Universality and Computability

Fundamental questions:

Q. What is a general-purpose computer?
Q. Are there limits on the power of digital computers?
Q. Are there limits on the power of machines we can build?

Pioneering work in the 1930s.

- Princeton == center of universe.
- Automata, languages, computability, universality, complexity, logic

David Hilbert   Kurt Gödel   Alan Turing   Alonzo Church   John von Neumann

Context: Mathematics and Logic

Mathematics. Any formal system powerful enough to express arithmetic.

Principia Mathematics
Peano arithmetic
Zermelo-Fraenkel set theory

Complete. Can prove truth or falsity of any arithmetic statement.
Consistent. Can't prove contradictions like 2 + 2 = 5.
Decidable. Algorithm exists to determine truth of every statement.

Q. [Hilbert, 1900] Is mathematics complete and consistent?
A. [Gödel's Incompleteness Theorem, 1931] No!!!

Q. [Hilbert's Entscheidungsproblem] Is mathematics decidable?
A. [Church 1936, Turing 1936] No!

7.4 Turing Machines (revisited)

Alan Turing (1912-1954)

Turing Machine

Desiderata. Simple model of computation that is "as powerful" as conventional computers.

Intuition. Simulate how humans calculate.

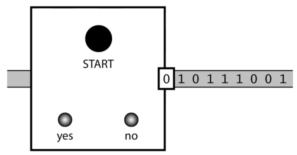
Ex. Addition.

			1	2	3	4	5	6		
		+	3	1	4	1	5	9		

Last lecture: DFA

Tape.
- Stores input.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.
- Points to one cell of tape.
- Reads a symbol from active cell.
- Moves right one cell at a time.



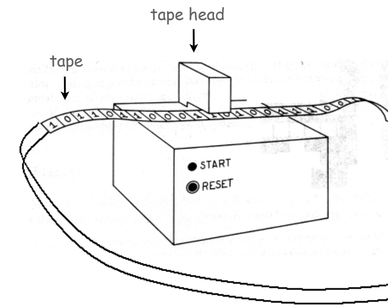
tape head ↓

tape	...	1	1	1	0	0	1	1	0	1	1	0	...

This lecture: Turing machine

Tape.
- Stores input, output, and intermediate results.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.
- Points to one cell of tape.
- Reads a symbol from active cell.
- Writes a symbol to active cell.
- Moves left or right one cell at a time.

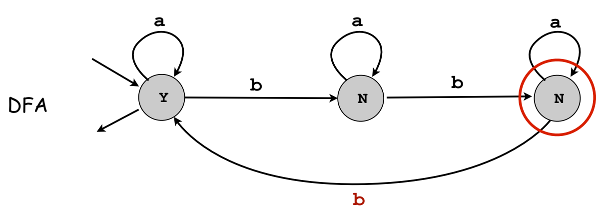


tape head ↓

tape	...	#	1	1	0	0	+	1	0	1	1	#	...

Last lecture: Deterministic Finite State Automaton (DFA)

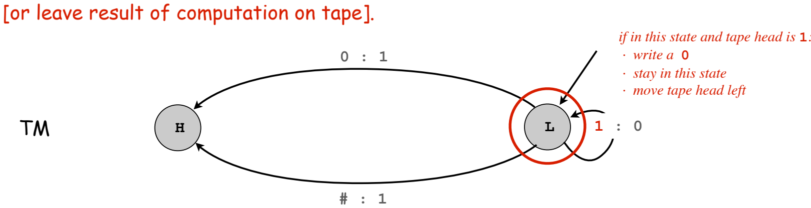
Simple machine with N states.
- Begin in start state.
- Read first input symbol.
- Move to new state, depending on current state and input symbol.
- Repeat until last input symbol read.
- Accept input string if last state is labeled Y.



