Introduction to Turing Machines

SITE : http://www.sir.blois.univ-tours.fr/˜mirian/

Problems that computers cannot solve

- **Problem:** What a program does?

- Not knowing when, if ever, something will occur is the ultimate cause of our inability to tell what a program does

Programs that print ”Hello,Word” (1)

Kernighan and Ritchie’s hello-world program

```
main()
       {
       printf("Hello, world");
       }
```

Programs that print "Hello,Word" (2)

Femat's last theorem expressed as a hello-world program

```
main()
      {
      int n, total, x, y, z;
      scanf("%", &n);
      total = 3;
      while (1) {
            for (x = 1; x <= total -2; x++)
                for (y = 1; y <= total -1 ; y++) {
                    z = total - x -y;
                    if (exp(x, n) + exp (y, n) == exp(z, n))
                    printf ("hello, word");
                }
            total ++
      }
      }
```

Programs that print "Hello,Word" (3)

- The program (Fermat) takes an input n and looks for positive integer solutions to equation

$$x^n + y^n = z^n$$

  - If the program finds a solution, it prints `hello, world`
  - If it never finds integer x, y, z to satisfy the equation, then it continues searching **forever**, and never prints `hello, world`
- If the value of n is 2, then it will find combinations of integers and thus:
  For input $n = 2$ the program prints `hello, world`
- For any integer $n > 2$, the program will never find a triple of positive integers to satisfy $x^n + y^n = z^n$ (300 years to prove!)

Our hello-world problem

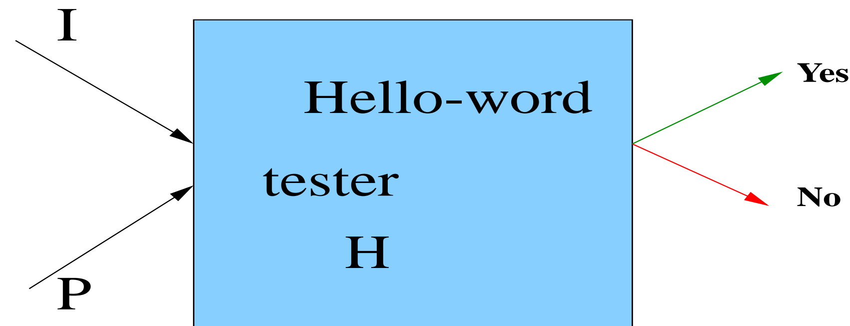
Determine whether a given C program, with a given input, prints `hello, world` as the first 12 characters that it prints

- Is it possible to have a program that proves the correction of programs?
- Any of the problems that mathematicians have not been able to resolve can be turned into a question of the form

  Does this program, with this input, print `hello, world`?

- Is it possible to have a program that could examine any program P and input I for P, and tell whether P, run with I as its input, would print `hello, world` ?

