# Chapter 3 - The Church Turing Thesis

We've seen several computational models:

- Finite automata (DFA, NFA, GNFA): good models for devices that have a small amount of memory
- PDA: good models for devices that have an unlimited memory that is usable in LIFO manner

However, we saw that some very simple tasks are beyond the capabilities of these models (recall P.L.) Hence they are too restricted to serve as models of general purpose computers.

## 3.1 - Turing machines (T.M.)

More powerful computational model:

- Proposed in 1936 (just before World War II) by Alan Turing who is considered one of the fathers of computer science (have you seen "The imitation game?")
- Similar to a PDA but with an unlimited and unrestricted memory (No LIFO = P)
- Much more accurate model of a general purpose computer: can do everything that a real computer can do.

T.M. uses an infinite tape as its unlimited memory:

- Tape head can read/write symbols and move around on the tape.

- Initially: tape contains only the input string and is blank everywhere else.

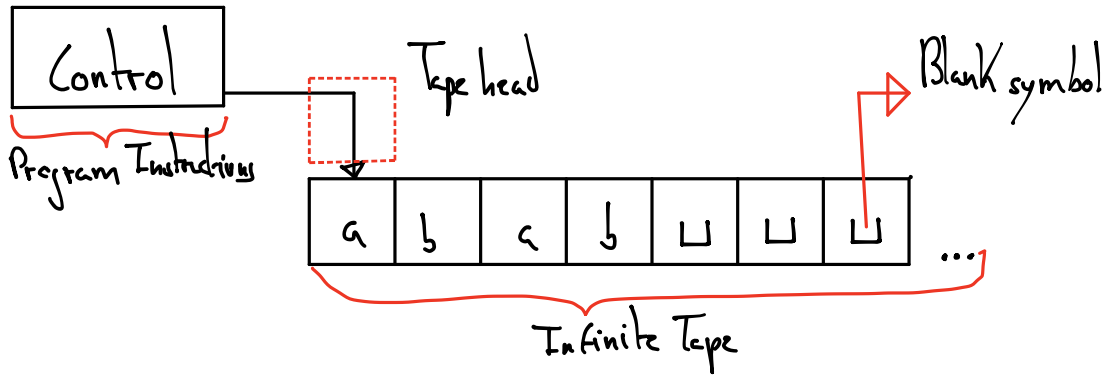
- To read the information that it has written: machine can move its head back over it.

On the computation of a TM:

- Machine computes until it decides to produce an output.

- The outputs "accept" and "reject" are obtained by entering designated accepting and rejecting states.

- If the machine does not enter accepting/rejecting state: will go on forever, never halting (Entscheidungs problem/halting problem).



## Key differences between finite automata and TMs:

① A TM can both write on the tape and read from it.

② The read-write head can move both to the left and to the right (which is ≠ from LIFO)

③ The tape is infinite.

④ The special states for rejecting and accepting take effect immediately.

**Example**

Separated by the # symbol

Let's introduce a TM $M_1$ for $B = \{w\#w \mid w \in \{0,1\}^*\}$.

### Main idea for $M_1$:

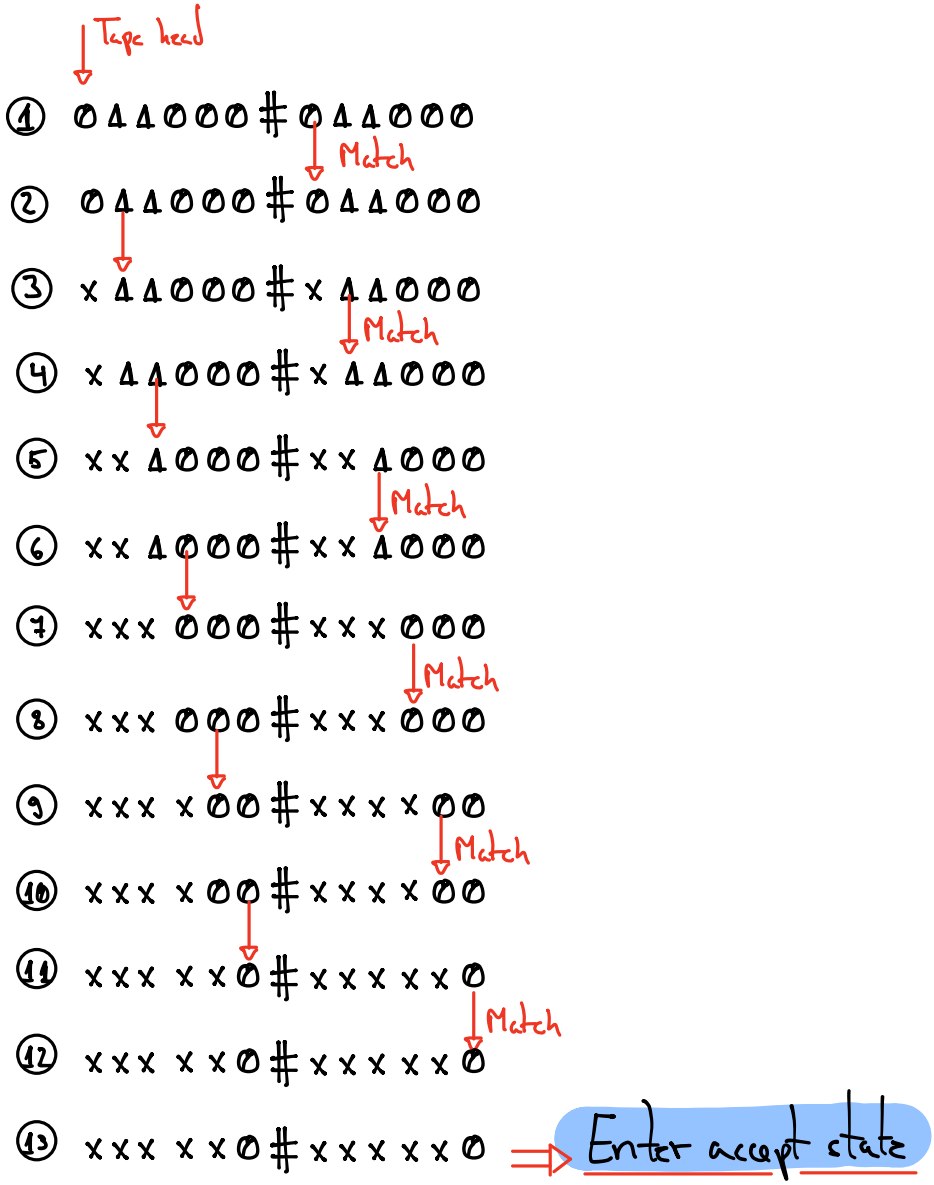
① We want $M_1$ to accept the input if it is a member of B and reject otherwise.

