OTHER MODELS OF TURING MACHINES

Chap. 10

1

Summary

- We essentially challenge Turing's thesis by examining a number of ways the Standard Turing Machine can be complicated to see if these complications increase its power.
- We look at a variety of *different storage devices* and even allow *nondeterminism*.
- *None* of these complications increases the essential power of the standard machine, which lends credibility to Turing's thesis.

2

Learning Objectives

- The concept of *equivalence* between classes of *Automata*.
- Describe how a *TM* with a *stay-option* can be simulated by a standard-TM.
- Describe how a standard-TM can be simulated by a machine with a *semi-infinite tape.*
- Describe how *off-line and multidimensional TMs* can be simulated by standard-TMs.
- Construct *two-tape TMs* to accept simple languages.
- Describe the operation of *Nondeterministic TMs* and their relationship to deterministic TMs.
- Describe the components of a *Universal Turing Machine*
- Describe the operation of *Linear Bounded Automata* and their relationship to standard TM.

3

Equivalence of Classes of Automata

- Definition 10.1: Two automata are *equivalent* if they accept the same language.

  Given two classes of automata C_1 and C_2,
  if for every automaton M_1 in C_1, there is an equivalent automaton M_2 in C_2 s.t. $L(M_1)=L(M_2)$,
  then the class C_2 is *at least as powerful* as C_1.
  If the class C_1 is at least as powerful as C_2, and the converse also holds, then the classes C_1 and C_2 are *equivalent*.
- Equivalence can be established either through a constructive proof or by simulation.

4

Turing Machines with a Stay-Option

- In a *Turing Machine with a Stay-Option*,
  the read-write head has the option to stay in place
  after rewriting the cell content: δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$
- Theorem 10.1: The class of TMs with a stay-option is equivalent to the class of Standard TMs.
- To show equivalence, we argue that any machine with
  a stay-option can be simulated by a standard TM,
  since the stay-option can be accomplished by
  - A rule that rewrites the symbol and moves Right, and
  - A rule that leaves the tape unchanged and moves Left.

5

Turing Machines with a Stay-Option

- Theorem 10.1: The class of TMs with a stay-option is equivalent to the class of Standard TMs.

Proof) Since a TM with a stay-option is an extension of the standard model, any standard TM can be simulated by TM with a stay-option.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ be a TM with a stay-option to be simulated by a standard TM $M' = (Q', \Sigma, \Gamma, \delta', q_0', \square, F)$.

For each move of *M*,

- If the move of *M* does not involve the stay-option, *M*' performs one move that is identical to *M*'s move.
- If the move of *M* involves *S*, then *M*' will make two moves:
  - The first *rewrites* the symbol and moves the head *right*;
  - the second moves the head *left*, leaving the tape contents unchanged.

M' can be constructed from M by defining δ' as follows:

6

TMs with a Stay-Option (cont.)

- Theorem 10.1: The class of Turing Machines with a stay-option is equivalent to the class of Standard TMs.

Proof: cont.)

M' can be constructed from M by defining δ' as follows:

For each transition,

- $\delta(q_i, a) = (q_j, b, L/R) \Rightarrow \delta'(q_i', a) = (q_j', b, L/R)$

For each S-transition,

- $\delta(q_i, a) = (q_j, b, S) \Rightarrow \delta'(q_i', a) = (q_{js}', b, R)$
  and $\delta'(q_{is}', c) = (q_j', c, L)$ for all $c \in \Gamma$.

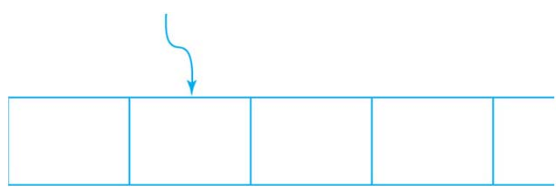
So, every computation of M has a corresponding computation of M'.

Thus, *M'* can simulates *M*.

