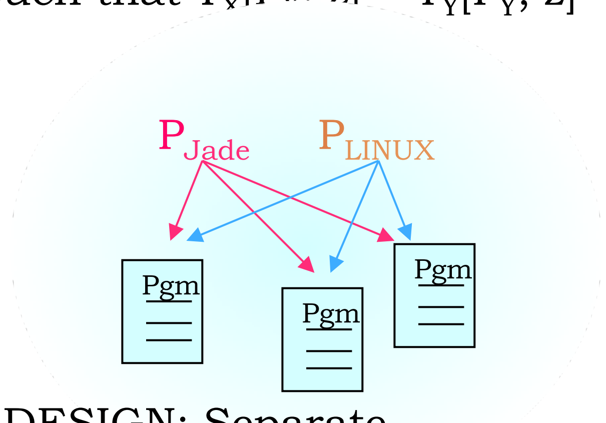
data vs hardware



Algorithms as data: enables

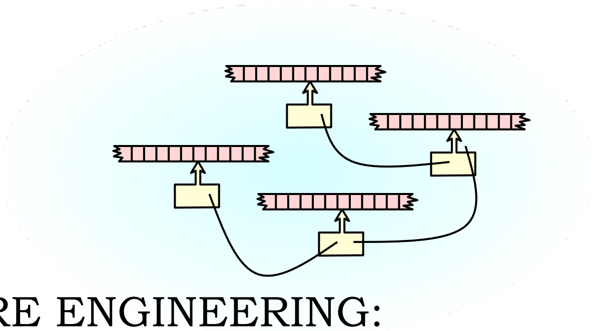
COMPILERS: analyze, optimize, transform behavior

$T_{\text{COMPILER-X-to-Y}}[P_X] = P_Y$, such that $T_X[P_X, z] = T_Y[P_Y, z]$



LANGUAGE DESIGN: Separate specification from implementation

- C, Java, JSIM, Linux, ... all run on X86, Sun, ARM, JVM, CLR, ...
- Parallel development paths:
  - Language/Software design
  - Interpreter/Hardware design



SOFTWARE ENGINEERING:

Composition, iteration, abstraction of coded behavior

$F(x) = g(h(x), p((q(x)))$

Uncomputability (!)

Uncomputable functions: There are well-defined discrete functions that a Turing machine cannot compute

- No algorithm can compute f(x) for arbitrary x in finite number of steps
- Not that we don't know algorithm - can prove no algorithm exists
- Corollary: Finite memory is not the only limiting factor on whether we can solve a problem

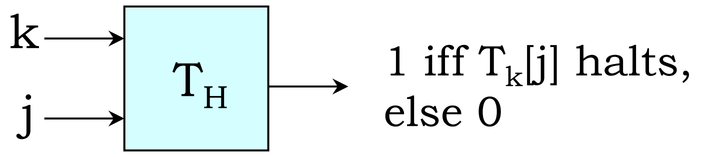
The most famous uncomputable function is the so-called Halting function, $f_H(k, j)$, defined by:

$$f_H(k, j) = \begin{cases} 1 & \text{if } T_k[j] \text{ halts;} \\ 0 & \text{otherwise.} \end{cases}$$

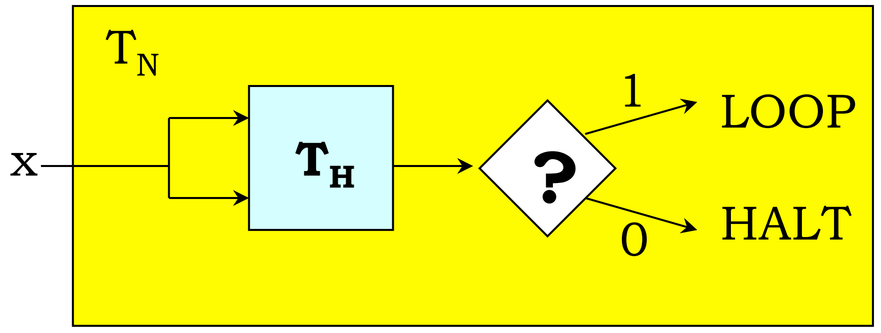
$f_H(k, j)$ determines whether the k^{th} TM halts when given a tape containing j.

Why f_H is Uncomputable

If f_H is computable, it is equivalent to some TM (say, T_H):



Then T_N (N for "Nasty"), which must be computable if T_H is:



$T_N[x]$: LOOPS if $T_x[x]$ halts; HALTS if $T_x[x]$ loops

Finally, consider giving N as an argument to T_N:

$T_N[N]$: LOOPS if $T_N[N]$ halts; HALTS if $T_N[N]$ loops

Contradiction!

T_N can't be computable, hence T_H can't either!