Input	b	b	a	a	b	b	a	b	b

This lecture: Turing Machine

Simple machine with N states.
- Begin in start state.
- Read first input symbol.
- Move to new state and write new symbol on tape, depending on current state and input symbol.
- Move tape head left if state is labeled L, right if state is labeled R.
- Repeat until entering a state labelled Y, N, or H.
- Accept input string if state is labeled Y, reject if N [or leave result of computation on tape].

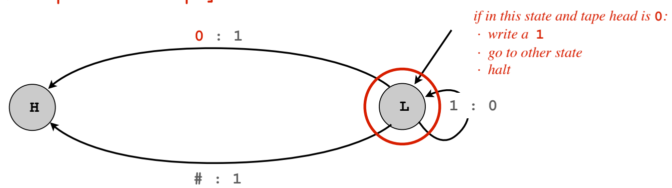


Input	#	#	1	0	1	1	1	0	1

TM Example

Simple machine with N states.
- Begin in start state.
- Read first input symbol.
- Move to new state and write new symbol on tape, depending on current state and input symbol.
- Move tape head left if state is labeled L, right if state is labeled R.
- Repeat until entering a state labelled Y, N, or H.
- Accept input string if state is labeled Y, reject if N
  [or leave result of computation on tape].

TM

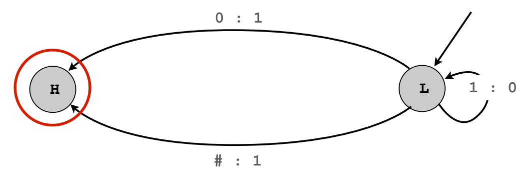


Input									
	#	#	1	0	1	1	1	0	0

TM Example

Simple machine with N states.
- Begin in start state.
- Read first input symbol.
- Move to new state and write new symbol on tape, depending on current state and input symbol.
- Move tape head left if state is labeled L, right if state is labeled R.
- Repeat until entering a state labelled Y, N, or H.
- Accept input string if state is labeled Y, reject if N
  [or leave result of computation on tape].

TM

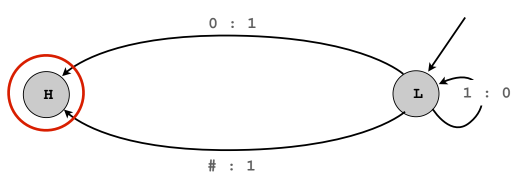


Output									
	#	#	1	0	1	1	1	1	0

TM Example

Simple machine with N states.
- Begin in start state.
- Read first input symbol.
- Move to new state and write new symbol on tape, depending on current state and input symbol.
- Move tape head left if state is labeled L, right if state is labeled R.
- Repeat until entering a state labelled Y, N, or H.
- Accept input string if state is labeled Y, reject if N
  [or leave result of computation on tape].

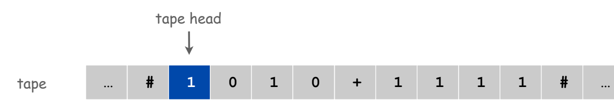
TM



Input	#	#	1	0	1	1	1	0	1
Output	#	#	1	0	1	1	1	1	0

Turing Machine: Initialization and Termination

Initialization. Set input on some portion of tape; set tape head.

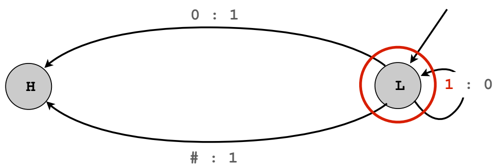


Termination. Stop if enter `yes`, `no`, or `halt` state.

Note: infinite loop possible

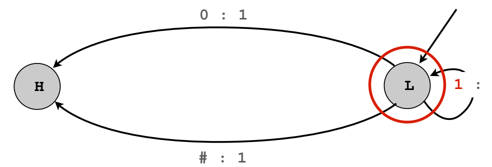
Output. Contents of tape.