7

TMs with Semi-Infinite Tape

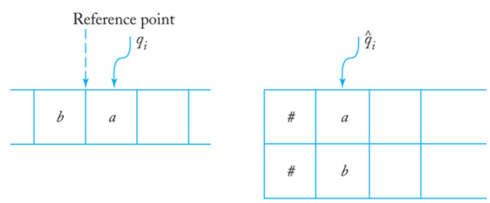
- As shown in Figure below, a common variation of the standard TM is one in which the tape is *unbounded only in one direction*.
- A *TM with semi-infinite tape* is otherwise identical to the standard model, except that *no left move* is possible when the read-write head is at the tape *boundary.*



8

Equivalence of Standard TMs and Semi-Infinite Tape Machines

- The classes are equivalent because any standard TM, *M*, can be simulated by a TM with a semi-infinite tape, *M'*.
- The simulating machine, M', has two tracks:
  - the *upper track* contains the symbols to the *right* of an arbitrary reference point, while
  - the *lower track* contains those to the *left* of the reference point in *reverse order*



(a) (b)

TM to be simulated, M. Simulating machine, M'.

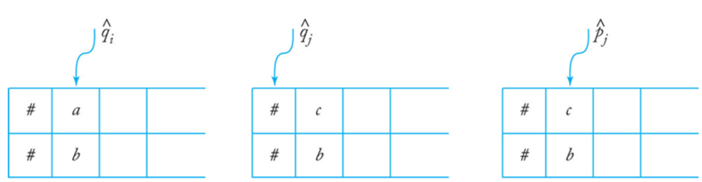
9

Cont.

- M' uses information on the upper track only as long as a head of M is to the right of the reference point, and
- M' works on the lower track as M moves into the left part of its tape.
- $Q' = Q_U \cup Q_L$: the states working on the upper/lower tracks.
- # : the end marker on the left boundary of the tape, switching the track.
- E.g.) $\delta(q_i, a) = (q_j, c, L)$ in M $\Rightarrow \delta'(q_i', (a,b)) = (q_j', (c,b), L)$ in M', where $q_i' \in Q_U$.

  Then, $\delta'(q_j', (\#,\#)) = (p_j', (\#,\#), R)$ where $p_i' \in Q_L$.

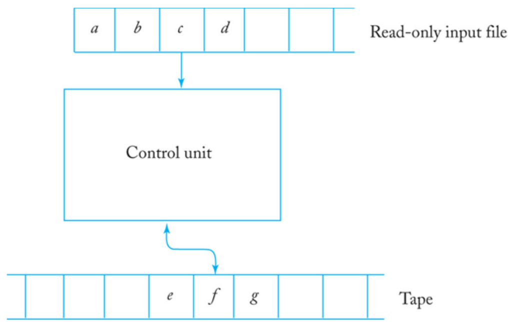
  Now, M' works on the lower track.



10

The Off-Line Turing Machine

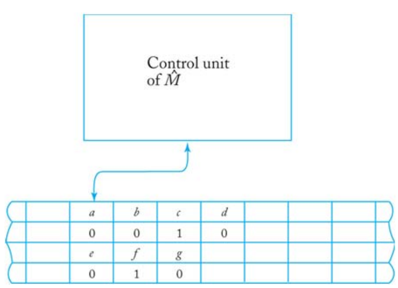
- An *Off-Line TM* has a read-only input file in addition to the read-write tape.
- Transitions are determined by both the current input symbol and the current tape symbol.



11

Equivalence of Standard TMs and Off-Line TMs

- The classes are equivalent because a standard TM with four tracks can simulate the computation of an off-line machine.
- Two tracks are used to store the *input file contents* and *current position*, while the other two tracks store the *contents* and *current position of the read-write tape*.



12

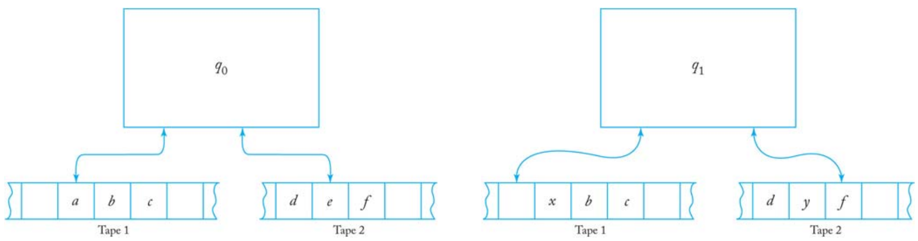
Multitape Turing Machines

- A *MultiTape Turing Machine* has several tapes, each with its own independent read-write head.

$$\delta: Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{L, R\}^n$$

- A sample transition rule for a two-tape machine must consider the current symbols on both tapes:

