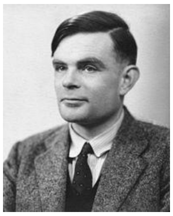
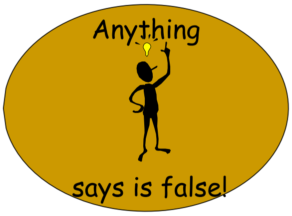
Great Theoretical Ideas in CS

V. Adamchik
Carnegie Mellon University

CS 15-251
Lecture 21

Turing's Legacy





Outline

Turing Machine
Decidable languages
Computable functions
Church–Turing Thesis
Halting Problem
(is undecidable)

23 Hilbert's problems

In 1900 Hilbert presented a list of 23 challenging problems in math

#1 The Continuum Hypothesis
#8 The Riemann Hypothesis
#10 On solving a Diophantine equations
#18 The Kepler Conjecture

What is a computation/algorithm?

Hilbert's 10th problem (1900):

Given a multivariate polynomial w/ integer coeffs, e.g. $4x^2y^3 - 2x^4z^5 + x^8$,
"*devise a process according to which it can be determined in a finite number of operations*" whether it has an integer root.

Mathematicians: "we should probably try to formalize what counts as an 'algorithm' ".

What is a computation/algorithm?

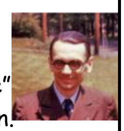
It took 30 years before it was described precisely

Hilbert's Entscheidungsproblem (1928):
("decision problem")

Given a sentence in first-order logic, give an "*effectively calculable procedure*" for determining if it's provable.

Mathematicians: "we should probably try to formalize what counts as an 'algorithm' ".

Gödel (1934):
Discusses some ideas for definitions of what functions/languages are "computable" but isn't confident what's a good definition.



Church (1936):
Invents lambda calculus,
claims it should be the definition.

Gödel, Post (1936):
Arguments that Church's isn't justified.

Meanwhile… a certain British grad student in Princeton, unaware of all these debates…

Alan Turing (1936, age 22):
Describes a new model of computation,
now known as the Turing Machine

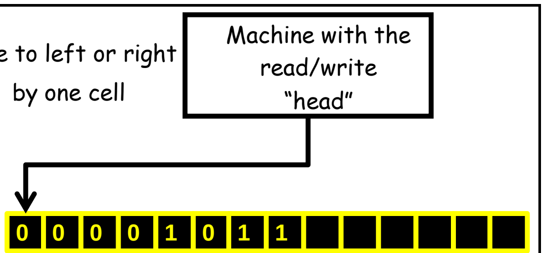
PH.D. student
of A. Church

Gödel, Kleene, and even Church:
"Um, he nailed it. Game over, computation defined."
1937: Turing proves TM's ≡ lambda calculus

Turing's Inspiration

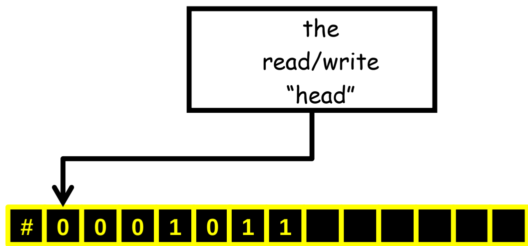
Human writes symbols on paper
WLOG, the paper is a sequence of squares
No upper bound on the number of squares
At most finitely many kinds of symbols
Human observes one square at a time
Human has only finitely many mental states
Human can change symbols and change focus to a neighboring square, but only based on its state and the symbol it observes
Human acts deterministically

move to left or right by one cell



input or the "tape" indefinitely extensible to the right.

each cell consists of a symbol



the "tape alphabet" in this example is $\{0,1,\#,\sqcup\}$ (blank)

Once we read a cell we replace it with #

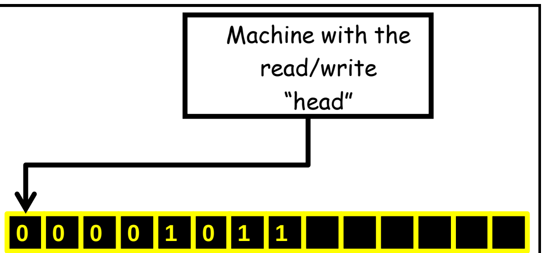


Illustration of a TM trying to decide $\{0^n1^n : n \in \mathbb{N}\}$

The idea is to match the first zero with the last 1 in the tape. So we will go forth and back in the tape.