TM Example 1: Binary Increment



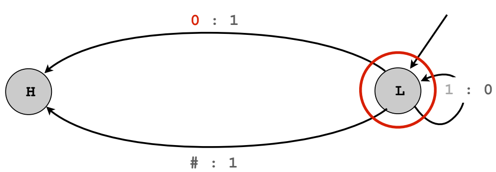
...	#	1	0	1	1	#	...

TM Example 1: Binary Increment



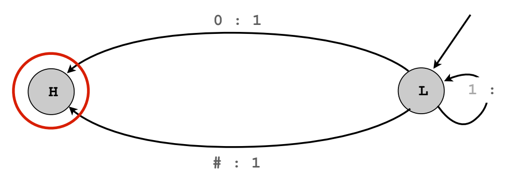
...	#	1	0	1	1	#	...
...	#	1	0	1	0	#	...

TM Example 1: Binary Increment



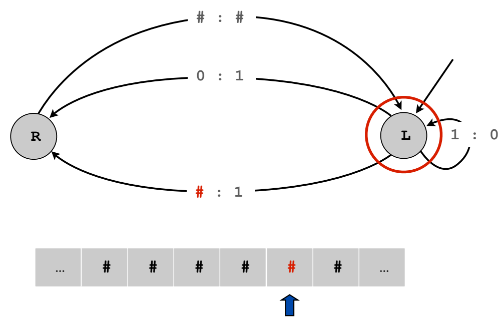
...	#	1	0	1	1	#	...
...	#	1	0	1	0	#	...
...	#	1	0	0	0	#	...

TM Example 1: Binary Increment

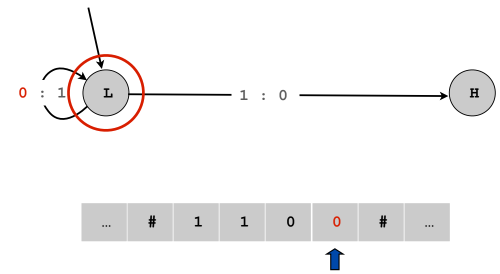


...	#	1	0	1	1	#	...
...	#	1	0	1	0	#	...
...	#	1	1	0	0	#	...

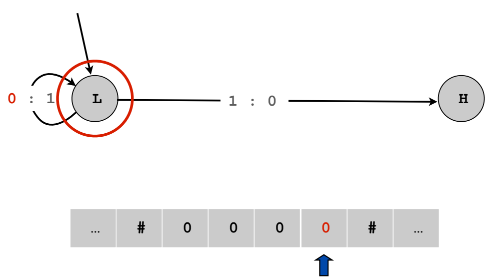
TM Example 2: Continuous Binary Counter



TM Example 3: Binary Decrement

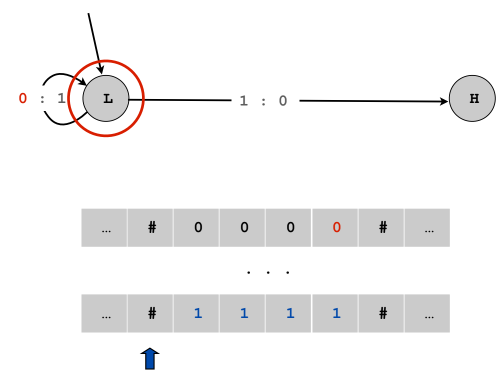


TM Example 3: Binary Decrement



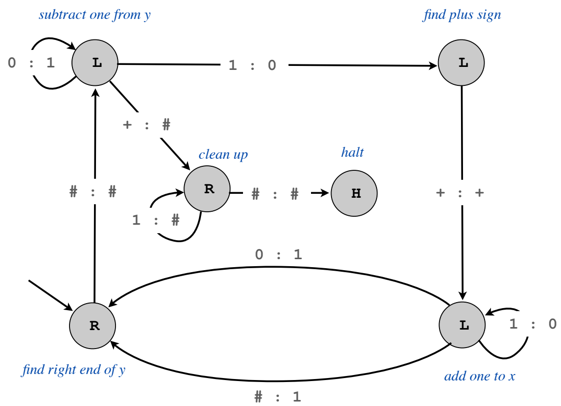
Q. What happens if we try to decrement 0 ?

