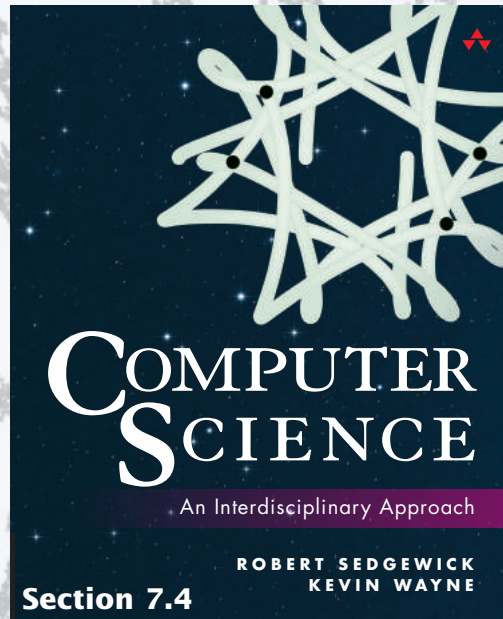


**COMPUTER SCIENCE**  
**SEDGEWICK / WAYNE**

PART II: ALGORITHMS, MACHINES, and THEORY



<http://introcs.cs.princeton.edu>

# 15. Turing Machines

## 15. Turing Machines

- **Context**
- A simple model of computation
- Universality
- Computability
- Implications

# Universality and computability

---

## Fundamental questions

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

Pioneering work at Princeton in the 1930s.



David Hilbert  
1862–1943

Asked the questions



Kurt Gödel  
1906–1978

Solved the math  
problem



Alonzo Church  
1903–1995

Solved the decision  
problem



Alan Turing  
1912–1954

Provided THE answers

## 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.** *Cannot* prove contradictions like  $2 + 2 = 5$ .

**Decidable.** An 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 (!!)**

*Image sources*

[http://en.wikipedia.org/wiki/David\\_Hilbert#/media/File:Hilbert.jpg](http://en.wikipedia.org/wiki/David_Hilbert#/media/File:Hilbert.jpg)

[http://en.wikipedia.org/wiki/Kurt\\_Gödel#/media/File:Kurt\\_gödel.jpg](http://en.wikipedia.org/wiki/Kurt_Gödel#/media/File:Kurt_gödel.jpg)

[http://en.wikipedia.org/wiki/Alonzo\\_Church#/media/File:Alonzo\\_Church.jpg](http://en.wikipedia.org/wiki/Alonzo_Church#/media/File:Alonzo_Church.jpg)

[http://en.wikipedia.org/wiki/Alan\\_Turing#/media/File:Alan\\_Turing\\_photo.jpg](http://en.wikipedia.org/wiki/Alan_Turing#/media/File:Alan_Turing_photo.jpg)

## 15. Turing Machines

- Context
- **A simple model of computation**
- Universality
- Computability
- Implications

## Starting point

---

### Goals

- Develop a model of computation that encompasses all known computational processes.
- Make the model as simple as possible.

**Example:** A familiar computational process.

		1	0	1	0			
			3	1	4	2		
			7	1	8	2		
		1	0	3	2	4		

### Characteristics

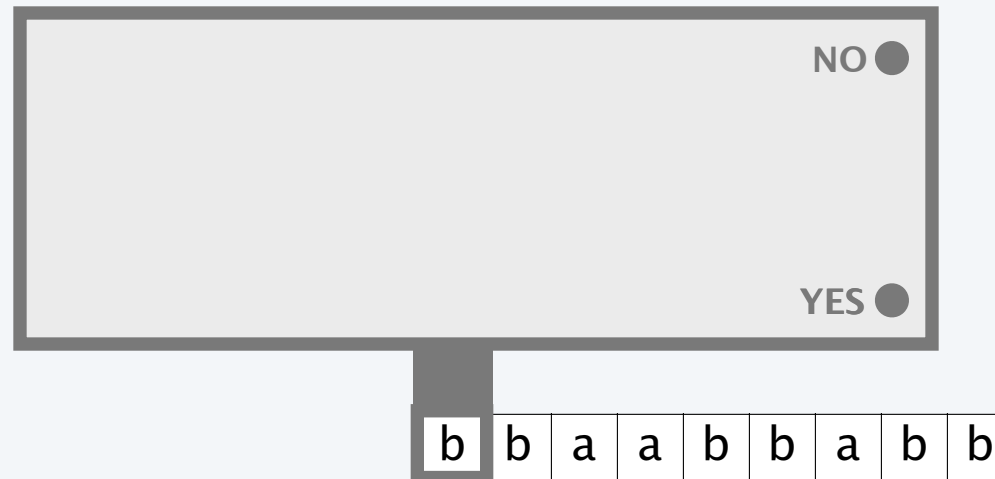
- Discrete.
- Local.
- States.

## Previous lecture: DFAs

A **DFA** is an abstract machine that solves a pattern matching problem.

- A string is specified on an input tape (no limit on its length).
- The DFA reads each character on input tape once, moving left to right.
- The DFA lights "YES" if it *recognizes* the string, "NO" otherwise.

Each DFA defines a *set* of strings (all the strings that it recognizes).

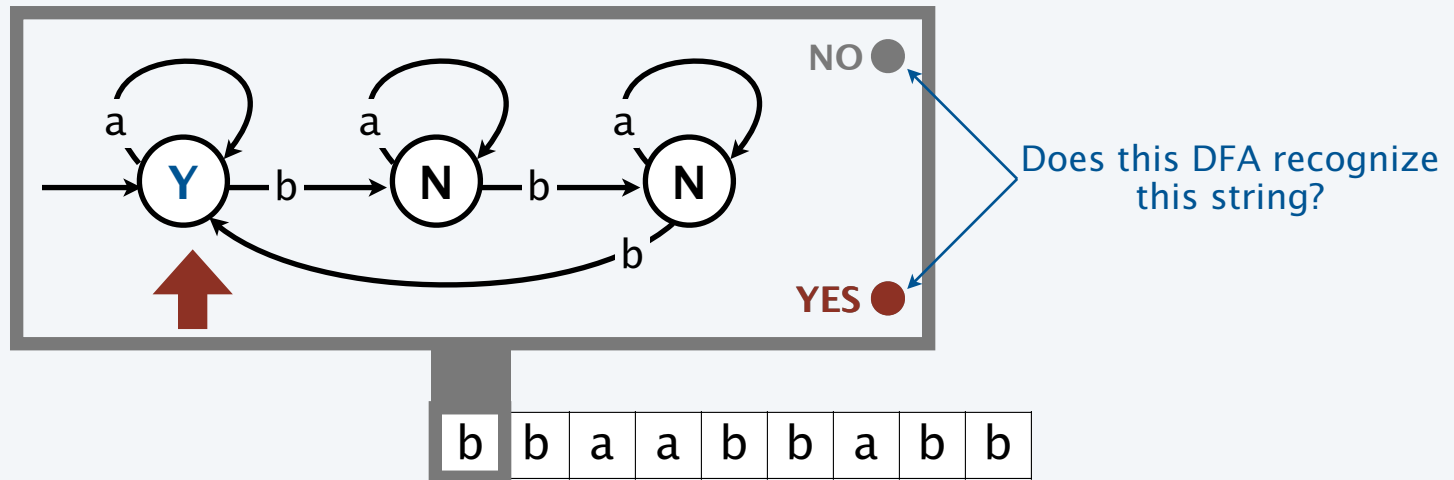




## Previous lecture: DFA details and example

A **DFA** is an abstract machine with a finite number of *states*, each labelled Y or N and *transitions* between states, each labelled with a symbol. One state is the *start* state.

- Begin in the *start* state.
- Read an input symbol and move to the indicated state.
- Repeat until the last input symbol has been read.
- Turn on the "YES" or "NO" light according to the label on the final state.



## This lecture: Turing machines

A **Turing machine (TM)** is an abstract model of computation.

