Theory of Computation (CS F351)

BITS Pilani
Hyderabad Campus

Dr.R.Gururaj
CS&IS Dept.

Turing Machines (Ch.4)

Introduction

- Neither FA nor PDA can be regarded as truly general models for computers.
- Because they can not recognize some languages.
- FA can not recognize $L=\{a^n b^n : n \geq 0\}$
- PDA can not recognize $L=\{a^n b^n c^n : n \geq 0\}$

- Here we study a new device which can recognize these and many more complicated languages.
- These machines are called *Turing Machines*, named after the inventor

  Allan Turing(1912-54).
- These are more general than automata we studied earlier.

The basic appearance is similar to that of automata.

A Turing Machine consists of the following.

- Finite Control
- A Tape
- A R/W head

- The formal definition of a TM and their operations are in the same mathematical style as those used for FA and PDA.
- TM is not simply one more class of automata to be replaced by a more powerful type.
- We see that as primitive as TM seem to be, attempts to strengthen them do not have any effect.

Ex: Even we augment this TM model with RA memory like what we have in any computer will not increase the power of a TM.

- So any fancier machine can be converted into a TM with analogous functionality.
- Hence any computation carried out by a fancier machine can be actually be carried out by a TM.

Hence the TM seem to form a stable and maximal class of computational devices, in terms of computations they can perform.

The criteria to be satisfied by a TM:

- They should be automata: That is their construction and functionality should be in the same general spirit of the devices studied earlier.
- They should be as simple as possible to describe, define and reason about.
- They should be as general as possible in terms of the computations they carry out.

- TM has : Finite control, Tape, and a R/W head
- A Finite control operates in discrete steps.
- At each step it performs two functions in a way that depends on its current state and the tape symbol currently scanned by the R/W head.

1. Put the Finite Control in a new state.
2. Either
   (a) Write a symbol in the tape square currently scanned, i.e., replacing the one that is already there.

   Or

   (b) Move the head one tape square to the left or right.

Note: The tape has a left end but no right end (extends infinitely).

To prevent the machine moving its head off the left end, we assume that the left most end/cell of the tape is marked with a special symbol.

- All the TMs are so designed that, when the head reads ► it immediately moves to the right.
- We use distinct symbols → , and ← to denote head movements. We also assume that the two symbols are not part of any alphabet.
- We give input to the TM by inscribing the string on the tape.
- The rest of the tape is initially contains blanks denoted by Ц.

- The machine is free to alter its input in any way as it needs.
- It can also write on the blank portion of the tape to the right.
- The RW head can move by only one square to the left or right at a time.

Formal definition

A Turing Machine is a quintuple- ($K, \sum, \delta, s, H$)

K *is a finite set of states*

$\sum$ is an alphabet (containing blank symbol and left end marker but not → and ←.

$s \in K$ *is the initial state*

H *subset of K is the set of halting states*

δ is the transition function from $(K-H) X \sum \rightarrow K X(\sum U\{\leftarrow, \rightarrow\})$

δ is the transition function from $(K\text{-}H) \times \Sigma \rightarrow K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$

Such that

(a) For all $q \in K\text{-}H$, *if* $\delta(q, \blacktriangleright) = (p, b)$ then $b = \rightarrow$

(b) For all $q \in K\text{-}H$, *and* $a \in \Sigma$, *if* $\delta(q, a) = (p, b)$ then $b \neq \blacktriangleright$

If $q \in K\text{-}H$, and $a \in \sum$, and $\delta(q,a) = (p,b)$ then then M when in state q and scanning symbol a, will enter state p, and

(1) If b is symbol in ∑, M will rewrite currently scanned symbol a with b.

or

(2) If b is ← or →, M will move its in the direction of b

The operations will stop only when it enters a halt state.

Notice that whenever it sees the left end marker (►) it immediately moves to the right.

Hence the left end marker is never erased.

Notation for TMs

- The TMs we have seen so far have been extremely simple.
- Their tabular form is more complex and hard to interpret. Hence we need some graphical representation to manage the complexity.

- We shall adopt a notation similar to that of the automata we have seen earlier (states and arrows representing the transitions).
- Here in this case things that are joined are TMs themselves.
- In other terms we use a hierarchical notation, in which more complex machines are built from simpler machines.
- We shall define a set of basic machines and rules to combine them.