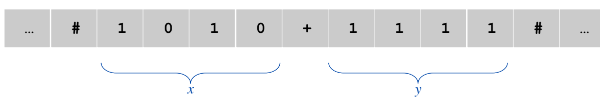
TM Example 3: Binary Decrement



Q. What happens if we try to decrement 0 ?
A. Doesn't halt! (TMs can have bugs, too.)

TM Example 4: Binary Adder





Ex. Use simulator to understand how this TM works.

7.5 Universality

Universal Machines and Technologies


Dell PC


iMac

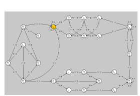

Diebold voting machine

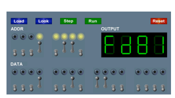

iPod


Printer


Xbox

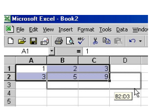

Tivo


Turing machine

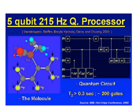

TOY

JAVA

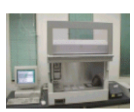
Java language

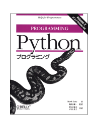

MS Excel


Blackberry



Quantum computer


DNA computer



Python language

Program and Data

Data. Sequence of symbols (interpreted one way).
Program. Sequence of symbols (interpreted another way).

Ex 1. A compiler is a program that takes a program in one language as input and outputs a program in another language.

Java

machine language

Your program

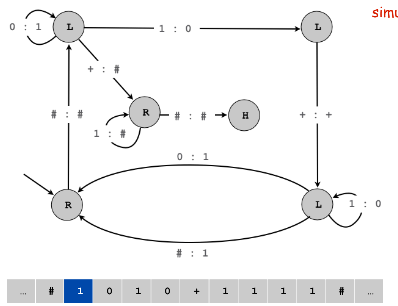
```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World");
    }
}
```

is DATA to a compiler

Program and Data

Data. Sequence of symbols (interpreted one way).
Program. Sequence of symbols (interpreted another way).

Ex 2. A simulator is a program that takes a program for one machine as input and simulates the operation of that program.



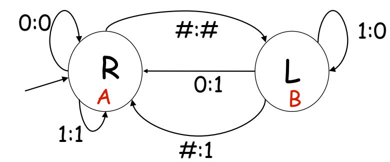
Data for simulator

```
% more adder.tur
vertices
2 R
0 L
1 L
3 L
4 R
5 H

edges
0 0 0 1
0 1 1 0
0 4 + #
1 3 + +
2 0 # #
3 2 # 1
3 2 0 1
3 3 1 0
4 4 1 #
4 5 # #

tape
[1] 0 1 0 + 1 1 1 1
```

is a PROGRAM!

Representations of a Turing Machine

Graphical:



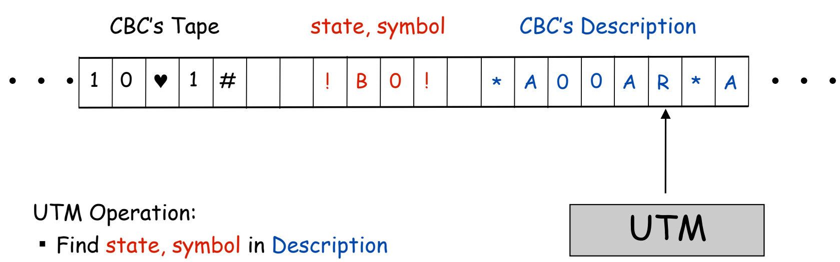
Continuous Binary Counter

Tabular:

Current state	Symbol read	Symbol to write	Next State	Direction
A	0	0	A	R
A	1	1	A	R
A	#	#	B	L
B	0	1	A	R
B	1	0	B	L
B	#	1	A	R

Linear: * A 0 0 A R * A 1 1 A R * A # # B L * B 0 1 A R * B 1 0 B L . . .