② We are allowed to move back and forth over the input that is stored on the tape and make marks on it.

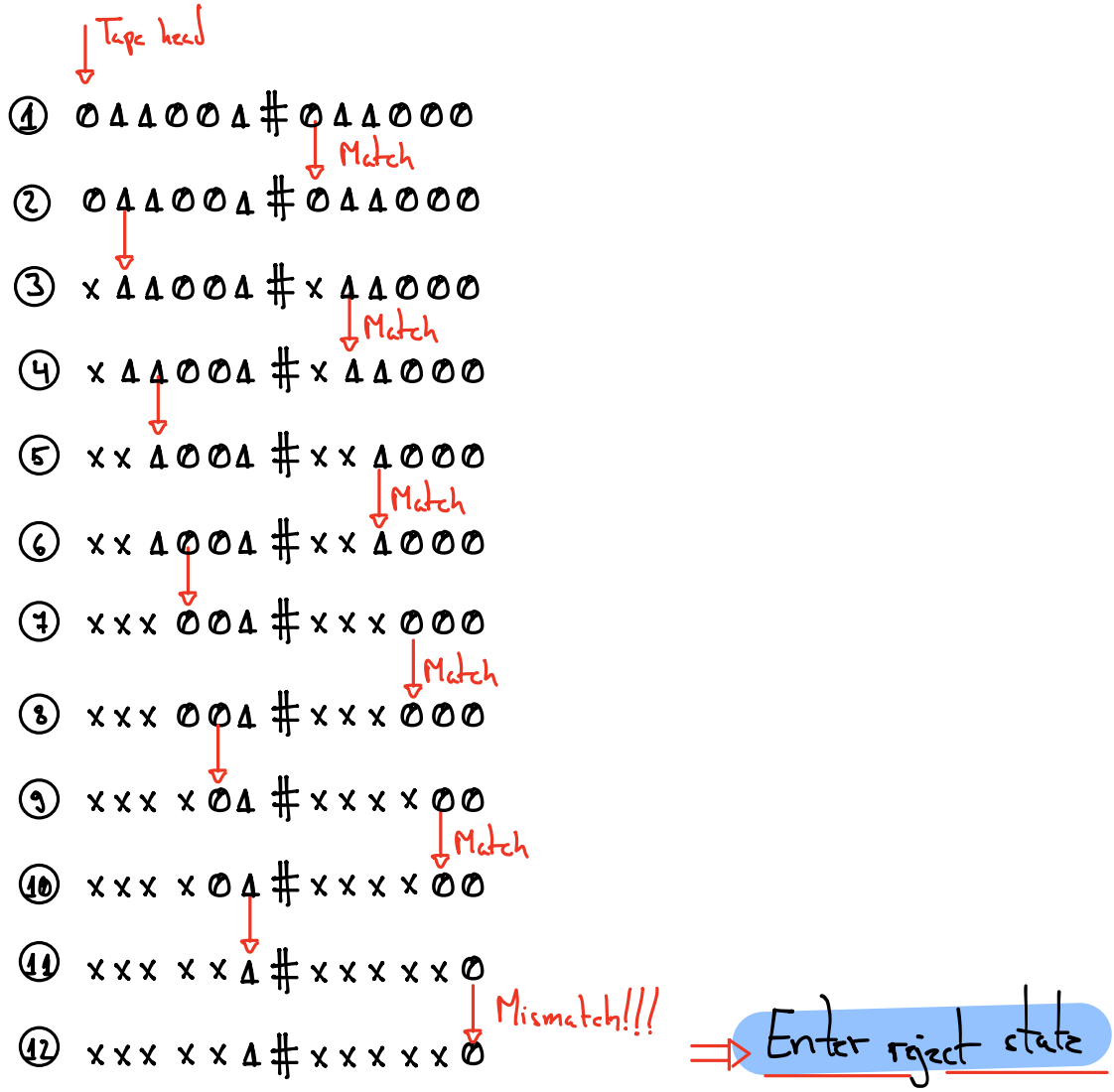
③ Strategy: Check whether the corresponding places on the two sides of the # symbol match. Place marks on the tape to keep track of which places correspond.

④ If all symbols are matched: accept, otherwise, reject

Consider $M_3$'s behavior on input 0∆∆000#0∆∆000:



Consider $M_1$'s behavior on input 0△△00△#0△△000:



# Formal definition of a TM

## Definition 3.3

A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject})$ where $Q, \Sigma, \Gamma$ are all finite sets and:

1. $Q$ is the set of states
2. $\Sigma$ is the input alphabet not containing the blank symbol $\sqcup$
3. $\Gamma$ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function
5. $q_0 \in Q$ is the start state
6. $q_{accept} \in Q$ is the accept state
7. $q_{reject} \in Q$ is the reject state ($q_{accept} \neq q_{reject}$)

Q: But wait! What does this all mean?

$\delta$ takes the form $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, i.e. if $\delta(q, a) = (r, b, \{L, R\})$ then:

- $q \in Q$ is the current state
- $a \in \Gamma$ is the tape symbol being read
- $r \in Q$ is the next state
- $b \in \Gamma$ is the symbol written on the tape
- L or R indicates whether to move the input head to the left or right after writing.

A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ computes as follows:

1. M receives its input $w = w_1 w_2 \ldots w_n \in \Sigma^*$ on the leftmost n squares of the tape, and the rest of the tape is filled with $\sqcup$ symbols.

2. Head starts on the leftmost square of the tape.

3. Once M has started, the computation proceeds according to the $\delta$ rules.

4. If M ever tries to move its head to the left off the left-hand end of the tape, then the head stays in the same place for that move, even though $\delta$ indicates L

5. Computation continues until it enters either the accept or reject states, at which point it halts. If neither occurs, M goes on forever.

As a TM computes, changes occur in:

- current state
- current tape contents
- current head location

} This is a **configuration** of the TM

Configurations are represented in a special way $u\,q\,v$ where:

- $u$ is a string over the tape alphabet $\Gamma$ (i.e. $u \in \Gamma^*$) representing the symbols that have been read.