Symbol-writing machines:

Head-moving machines:

alphabet $a \in (\sum \cup \{\leftarrow, \rightarrow\} - \{\blacktriangleright\})$

$M_a = (\{s,h\}, \sum, \delta, s, \{h\})$

Where for each $b \in \sum - \{\blacktriangleright\}$, $\delta(s,b) = (h,a)$

$\delta(s, \blacktriangleright)$ is still $(s,\rightarrow)$

Only thing this machine does is to perform action a-writing a if $a \in \sum$, moving in the direction indicated by a if $a \in \{\rightarrow, \leftarrow\}$ and then to immediate halt.

We abbreviate symbol-writing machine M_a with a

That is if $a \in \sum$, *a-writing* machine will be simply denoted by a

We abbreviate head-moving machine $M_{\leftarrow}$ with L

We abbreviate head-moving machine $M_{\rightarrow}$ with R

Rules for combining machines

Individual machines are like states of a finite automaton.

And the machines may be connected to each other in the way that the states of a automaton are connected to each other.

The connection from one machine to the other is not pursued until the first machine halts; the other machine is then started from its initial state with the tape and head positions as they are left by the prev machine.

That is if $a \epsilon \sum$, *a-writing* machine will be simply denoted by a

We abbreviate head-moving machine $M_{\leftarrow}$ with L

We abbreviate head-moving machine $M_{\rightarrow}$ with R

Computing Machines

So far we have seen only how a TM works, without much indication about how it can be used for performing computations

Ex: recognizing a language.

We fix some conventions for the use of TMs:

1. The input string, with no blanks in it, is written to the right of the leftmost symbol Δ, with a blank to its left and a blank to its right.
2. The head is positioned at the tape square containing the blank between Δ and the input.
3. The machine starts operating in its initial state.

Ex: If M= ($K, \sum, \delta, s, H$) is a TM

and w ϵ ($\sum$ - {␣, , ►})

Then initial configuration= (s, ► ␣ w)

Let $M = (K, \sum, \delta, s, H)$ be a TM such that $H=\{y, n\}$

consists of two distinguished halting states

(y for "yes", and n for "no")

Any configuration whose state component is y is called as an *accepting* configuration.

A halting configuration whose state component n is called a *rejecting* configuration.

We say that M accepts string $w \in (\sum - \{␣, , ▶\})^*$

If $(s, ▶ \underline{␣}\ w)$ yields an accepting configuration

We say that M rejects string $w \in (\sum - \{␣, , ▶\})^*$

If $(s, ▶ \underline{␣}\ w)$ yields a rejecting configuration

The Turing Machine *M* decides a language *L*

1. if it accepts or
2. Rejects it

That is a TM decides a language L if, *when started with input w, it always halts*

No guarantees are given about wrong input.

A language *L* is recursive if there is a TM that decides it.

$L=\{a^n b^n c^n : n \geq 0\}$

Let $M = (K, \Sigma, \delta, s, \{h\})$ be a TM,

Σ_0 subset of $(\Sigma - \{Ц, , ►\})$ be an alphabet let $w \in \Sigma_0^*$

Suppose M halts on input w,

and that $(s, ► Ц\ w) \rightarrow^*_M (h, Ц\ y)$; for some $y \in \Sigma_0^*$

Then y is called as output of M on w as input.

This is denoted by $M(w)$

$M(w)$ is defined only if M halts on w

We say that M computes function f if, for all w $\epsilon \sum_0{}^*$

M(w) = f(w)

When machine halts it contains the string on the tape ► Ц f(w)

A function f is called *recursive* if there is a TM *M* that computes *f*.

Let $M = (K, \Sigma, \delta, s, \{h\})$ be a TM,

Σ_0 subset of $(\Sigma - \{Ц, , ►\})$ be an alphabet

And let L is subset of Σ_0^*

We say that M semidecisde L if for any $w \in \Sigma_0^*$

if and only if M halts on input w.

A language L is *recursively enumerable* if and only if there is a TM M that semidecides L.

If a L is recursive, then it is also recursively enumerable.

If a L is recursive, then it is complement is also.

Extensions of the Turing Machine

It is clear that Turing machines can perform fairly powerful computations.

We shall consider the effect of extending the TM model in various directions.

We shall see that in each case the additional features do not add to the classes of computable functions or decidable languages : the “new, improved model” of the TM in each instance be simulated by the standard model.

Turing Machines with multiple tapes

innovate achieve lead

We can think of a TMs that have several tapes.

Each tape is connected to the finite control by means of a R/W head(one on each tape).

The machine can in one step read the symbols scanned by all its heads and then depending on those symbols and its current state, write some of those scanned squares and move some of heads to the right or left, in addition to changing state.

For any fixed integer $k \geq 1$, a k-tape Turing machine is a Turing machine equipped as above with k tapes and with corresponding heads.