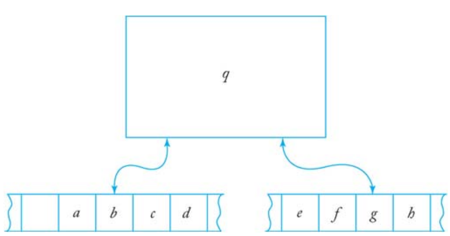
$$\delta(q_0, (a, e)) = (q_1, (x, y), (L, R))$$

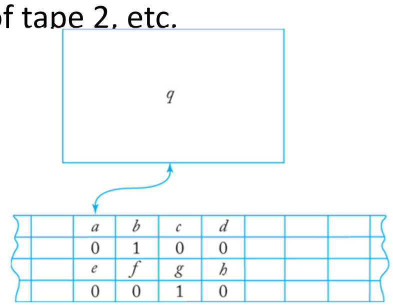


13

Equivalence of Standard TMs and MultiTape TMs

- The classes are equivalent because a standard TM with multi tracks can simulate the computation of an multitape machine.
- Two tracks are used to store the contents and current position of tape 1, while the other two tracks store the contents and current position of tape 2, etc.





14

Equivalence of Standard TMs and MultiTape TMs

Example 10.1: L = {$a^n b^n$}.

TM with one tape to accept L in Example 9.7.

A two-tape machine to accept L? -- easier.

Assume that an initial string $a^n b^n$ is written on tape 1 at the beginning of the computation.

Then, we read all the *a*'s, copying them onto tape 2.

When we reach the end of the *a*'s, we match the *b*'s on tape 1 against the copied a's on tape 2.

By this way, we can determine whether there are an equal number of *a*'s and *b*'s without repeated back-and-forth movement of the read-write head.

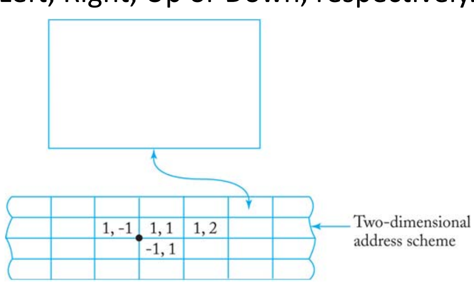
15

Multidimensional Turing Machines

- A *Multidimensional Turing Machine* has a tape that can extend infinitely in more than one dimension.
- In the case of a two-dimensional machine, the transition function must specify movement along both dimensions:

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\}$$ where

L, R, U, D specify movement of the head Left, Right, Up or Down, respectively.



16

Equivalence of Standard TMs and Multidimensional TMs

- The classes are equivalent because a standard TM with two tracks can simulate the computation of a two-dimensional machine.
- In the simulating machine, one track is used to store the cell contents and the other one to keep the associated address.
- The configuration in which cell (1, 2) contains *a* and cell (10, −3) contains *b* is shown below.
- The address track uses a variable field-size arrangement, using a special symbol to delimit the field.

	a				b						
	1	#	2	#	1	0	#	-	3	#	

17

Equivalence of Standard TMs and Multidimensional TMs

- Let's simulate multi(2)-dim TM M in the standard TM M'.
- Assume that, at the start of the simulation of each move,

the RW-head of the 2-dim machine M and the RW-head of the simulating machine M' are always on corresponding cells.

To simulate a move, the simulating machine M' first computes the address of the cell to which M is to move.

Once the address is computed, M' finds the cell with this address on track 2 and then changes the cell contents to account for the move of M. In such a way, M can be simulated in M'.

Thus, a multi-dim. TM is equivalent to a standard TM.

18

Nondeterministic Turing Machines (NTM)

- A *Nondeterministic Turing Machine* is one with potentially many transition choices for a given (state, symbol) combination.

$$\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

- Example 10.2: A transition rule for Nondeterministic TM:

$$\delta(q_0, a) = \{(q_1, b, R), (q_2, c, L)\}$$

Both $q_0aaa \vdash bq_1aa$ and $q_0aaa \vdash q_2\square caa$ are possible.