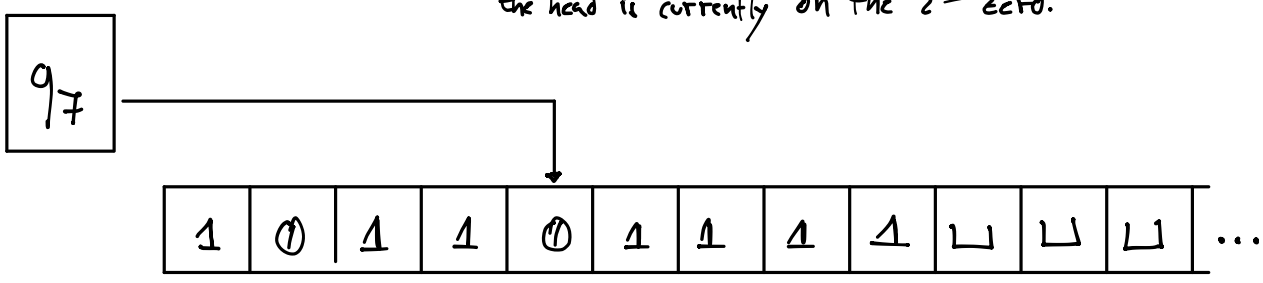
- $q$ is the current state of the machine after reading $u$.

- $v$ is a string over the tape alphabet $\Gamma$ (i.e. $v \in \Gamma^*$) representing the symbols that remain to be read.

The tape contains only blanks $\sqcup$ after the last symbol of $v$.

**Configuration example:**

$1011\ q_7\ 01111$ $\Rightarrow$ Configuration when the tape is 10110111, the current state is $q_7$ and the head is currently on the 2nd zero.



# Formalization of a TM computation:

- Configuration $C_1$ yields configuration $C_2$ if the TM can legally go from $C_1$ to $C_2$ in a single step. Namely, let:

  - $a, b$ and $c \in \Gamma$
  - $u$ and $v \in \Gamma^*$
  - $q_i$ and $q_j \in Q$
  - $C_1 = u\, a\, q_i\, b\, v$
  - $C_2 = u\, q_j\, a\, c\, v$

- Then $u a q_i b v$ yields $u q_j a c v$ if $\delta(q_i, b) = (q_j, c, L)$ (left head movement)

- Then $u a q_i b v$ yields $u a c q_j v$ if $\delta(q_i, b) = (q_j, c, R)$ (right head movement)

- Special cases: When head is at one of the ends of the configuration.

  ① Left-hand end:

    ①.① Left moving transition: Configuration $q_i b v$ yields $q_j c v$

    ①.② Right moving transition: Configuration $q_i b v$ yields $c q_j v$

  ② Right-hand end: Configuration $u a q_i$ is equivalent to $u a q_i \sqcup$ (blanks follow $u a$). I.e. there will always be a symbol at the right-hand end.

(formalization continues on next page)

- Start configuration: $q_0 w$ (Input String) (machine is in the start state $q_0$ with head at leftmost position of the tape)

- Accepting configuration: State of the configuration is $q_{accept}$

- Rejecting configuration: State of the configuration is $q_{reject}$

} Both of these are halting configurations and do not yield further configurations.

- A TM M accepts input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where:

  1. $C_1$ is the start configuration of M on input w
  2. Each $C_i$ yields $C_{i+1}$,
  3. $C_k$ is an accepting configuration

- $L(M) :=$ the language of machine M, i.e., the collection of strings that M accepts

(yay!!! we finished specifying how a TM computes, let's see some definitions!)

## Definition 3.5

A language is **Turing-recognizable** if some TM recognizes it.

**Notes:** This is similar to how a language is regular if a FA that recognizes it.

- A TM can have three possible outcomes: accept, reject, loop (i.e. does not halt).

- A TM M can fail to accept an input by entering the $q_{reject}$ state or by looping. Sometimes it is difficult to distinguish a machine that is looping from one that is merely taking a long time.

  - For this reason it is preferable to have TM that halt on all inputs, such machines never loop.

  - These machines are called **deciders:** they always make a decision to accept or reject. A decider that recognizes some language also is said to **decide** that language.

## Definition 3.6

A language is **Turing-decidable** or simply **decidable** if some TM decides it.

# Examples of Turing machines

Disclaimer: Cumbersome to provide a description of all the details of a TM. Therefore, we will give only high level descriptions, which are easier to understand.

Example 3.7 (only finishes on page 18)

Notes:

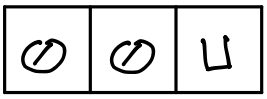
$A = \{0^1, 0^2, 0^4, 0^8, 0^{16}, \dots\}$

Describe TM $M_2$ that decides $A = \{0^{2^n} \mid n \geq 0\}$, i.e. the language consisting of all strings of 0s whose length is a power of 2.

Idea:

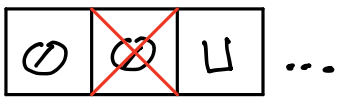
1. Sweep left to right across the tape, crossing off every other 0.
2. Check if the tape:
   - 2.1 Contains a single uncrossed 0: If so then accept
   - 2.2 Contains more than a single 0 and the number of 0s (both crossed and uncrossed) is odd: reject
3. Return the head to the left-hand end of the tape.
4. Repeat the procedure

## Idea Example 1

① Input string w: 

...

② Sweep left to right across the tape, crossing off every other 0.

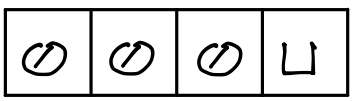


...

② Check if the tape:

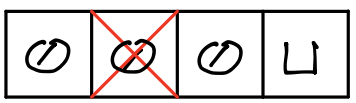
2.1 Contains a single uncrossed 0: If so then accept

## Idea Example 2

① Input string w: 

...

② Sweep left to right across the tape, crossing off every other 0.

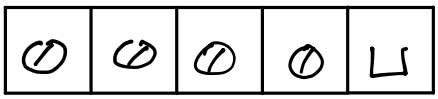


...

② Check if the tape:

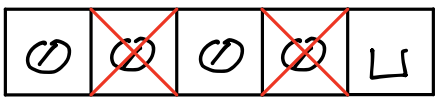
2.2 Contains more than a single 0 and the number of 0s (both crossed and uncrossed) is odd: reject

# Idea Example 3

① Input string w: 

...

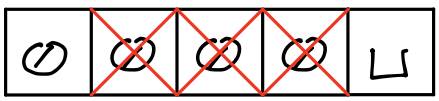
② Sweep left to right across the tape, crossing off every other 0.



...

③ Repeat the procedure

③.1 Sweep left to right across the tape, crossing off every other 0.



...