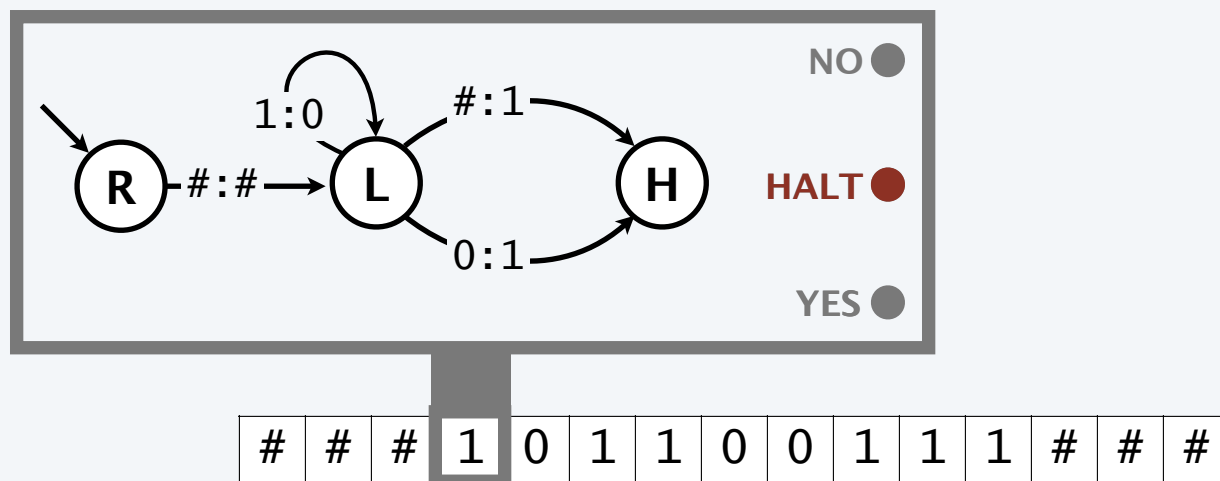
- A string is specified on a tape (no limit on its length).
- The TM reads **and writes** characters on the tape, moving left **or right**.
- The TM lights "YES" if it recognizes the string, "NO" otherwise.
- **The TM may *halt*, leaving the result of the computation on the tape.**



## This lecture: Turing Machine details and example

A **Turing Machine** is an abstract machine with a finite number of *states*, each labeled Y, N, H, L, or R and *transitions* between states, each labeled with a read/write pair of symbols.

- Begin in the *start* state.
- Read an input symbol, move to the indicated state **and write the indicated output**.
- **Move tape head left if new state is labeled L, right if it is labeled R.**
- **Repeat until entering a state labeled Y, N, or H ( and turn on associated light).**



## DFAs vs TMs

---

### Similarities

- Simple model of computation.
- Input on tape is a finite string with symbols from a finite alphabet.
- Finite number of states.
- State transitions determined by current state and input symbol.

### Differences



#### DFAs

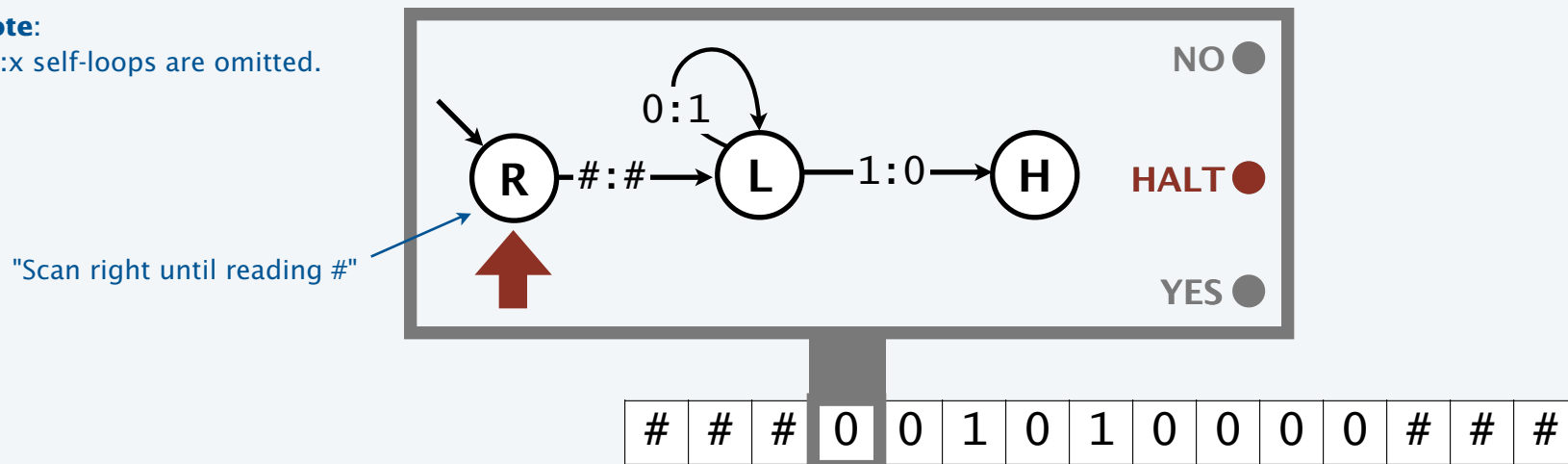
- Can read input symbols from the tape.
- Can only move tape head to the right.
- Tape is finite (a string).
- One step per input symbol.
- Can *recognize* (turn on "YES" or "NO").

#### TMs

- Can read from or write onto the tape.
- Can move tape head either direction.
- Tape does not end (either direction).
- No limit on number of steps.
- Can also *compute* (with output on tape).