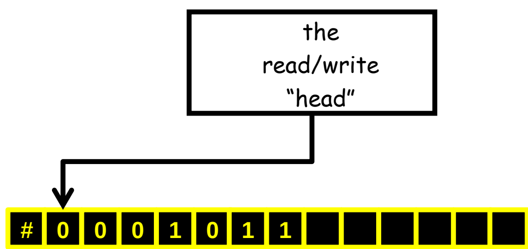
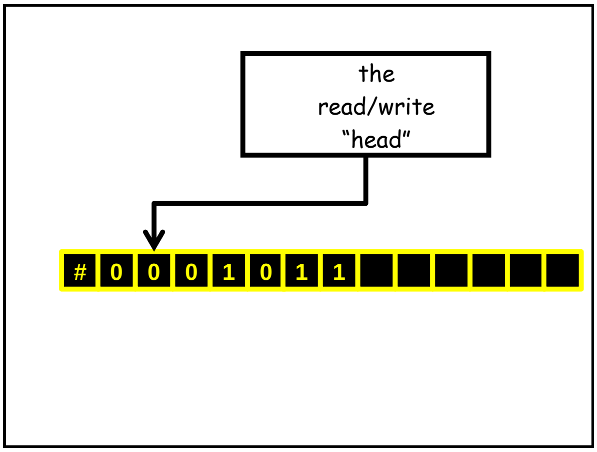
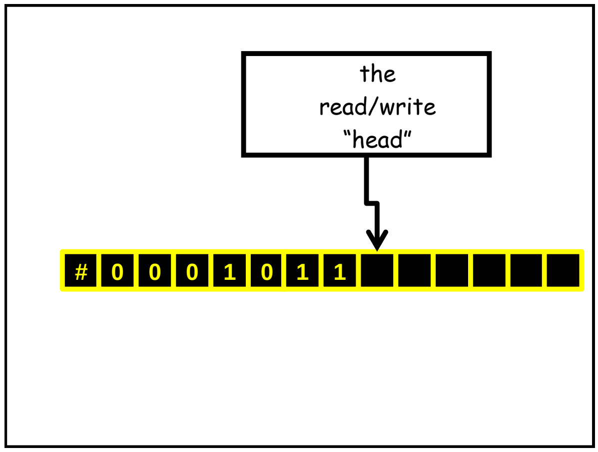
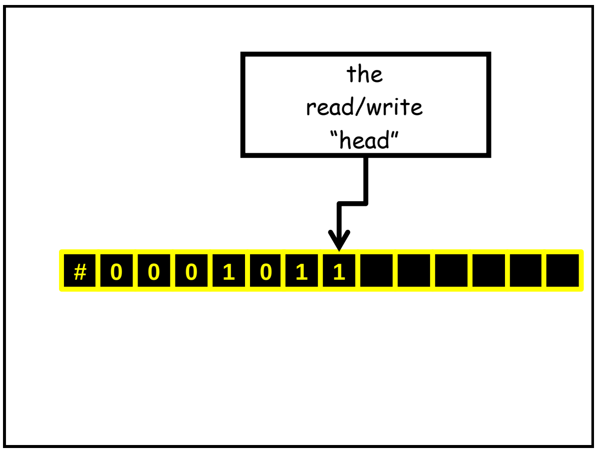
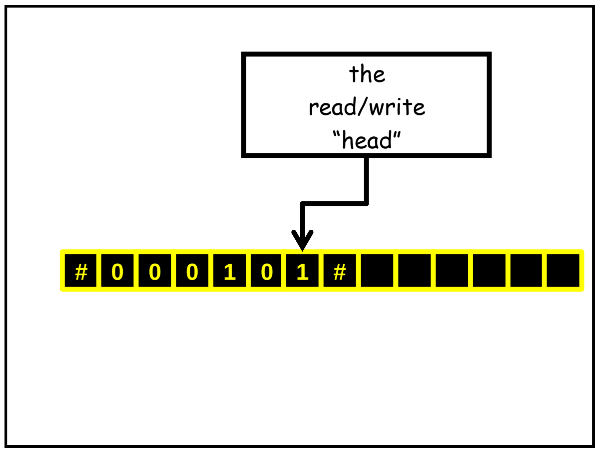