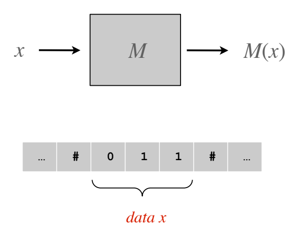
Universal Turing Machine



UTM Operation:
- Find state, symbol in Description
- Copy new symbol to CBI's tape
- Move ♥ L or R
- Update state, symbol
- Repeat

Universal Turing Machine

Turing machine M. Given input tape x, Turing machine M outputs $M(x)$.

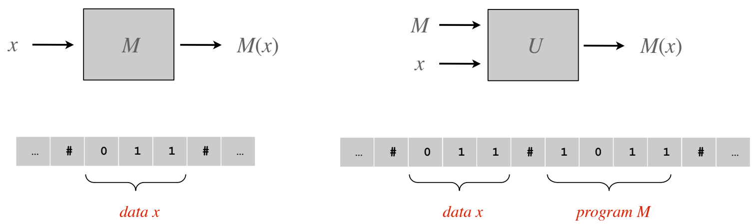


TM intuition. Software program that solves one particular problem.

Universal Turing Machine

Turing machine M. Given input tape x, Turing machine M outputs $M(x)$.

Universal Turing machine U. Given input tape with x and M, universal Turing machine U outputs $M(x)$.



TM intuition. Software program that solves one particular problem.
UTM intuition. Hardware platform that can implement any algorithm.

Universal Turing Machine

Consequences. Your laptop (a UTM) can do any computational task.

- Java programming.
- Pictures, music, movies, games.
- Email, browsing, downloading files, telephony.
- Word-processing, finance, scientific computing.
- …

even tasks not yet contemplated when laptop was purchased

Wenger Giant Swiss Army Knife

Universal Turing Machine

Consequences. Your laptop (a UTM) can do any computational task.

- Java programming.
- Pictures, music, movies, games.
- Email, browsing, downloading files, telephony.
- Word-processing, finance, scientific computing.
- …

even tasks not yet contemplated when laptop was purchased

" Again, it [the Analytical Engine] might act upon other things besides numbers… the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent. " — Ada Lovelace

Church-Turing Thesis

Church Turing thesis (1936). Turing machines can do anything that can be described by any physically harnessable process of this universe.

Remark. "Thesis" and not a mathematical theorem because it's a statement about the physical world and not subject to proof.

but can be falsified

Use simulation to prove models equivalent.

- TOY simulator in Java
- Java compiler in TOY.

Implications.

- No need to seek more powerful machines or languages.
- Enables rigorous study of computation (in this universe).

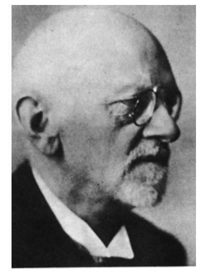
Bottom line. Turing machine is a simple and universal model of computation.

Church-Turing Thesis: Evidence

Evidence.
- 7 decades without a counterexample.
- Many, many models of computation that turned out to be equivalent. ("universal")

model of computation	description
enhanced Turing machines	multiple heads, multiple tapes, 2D tape, nondeterminism
untyped lambda calculus	method to define and manipulate functions
recursive functions	functions dealing with computation on integers
unrestricted grammars	iterative string replacement rules used by linguists
extended L-systems	parallel string replacement rules that model plant growth
programming languages	Java, C, C++, Perl, Python, PHP, Lisp, PostScript, Excel
random access machines	registers plus main memory, e.g., TOY, Pentium
cellular automata	cells which change state based on local interactions
quantum computer	compute using superposition of quantum states
DNA computer	compute using biological operations on DNA