# TM example 1: Binary decrementer

**Note:**  
x:x self-loops are omitted.

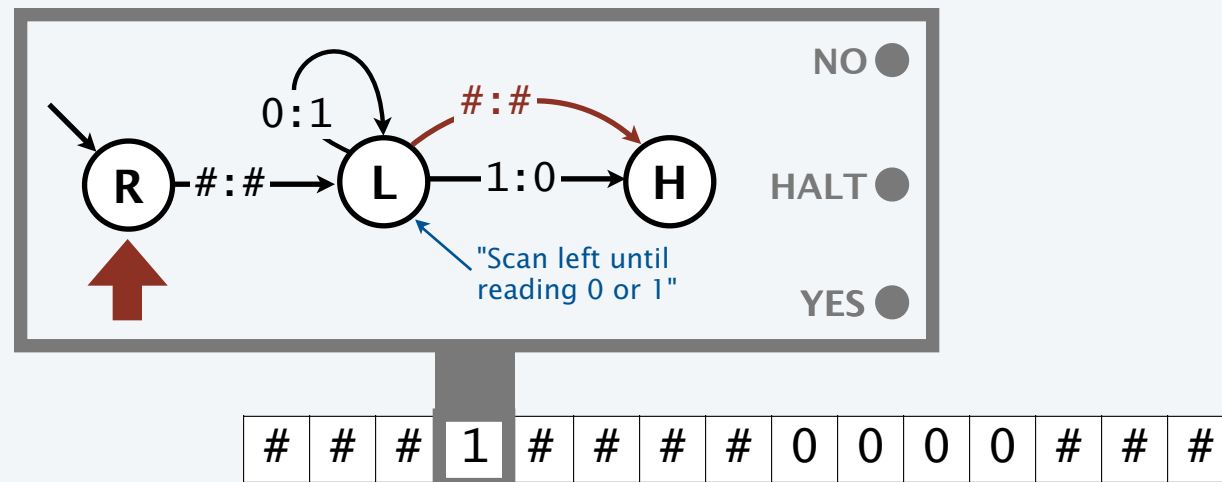


**Input**    1 0 1 0 1 0 0 0 0

**Output**   1 0 1 0 0 1 1 1 1

## TM example 1: Binary decrementer

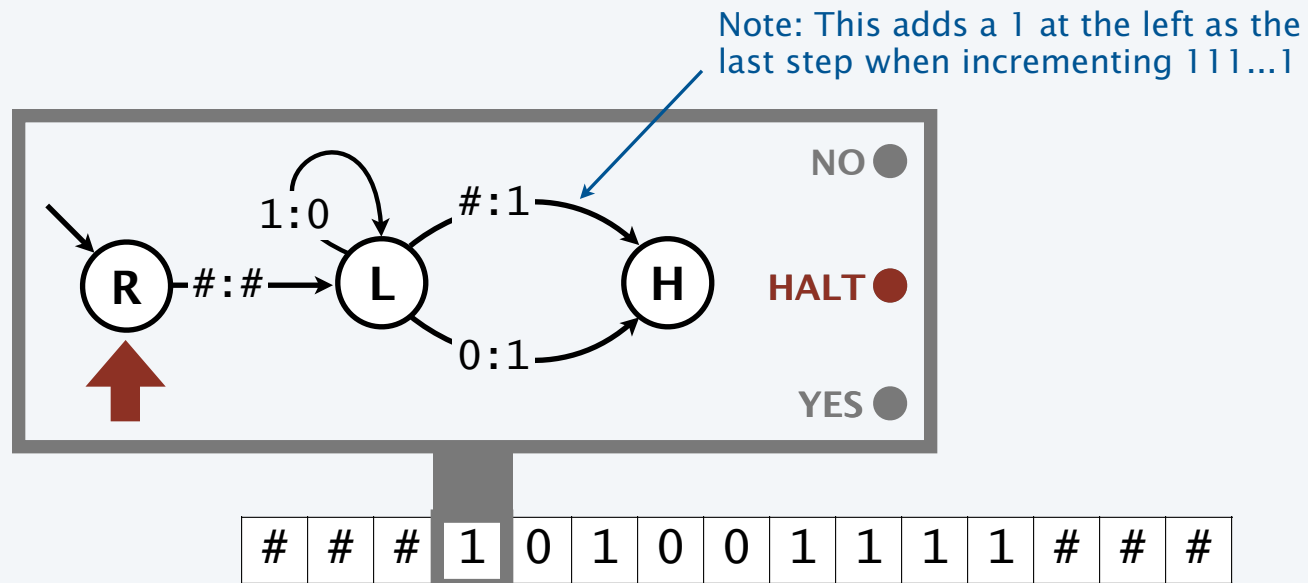
Q. What happens when we try to decrement 0?



A. Doesn't halt! TMs can have bugs, too.

Fix to avoid infinite loop. Check for #.

## TM example 2: Binary incrementer



Input 

1	0	1	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---

Output 

1	0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---

## TM example 3: Binary adder (method)

To compute  $x + y$

- Move right to right end of  $y$ .
- Decrement  $y$ .
- Move left to right end of  $x$  (left of  $+$ ).
- Increment  $x$ .
- Continue until  $y = 0$  is decremented.
- Clean up by erasing  $+$  and 1s.

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

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

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

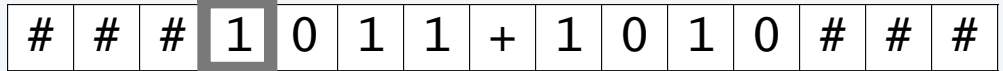
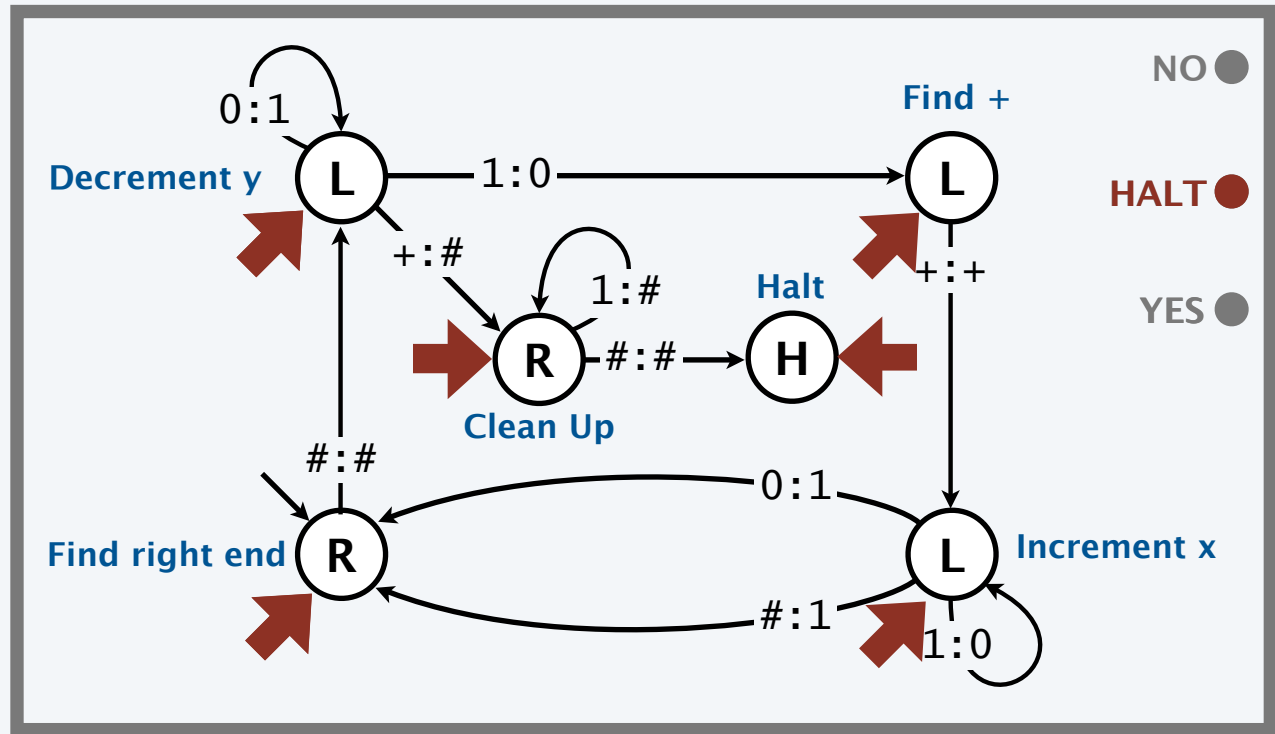
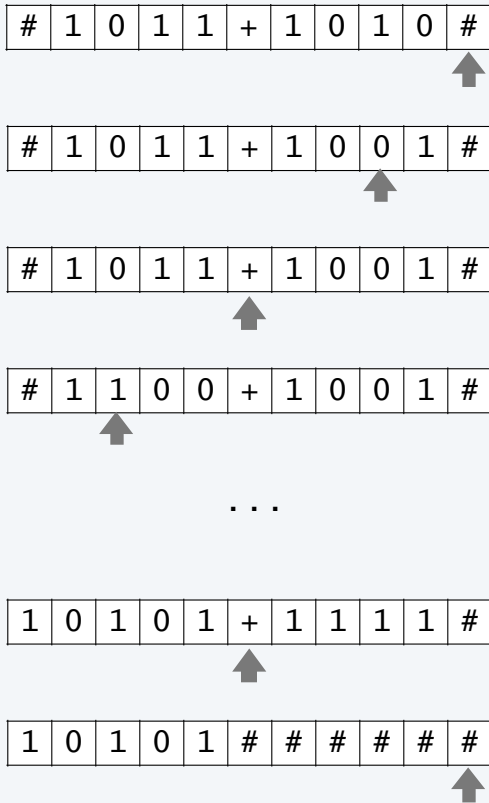
... # # 1 1 0 0 + 1 0 0 1 # # ...

... # 1 0 1 0 1 + 1 1 1 1 # # ...  
  
 ← Found + when seeking 1? Just decremented 0.