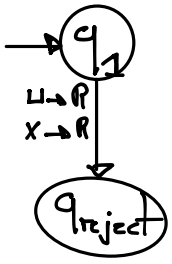
③.2 Check if the tape:

③.2.1 Contains a single 0: If so then accept

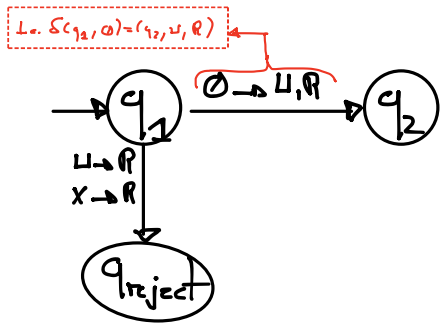
Formal description of $M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$
- $\Sigma = \emptyset$
- $\Gamma = \{\emptyset, X, \sqcup\}$
- Start state = $q_1$
- Accept state = $q_{accept}$
- Reject state = $q_{reject}$
- $\delta$ can described with the following state diagram:

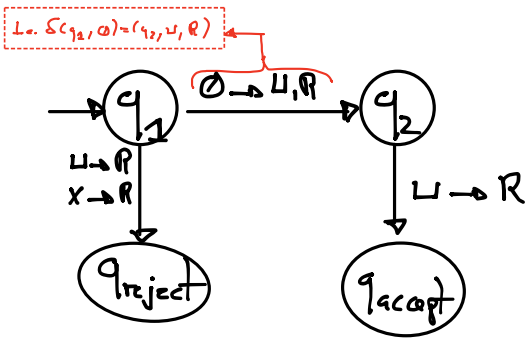
① If the tape is blank or the first symbol is a X then reject



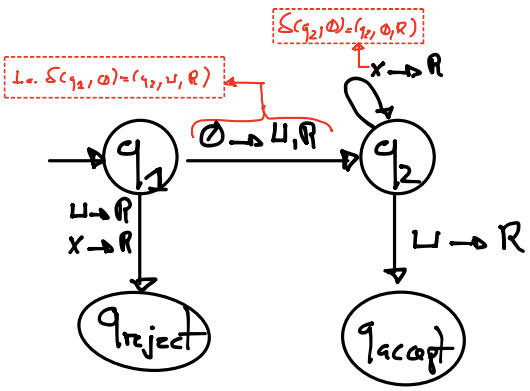
② Let's mark the first 0 as a ⊔ so that we know where the tape starts:



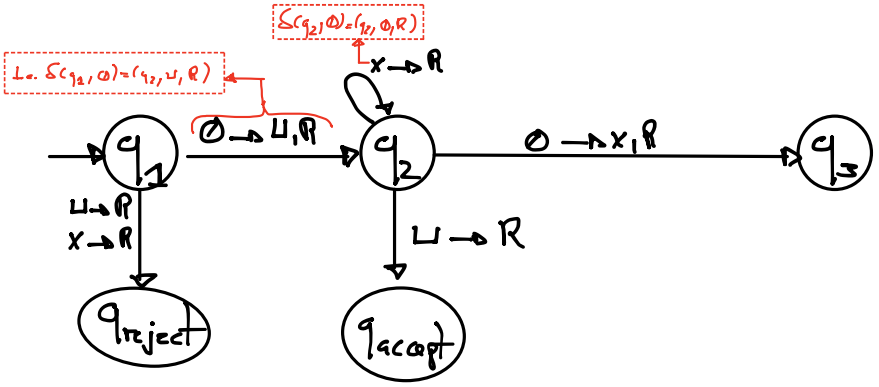

③ If a single 0 has been read and the tape finishes then we should accept:



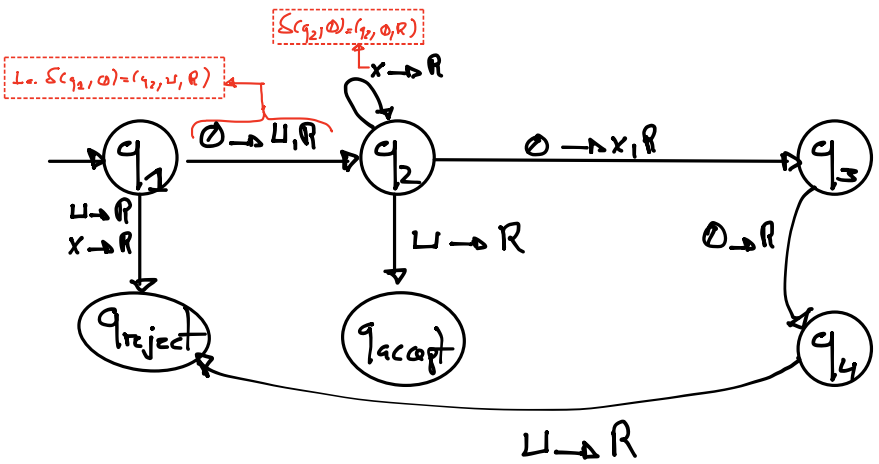

④ If we just read X's then we need to move the head to the right to see the impact of the remaining symbols



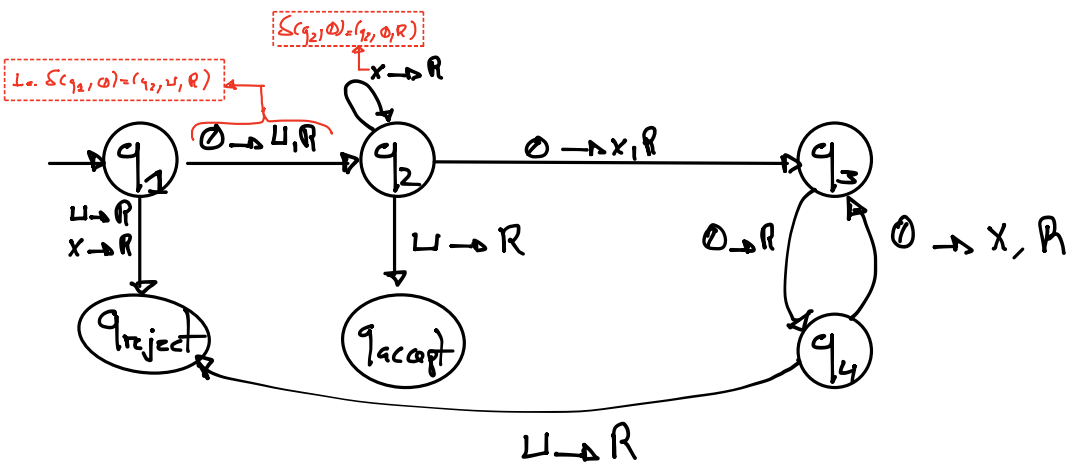
⑤ The next 0 that is read should be replaced by a X