- Since multiple transitions may be applied at each step, the machine may have multiple active simultaneous threads, any of which may accept the input string when the thread halts.
- For every Nondeterministic TM, there is an equivalent deterministic TM that can simulate its operation. (+: separator of IDs, x: to delimit the area of interest)

□	×	a	q_0	a	a	+	b	b	q_1	a	×	□

19

Equivalence of NTM and DTM

- Theorem 10.2: The class of Deterministic TM and Nondeterministic TM are equivalent.

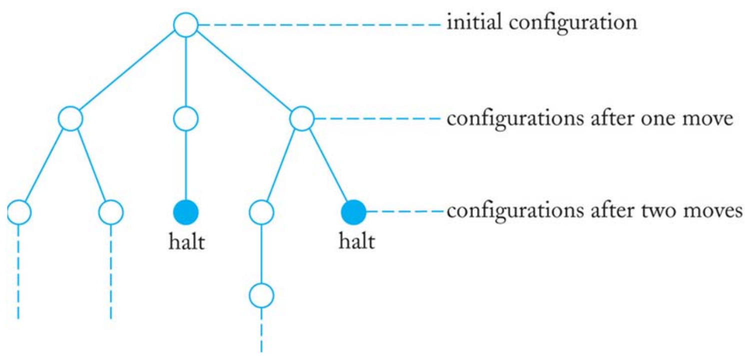
Proof) Construct D-TM to simulate N-TM.

- Since multiple transitions may be applied at each step, the machine may have multiple active simultaneous threads, any of which may accept the input string when the thread halts.
- Keep all possible Instantaneous Descriptions(IDs) of NTM on its tape, separated by some convention (e.g. +): E.g.) aq_0aa, bbq_1a
- For every NTM, there is an equivalent DTM that can simulate its operation. (+: separator of IDs, x: delimiter)
- The simulating machine looks at all active configurations and updates them according to the program of the NTM. New configuration or expanding IDs involve moving 'x' marker.

□	×	a	q_0	a	a	+	b	b	q_1	a	×	□

20

Configuration Tree of NTM



The width depends on the # of options available on each move.

If k = max. branching, M = k^n is the max # of configurations after n moves.

21

Equivalence of NTM and DTM

- Definition 10.3:

  A Nondeterministic TM M is said to ***accept*** a language *L*

  if $\forall w \in$ L, *at least one of the possible configurations accepts w*:

  $$q_0w \vdash^* x_1q_fx_2 \text{ where } q_f \in F,\ x_1, x_2 \in \Gamma^*$$

  i.e. L = { w | $\exists\ q_0w \vdash^* x_1q_fx_2$ where $q_f \in F$, $x_1, x_2 \in \Gamma^*$ }.

  There may be branches that lead to *nonaccepting configurations*, while some may put the machine into an *infinite loop*. But these are irrelevant for acceptance.

- A Nondeterministic TM M is said to ***decide*** *a language L* if, for all $w \in \Sigma^*$, there is a path that leads either *to accept w* or *to reject w* then, *halt.*

22

Decidable vs. Acceptable Language: i.e. Recursive vs. Recursively Enumerable Language

- A language is *recursively enumerable* (*r.e.*) if it is the set of strings *accepted* by some TM, i.e. Turing acceptable language.
- A language is *recursive* (rec.) if it is the set of strings *accepted* by some TM that ***halts on every input***.
- For example, any regular language is recursive.

23

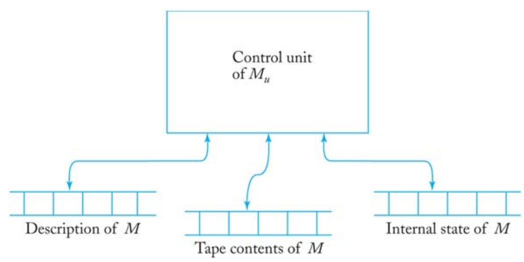
A Universal Turing Machine

- A standard TM is a special purpose computer: Once δ is defined, TM is restricted to carry out one particular type of computation.
- Digital computer is more general-purpose machine that can be programmed to do different jobs at different times → need a TM more in the general purpose machine?
- A *Universal Turing Machine* is a reprogrammable TM which, given as input the description of a TM M and a string *w*, can simulate the computation of M on *w*.
- A Universal TM has the structure of a multitape machine.

