Turing Machines

Informal discussion

- A Turing machine (TM) is similar to a finite automaton with an unlimited and unrestricted memory
- A Turing machine is however a more accurate model of a general purpose computer
- A Turing machine can do everything that a real computer can do

Note: even a Turing machine cannot solve certain classes of problems

Real computer

A real computer is a triple
$Computer = \langle Processor, Memory, IOdevices \rangle$
which performs the action:

```
RunProgram::
while (PluggedIn and PowerOn)
     Execute (PC);
     PC := Next (PC);
```

Implications:

1. Processor = (PC, Instructions)
2. Program is stored in memory as a stream of instructions
3. PC always points to the next instruction to execute

Informal characterization

- **TM memory:** infinite tape
- **TM I/O:** a tape-head that can read/write symbols and move around on the tape
- **TM Processor:** a control device that performs transformations of the symbols written on the tape while moving the head around.

Comments:

1. What are the similarities to a real computer?
2. What are the differences from a real computer?

Initial condition

- Initially the tape contains only the input string and is blank everywhere else
- If TM needs to store info, it may write it on the tape
- To read the info that it has written, TM can move its head back over its tape
- Machine continues computing until it decides to produce an output

The output

The computation performed by a TM ends up with an output, which is one of:

accept, *reject*, or *compute forever*.

Graphic

Figure 1 shows the schematic of a TM

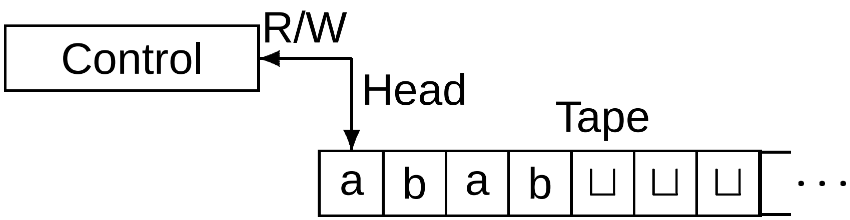

Figure 1: Schematic of a Turing machine

TM versus FA

1. A TM can both write on the tape and read from it;
   a FA can only read its input
2. The read/write head of a TM can move both to the left and to the right;
   a FA can move in one direction only
3. The tape of a TM is infinite;
   the input of a FA is finite
4. The special states of a TM for rejecting and accepting the input take immediate effect;
   FA terminates when input is entirely consumed

Example TM computation

Construct a TM M_1 that tests the membership in the language $B = \{w\#w | w \in \{0,1\}^*\}$

In other words: we want to design M_1 such that $M_1(w) = accept$, if $w \in B$

Note: with regard to a real computer this problem becomes: *construct a program that solve the above problem*

Hence, a TM is an algorithm (i.e., a program).

Note

To understand this problem we assume that we are TMs, i.e., we simulate the actions performed by TM by ourselves.

- We have an input $w \in \{\#, 0, 1\}^*$
- We cam examine w consuming it in any direction, as long as necessary
- We can write to remember anything we want

Strategy

- Identify first the character $\#$ in w
- Zig-zag around $\#$ to determine whether or not the corresponding places on the two sides of $\#$ match
- We can mark the places we have already visited

Design of M_1

M_1 works following the strategy specified above:

- M_1 makes multiple passes over the input with the read/write head
- On each pass M_1 matches one of the characters on each side of $\#$ symbol
- To keep track of which symbols have been checked M_1 crosses off each symbol as it is examined
- If M_1 crosses all symbols it accepts, otherwise it rejects.

M_1 = "On input w

1. Scan the input tape to be sure that it contains a single $\#$. If not, *reject*
2. Zig-zag across the tape to corresponding positions on either side of $\#$ to check whether these positions contain the same symbol. If they do not, *reject*. Cross off the symbols as they are checked
3. When all symbols to the left of $\#$ have been crossed off, check for the remaining symbols to the right of $\#$. If any symbol remain, *reject*; otherwise *accept*."

Illustration, Fig 2

0	1	1	0	0	0	#	0	1	1	0	0	0	⊔	···
x	1	1	0	0	0	#	0	1	1	0	0	0	⊔	···
x	1	1	0	0	0	#	x	1	1	0	0	0	⊔	···
x	1	1	0	0	0	#	x	1	1	0	0	0	⊔	···
x	x	1	0	0	0	#	x	1	1	0	0	0	⊔	···
x	x	x	x	x	x	#	x	x	x	x	x	x	⊔	···
													accept	

Figure 2: Snapshots of M_1 computing

Preparing a formal definition

- The control device (the processor) of a TM can be in a finite set of states. We denote this set by Q
- The tape (i.e. the memory) of a TM is split into an infinite number of locations called *squares* or *cells*. Each square can hold a symbol of a given alphabet, Γ
- The tape-head can move to the right (R) or to the left (L) one square at each step of the computation performed by the TM.