7.6 Computability



Take any definite unsolved problem, such as the question as to the irrationality of the Euler-Mascheroni constant γ, or the existence of an infinite number of prime numbers of the form 2^n-1. However unapproachable these problems may seem to us and however helpless we stand before them, we have, nevertheless, the firm conviction that their solution must follow by a finite number of purely logical processes.
-David Hilbert, in his 1900 address to the International Congress of Mathematics

Introduction to Computer Science · Sedgewick and Wayne · Copyright © 2007 · http://www.cs.Princeton.EDU/IntroCS

A Puzzle: Post's Correspondence Problem

Given a set of cards:
- N card types (can use as many copies of each type as needed).
- Each card has a top string and bottom string.

Example 1:

BAB	A	AB	BA
A	ABA	B	B
0	1	2	3

N = 4

Puzzle:
- Is it possible to arrange cards so that top and bottom strings match?

A Puzzle: Post's Correspondence Problem

Given a set of cards:
- N card types (can use as many copies of each type as needed).
- Each card has a top string and bottom string.

Example 1:

BAB	A	AB	BA
A	ABA	B	B
0	1	2	3

N = 4

Puzzle:
- Is it possible to arrange cards so that top and bottom strings match?

Solution 1.
✎ Yes.

A	BA	BAB	AB	A
ABA	B	A	B	ABA
1	3	0	2	1

Given a set of cards:
- N card types (can use as many copies of each type as needed).
- Each card has a top string and bottom string.

Example 2:

	0	1	2	3
top	A	ABA	B	A
bottom	BAB	B	A	B

N = 4

Puzzle:
- Is it possible to arrange cards so that top and bottom strings match?

Given a set of cards:
- N card types (can use as many copies of each type as needed).
- Each card has a top string and bottom string.

Example 2:

	0	1	2	3
top	A	ABA	B	A
bottom	BAB	B	A	B

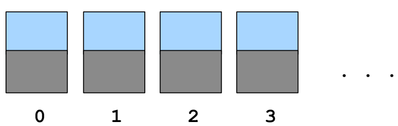
N = 4

Puzzle:
- Is it possible to arrange cards so that top and bottom strings match?

Solution 2.

✎ No. First card in solution must contain same letter in leftmost position.

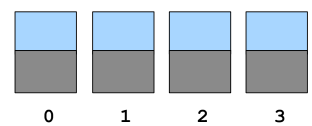
Given a set of cards:
- N card types (can use as many copies of each type as needed).
- Each card has a top string and bottom string.



Puzzle:
- Is it possible to arrange cards so that top and bottom strings match?

Challenge:
- Write a program to take cards as input and solve the puzzle.

Given a set of cards:
- N card types (can use as many copies of each type as needed).
- Each card has a top string and bottom string.



Puzzle:
- Is it possible to arrange cards so that top and bottom strings match?

Challenge:
- Write a program to take cards as input and solve the puzzle.

Surprising fact:
- It is NOT POSSIBLE to write such a program!

Halting Problem

Halting problem. Write a Java function that reads in a Java function `f` and its input `x`, and decides whether `f(x)` results in an infinite loop.

Easy for some functions, not so easy for others.

Ex. Does `f(x)` terminate?

```
public void f(int x)
{
   while (x != 1)
   {
      if   (x % 2 == 0) x = x / 2;
      else(x % 2 == 0) x = 3*x + 1;
   }
}
```

relates to famous open math conjecture

```
f(6):    6 3 10 5 16 8 4 2 1
f(27):   27 82 41 124 62 31 94 47 142 71 214 107 322 … 4 2 1
f(-17):  -17 -50 -25 -74 -37 -110 -55 -164 -82 -41 -122 …  -17 …
```


Undecidable Problem

A yes-no problem is undecidable if no Turing machine exists to solve it.

↖ and (by universality) no Java program either

Theorem. [Turing 1937] The halting problem is undecidable.