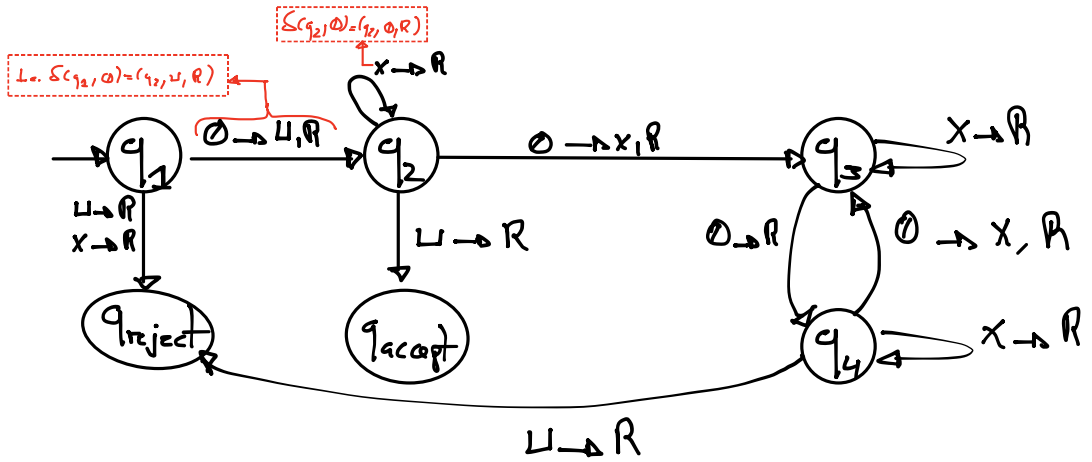

⑥ Now we need to check if an odd number of 0s exists:



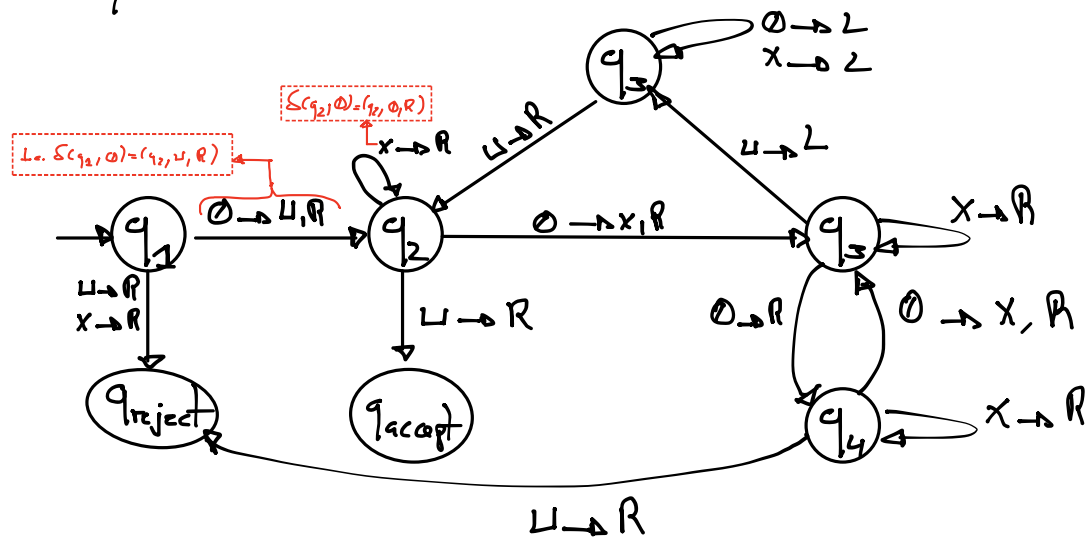

⑦ Now we need to check if an even number of 0s exists:



(8) We also need to process X's that appear between 0's:




(8) Finally, we need to move the head to the left once we reach the end of the tape:




Lets see how this machine runs on input 0000:

(1) $q_1 0000 \sqcup$
(2) $\sqcup q_2 000 \sqcup$
(3) $\sqcup X q_3 00 \sqcup$
(4) $\sqcup X 0 q_4 0 \sqcup$
(5) $\sqcup X 0 X q_3 \sqcup$
(6) $\sqcup X 0 q_5 X \sqcup$
(7) $\sqcup X q_5 0 X \sqcup$
(8) $\sqcup q_5 X 0 X \sqcup$
(9) $q_5 \sqcup X 0 X \sqcup$
(10) $\sqcup q_2 X 0 X \sqcup$
(11) $\sqcup X q_2 0 X \sqcup$
(12) $\sqcup X X q_3 X \sqcup$
(13) $\sqcup X X X q_3 \sqcup$
(14) $\sqcup X X q_5 X \sqcup$
(15) $\sqcup X q_5 X X \sqcup$
(16) $\sqcup q_5 X X X \sqcup$
(17) $q_5 \sqcup X X X \sqcup$
(18) $\sqcup X X q_2 X \sqcup$
(19) $\sqcup X X X q_2 \sqcup$
(20) $\sqcup X X X \sqcup q_{accept}$

(end of example 3.7)

## Example 3.9

Describe TM $M_1$ that decides the language $B = \{w\#w \mid w \in \{0,1\}^*\}$ (this is the TM we saw on page 3).

[resolution:]

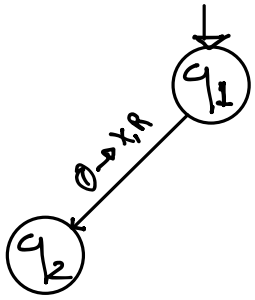
- $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$
- $Q = \{q_1, \dots, q_8, q_{accept}, q_{reject}\}$
- $\Sigma = \{0, 1, \#\}$
- $\Gamma = \{0, 1, \#, x, \sqcup\}$
- Start state = $q_1$
- Accept State = $q_{accept}$
- Reject state = $q_{reject}$

- $\delta$ can described with the following state diagram:

(1) First, lets assume that the input string always $\in B$.

(2) Lets start by observing that if a 0 is read then we need to cross it and move the head to the right until a 0 in the corresponding position is found.