Thus a standard TM with single tape can be seen as a k-tape TM, where $k=1$.

Let $M = (K, \sum, \delta, s, H)$ be a k-tape TM, where $k \geq 1$

Where

K, $\sum$, s, *and* H have as in the definition of the ordinary TM, and δ, transition function is a function from

$(K-H) \times \sum^k \rightarrow K \times (\sum \cup \{\leftarrow, \rightarrow\})^k$

Ex: $\delta(q, (a_1, .., a_k)) = (p, (b_1, .., b_k))$

Ex: to transform ►Ц *w*Ц into ►Ц *w* Ц *w* Ц Move the heads on both tapes to the right, copying each symbol on the first tape onto the second tape, until a blank is found on the first tape. The square of the second tape should be left blank.

1. Move the head on the second tape to the left until a blank is found.
2. Again move the heads on both tapes to the right, this time copying symbols from the second tape onto the first one.
3. Halt when a blank is found on the second tape.

Two-way infinite tapes

We can think of a TM that has a tape which is infinite on both directions. All squares are initially blank, except of those containing input.

The head is initially to the left of the input.

Hence the convention of left end marked is meaningless here.

Still this fancy machine with multiple tapes do not add substantial power to TMs. And can be simulated by standard basic TM.

Multiple heads

innovate achieve lead

What if we have one single tape with multiple heads.

In one step, the heads all sense the scanned symbols and move or write independently.

The issue is what if two heads scan and try to write two different symbols on to a tape square. Some convention need to used there to address this.

Two-dimensional tape

Another organization is that a TM has type as a infinite 2D grid.

Such a device could be much more useful than the standard one in solving puzzles.

summary

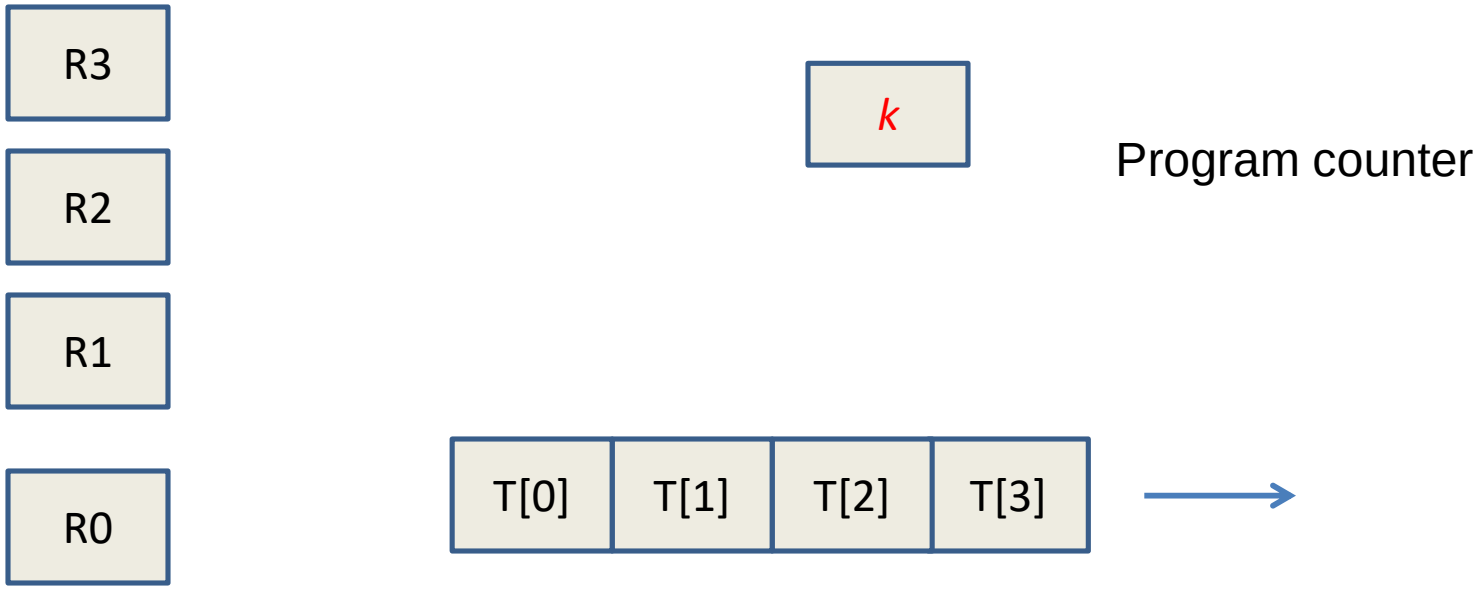
Any language *decided* or *semidecided*, and any function *computed* by TM with several tapes, heads, two-way infinite tapes, or multi-dimensional tapes, can be *decided*, *semidecided*, or *computed*, respectively by a standard TM.