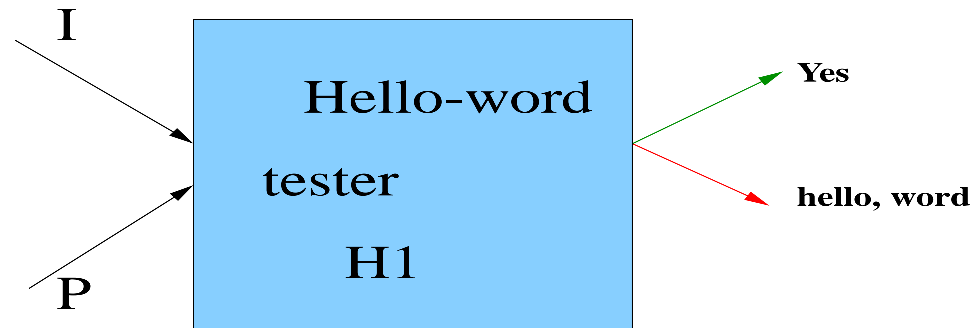
The Hypothetical "Hello World" Tester



- Assume there is a program (H) that takes as input
  - A program P
  - Input I

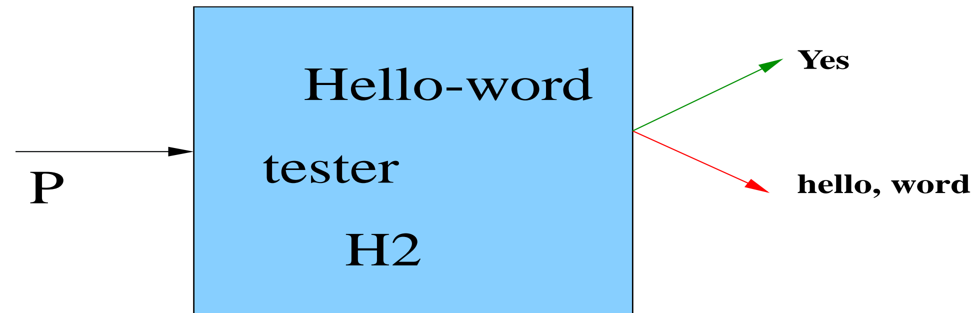
  and tells whether P within input I prints `hello, world`
  (Output is either **Yes** or **No**)
- If a problem has an algorithm like H, that always tells correctly whether an instance of the problem has answer **Yes** or **No**, then the problem is said to be decidable. Otherwise, the problem is undecidable.
- GOAL: To prove that H does not exist (proof by contradiction).

The Hypothetical "Hello World" Tester



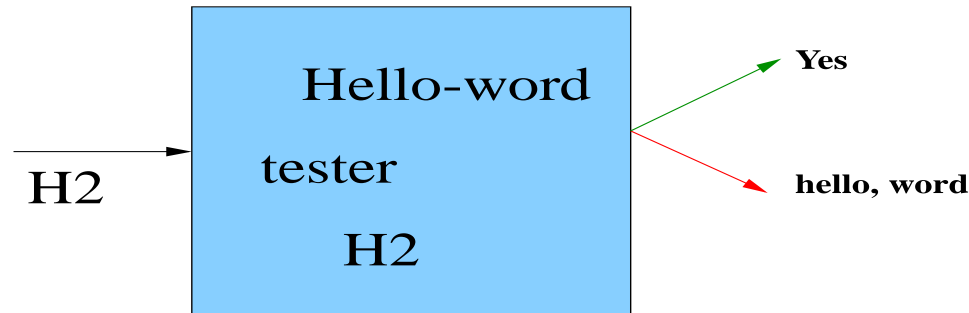
- Assume H exists.
- First modification: $H1$ prints **hello, world** exactly when H would print **No**

The Hypothetical "Hello World" Tester



- We want to restrict $H1$ so it:
  - Takes only input P, not P and I.
  - Asks what P would do if its inputs were its own code, *i.e.*, what would $H1$ do on inputs P **as program** and P **as input** I as well?
- Second modification:
  - $H2$ first reads the entire input P and stores it in an array A.
  - $H2$ then simulates $H1$ but whenever $H1$ would read input from P or I, $H2$ reads from the stored copy in A.

How can we prove that H2 cannot exist?