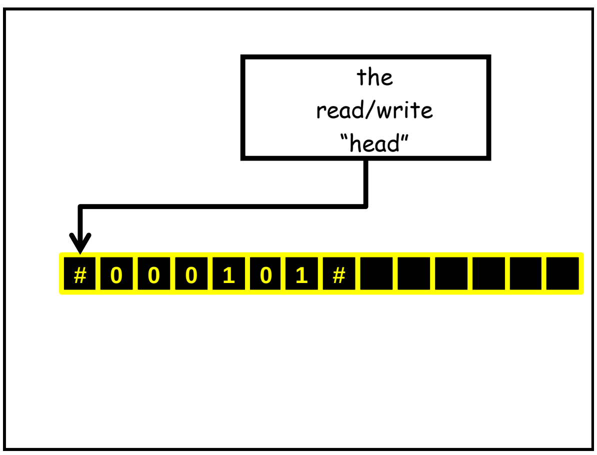
Illustration of a TM trying to decide $\{0^n1^n : n \in \mathbb{N}\}$

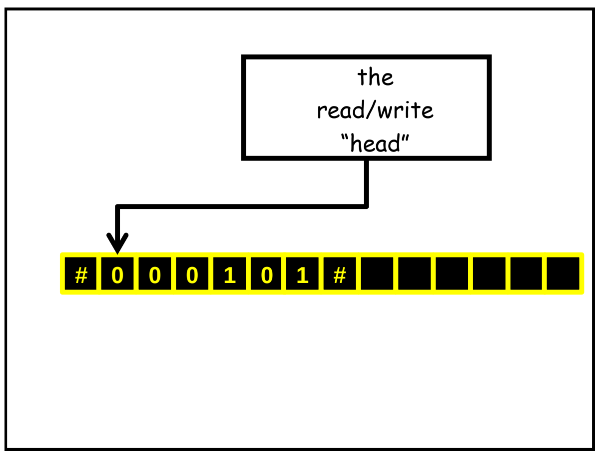

the
read/write
“head”
0 0 0 1 0 1 1

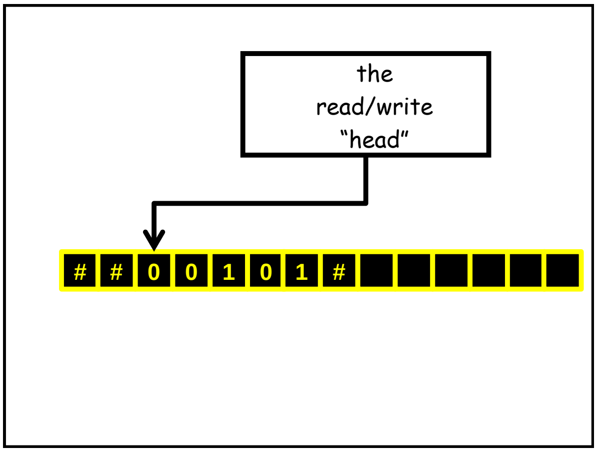

the
read/write
“head”
0 0 0 1 0 1 1

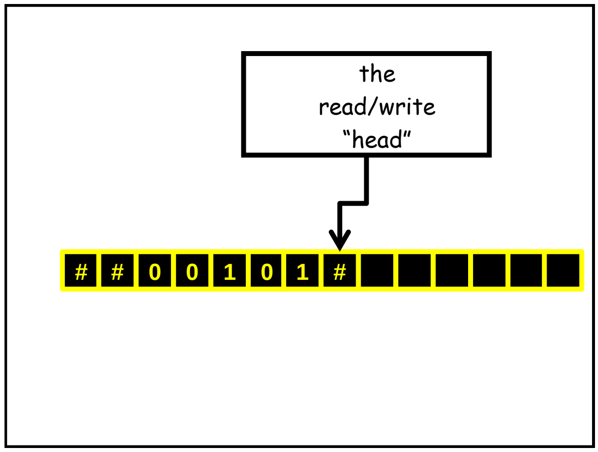

the
read/write
“head”
0 0 0 1 0 1 1

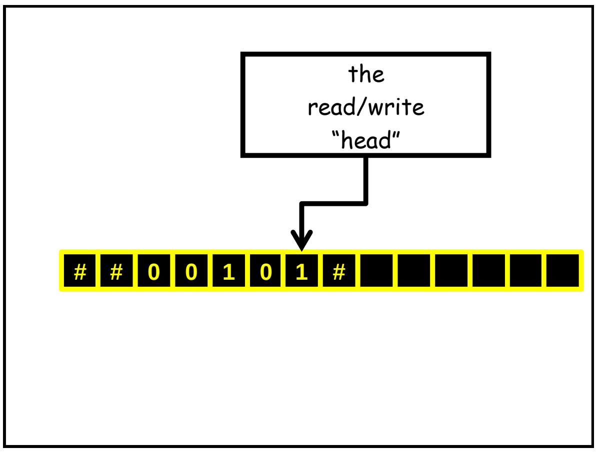

the
read/write
“head”
0 0 0 1 0 1

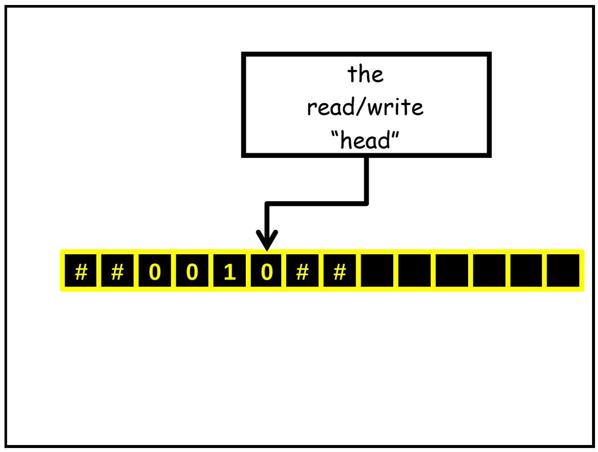

the
read/write
“head”
0 0 0 1 0 1

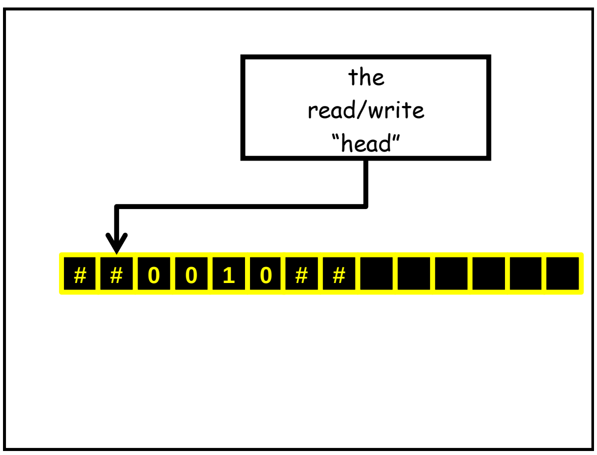

the
read/write
“head”
0 0 0 1 0 1

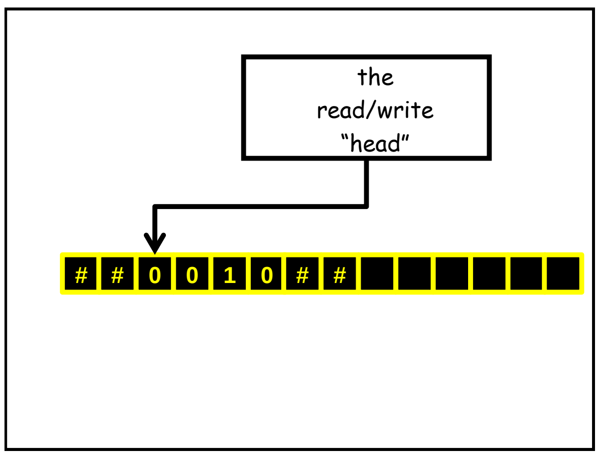

the
read/write
“head”
0 0 1 0 1

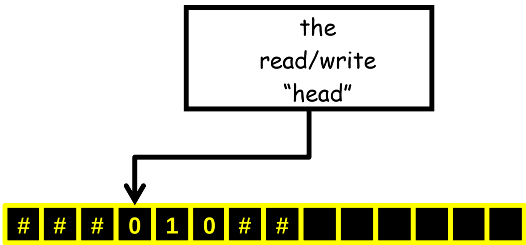

the
read/write
“head”
0 0 1 0 1

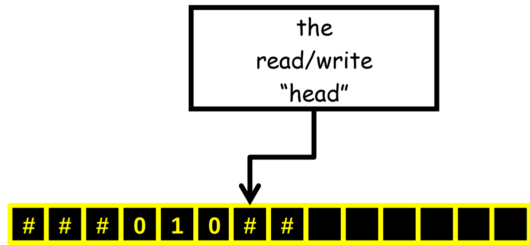

the
read/write
“head”
0 0 1 0 1

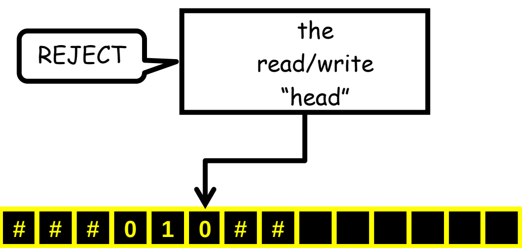

the
read/write
“head”
0 0 1 0 #


the
read/write
“head”
0 0 1 0 #

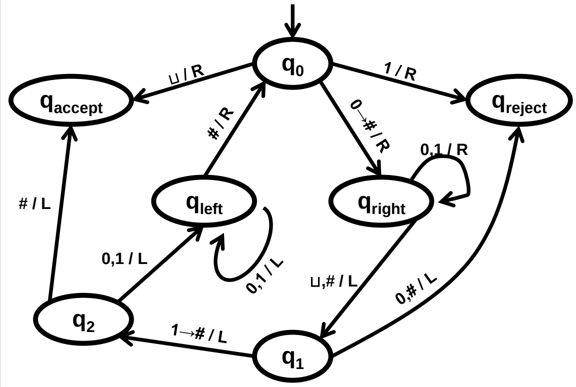