24

A Universal Turing Machine

- A *Universal TM (M_U)* is a reprogrammable TM which, given as input the description/encoding of a TM M (<M>), and a string *w* (<*w*>), can simulate the computation of M on *w*.
- A Universal TM has the structure of a multitape machine:
  - Tape 1: Description of M
  - Tape 2: Tape Contents of M
  - Tape 3: Internal state of M



25

A Universal Turing Machine

- A universal-TM M_u is an automaton that, given as input the *description of any TM M* and *a string w*, can simulate the computation of *M* on *w*.
- To construct such an M_u, we first choose a standard way of describing Turing Machines: i.e. encoding of TM M, <M>

<u>Encoding of the *states Q* and the *tape symbols* Γ:</u>

Assume that $Q = \{q_1, q_2, \ldots, q_n\}$, $\Gamma = \{a_1, a_2, \ldots, a_m\}$, with q_1 the initial state, q_2 the single final state, and where a_1 represents the blank. We then select an encoding in which q_1 is represented by 1, q_2 is represented by 11, and so on.

Similarly, a_1 is encoded as 1, a_2 as 11, etc.

The symbol 0 will be used as a separator between the 1's.

With the initial and final state and the blank defined by this convention, any TM can be described completely with δ only.

26

A Universal Turing Machine

Encoding of the transition function δ:

- The transition function is encoded according to this scheme, with the arguments and result in some prescribed sequence.
- For example, $\delta\ (q_1, a_2) = (q_2, a_3, L)$ might appear as

  ⋯ 1 0 1 1 0 1 1 0 1 1 1 0 1 0 ⋯.
- So, any TM has a *finite encoding* as a *string* on $\{0, 1\}^+$ and that, given any encoding of *M* (<M>), we can decode it uniquely.

  Some strings will not represent any TM (e.g., the string 00011).
- A universal-TM, M_u, then has an input alphabet including {0, 1} and the structure of a multitape machine.

(1) M_u looks at the contents of tapes 2 (input symbol) and tape 3 (current state) to determine the configuration of M.

(2) Then, M_u consults tape 1 for a transition δ of M in this configuration.

(3) Finally, tapes 2 and 3 will be modified to reflect the result of the transition.

27

Cantor and Infinity (from Goddard's: chap. 14)

Equal-Size Sets:

- If two *finite sets* are the equal size, one can pair the sets off: e.g.)10 apples with 10 oranges. This is called a *1–1 correspondence*: every apple and every orange is used up.
- So, we say two *infinite sets, A and B* are the equal size ($|A|=|B|$) if there exists a 1–1 *correspondence*.

  In mathematics, there exists a *bijective function f* (both injection (=into) and surjection (=onto))

  f: A $\rightarrow$ B.

28

Cantor and Infinity (from Goddard's: chap. 14)

Countable Sets:

- Define N to be the set of all positive integers: {1, 2, 3, . . .}.
- The set of the positive *even numbers* are the equal size as N:
  - $\exists f: i \rightarrow 2i \ \ \forall i \in \mathrm{N}$
  - one can pair 1 with 2, 2 with 4, 3 with 6, and so on.

  Note that the even numbers are used up: 1 – 2, 2 – 4, 3 – 6, ..
- A set is *countably infinite* if the equal size as N: |A| = |N|
- A set is *countable* if *finite* or *countably infinite.*

  i.e. there is a numbered *enumeration* of all elements.
- E.g.) The rational numbers are countable.
- But, there are sets that are *NOT countable*:

  uncountably infinite or uncountable: e.g.) The real numbers.

29

Cantor and Infinity (from Goddard's: chap. 14 & chap.10 of Lintz)

- Example: The rational numbers (Q) are countable.

Enumeration procedure: How to enumerate the set of rational numbers?

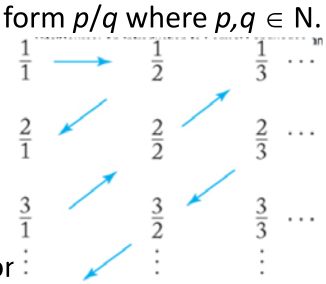
A rational number is a quotient of the form *p/q* where *p,q* ∈ N.