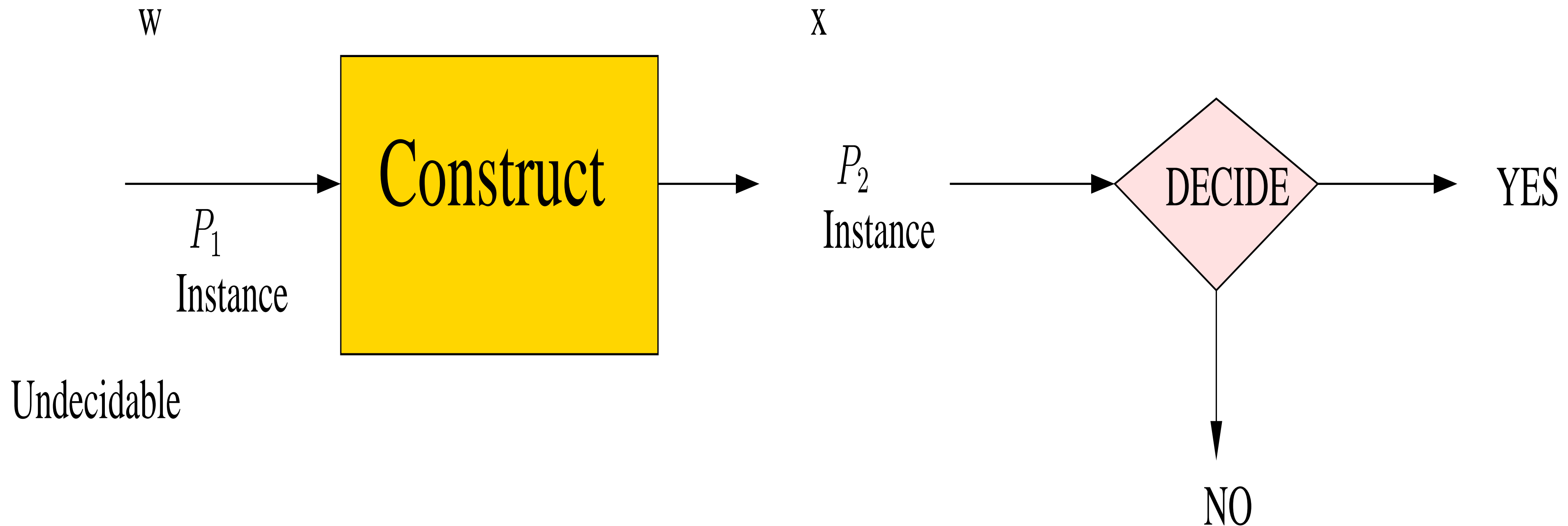


- **What $H2$ does when given itself as input**?
- $H2$, given any program P as input:
  - Makes output **Yes** if P prints `hello, world` when given itself as input.
  - Prints **hello, world** if P (given itself as input) does not print `hello, world`.
- Suppose that $H2$ makes the output **Yes**: Then the $H2$ in the box is saying about its input $H2$ that $H2$ (given itself as input) prints `hello, world` as its first output.
  But we just supposed that the first output $H2$ makes in this situation is **Yes**!
- Thus it appears that the output of the box above is **hello, world**.
  But if H2 (given itself as input) prints `hello, world`, then the output of the box must be **Yes**
- **Whichever output we suppose $H2$ makes, we can argue that it makes the other output.** A PARADOX!!!!

Undecidable Problems and Reduction

- A problem that cannot be solved by computer is called **undecidable**
  - A problem is the question of deciding whether a given string is a member of some particular language.
- How to prove that a given problem is undecidable?
  - Once we have one problem that we know is undecidable, we no longer have to prove the existence of a paradoxical situation.
    It is sufficient to show that if we could solve the new problem, then we could use that solution to solve a problem we already know is undecidable
- This strategy is called **reduction**

Reduction of P_1 to P_2



Let P_1 be an undecidable problem (we know that)
Let P_2 be a new problem that we would like to prove is undecidable as well

- We invent a construction that converts instances of P_1 into instances of P_2, *i.e.*:
  - any string in the language P_1 is converted into a string un the language P_2 and
  - and any string that is **not** in the language of P_1 is converted into a string that is **not** in the language of P_2.

Reduction of P_1 to P_2

- Once we have this construction, we can solve P_1 as follows:
  - Given an instance of P_1, *i.e.*, given a string w that may or may not be in the language P, apply the construction algorithm to produce a string x.
  - Test whether x is in P_2, and give the same answer about w and P_1.
- **If $w \in P_1$, then $x \in P_2$; so the algorithm says YES**
- **If $w \notin P_1$, then $x \notin P_2$; so the algorithm says NO**
- Since we assumed that NO algorithm to decide the membership of a string in P_1 exists, we have a proof BY CONTRADICTION, that the hypothetical decision algorithm for P_2 does not exist
  P_2 **is undecidable**

What are you doing?