... # 1 0 1 0 1 # # # # # # ...  
 Clean up →



# TM example 3: Binary adder



## Simulating an infinite tape with two stacks

Q. How can we simulate a tape that is infinite on both ends?

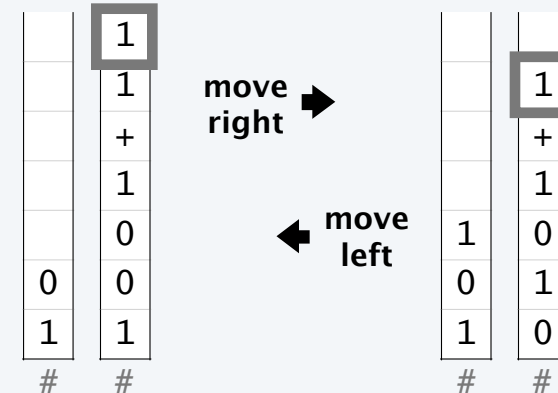
A. Use two stacks, one for each end.

```
private Stack<Character> left;
private Stack<Character> right;
private char read()
{
    if (right.isEmpty()) return '#';
    else return right.pop();
}
private char write(char c)
{ right.push(c); }
private void moveRight()
{
    if (right.isEmpty())
        left.push('#')
    else left.push(right.pop());
}
private void moveLeft()
{
    if (left.isEmpty())
        right.push('#')
    else right.push(left.pop());
}
```

to simulate TM tape,  
need to call write()  
exactly once after  
each call on read()

#	#	#	1	0	1	1	+	1	0	1	0	#	#	#
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

"tape head" is top of right stack



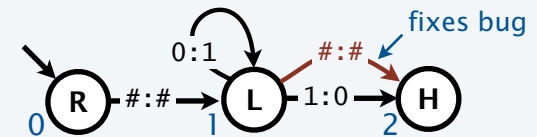
empty? assume # is there

# Simulating the operation of a Turing machine

```

public class TuringMachine
{
    private int state;
    private int start;
    private char[] action;
    private ST<Character, Integer>[] next;
    private ST<Character, Character>[] out;
    /* Stack code from previous slide */
    public TM(String filename)
    { /* Fill in data structures */ }
    public String simulate(String input)
    {
        state = start;
        for (int i = input.length()-1; i >= 0; i--)
            right.push(input.charAt(i));
        while (action[state] != 'H')
        {
            char c = read();
            write(out[state].get(c));
            state = next[state].get(c);
            if (action[state] == 'R') moveRight();
            if (action[state] == 'L') moveLeft();
        }
        return action[state];
    }
    public static void main(String[] args)
    { /* Similar to DFA's main() */ }
}

```



action[]		next[]				out[]			
		0	1	#		0	1	#	
0	R	0	0	0	1	0	0	1	#
1	L	1	1	2	2	1	1	0	#
2	H	2	2	2	2	2	0	1	#

entries in gray are implicit in graphical representation

# states, alphabet, start  
 type, transitions, output  
 for each state

```

% more dec.txt
3 01# 0
R 0 0 1 0 1 #
L 1 2 2 1 0 #
H 2 2 2 0 1 #

% java TM dec.txt
000111
000110
010000
001111
000000
111111

```



COMPUTER SCIENCE  
SE D G E W I C K / W A Y N E

CS.15.B.Turing.Machine

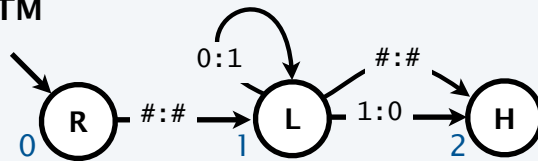
## 15. Turing Machines

- Context
- A simple model of computation
- **Universality**
- Computability
- Implications

## Representing a Turing machine

**Turing's key insight.** A TM is nothing more than a finite sequence of symbols.

decrementer TM

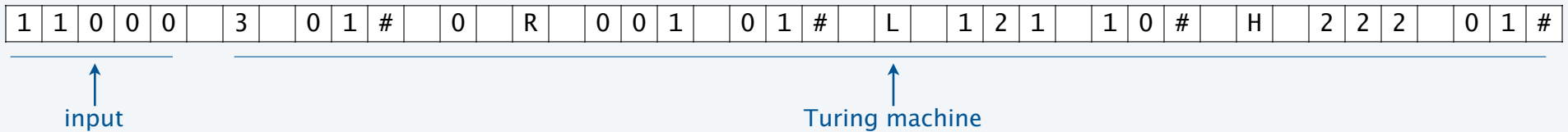


dec.txt

```

3 01# 0
R 0 0 1 0 1 #
L 1 2 2 1 0 #
H 2 2 2 0 1 #
  
```

**Implication.** Can put a TM and its input on a TM tape.

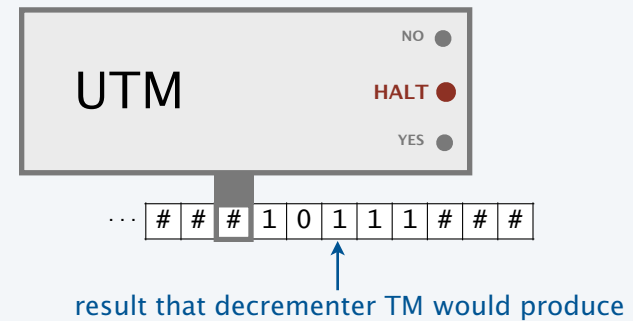
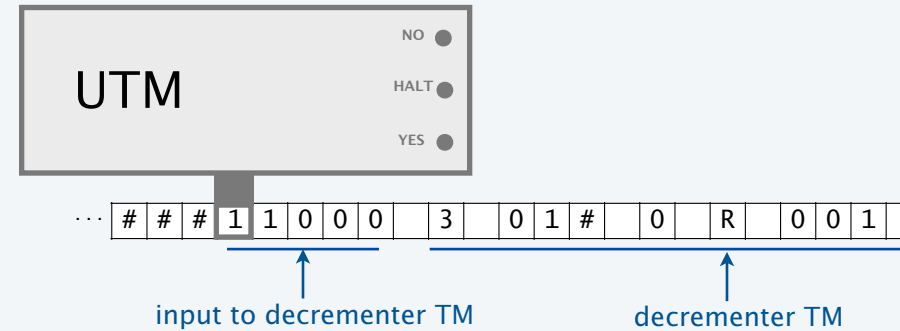


**Profound implication.** We can use a TM to simulate the operation of any TM.

## Universal Turing machine (UTM)

**Universal Turing machine.** A TM that takes as input any TM and input for that TM on a TM tape.

**Result.** Whatever would happen if that TM were to run with that input (could loop or end in Y, N, or H).



**Turing.** Simulating a TM is a simple computational task, so there exists a TM to do it: A UTM.

**Easier for us to think about.** Implement Java simulator as a TM.

# Implementing a universal Turing machine

Java simulator gives a roadmap

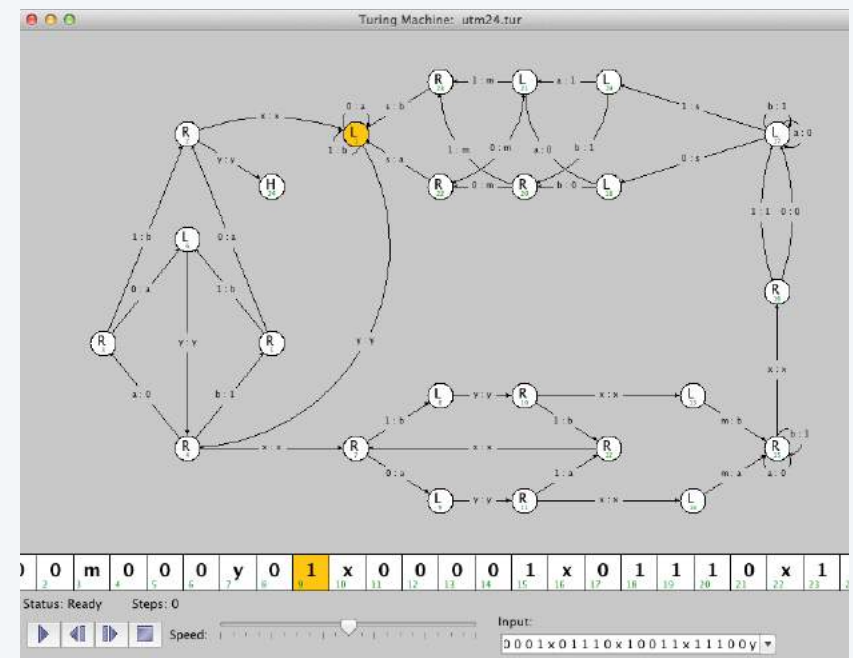
- No need for a constructor as in Java because everything is already on the tape.
- Simulating the infinite tape is a bit easier than in Java because TM has an infinite tape.
- Critical part of the calculation is to update **state** as indicated.

Want to see the details or build your own TM?  
Use the booksite's TM development environment.



**Warning.** TM development may be addictive.

## A 24-state UTM



Amazed that it's only 24 states?  
The record is 4 states, 6 symbols.