Note: the input data on which a TM operates come

Transitions

- The heart of a formal definition of a TM is the transition function δ because it tells how is the machine going from one step to the next.
- The signature of δ is:
  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- In other words: when TM is in a state $q \in Q$ and the head is over a tape square containing a symbol $a \in \Gamma$

  if $\delta(q, a) = (r, b, L)$ the machine replaces a with b, moves to the state r and moves the head to the left (L)

  if $\delta(q, a) = (r, b, R)$ the machine replaces a with b, moves to the state r and moves the head to the right (R)

Formal definition

A Turing machine is a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

where Q, Σ, Γ are finite sets and

1. Q is a set of states
2. Σ is the input alphabet and blank $\sqcup \notin \Sigma$
3. Γ is the tape alphabet, $\sqcup \in \Gamma, \Sigma \subset \Gamma$
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function
5. $q_0 \in Q$ is the initial state
6. $q_{accept} \in Q$ is the accept state (sometimes denoted q_a)
7. $q_{reject} \in Q$ is the reject state (sometimes denoted q_r)

Other definitions

Hopcroft and Ullman 1979:

A Turing machine M is a 7-tuple $M = (Q, \Sigma, \Gamma, q_0, \delta, B, F)$ where:

1. Q is a set of states,
2. Γ is a fi nite set of allowable tape symbols,
3. B is a symbol from Γ called blank,
4. $\Sigma \subset \Gamma$, $B \notin \Sigma$,
5. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ (δ may be undefi ned on some arguments),
6. $q_0 \in Q$ is the start state,
7. $F \subseteq Q$ is the set of fi nal states.

Different notation

Fleck 2001:

A Turing machine M is a 7-tuple $M = (S, \Sigma, \Gamma, s_0, \delta, B, R)$ where

1. S is the set of states,
2. $s_0 \in S$ is the start state,
3. $R \subseteq S$ is the set of recognizing or accepting states.

The other components of M are as in Hopcroft and Ullman

TM as a quintuple

Lewis and Papadimitriou (1981) and Kimber and Smith, (2001)

A Turing machine M is a 5-tuple $M = (S, \Sigma, \delta, s, H)$ where:

1. S is a set of states,
2. Σ is an alphabet containing $\triangleright$ (left marker) and $\sqcup$ (blank), but $\leftarrow, \rightarrow \notin \Sigma$,
3. $s \in S$ is the initial state,
4. $H \subseteq S$ is the set of halting states, and
5. $\delta : (S \setminus H) \times \Sigma \rightarrow S \times (\Sigma \cup \{\leftarrow, \rightarrow\})$, the transition functions, is such that:
   - $\forall q \in S \setminus H, \delta(q, \triangleright) = (p, \rightarrow)$ for some $p \in S$;

Computations

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

- M receives as input $w = w_1 w_2 \ldots w_n \in \Sigma^*$ written on the leftmost squares of the tape; the rest of the tape is blank (i.e., filled with $\sqcup$)
- The head starts on the leftmost square of the tape
- The first blank encountered shows the end of the input
- Once it starts, it proceeds by the rules describing δ
- If M ever tries to move to the left of the leftmost square the head stays in the leftmost square even though δ indicate L
- Computation continues until M enters q_{accept}, q_{reject} at which points it halts. If neither occurs M goes on forever

Configuration

A configuration C of M is a tuple
$C = (q \in Q, tapeContents, headLocation)$

- Configurations are used to formalize machine computation and are represented by special symbols
- For $q \in Q$, $u, v \in \Gamma^*$, $u\ q\ v$, also denoted $C = (u, q, v)$, represents the configuration where current state is q, tape contains uv, and head is on the first symbol of v.
- **Notation:** $C = uqv$

Note: tape contains only $\sqcup$ following the last symbol of v.

Example configurations

Consider again the Figure 3 representing a snapshot of TM M_1 recognizing the language $L = \{x | x = w\#w, w \in \{0,1\}^*\}$.

$$0\ 1\ 1\ 0\ 0\ 0\ \#\ 0\ 1\ 1\ 0\ 0\ 0\ \sqcup\ \cdots$$

$$x\ 1\ 1\ 0\ 0\ 0\ \#\ 0\ 1\ 1\ 0\ 0\ 0\ \sqcup\ \cdots$$

$$x\ 1\ 1\ 0\ 0\ 0\ \#\ x\ 1\ 1\ 0\ 0\ 0\ \sqcup\ \cdots$$

$$x\ 1\ 1\ 0\ 0\ 0\ \#\ x\ 1\ 1\ 0\ 0\ 0\ \sqcup\ \cdots$$

$$x\ x\ 1\ 0\ 0\ 0\ \#\ x\ 1\ 1\ 0\ 0\ 0\ \sqcup\ \cdots$$

$$x\ x\ x\ x\ x\ x\ \#\ x\ x\ x\ x\ x\ x\ \sqcup\ \cdots$$

accept

Figure 3: Snapshots of M_1 computing

Configurations of M_1

The following are configurations from from M_1's computations.