the
read/write
“head”
0 0 1 0 #







Transition rules



Formal definition of Turing Machine

A Turing Machine is a 7-tuple
$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$:

Q is a finite set of states,

$q_0 \in Q$ is the start state,

$q_{accept} \in Q$ is the accept state,

$q_{reject} \in Q$ is the reject state, $q_{reject} \neq q_{accept}$.

Σ is a finite input alphabet (with $\sqcup \notin \Sigma$),

Γ is a finite tape alphabet (with $\sqcup \in \Gamma$, $\Sigma \subset \Gamma$)

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$ is transition function,

Rules of computation

Tape starts with input $x \in \Sigma^*$, followed by infinite $\sqcup$'s.

Control starts in state q_0, head starts in leftmost square.

If the current state is q and head is reading symbol $s \in \Gamma$, the machine transitions according to $\delta(q,s)$, which gives:

the next state,
either erase or write a symbol
move the head Left or Right.

Continues until either the accept state or reject state reached.

When accept/reject state is reached, M halts.

M might also never halt, in which case we say it loops.

Decidable languages

Definition:

A language $L \subseteq \Sigma^*$ is decidable if there is a Turing Machine M which:

1. Halts on every input $x \in \Sigma^*$.
2. Accepts inputs $x \in L$ and rejects inputs $x \notin L$.

Such a Turing Machine is called a decider. It 'decides' the language L.

We like deciders. We don't like TM's that sometimes loop.

Decidable

A problem P is *decidable* if it can be solved by a Turing machine T that always halt. (We say that P has an effective algorithm.)

Note that the corresponding language of a decidable problem is *recursive*.

Undecidable

A problem is *undecidable* if it cannot be solved by any Turing machine that halts on all inputs.

Note that the corresponding language of an undecidable problem is *non-recursive*.