# Universality

UTM: A *simple* and *universal* model of computation.

**Definition.** A task is **computable** if a Turing machine exists that computes it.

**Theorem (Turing, 1936).** *It is possible to invent a single machine which can be used to do any computable task.*



## Profound implications

- Any machine that can simulate a TM can simulate a universal Turing machine (UTM).
- Any machine that can simulate a TM can do *any* computable task.
- Don't need separate devices for solving scientific problems, playing music, email, . . .



# Conway's game of life

## A cellular automaton

- Cells live and die in an infinite square grid.
- Time proceeds in discrete steps.

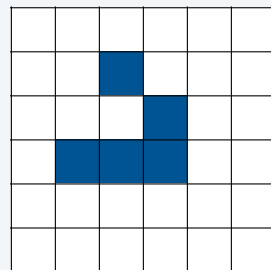
**Survival.** Each cell lives or dies depending on its 8 neighbors:

- Too few neighbors alive? (0 or 1) Die of loneliness.
- Number of living neighbors just right (2 or 3) Survive to next generation.
- Too many neighbors alive? (more than 3) Die of overcrowding.

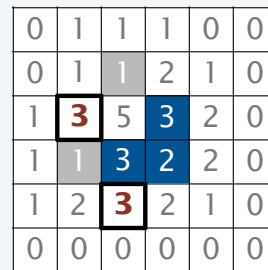
**Birth.** Cell born when it has exactly 3 living neighbors.



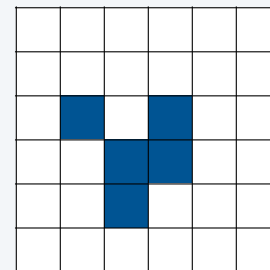
John Horton Conway



time t



living neighbors

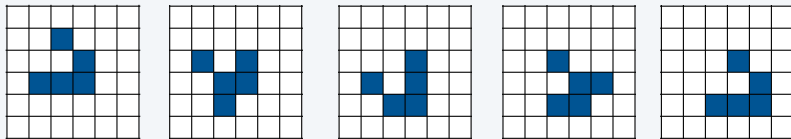


time t+1

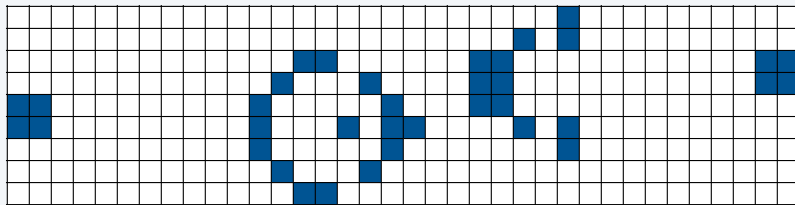
# Conway's Game of Life

**Lesson.** Simple rules can lead to complicated behavior

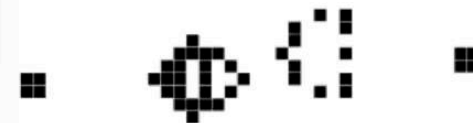
Example 1. Glider



Example 2. Glider gun (generates gliders)



Example 3. Glider gun breeder (generates glider guns)

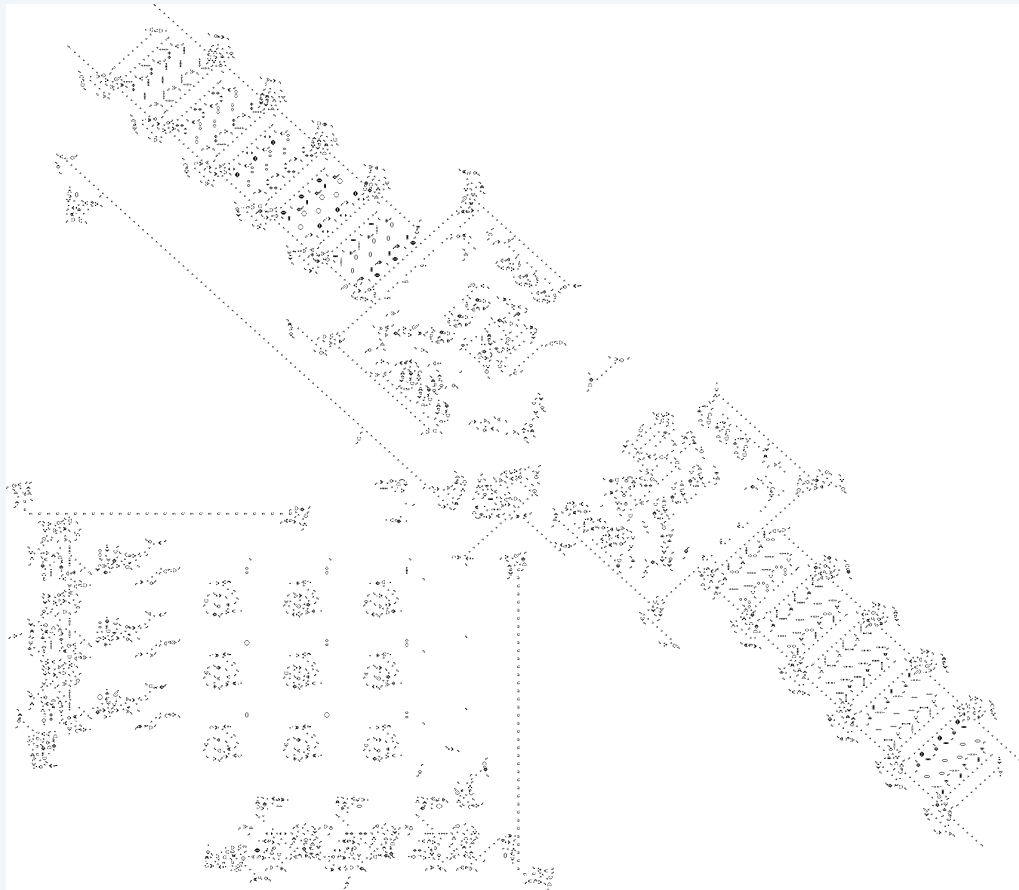


**Note.** YOU can write a program for the game of life (might have been an assignment).

## A complex initial configuration for the game of life

---

Q. What happens with this starting configuration?



A. Anything we can compute!  
(It is a UTM).

## A profound connection to the real world

**Church-Turing thesis.** Turing machines can do anything that can be described by *any* physically harnessable process of this universe: *All computational devices are equivalent.*

### Remarks

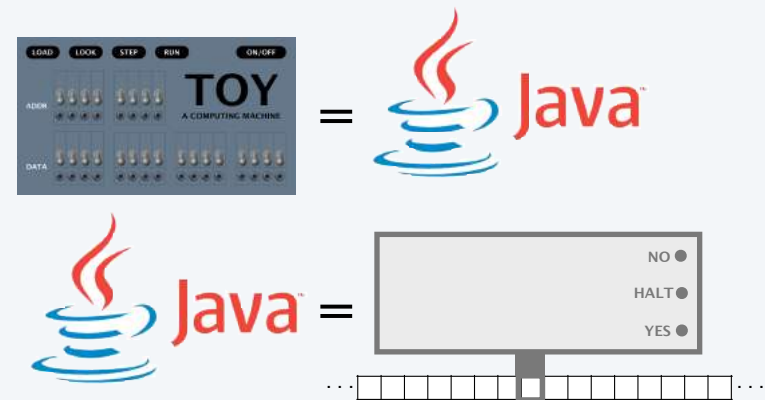
- A thesis, not a theorem.
- *Not* subject to proof.
- *Is* subject to falsification.

### New model of computation or new physical process?

- Use *simulation* to prove equivalence.
- Example: TOY simulator in Java.
- Example: Java compiler in TOY.