- We prove that:
  If P_2 is decidable then P_1 is decidable
- This is the same as to prove the contrapositive of the above statement:
  If P_1 is undecidable then P_2 is undecidable

Propose of the theory of undecidable problems

To provide guidance to programmers about what they might or might not be able to accomplish through programming

Pragmatic impact:

- **Intractable problems:**
  - Decidable problems.
  - Require large amount of time to solve them.
  - Contrary to undecidable problems, which are usually rarely attempted in practice, the intractable problems are *faced everyday*.
  - Yield to small modifications in the requirements or to heuristc solutions.

- We need tools that allow us to prove everyday questions undecidable or intractable
- We need to rebuild our theory of undecidability, based not on C programs, but based on a very simple model of computer called **Turing Machine**

The Turing Machine

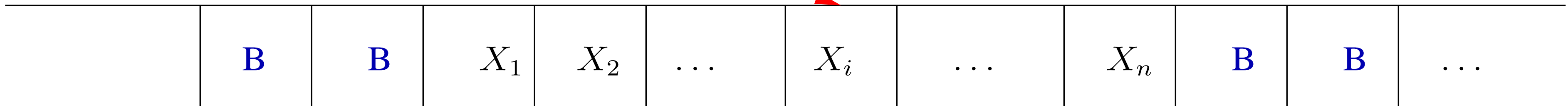
Informally...

- TM is essentially a **finite automaton** that has a single **tape of infinite length** on which it may read and write data.
- Advantage of TM over programs as representation of what can be computed: TM is sufficiently simple that we can represent its configuration precisely, using a simple notation much like ID's of a PDA.

History

- 1936: Alan Turing proposed the TM as a model of *any possible computation*.
- This model is computer-like, rather than program-like, even though true electronic or electromechanical computers were several years in the future.
- All the serious proposals for a model of computation have the same power; *i.e.*, they compute the same functions and recognize the same languages.

Notation for the TM



- **Finite control:** can be in any of a finite set of states
- **Tape:** divided into cells; each cell can hold one of a finite number of symbols.
- Initially the **input** (a finite-length string) is placed on the tape
- All other tape cells initially hold a special symbol: **blank**
- **Blank** is tape symbol (not an input symbol)
- **Tape head:** always positioned at one of the tape cell. Initially, the tape head is at the leftmost cell that holds the input.

A move of the TM

A move of the TM is a function of the state of the finite control and the tape symbol scanned. In one move the TM will

1. Change state
2. Write a tape symbol in the cell scanned.
3. Move the tape head left or right

TM: Formal notation

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

- Q: The finite set of states of the finite control
- Σ: The finite set of *input symbols*
- Γ: The complete set of *tape symbols*;
  Σ is always a subset of Γ
- δ: The transition function
  The arguments of $\delta(q, X)$ are: a state q and a tape symbol X.
  The value of $\delta(q, X)$, if it is defined, is (p, Y, D) where:
  - p is the next state, in Q
  - Y is the symbol, in Γ, written in the cell being scanned, replacing whatever symbol was there.
  - D is a *direction* (either L or R), telling us the direction in which the head moves.
- q_0: The *start state* ($q_0 \in Q$) in which the finite control is found initially.
- B: blank symbol ($B \in \Gamma$ but $B \notin \Sigma$).
- F: the set of *final* or *accepting* states ($F \subseteq Q$).

Instantaneous Descriptions for TM

We use the instantaneous description to describe the configuration.

An ID is represented by the string:

$$X_1X_2\dots X_{i-1}qX_iX_{i+1}\dots X_n$$

where:

1. q is the state of the TM.
2. The tape head is scanning the ith symbol from the left.
3. $X_1X_2\dots X_n$ is the portion of the tape between the leftmost and the rightmost **nonblank**

Moves (1)

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$.
- We use the notation $\vdash_M$ (or $\vdash$) to represent moves of a TM M from one configuration to another.
- $\overset{*}{\vdash}_M$ is used as usual.