Random Access Turing Machines

All the TMs which we have seen so far have quite limiting common feature – *their memory is sequential*.

In contrast real computers have random access memories, each element of which can be accessed in a single step, if appropriately addressed.

What could happen if we equip our TMs with registers, capable of storing and manipulating the addresses of tape squares.



Registers

Random Access Turing Machines

1. A RATM has a fixed number of registers and one-way infinite tape (memory).
2. Each register and each tape square is capable of containing an arbitrary natural number.
3. The TM acts on its tape squares and registers as dictated by a fixed program- the analog of the transition function of ordinary TMs.
4. The program of a RATM is a sequence of instructions.

5. Initially the register values are 0, the program counter is *k =1*, and the tape content encode the input string.
6. Then the machine executes the first instruction of its program. This will change the contents of the registers or of the tape squares, the value of program counter *k*, an integer indicating the next instruction to be executed.
7. Register R_0 is an accumulator
8. This will continue until halt instruction is executed.
9. When halt is executed, then the *k=0*

Kind of instructions allowed are

Instruction	Operand	Semantics
read	*j*	$R_0 := T[R_j]$
write	*j*	$T[R_j] := R_0$
store	*j*	$R_j := R_0$
load	*j*	$R_0 := R_j$
load	*=c*	$R_0 = c$
add	*j*	$R_0 := R_0 + R_j$
add	*=c*	$R_0 := R_0 + c$
sub	*j*	$R_0 := max\{R_0 - R_j , 0\}$
sub	*=c*	$R_0 := max\{R_0 - c , 0\}$
half		$R_0 := floor[R_0 / 2]$

instructions cntd..

Instruction	Operand	Semantics
jump	*s*	*k:=s*
jpos	*s*	*If R_0 > 0 then k:=s*
jzero	*s*	*If R_0 = 0 then k:=s*
halt	*j*	*k=0*

Description

The description of a RA TM is a pair

M= (K, ∏), where k>0, is the number of registers and

∏ = (π_1 , π_2 , …, π_p), the program, is a finite sequence of instructions. The last instruction π_p

The configuration of RATM M= M= (K, ∏), is a K+2 tuple,

(k, R_0, , R_1 , … , R_{K-1}, T)

For halted configuration, k=0.

Description

T is the tape contents, is a finite set of pairs of positive integers – that is $(i, m) \in T$ means that the i^{th} tape square, currently contains the integer $m>0$. All other tape squares not appearing as the first component of a pair in *T* are assumed to contain *0*.

Non-deterministic Turing Machines

Another important feature of TM is non-determinism.

In case of FA, it is allowed to act non-deterministically, without increase in it power.

But the non-deterministic PDA are more powerful than deterministic ones.

We can also imagine Turing machines that act non-deterministically.

Such machines might have, on certain combination of state and scanned symbol, more than one possible choice of behavior.

Non-deterministic Turing Machines

A deterministic TM is $(K, \Sigma , \delta, s, H)$

Where δ *is a function from* $(K-H) X \Sigma$ *to* $KX(\Sigma U\{\leftarrow, \rightarrow\})$

A non-deterministic Turing machine is a quintuple

$M= (K, \Sigma , \Delta, s, H)$ where K, Σ ,s, *and H are as for standard TM*

Δ *is subset of*

$$((K-H) X \Sigma) X (KX(\Sigma U\{\leftarrow, \rightarrow\}))$$

Grammars (Unrestricted Grammars)

The class of language generated by *Grammars* is precisely the class of *recursively enumerable* ones.

A language is generated by a *Grammar* if and only if it is recursively enumerable.

The left-hand side of a rule may consist of more than one NT with terminals; but at least with on NT

Final product is a string of terminals.

Grammars (Unrestricted Grammars)

The class of language generated by *Grammars* is precisely the class of *recursively enumerable* ones.

A language is generated by a *Grammar* if and only if it is recursively enumerable.

The left-hand side of a rule may consist of more than one NT with terminals; but at least with on NT

Final product is a string of terminals.

Summary

innovate achieve lead

1. Introduction to Turing Machines
2. Working principle of a TM
3. Description , Configuration, transition of a TM
4. Combined Tm
5. 2-tape TM
6. Extensions of the Turing Machine
7. Random access TM
8. Grammar