### Implications

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



## Evidence in favor of the Church-Turing thesis

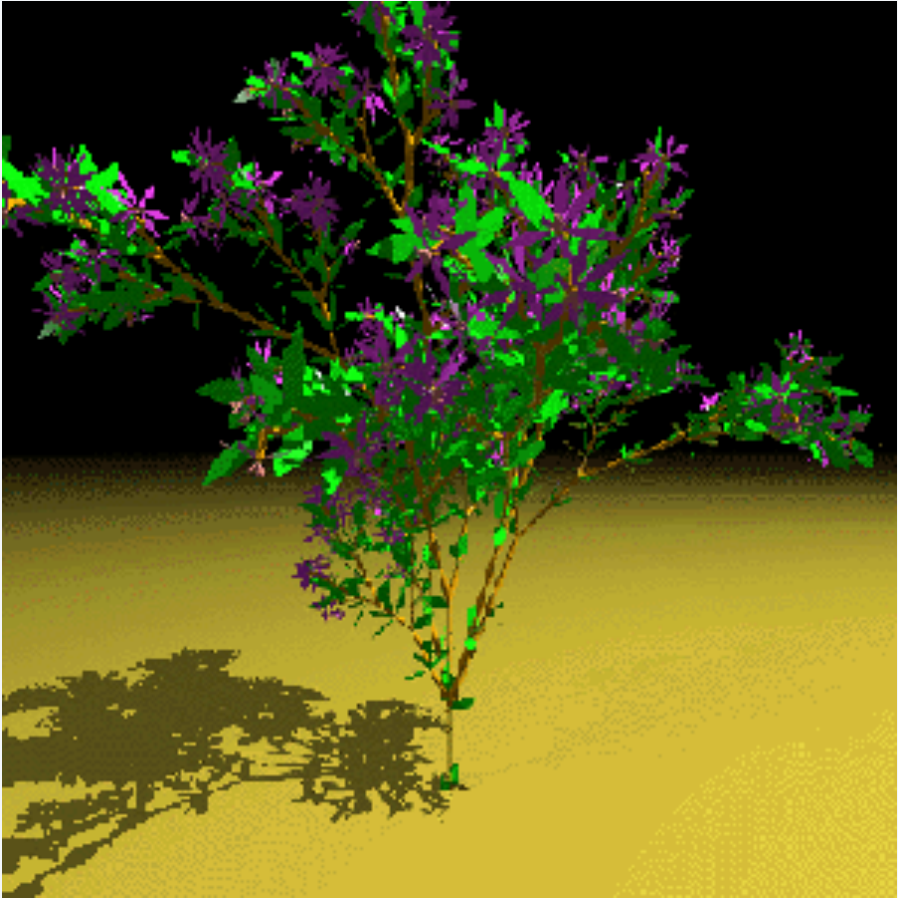
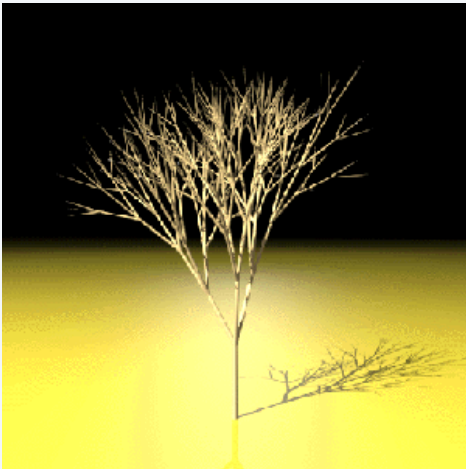
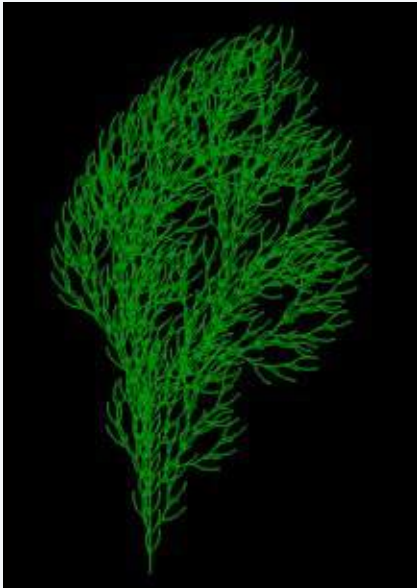
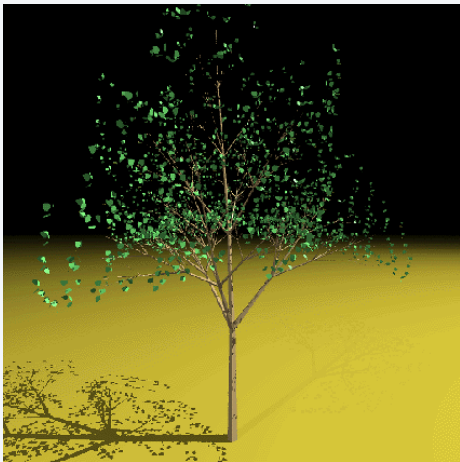
---

**Evidence.** Many, many models of computation have turned out to be equivalent (**universal**).

<i>model of computation</i>	<i>description</i>
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 Lindenmayer 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
PCP systems	string matching puzzles (stay tuned)

8 decades without a counterexample, and counting.

Example of a universal model: Extended Lindenmayer systems for synthetic plants



*Image sources*

<http://astronomy.swin.edu.au/~pbourke/modelling/plants>



## 15. Turing Machines

- Context
- A simple model of computation
- Universality
- **Computability**
- Implications

## Post's correspondence problem (PCP)

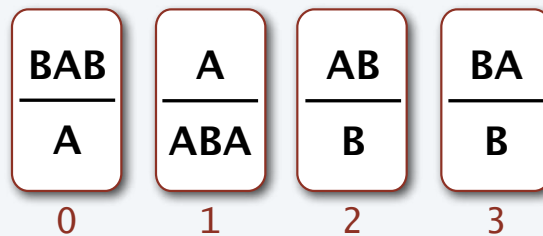
---

**PCP.** A family of puzzles, each based on a set of cards.

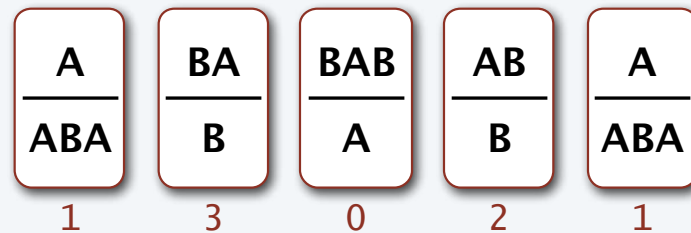
- $N$  types of cards.
- No limit on the number of cards of each type.
- Each card has a top string and bottom string.

Does there exist an arrangement of cards with matching top and bottom strings?

Example 1 ( $N = 4$ ).



Solution 1 (easy): YES.



## Post's correspondence problem (PCP)

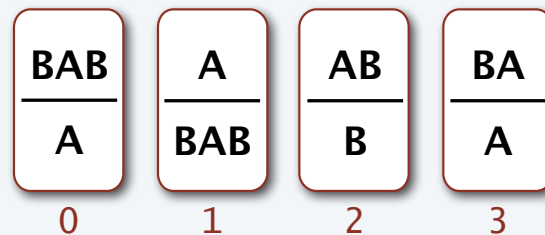
---

**PCP.** A family of puzzles, each based on a set of cards.

- $N$  types of cards.
- No limit on the number of cards of each type.
- Each card has a top string and bottom string.

Does there exist an arrangement of cards with matching top and bottom strings?

Example 2 ( $N = 4$ ).



**Solution 2 (easy):** NO. No way to match even the first character!

## Post's correspondence problem (PCP)

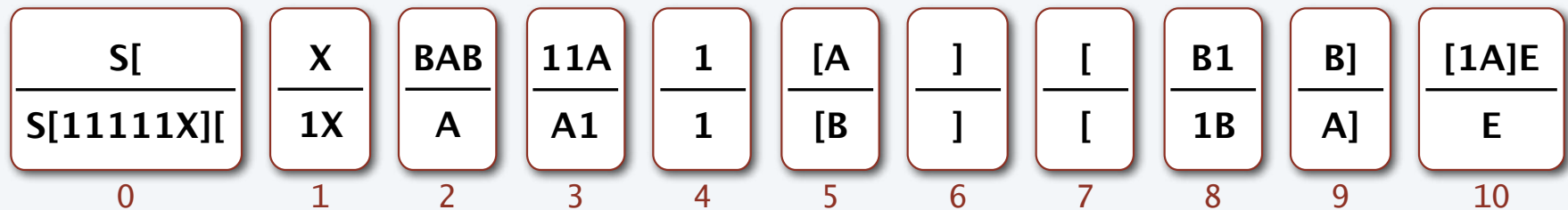
---

**PCP.** A family of puzzles, each based on a set of cards.

- $N$  types of cards.
- No limit on the number of cards of each type.
- Each card has a top string and bottom string.