1/1, 1/2, 1/3, 1/4, …

2/1, 2/2, 2/3, 2/4, … $\xRightarrow{\textit{enumerate}}$

3/1, 3/2, 3/3, 3/4, …

……



Count (or list, enumerate) them diagor

i.e. list *p/q* in the non-descending order of (*p+q*),

^{1}1/1, (^{2}½, ^{3}2/1), (^{4}3/1, ^{5}2/2, ^{6}1/3), (^{7}2/3, ^{8}3/2, ^{9}4/1), (^{10}5/1, ^{11}4/2, ^{12}3/3, ^{13}2/4, ^{14}1/5), …

So, there exists a function f: N → Q. Therefore, Q is countable.

30

Cantor's Diagonalization (from Goddard's)

- Given a list of words, one can construct a word not on the list:
  Start with the diagonal as a word, and then
  replace each letter by the next letter in the alphabet.
- Example:

1.	Q	U	I	E	T
2.	S	T	O	N	E
3.	O	F	F	E	R
4.	C	L	E	A	R
5.	P	H	L	O	X

  Here diagonalization produces RUGBY. This is not on the list.
- Diagonalization Always Gives New Word:
  The new word cannot be on the list:
  it is different from 1st word in 1st letter,
  different from 2nd word in 2nd letter, etc.
- Cantor's insight was that same idea works with infinite lists...

31

Cantor's Theorem (Goddard's and Th^m 11.1@Lintz)

- Theorem 11.1: Let S be an countably infinite set.
  Then, its powerset 2^S (or $\wp(S)$) is not countable.
- Cantor's Theorem: The powerset $\wp(N)$ is *not countable*.

Proof by Contradiction)

Suppose $\wp(N)$ is countable. It means we can write down a enumeration of all the subsets of N.

Maybe the list starts: 1 – N, 2 – {4, 7}, 3 – {2, 4, 6, 8}, 4 – $\varnothing$, ...

i.e. We have a function f: $N \rightarrow \wp(N)$ that maps numbers to subsets s.t. every subset appears in the list.

32

Cantor's Theorem (cont.)

- Cantor's Theorem: The powerset $\wp(N)$ is *not countable*.

Proof by Contradiction: cont.)

Now, define a subset T on N:

For each number $i \in N$, look up $f(i)$ and add i to T if $i \notin f(i)$.

But: T is not on list. Why?

- T $\neq f(1)$, because T and $f(1)$ differ on 1. (by definition $1 \in T \Leftrightarrow 1 \notin f(1)$).
- T $\neq f(2)$, because T and f(2) differ on 2. (by def., $2 \in T \Leftrightarrow 2 \notin f(2)$)
- and so on.

Contradiction!! i.e. f is a lie; it doesn't use up the sets in $\wp(N)$.

It means: such an enumeration doesn't exist.

i.e. There doesn't exist such a 1 – 1 function $f: N \rightarrow \wp(N)$.

Therefore, $\wp(N)$ is *not countable.* *Q.E.D.*

33

Cantor's Theorem (cont.)

Immediate Implication of Cantor's Theorem:

1) For any alphabet, the set of TMs is countable.
2) For any alphabet, the set of languages is uncountable.

- The set of TMs is countable because each TM can be represented by a *binary number* in its encoding (<M>), hence as an *integer.*
- However, the subsets of the integers are not countable and hence the number of languages is uncountable.
- Therefore, there exists the languages that are not accepted by any TM, i.e. not recursively enumerable.

34

Enumeration Procedure

- Definition 10.4: Let S be a set of strings on some alphabet Σ. Then, an *enumeration procedure* for S is a TM that can carry out the sequence of steps

$$q_0\square \vdash^* q_s x_1 \# s_1 \vdash^* q_s x_2 \# s_2 \ldots,$$

  with $x_i \in \Gamma^* - \{\#\}$, $s_i \in S$, in such a way that any $s \in S$ is produced in a finite number of steps.
  The state q_s is a state signifying membership in S; i.e., whenever q_s is entered, the string following # must be in S.
- Note: Not every set is countable, i.e. there are some uncountable sets.
- Any set for which an enumeration procedure exists is countable.
- Since S is infinite, an enumeration procedure can't be called an algorithm as it won't terminate.

35

Countability of TM

- Theorem 10.3: The set of all TMs is countable, although infinite.

Proof) Encode each TM using 0 and 1.

With this encoding, construct the following enumeration procedure.

1. Generate the next string in $\{0, 1\}^+$ in proper order.
2. Check the generated string to see if it defines a TM.
   If so, write it on the tape in the form required by Def. 10.4.
   If not, ignore the string.
3. Return to Step 1.