The next move is leftward:
If $\delta(q, X_i) = (p, Y, L)$ then:

$$X_1 X_2 \dots X_{i-1}\ q\ X_i X_{i+1} \dots X_n \vdash_M X_1 X_2 \dots X_{i-2}\ p\ X_{i-1}\ Y\ X_{i+1} \dots X_n$$

Exceptions:

1. If $i = 1$, then M moves to the blank to the left of X_1

$$q X_1 X_2 \dots X_n \vdash_M p\ \mathbf{B}\ Y\ X_2 \dots X_n$$

2. If $i = n$ and $Y = B$, then the symbol B written over X_n joins the infinite sequence of trailing blanks and does not appear in the next ID.

$$X_1 \dots X_{n-1} q X_n \vdash_M X_1 X_2 \dots X_{n-2}\ p\ X_{n-1}$$

Moves (2)

The next move is rightward:

If $\delta(q, X_i) = (p, Y, L)$ then:

$$X_1 X_2 \ldots X_{i-1}\ q\ X_i\ X_{i+1} \ldots X_n \vdash_M X_1 X_2 \ldots X_{i-2} X_{i-1}\ Y\ p\ X_{i+1} \ldots X_n$$

Exceptions:

1. If $i = n$ then the $i + 1$st cell holds a blank, and that cell was not part of the previous ID.

$$X_1 \ldots X_{n-1}\ q\ X_n \vdash_M X_1 X_2 \ldots X_{n-1}\ Y\ p\ \mathbf{B}$$

2. If $i = 1$ and $Y = B$, then the symbol B written over X_1 joins the infinite sequence of trailing blanks and does not appear in the next ID.

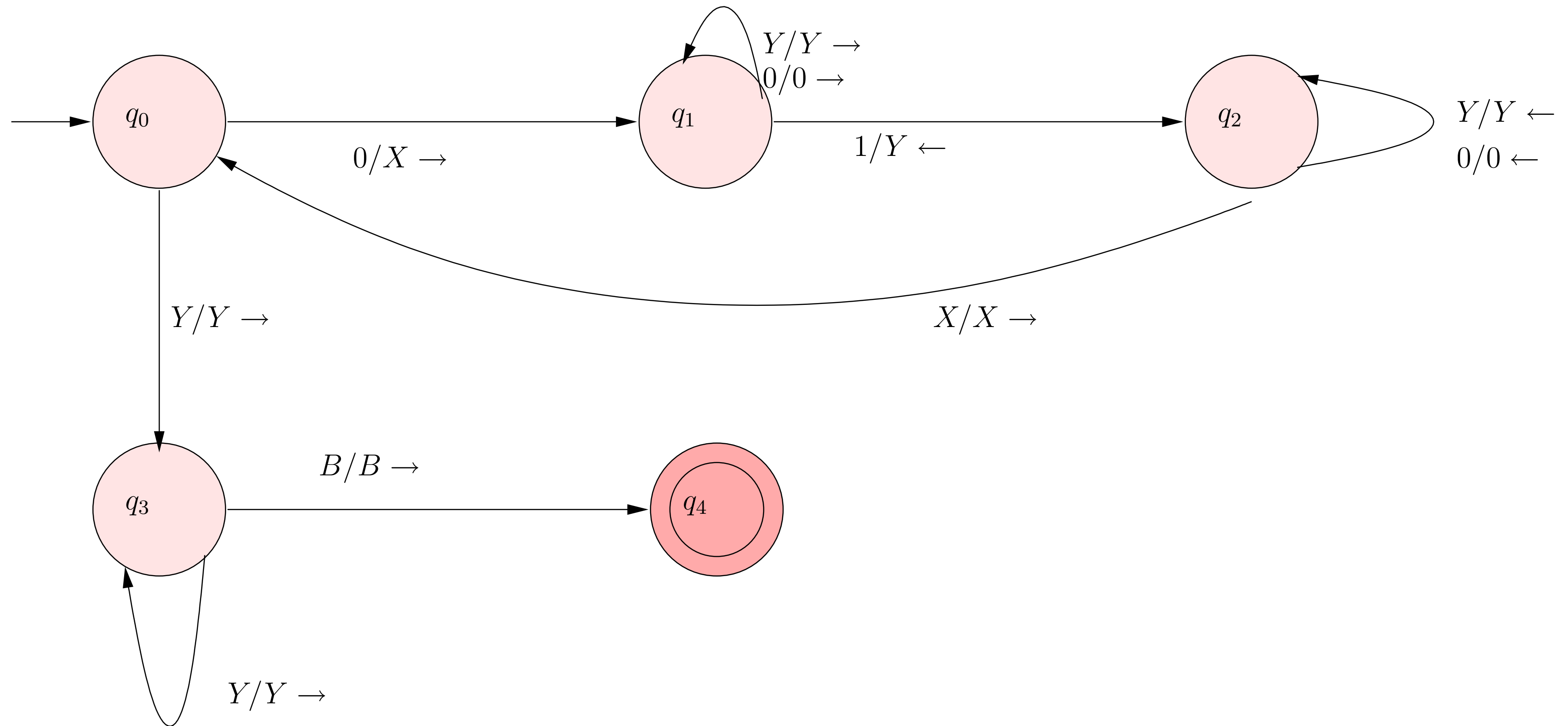
$$q\ X_1 X_2 \ldots X_n \vdash_M p\ X_2 \ldots X_n$$