Proof intuition: lying paradox.

- Divide all statements into two categories: truths and lies.
- How do we classify the statement: "I am lying".

Key element of lying paradox and halting proof: self-reference.

Halting Problem: Preliminaries

Some programs take other programs as input

- Java compiler, e.g.

Can a program take itself as input ??

Why not ?

- TextGenerator could take TextGenerator.java as input, produce a Markov model of itself, and generate Java-like text.
- GuitarHero could "play" the characters in GuitarHero.java.
- Almost always a peculiar thing to do, but we'll be interested only in whether the program halts, or goes into an infinite loop.

Halting Problem Proof

Assume the existence of `halt(f,x)`:

- Input: a function `f` and its input `x`.
- Output: `true` if `f(x)` halts, and `false` otherwise.

Note. `halt(f,x)` does not go into infinite loop.

We prove by contradiction that `halt(f,x)` does not exist.

- Reductio ad absurdum : if any logical argument based on an assumption leads to an absurd statement, then assumption is false.

encode f and x as strings

```
public boolean halt(String f, String x)
{
   if ( something terribly clever ) return true;
   else                             return false;
}
```

hypothetical halting function

Halting Problem Proof

Assume the existence of **`halt(f,x)`**:
- Input: a function **`f`** and its input **`x`**.
- Output: **`true`** if **`f(x)`** halts, and **`false`** otherwise.

Construct function **`strange(f)`** as follows:
- If **`halt(f,f)`** returns **`true`**, then **`strange(f)`** goes into an infinite loop.
- If **`halt(f,f)`** returns **`false`**, then **`strange(f)`** halts.

f is a string so it is legal (if perverse) to use it for second argument

```
public void strange(String f)
{
   if (halt(f, f))
   {
      while (true) { } // an infinite loop
   }
}
```


Halting Problem Proof

Assume the existence of **`halt(f,x)`**:
- Input: a function **`f`** and its input **`x`**.
- Output: **`true`** if **`f(x)`** halts, and **`false`** otherwise.

Construct function **`strange(f)`** as follows:
- If **`halt(f,f)`** returns **`true`**, then **`strange(f)`** goes into an infinite loop.
- If **`halt(f,f)`** returns **`false`**, then **`strange(f)`** halts.

In other words:
- If **`f(f)`** halts, then **`strange(f)`** goes into an infinite loop.
- If **`f(f)`** does not halt, then **`strange(f)`** halts.

Halting Problem Proof

Assume the existence of **`halt(f,x)`**:
- Input: a function **`f`** and its input **`x`**.
- Output: **`true`** if **`f(x)`** halts, and **`false`** otherwise.

Construct function **`strange(f)`** as follows:
- If **`halt(f,f)`** returns **`true`**, then **`strange(f)`** goes into an infinite loop.
- If **`halt(f,f)`** returns **`false`**, then **`strange(f)`** halts.

In other words:
- If **`f(f)`** halts, then **`strange(f)`** goes into an infinite loop.
- If **`f(f)`** does not halt, then **`strange(f)`** halts.

Call **`strange()`** with ITSELF as input.
- If **`strange(strange)`** halts then **`strange(strange)`** does not halt.
- If **`strange(strange)`** does not halt then **`strange(strange)`** halts.

Halting Problem Proof

Assume the existence of **`halt(f,x)`**:
- Input: a function **`f`** and its input **`x`**.
- Output: **`true`** if **`f(x)`** halts, and **`false`** otherwise.

Construct function **`strange(f)`** as follows:
- If **`halt(f,f)`** returns **`true`**, then **`strange(f)`** goes into an infinite loop.
- If **`halt(f,f)`** returns **`false`**, then **`strange(f)`** halts.

In other words:
- If **`f(f)`** halts, then **`strange(f)`** goes into an infinite loop.
- If **`f(f)`** does not halt, then **`strange(f)`** halts.