Since every TM has a finite description, any specific machine will eventually be generated by this process.

36

Linear Bounded Automata (LBA)

- The power of a standard TM can be restricted by limiting the area of the tape that can be used.
- E.g.) A PDA may be a N-TM with a tape that is restricted to being used like a stack.
- A *Linear Bounded Automaton* (LBA) is a TM that restricts the usable part of the tape to exactly the cells used by the input.

  i.e. |work space| = |input size|
- Input can be considered as bracketed by two special symbols or markers which can be neither overwritten nor skipped by the read-write head: e.g.) [w]
- LBAs are assumed to be *Nondeterministic-TM* and accept languages in the same manner as other TM accepters.

37

Linear Bounded Automata (LBA)

- Definition 10.5: A *Linear Bounded Automaton* is a Nondeterministic TM M = $(Q, \Sigma, \Gamma, \delta, q_0, \square, F)$, subject to the restriction that Σ must contain two special symbols [and], such that

  $\delta(q_i, [)$ can contain only elements of the form $(q_j, [, R)$, and

  $\delta(q_i,])$ can contain only elements of the form $(q_j,], L)$.

- Definition 10.6: A string w is *accepted* by a LBA if there is a possible sequence of moves

  $$q_0[w] \vdash^* [x_1 q_f x_2] \quad \text{for some } q_f \in F, x_1, x_2 \in \Gamma^*.$$

  The language accepted by the LBA is the set of all such accepted strings.

38

Languages Accepted by Linear Bounded Automata

- It can be shown that any Context-Free Language can be accepted by a Linear Bounded Automaton.
- In addition, LBA can be designed to accept languages which are *not* context-free, such as

$$L = \{ a^n b^n c^n \mid n \geq 1 \}$$

- Example 9.8: TM for L didn't require space outside the original input w, so it can be carried out by a LBA.
- LBA are *not as powerful as* standard Turing machines, while it is difficult to come up with a concrete and explicitly defined language to use as an example.

39

Languages Accepted by LBA (Cont.)

- Example 10.5: Find a LBA that accepts

$$L = \{ a^{n!} \mid n \geq 1 \}$$

Divide the number of a's successively by 2, 3, 4, ..., until we can either accept or reject the string.

If the input is in L, eventually there will be a single a left;

if not, at some point a nonzero remainder will arise.

In a multitrack LBA, use the extra tracks as works pace. Let's use 2-track tape.

1st track: contains the number of a's left during the process of division,

2nd track: contains the current divisor.

Using the divisor on the 2nd track, we divide the number of a's on the 1st track, by removing all symbols except those at multiples of the divisor.

After this, we increment the divisor by one, and continue until we either find a nonzero remainder ($w \notin L$) or are left with a single a ($w \in L$).

	[	a	a	a	a	a	a	]	a's to be examined
	[	a	a	a				]	Current divisor

40

Languages Accepted by LBA (Cont.)

- Example 10.5: Find a LBA that accepts

$L = \{ a^{n!} \mid n \geq 1 \}$

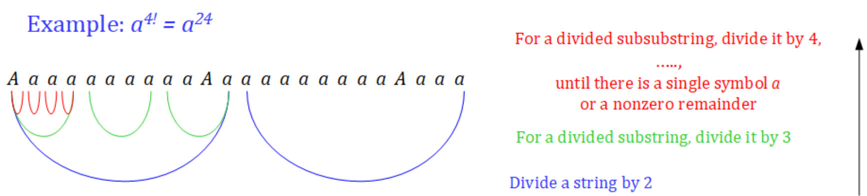
Divide the number of a's successively by 2, 3, 4, ..., until we can either accept or reject the string. In 2-track tape LBA,

1st track: contains the number of a's left during the process of division,

2nd track: contains the current divisor.

Using the divisor on the 2nd track, we divide the number of a's on the 1st track, by removing all symbols except those at multiples of the divisor.

After this, we increment the divisor by one, and continue until we either find a nonzero remainder or are left with a single a.



41

Languages Accepted by LBA (Cont.)

- So, LBA is more powerful than PDA in the Examples where neither of the languages is CFL.
- There exists CFL that can be accepted by an LBA not by a PDA.
- The class of LBA is less powerful than the class of unrestricted TMs.

42