(2.1) If a 0 is read then we need to cross it and move the head to the right

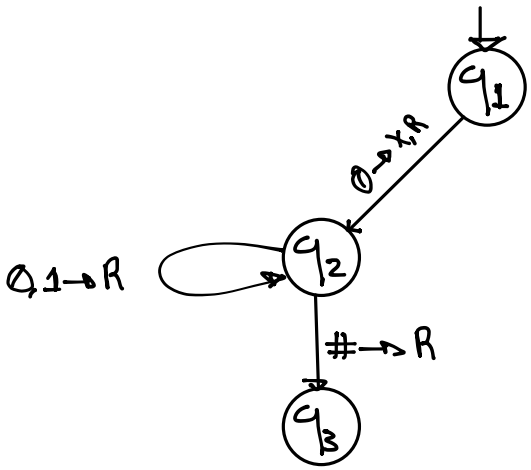



Example:

Head

x 0 0 1 # x 0 0 1

2.2 Now we need to move the head to the right until # symbol is found.

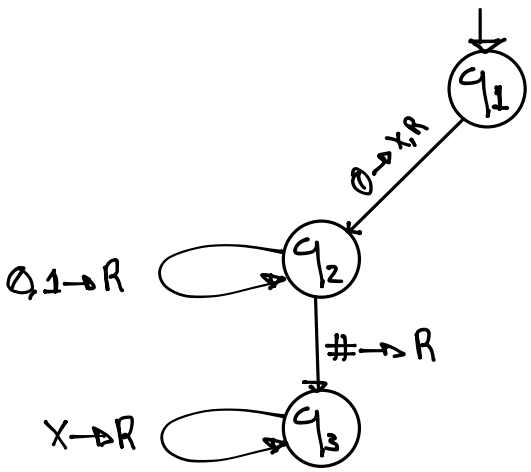



Example:

Head

X 0 0 1 # X 0 0 1

2.3 Next, we need to read all the potential X symbols that might exist.

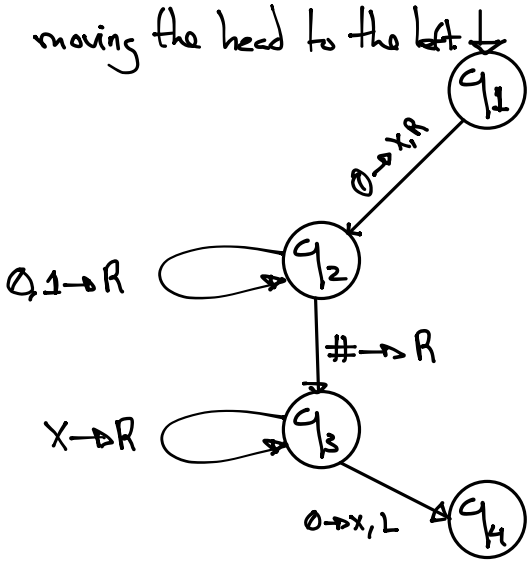



Example:

Head

X 0 0 1 # X 0 0 1

2.3 Now we need to mark the corresponding 0 with an X, we can also start moving the head to the left.

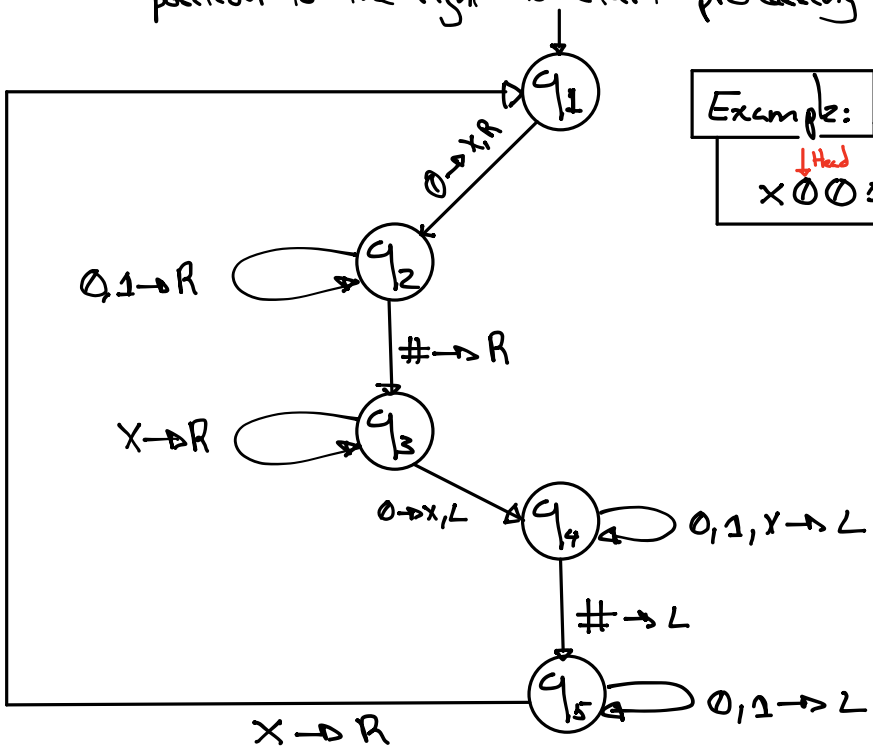



Example:

Head

X 0 0 1 # X 0 0 1

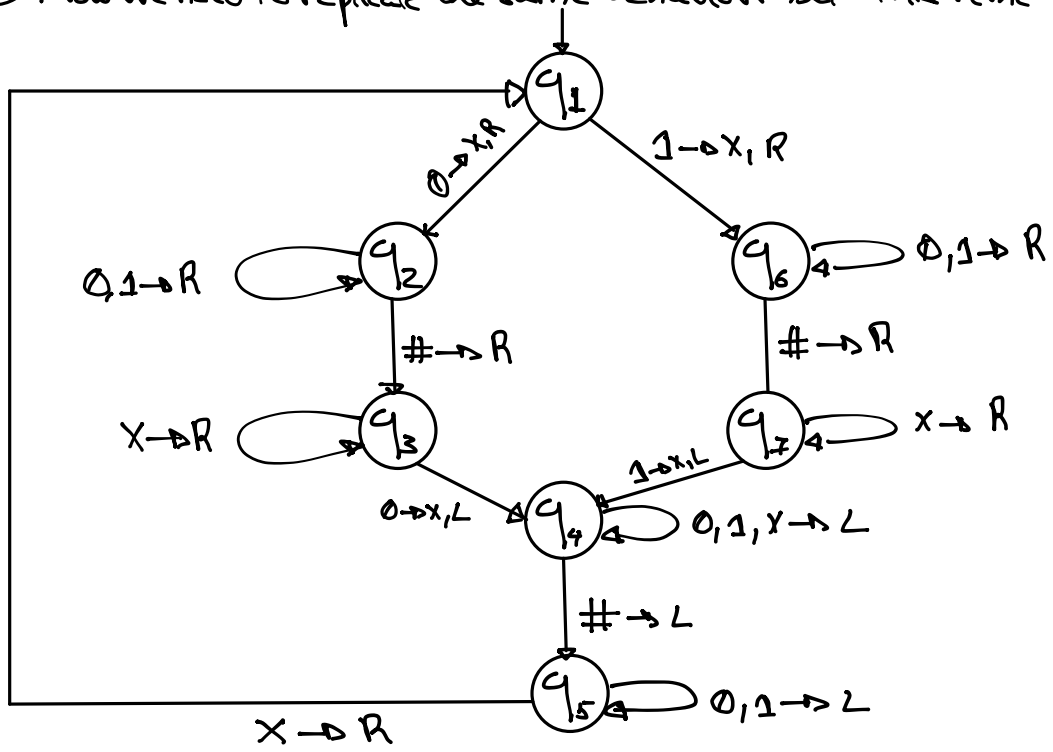
(2.4) Now we need to move the tape to the leftmost position that is not an X and start processing the next symbol. This requires moving the head all the way to the left until an X is found and then move the head one position to the right to start processing the next input symbol.