Does there exist an arrangement of cards with matching top and bottom strings?

**Example 3** (created by Andrew Appel).



**Challenge for the bored:** Find a solution that starts with a card of type 0.

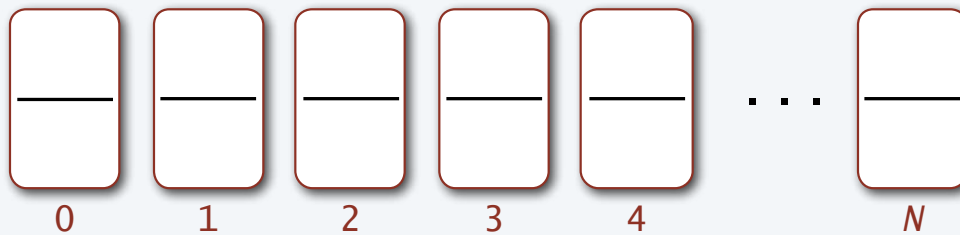
## Post's correspondence problem (PCP)

---

**PCP.** A family of puzzles, each based on a set of cards.

- $N$  types of cards.
- No limit on the number of cards of each type.
- Each card has a top string and bottom string.

Does there exist an arrangement of cards with matching top and bottom strings?



**A reasonable idea.** Write a program to take  $N$  card types as input and solve PCP.

**A surprising fact.** It is *not possible* to write such a program.

## Another impossible problem

**Halting problem.** Write a Java program that reads in code for a Java static method `f()` and an input `x`, and decides whether or not `f(x)` results in an infinite loop.

Example 1 (easy).

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

Halts only if `x` is a positive power of 2

Example 2 (difficulty unknown).

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

Involves *Collatz conjecture*  
(see Recursion lecture)

`f(7):`    7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1  
`f(-17):` -17 -50 -25 -74 -37 -110 -55 -164 -82 -41 -122 ... -17 ...

Next. A proof that *it is not possible* to write such a program.

## Undecidability of the halting problem

---

**Definition.** A yes-no problem is **undecidable** if no Turing machine exists to solve it.  
(A problem is **computable** if a Turing machine does exist that solves it.)

**Theorem (Turing, 1936).** The halting problem is undecidable.

### Profound implications

- There exists a problem that no Turing machine can solve.
- There exists a problem that no *computer* can solve.
- There exist *many problems* that no computer can solve (stay tuned).

## Warmup: self-referential statements

---

Liar paradox (dates back to ancient Greek philosophers).

- Divide all statements into two categories: *true* and *false*.
- Consider the statement "This statement is *false*."
- Is it *true*? If so, then it is *false*, a contradiction.
- Is it *false*? If so, then it is *true*, a contradiction.

Source of the difficulty: Self-reference.

*true*

$2 + 2 = 4$   
The earth is round.  
Starfish have no brains.  
Venus rotates clockwise.  
...  
This statement is false. **X**

*false*

$2 + 2 = 99$   
The earth is flat.  
Earthworms have 3 hearts.  
Saturn rotates clockwise.  
...  
This statement is false. **X**

Logical conclusion. Cannot label *all* statements as *true* or *false*.



## Proof of the undecidability of the halting problem

---

Theorem (Turing, 1936). The halting problem is undecidable.

### Proof outline.

- Assume the existence of a function `halt(f, x)` that solves the problem.

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

← By universality, may as well use Java.  
(If this exists, we could simulate it on a TM.)

- Arguments: A function `f` and input `x`, encoded as strings.
- Return value: `true` if `f(x)` halts and `false` if `f(x)` does not halt.
- `halt(f, x)` always halts.
- Proof idea: *Reductio ad absurdum*: if any logical argument based on an assumption leads to an absurd statement, then the assumption is false.

# Proof of the undecidability of the halting problem

**Theorem (Turing, 1936).** The halting problem is undecidable.

## Proof.

- Assume the existence of a function `halt(f, x)` that solves the problem.
- Create a function `strange(f)` that goes into an infinite loop if `f(f)` halts and halts otherwise.
- Call `strange()` with *itself* as argument.
- If `strange(strange)` halts, then `strange(strange)` goes into an infinite loop.
- If `strange(strange)` does not halt, then `strange(strange)` halts.
- *Reductio ad absurdum.*
- `halt(f, x)` cannot exist.

## Solution to the problem

```
public boolean halt(String f, String x)
{
    if ( /* f(x) halts */ ) return true;
    else return false;
}
```

## A client

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

## A contradiction

`strange(strange)`

**halts?  
does not halt?**



COMPUTER SCIENCE  
SE D G E W I C K / W A Y N E

CS.15.D.Turing.Computability

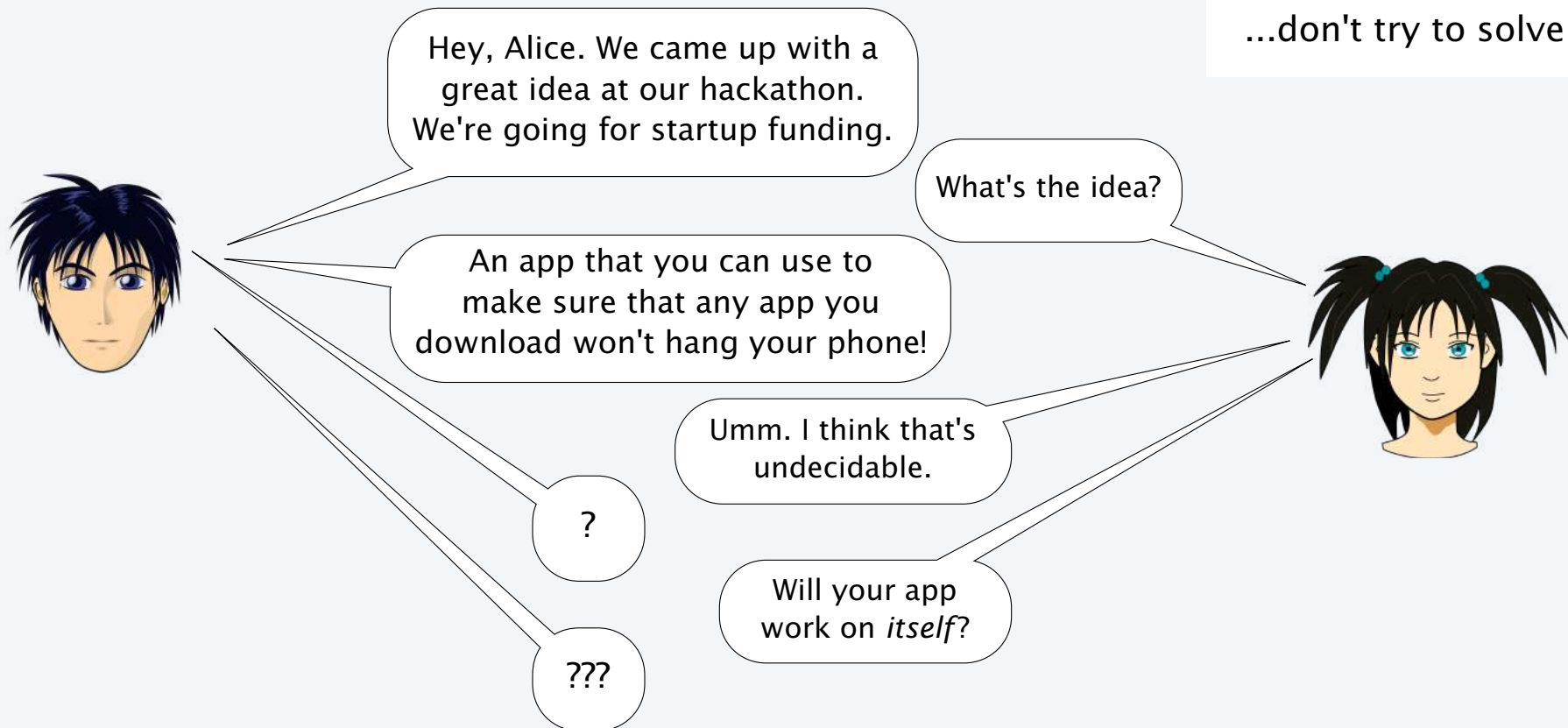
## 15. Turing Machines

- Context
- A simple model of computation
- Universality
- Computability
- **Implications**

## Implications of undecidability

Primary implication. If you know that a problem is undecidable...