Call **`strange()`** with ITSELF as input.
- If **`strange(strange)`** halts then **`strange(strange)`** does not halt.
- If **`strange(strange)`** does not halt then **`strange(strange)`** halts.

Either way, a contradiction. Hence **`halt(f,x)`** cannot exist.

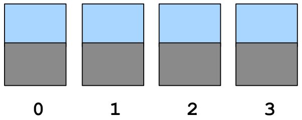
Consequences

Q. Why is debugging hard?
A. All problems below are undecidable.

Halting problem. Give a function f, does it halt on a given input x?
Totality problem. Give a function f, does it halt on every input x?
No-input halting problem. Give a function f with no input, does it halt?
Program equivalence. Do two functions f and always return same value?
Uninitialized variables. Is the variable x initialized before it's used?
Dead-code elimination. Does this statement ever get executed?

Post's Correspondence Problem

Given a set of cards:
- N card types (can use as many copies of each type as needed).
- Each card has a top string and bottom string.



Puzzle:
- Is it possible to arrange cards so that top and bottom strings match?

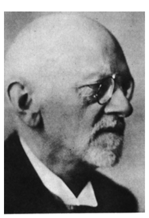
Challenge:
- Write a program to take cards as input and solve the puzzle.

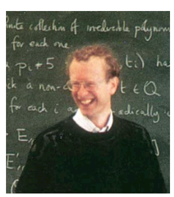
is UNDECIDABLE

More Undecidable Problems

Hilbert's 10th problem.
- "Devise a process according to which it can be determined by a finite number of operations whether a given multivariate polynomial has an integral root."

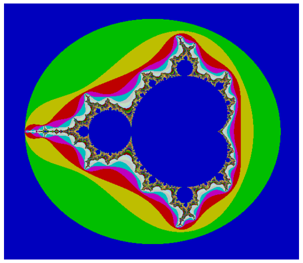
Examples.
- $f(x, y, z) = 6x^3 y z^2 + 3xy^2 - x^3 - 10$. ⬅ yes: $f(5, 3, 0) = 0$
- $f(x, y) = x^2 + y^2 - 3$. ⬅ no
- $f(x, y, z) = x^n + y^n - z^n$ ⬅ yes if $n = 2, x = 3, y = 4, z = 5$ ⬅ no if $n \geq 3$ and $x, y, z > 0$. (Fermat's Last Theorem)


Hilbert


Andrew Wiles, 1995

More Undecidable Problems

Optimal data compression. Find the shortest program to produce a given string or picture.


Mandelbrot set (40 lines of code)

More Undecidable Problems

Virus identification. Is this program a virus?

```
Private Sub AutoOpen()
On Error Resume Next
If System.PrivateProfileString("", CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security",
                               "Level") <> "" Then
CommandBars("Macro").Controls("Security...").Enabled = False
. . .
For oo = 1 To AddyBook.AddressEntries.Count
   Peep = AddyBook.AddressEntries(x)
   BreakUmOffASlice.Recipients.Add Peep
   x = x + 1
   If x > 50 Then oo = AddyBook.AddressEntries.Count
Next oo
. . .
BreakUmOffASlice.Subject = "Important Message From " & Application.UserName
BreakUmOffASlice.Body = "Here is that document you asked for ... don't show anyone else ;-)"
. . .
```

Can write programs in MS Word.
This statement disables security.

Melissa virus
March 28, 1999

Turing's Key Ideas

Turing machine.
formal model of computation

Program and data.
encode program and data as sequence of symbols

Universality.
concept of general-purpose, programmable computers

Church-Turing thesis.
computable at all == computable with a Turing machine

Computability.
inherent limits to computation

Hailed as one of top 10 science papers of 20th century.

Reference: On Computable Numbers, With an Application to the Entscheidungsproblem by A. M. Turing.
In Proceedings of the London Mathematical Society, ser. 2. vol. 42 (1936-7), pp.230-265.

Alan Turing
1912-1954