A TM for the language $\{0^n 1^n \mid n \geq 1\}$

$$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

State	Symbol				
	0	1	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	$(q_1, 0, R)$	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Transition diagram for TM



$q_0 0011 \vdash Xq_1 011 \vdash X0q_1 11 \vdash Xq_2 0Y1 \vdash q_2 X0Y1 \vdash Xq_0 0Y1$
$\vdash XXq_1 Y1 \vdash XXYq_1 1 \vdash XXq_2 YY \vdash Xq_2 XYY \vdash XXq_0 YY$
$\vdash XXYq_3 Y \vdash XXYYq_3 B \vdash XXYYBq_4 B$

The language of a TM

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a TM. Then $L(M)$ is the set of strings $w \in \Sigma^*$ such that

$$q_0\, w \overset{*}{\vdash} \alpha\, p\, \beta$$

for some state $p \in F$ and any tape string α and β.

The language of a TM

- The set of languages we can accept using a TM is often called the recursively enumerable languages or RE languages.
- There is another notion of *acceptance* that is commonly used for TM: **acceptance by halting**.
- We say a TM **halts** if it enters a state q scanning a tape symbol X, and there is no move in this situation, *i.e.*, $\delta(q, X)$ is undefined.
  - We can always assume that a TM halts if it accepts
  - Unfortunately, it is not always possible to require that a TM halts even if it does not accept
- **Recursive languages:** Languages for which TM do halt, regardless of whether or not they accept
  TM that always halt, regardless of whether or not they accept, are good model of an algorithm
- If an algorithm to solve a given problem exists, then we say the problem is **decidable**.

Alan Turing, the father of modern computer science. - Biography

- June 23, 1912 - June 7, 1954: English mathematician, logician, and cryptographer
- With the Turing test, Turing made a significant and characteristically provocative contribution to the debate regarding artificial intelligence: whether it will ever be possible to say that a machine is conscious and can think.
- He provided an influential formalisation of the concept of the algorithm and computation with the Turing machine, formulating the now widely accepted "Turing" version of the Church-Turing thesis: **any practical computing model has either the equivalent or a subset of the capabilities of a Turing machine**.
- During World War II, Turing worked at Britain's code breaking centre, and was for a time head of the section responsible for German Naval cryptanalysis. He devised a number of techniques for breaking German ciphers, including the method of the bombe, an electromechanical machine which could find settings for the Enigma machine.
- After the war, he worked at the National Physical Laboratory, creating one ofthe first designs for a stored-program computer, although it was never actually built. In 1947 he moved to the University of Manchester to work, largely on software, on the Manchester Mark I then emerging as one of the world's earliest true computers.
- In 1952, Turing was convicted of "acts of gross indecency" after admitting to a sexual relationship with a man in Manchester. He was placed on probation and required to undergo hormone therapy. Turing died after eating an apple laced with cyanide in 1954, sixteen days short of his 42nd birthday. His death is regarded by most as an act of suicide.