Computable functions

Note the equivalence between languages and functions:

function $f : \{0,1\}^* \to \{0,1\} \equiv$ subset $L \subseteq \{0,1\}^*$

$L \subseteq \{0,1\}^*$

$$f(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

If L is decidable we call f computable, and vice versa.

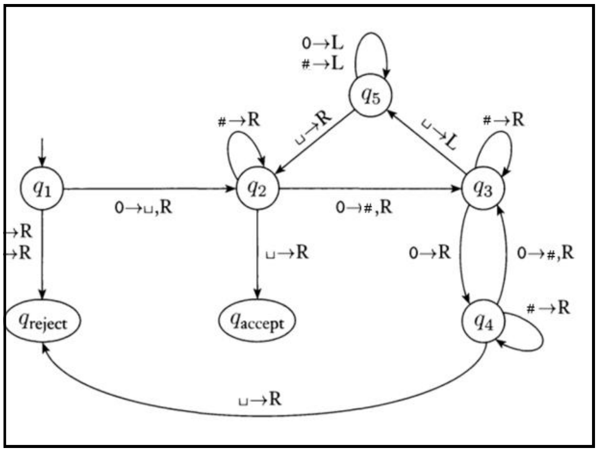
Decidable languages

Examples:

Hopefully you're convinced that $\{0^n 1^n : n \in N\}$ is decidable. (Recall it's not "regular".)

The language $\{0^{2^n} : n \in N\} \subseteq \{0\}^*$, i.e. {0, 00, 0000, 00000000, …}, is decidable.

Proof: we'll describe a decider TM for it.



Describing Turing Machines

Low Level:

Explicitly describing all states and transitions.

Medium Level:

Carefully describing in English how the TM operates. Should be 'obvious' how to translate into a Low Level description.

High Level:

Skips 'standard' details, just highlights 'tricky' details. For experts only!

$\{0^{2^n} : n \in \mathbb{N}\}$ is decidable

Medium Level description:

1. Sweep from left to right across the tape, overwriting a # over top of every *other* 0.
2. If you saw one 0 on the sweep, accept.
3. If you saw an odd number of 0's, reject.
4. Move back to the leftmost square.
5. Repeat, go to step 1

TM programming exercises

Convert input $x_1x_2x_3\cdots x_n$ to $x_1\sqcup x_2\sqcup x_3\sqcup\cdots\sqcup x_n$.

Simulate a big Γ by just $\{0,1,\sqcup\}$.

Increment/decrement a number in binary.

Copy sections of tape from one spot to another.

Simulate having 2 tapes, with separate heads.

Create a Turing Machine U(a,b) whose input is
$\langle M\rangle$, the encoding of a TM M, and
x, a string
and which simulates the execution of M on x.

Like writing a Py simulator in Py!

Church-Turing Thesis:

"Any natural / reasonable notion of computation can be simulated by a TM."

This is not a theorem.

Is it… *…an observation?*
…a definition?
…a hypothesis?
…a law of nature?
…a philosophical statement?

Well, whatever. Everyone believes it.

Is every language in $\{0,1\}^*$ decidable?
Is every function $f : \{0,1\}^* \to \{0,1\}$ computable?

Answer: No

Every TM is encodable by a finite string.
Therefore the set of all TM's is countable.

So the subset of all *decider* TM's is countable. Thus the set of all decidable languages is countable.

But the set of all languages is uncountable.

$|P(\{0,1\}^*)| > |\{0,1\}^*|$

The HELLO Assignment

Write a program to output the word "HELLO" on the screen and halt.

Space and time are not an issue.
The program is for an ideal computer.

PASS for any working HELLO .
No partial credit.

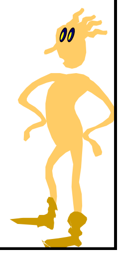
Grading Script

The grading script G must be able to take any program P and grade it.

$$G(P) = \begin{cases} \text{Pass, if P prints only the word "HELLO" and halts.} \\ \text{Fail, otherwise.} \end{cases}$$

How exactly might such a script work?

What kind of program could a student who hated his TA hand in?



Nasty Program

```
n:=0;
while (n is not a counter-example
     to the Riemann Hypothesis) {
  n++;
}
print "Hello";
```

The nasty program is a PASS if and only if the Riemann Hypothesis is true.