Example:

Head
X 0 0 1 # X 0 0 1

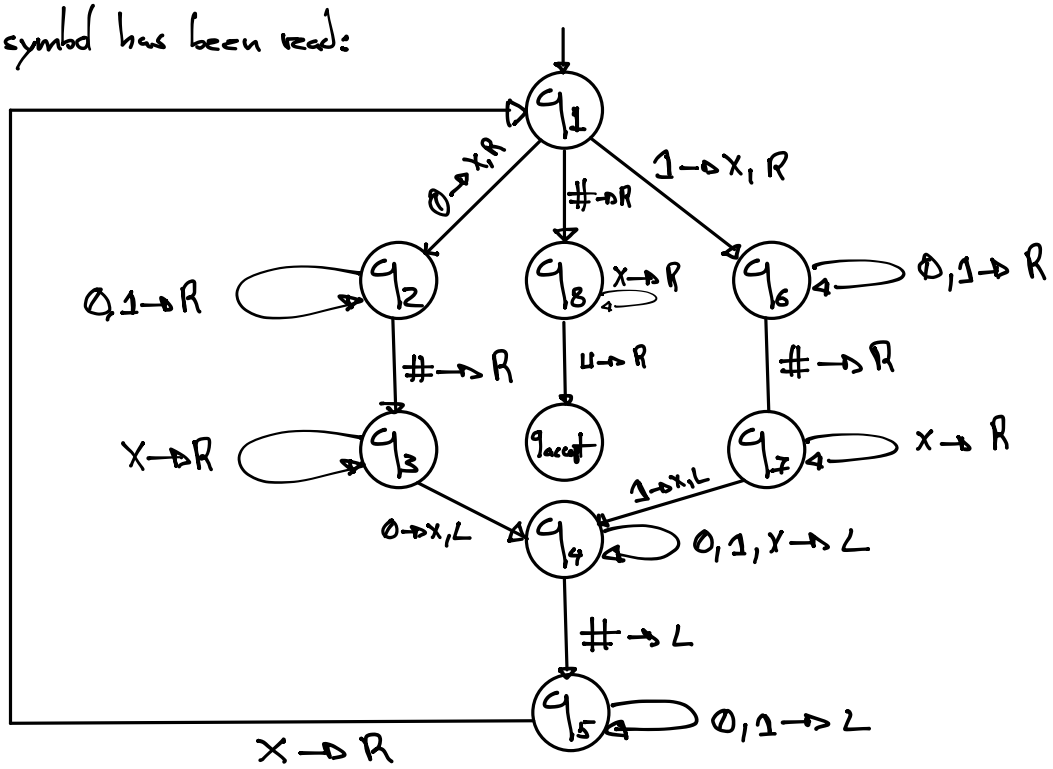
(3) Now we need to replicate the same behaviour but this time when a 1 is read:




Example:

Head
X X X 1 # X X X 1

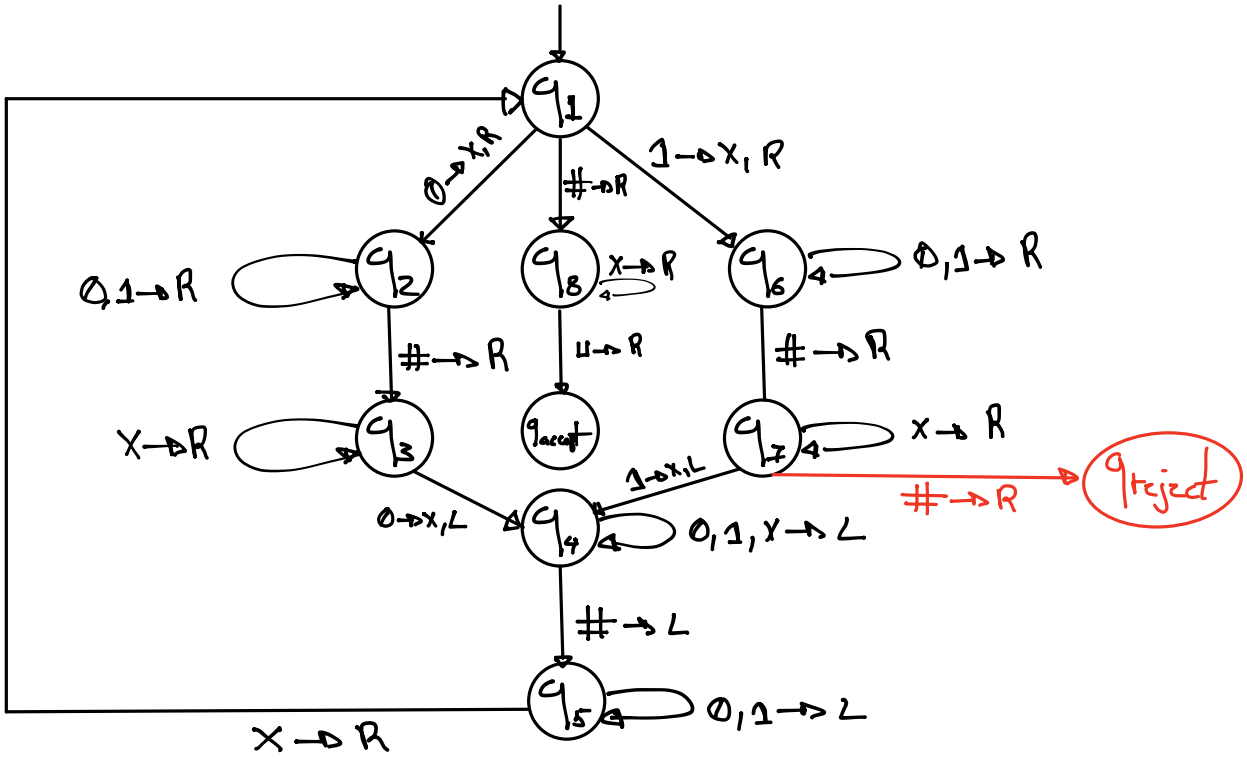
(4) Finally, we need to terminate the computation if the string on the left side of the input has been successfully matched against the right-hand side. This can be done if a # symbol has been read:




Q: But wait, is something missing?