- In first line $C = (\epsilon, q_1, 011000\#011000)$
- In second line $C = (x, q_2, 11000\#011000)$
- In third line $C = (x11000\#, q_3, x11000)$
- In fourth line $C = (\epsilon, q_4, x11000\#x11000)$
- In fifth line $C = (x, q_5, x000\#xx1100)$
- In sixth line $C = (xxxxxxx\#xxxxxxx, q\sqcup)$

where ϵ denotes the empty word in Σ^* and x denotes a crossed symbol.

Formalizing TM computation

- A configuration C_1 **yields** a configuration C_2 if the TM can legally go from C_1 to C_2 in a single step
- **Formally:** suppose $a, b, c \in \Gamma$, $u, v \in \Gamma^*$ and $q_i, q_j \in Q$.
  1. We say that $ua\ q_i\ bv$ yields $uac\ q_j\ v$ if $\delta(q_i, b) = (q_j, c, R)$; (machine moves rightward)
  2. We say that $ua\ q_i\ bv$ yields $u\ q_j\ acv$ if $\delta(q_i, b) = (q_j, c, L)$; (machine moves leftward)

Head at one input end

- For the left-hand end, i.e.$u = \epsilon$:
  - the confi guration $q_i \; bv$ yields $q_j \; cv$ if the transition is left moving, i.e., $\delta(q_i, b) = (q_j, c, L)$
  - the confi guration $q_i \; bv$ yields $c \; q_j \; v$ for the right moving transition, i.e., $\delta(q_i, b) = (q_j, c, R)$
- For the right-hand end, i.e., $v = \epsilon$:
  - the confi guration $ua \; q_i$ is equivalent to $ua \; q_i\sqcup$ because we assume that blanks follow the part of the tape represented in confi guration. Hence we can handle this case as the previous

Special configurations

- If the input of M is w and initial state is q_0 then $q_0\ w$ is the *start configuration*
- $ua\ q_{accept}bv$ is called the *accepting configuration*
- $ua\ q_{reject}bv$ is called the *rejecting configuration*

Note: accepting and rejecting configurations are also called *halting configurations*

Accepting an input w

A Turing machine M accepts the input w if a sequence of configurations $C_1, C_2, \ldots, C_n$ exists such that:

1. C_1 is the start configuration, $C_1 = (\epsilon, q_0, w)$
2. Each C_i yields C_{i+1} denoted $C_i \vdash C_{i+1}$, $i = 1, 2, \ldots, n-1$
3. C_n is an accepting configuration

Note: The sequence $C_1 \vdash C_2 \vdash \ldots \vdash C_k$ is although denoted by $C_1 \models C_k$.

Language of M

The language recognized by a Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$$

is denoted by $L(M)$ and is defined by

$$L(M) = \{w \in \Sigma^* | q_0 w \models u q_a v\}$$

That is, $L(M)$ is the set of strings $w \in \Sigma^*$ accepted by M.

Note: The language $L(M)$ recognized by a Turing machine M is also called *the language of* M.

Turing-recognizable language

A language L is Turing-recognizable if there is a Turing machine M that recognizes it

Note

When we start a TM on an input w three cases can happen:

1. TM may accept w
2. TM may reject w
3. TM may **loop** indefinitely, i.e., TM does not halt.

Note: looping does not mean that machine repeats the same steps over and over again; looping may entail any simple or complex behavior that never leads to a halting state.

Question: is this real? I.e., can you indicate a computation that takes infinite many steps without repetition?

Fail to accept

- A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ fails to accept $w \in \Sigma^*$ by:
  1. $q_0 w \models u q_r v$, i.e., entering q_{reject}, and thus rejecting
- When M is looping, that is $q_0 \models u_m q_m v_m \models \ldots$ one cannot say if M accepts or rejects because we don't know if M will ever enter a q_m for $q_m \in \{q_a, q_r\}$.

Fail to reject

- A TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ fails to rejects $w \in \Sigma^*$ by:
  1. $q_0 w \models u q_a r v$, i.e., entering q_{accept}, and thus accepting
- When M is looping, that is $q_0 \models u_m q_m v_m \models \ldots$ one cannot say if M accepts or rejects because we don't know if M will ever enter a q_m for $q_m \in \{q_a, q_r\}$.

Note

Sometimes it is difficult to distinguish a machine that fail to reject from one that merely takes long-time to halt.

Decider

A TM that halts on all inputs is called a decider.

Note: a decider always halts, accepting or rejecting its input.

Turing-decidability

- A decider that recognizes some language is also said to *decide* that language
- A language is called *Turing-decidable* or simple *decidable* if some TM decides it.

Note

- Every decidable language is Turing-recognizable

  Remember: a language is Turing-recognizable if it is recognized by a TM M, i.e $\forall w \in \Sigma^*$: M accepts w or M rejects w or M is looping on w.

- Certain Turing-recognizable languages are not decidable

  Remember: to be decidable means to be decided by a TM which halts on all inputs, i.e., $\forall w \in \Sigma^*$: M accepts w or M rejects w.