Despite the simplicity of the HELLO assignment, there is no program to correctly grade it!

And we will <u>prove</u> this.

Some uncomputable functions

This one is called The Halting Problem.

Given a TM description ⟨M⟩ and an input x, does M halt on input x?

Turing's Theorem:
The Halting Problem is undecidable.

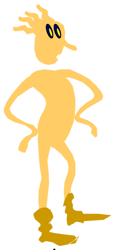
Notation And Conventions

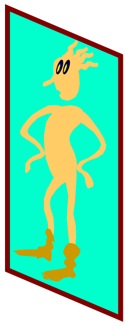
Fix a single programming language

When we write program P we are talking about the text of the source code for P

P(x) means the output that arises from running program P on input x, assuming that P eventually halts.

The meaning of P(P)





P(P) means the output obtained when we run P on the text of its own source code.

The Halting Set K

Definition:
K is the set of all programs P such that P(P) halts.

K = { program P | P(P) halts }

The Halting Problem
K = {P | P(P) halts }

Is the Halting Set K decidable?
Is there a program HALT such that:

HALT(P) = yes, if $P \in K$, so P(P) halts
HALT(P) = no, if $P \notin K$, so P(P) doesn't halt

HALT decides whether or not any given program is in K.

THEOREM: There is no program to solve the halting problem (Alan Turing, 1937)

Suppose a program HALT that solved the halting problem is indeed exist.

We will call HALT as a subroutine in a new program called CONFUSE.

CONFUSE

```
bool CONFUSE(P)
{
  if (HALT(P) == True)
  then loop forever;

  else return True;

}
```

We assume that HALT solves the halting problem

Does CONFUSE(CONFUSE) halt?

Does CONFUSE(CONFUSE) halt?

```
boolean CONFUSE(P)
{
  if (HALT(P) == True) then loop forever;
  else return True;
}
```

Consider two cases:

1. CONFUSE(CONFUSE) halts

   then (by def.) HALT(CONFUSE) is True.

   But then CONFUSE will loop forever.

Does CONFUSE(CONFUSE) halt?

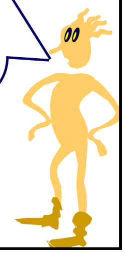
```
boolean CONFUSE(P)
{
  if (HALT(P) == True) then loop forever;
  else return True;
}
```

2. CONFUSE(CONFUSE) does not halt

then (by def.) HALT(CONFUSE) is False.

But then CONFUSE halts.

Turing's argument is essentially the reincarnation of Cantor's Diagonalization argument



All Programs (the input)

All Programs

	P_0	P_1	P_2	…	P_j	…
P_0						
P_1						
…						
P_i						
…						

Programs (computable functions) are countable, so we can put them in a (countably long) list

---

All Programs (the input)

All Programs

	P_0	P_1	P_2	…	P_j	…
P_0						
P_1						
…						
P_i						
…						

Yes, if $P_i(P_j)$ halts
No, otherwise

All Programs (the input)

All Programs

	P_0	P_1	…	P_i	…
P_0	d_0				
P_1		d_1			
…			…		
P_i				d_i	
…					…

Let d_i = HALT(P_i)

CONFUSE(P_i) halts iff d_i = no
(The CONFUSE function is the negation of the diagonal.) Hence CONFUSE cannot be on this list.

Alan Turing (1912-1954)

Theorem: [1937]
There is no program to solve the halting problem

Given some code, determine if it terminates.

We know that it is unsolvable by any algorithm.

Hilbert's 10^{th} problem

Input: Multivariate polynomial w/ integer coeffs.

Question: Does it have an integer root?
Undecidable.

Question: Does it have a real root?
Decidable.

Question: Does it have a rational root?
Not known if it's decidable or not.

Richardson's Problem

Make an expression E using the rational numbers, two real numbers π and ln(2), the variable x, and operations +, −, ·, sin, exp, abs.

Question: Can you make an E such that $E \equiv 0$?

Theorem (Richardson, 1968): Undecidable.

- Turing Machine
- Decidable languages
- Computable functions
- Church–Turing Thesis
- Halting Problem

Here's What You Need to Know…