Q: We dealt with the cases that the input string $\in B$. But what about when the input string $\notin B$? (i.e. $q_{reject}$)

↳ To simplify the figure, we don't show the reject state or the transitions to the reject state. These transitions occur implicitly whenever a state lacks an outgoing transition for a particular symbol. Example: In state $q_7$ no outgoing arrow with an # is present. If a # occurs in $q_7$ the next state should be $q_{reject}$. See the next figure for one example of such a transition.


$q_1$
$0 \to X, R$
$\# \to R$
$1 \to X, R$
$q_2$
$0,1 \to R$
$q_8$
$X \to R$
$q_6$
$0,1 \to R$
$\# \to R$
$\sqcup \to R$
$\# \to R$
$q_3$
$X \to R$
$q_{accept}$
$q_7$
$X \to R$
$1 \to X, L$
$q_{reject}$
$\# \to R$
$0 \to X, L$
$q_4$
$0,1,X \to L$
$\# \to L$
$q_5$
$0,1 \to L$
$X \to R$

# 3.3 - The definition of algorithm

Q: So, in your opinion, what is an algorithm?

Algorithm := collection of simple instructions for carrying some task.

- The notion of an algorithm was not precisely defined until the 20th century.

- Before that, mathematicians had an intuitive notion of what algorithms were.

- But that intuitive notion was insufficient for gaining a deeper understanding of algorithms.

- The following story relates how the precise definition of algorithm was crucial to one important mathematical problem.

## Hilbert's problem

- In 1900, mathematician David Hilbert delivered a now-famous address at the International Congress of Mathematicians in Paris.

- In his lecture, he identified 23 mathematical problems and posed them as a challenge for the coming century. The 10th problem concerned algorithms. Before describing the problem let's briefly discuss polynomials.

# Q: What is a polynomial?

- **Polynomial** := sum of terms, where each term is a product of certain **variables** and a constant, called a **coefficient**.

Examples:

- $6x^3yz^2$ (polynomial with one term, over the variables x, y and z.)
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ (polynomial with four terms, over the variables x, y and z.)

- **Root of a polynomial** := assignement of values to its variables so that the value of the polynomial is 0.

Examples:

- $6x^3yz^2$: Trivial solution $x=0 \vee y=0 \vee z=0$
- $6x^3yz^2 + 3xy^2 - x^3 - 10$: $x=5, y=3, z=0$

**Integral root** because all variables are assigned integer values. Some polynomials have an integral root and some don't.

**Hilbert's 10th problem:** devise an algorithm that tests whether a polynomial has an integral root.

> Q: Can you see anything wrong with Hilbert's 10th problem?

- It assumes that one such algorithm exists...

- Guess what... We now know that no such algorithm exists.

- **Algorithmically unsolvable:** this concept did not exist for mathematicians of that time:

  - Their intuitive notion (i.e., not formalized) was useless for showing that no algorithm exists for a particular task.

- Proving that an algorithm does not exist requires having a clear definition of algorithm.

- Progress on the 10th problem had to wait for that definition, which arrived in 1936 with the scientific journal papers of Alonzo Church and Alan Turing:
  (PhD supervisor) ⇢ (PhD student)

  - Church used a notation system: λ-calculus
  - Turing used his "machines"

  } Both were later proven to be equivalent.

This connection between the informal notion of algorithm and the precise definition has come to be called the **Church-Turing thesis.**

## Church-Turing Thesis

| | | |
|---|---|---|
| Intuitive notion of algorithms | equals | Turing machine algorithms |

- In 1970 it was shown that no algorithm exists for testing whether a polynomial has integral roots

- Chapter 4 of the book develops the techniques for proving that some problems are algorithmically unsolvable.

- Let's phrase Hilbert's 10th problem in the following terminology. Let

$$D = \{p \mid p \text{ is a polynomial with an integral root}\}.$$

Hilbert's 10th problem asks in essence whether the set $D$ is **decidable:**

> I.e. does a TM exists that either accepts or rejects language $D$?

→ As we now know: the answer is negative! = P

In contrast, we can show that D is Turing-recognizable, i.e., it is possible to build a TM that recognizes it.

Consider the simpler Hilbert's 10th problem for polynomials that have only a single variable, e.g.: $2x^2 + x - 7$. Let

$$D_1 = \{p \mid p \text{ is a polynomial over } x \text{ with an integral root}\}.$$

Here is a TM $M_1$ that recognizes $D_1$:

$M_1$ = On input $p$: where $p$ is a polynomial over the variable $x$

(1) Evaluate $p$ with $x = \{0, 1, -1, 2, -2, 3, -3, \ldots\}$.

test cycle for variable x

(1.1) If at any point the polynomial evaluates to 0: accept

$\therefore_1$ If $p$ has an integral root, $M_1$ eventually will find it and accept

$\therefore_2$ If $p$ does not have an integral root, $M_1$ will run forever (i.e. will not halt)

$\therefore_3$ Easy to extend to the multivariable case: just add a test cycle for each variable.

# $M_1$ is a recognizer but not a decider:

① $M_1$ can be converted to a decider $D_1$ because we can calculate bounds within which the roots of a single variable polynomial must lie and restrict the search to these bounds.

② Namely, the roots of such a polynomial must lie between the values $\pm k\frac{c_{max}}{c_1}$ where $k$ is the number of terms in the polynomial, $c_{max}$ is the coefficient with the largest absolute value, and $c_1$ is the coefficient of the highest order term.

③ If the root is not found within these bounds: the machine rejects

Matijasevic's theorem shows that calculating such bounds for multivariable polynomials is impossible

∴ Impossible to build a multivariable decider for D.