...don't try to solve it!



## Implications for programming systems

---

Q. Why is debugging difficult?

A. All of the following 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  $g$  always return same value?

**Uninitialized variables.** Is the variable  $x$  initialized before it's used?

**Dead-code elimination.** Does this statement ever get executed?

**UNDECIDABLE**

↑  
Prove each by reduction from the halting problem: A solution would solve the halting problem.

Q. Why are program development environments complicated?

A. They are programs that manipulate programs.

## Another undecidable problem

The Entscheidungsproblem (Hilbert, 1928) ← "Decision problem"

- Given a first-order logic with a finite number of additional axioms.
- Is the statement provable from the axioms using the rules of logic?

UNDECIDABLE



David Hilbert  
1862–1943

### Lambda calculus

- Formulated by Church in the 1930s to address the Entscheidungsproblem.
- Also the basis of modern functional languages.



Alonzo Church  
1903–1995

Theorem (Church and Turing, 1936). The Entscheidungsproblem is undecidable.

## Another undecidable problem

### Post's correspondence problem (PCP)

PCP. A family of puzzles, each based on a set of cards.

- $N$  types of cards.
- No limit on the number of cards of each type.
- Each card has a top string and bottom string.

Does there exist an arrangement of cards with matching top and bottom strings?



**UNDECIDABLE**

**A reasonable idea.** Write a program to take  $N$  card types as input and solve PCP.

**Theorem (Post, 1946).** Post's correspondence problem is undecidable.



## Examples of undecidable problems from computational mathematics

### Hilbert's 10th problem

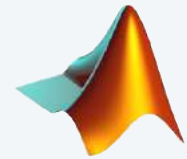
- Given a multivariate polynomial  $f(x, y, z, \dots)$ .
- Does  $f$  have integral roots? (Do there exist integers  $x, y, z$ , such that  $f(x, y, z, \dots) = 0$ ?)

UNDECIDABLE

Ex. 1  $f(x, y, z) = 6x^3yz^2 + 3xy^2 - x^3 - 10$

YES  $f(5, 3, 0) = 0$

Ex. 2  $f(x, y) = x^2 + y^2 - 3$  NO



Ex. 1  $\frac{\cos(x)}{1+x^2}$

YES

$$\int_{-\infty}^{\infty} \frac{\cos(x)}{1+x^2} dx = \frac{\pi}{e}$$

Ex. 2  $\frac{\cos(x)}{1-x^2}$

NO

### Definite integration

- Given a rational function  $f(x)$  composed of polynomial and trigonometric functions.

- Does  $\int_{-\infty}^{\infty} f(x) dx$  exist?

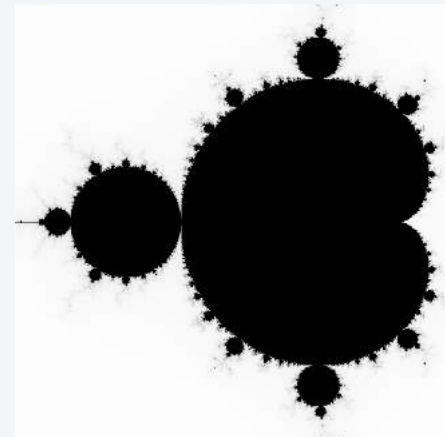
UNDECIDABLE

## Examples of undecidable problems from computer science

### Optimal data compression

- Find the shortest program to produce a given string.
- Find the shortest program to produce a given *picture*.

UNDECIDABLE



produced by a 34-line Java program

### Virus identification

- Is this code equivalent to this known virus?
- Does this code contain a virus?

UNDECIDABLE

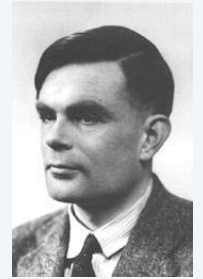
```
Private Sub AutoOpen()  
On Error Resume Next  
If System.PrivateProfileString("", CURRENT_USER\Software  
\Microsoft\Office\9.0\Word\Security",  
"Level") <> "" Then  
CommandBars("Macro").Controls("Security...").Enabled = False  
. . .  
For oo = 1 To AddyBook.AddressEntries.Count  
Peep = AddyBook.AddressEntries(x)  
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 ;-)"  
. . .
```

Melissa virus (1999)

## Turing's key ideas

---

Turing's paper in the *Proceedings of the London Mathematical Society* "On Computable Numbers, With an Application to the Entscheidungsproblem" was one of the most impactful scientific papers of the 20th century.



Alan Turing  
1912–1954

**The Turing machine.** A formal model of computation.

**Equivalence of programs and data.** Encode both as strings and compute with both.

**Universality.** Concept of general-purpose programmable computers.

**Church-Turing thesis.** If it is computable at all, it is computable with a Turing machine.

**Computability.** There exist inherent limits to computation.

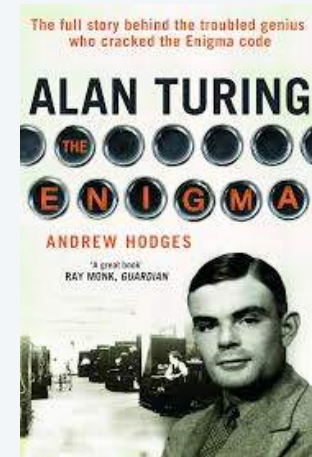
Turing's paper was published in 1936, *ten years before* Eckert and Mauchly worked on ENIAC (!)

John von Neumann read the paper... ← Stay tuned

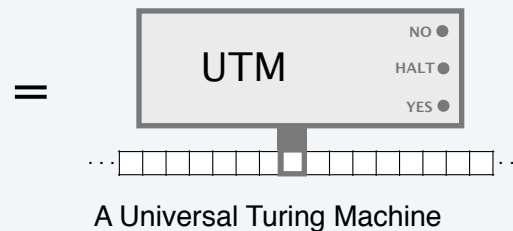
# Alan Turing: the father of computer science

*It was not only a matter of abstract mathematics, not only a play of symbols, for it involved thinking about what people did in the physical world.... It was a play of imagination like that of Einstein or von Neumann, doubting the axioms rather than measuring effects.... What he had done was to combine such a naïve mechanistic picture of the mind with the precise logic of pure mathematics. His machines – soon*

– John Hodges, in *Alan Turing, the Enigma*



A Google data center



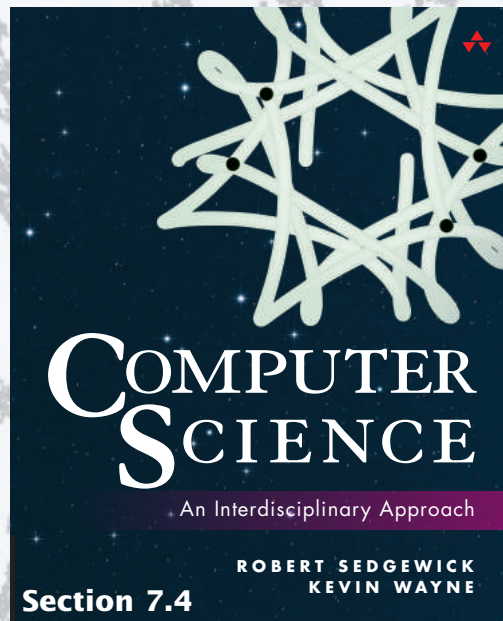


COMPUTER SCIENCE  
SEDEGWICK / WAYNE

CS.15.E.Turing.Implications

**COMPUTER SCIENCE**  
**SEDGEWICK / WAYNE**

PART II: ALGORITHMS, MACHINES, and THEORY



<http://introcs.cs.princeton.edu>

# 15. Turing Machines