Contents

TURING MACHINE

8.1 Problems that Computer Cannot Solve

8.1.1 Programs that Print "Hello World"

Consider a simple C program with a simple print statement

```
main()
{
    printf("hello, world\n");
}
```

It is easy to discover that this program prints hello, world and terminates.

However, we may have another program that might print hello, world as shown below. It takes an input n, and looks for positive integer solutions to the equation xn+yn=zn. If it finds one, it prints hello, world. If it never finds integers x, y, and z to satisfy the equation, then it continues searching forever and never prints hello, world.

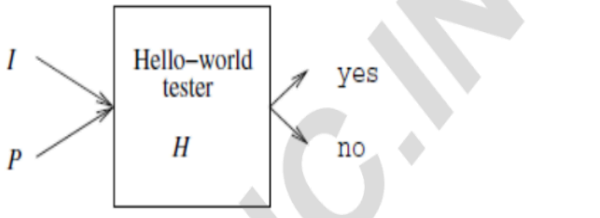
```
int exp(int i, n)
/* computes i to the power n */
{
     int ans, j;
     ans = 1;
     for (j=1; j<=n; j++) ans *= i;
     return(ans);
}

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

If the value of n that the program reads is 2, then it will eventually find combinations of integers such as `total` $= 12$, $x = 3$, $y = 4$, and $z = 5$, for which $x^n + y^n = z^n$. Thus, for input 2, the program *does* print `hello, world`.

However, for any integer $n > 2$, the program will never find a triple of positive integers to satisfy $x^n + y^n = z^n$, and thus will fail to print `hello, world`. Interestingly, until a few years ago, it was not known whether or not this program would print `hello, world` for some large integer n. The claim that it would not, i.e., that there are no integer solutions to the equation $x^n + y^n = z^n$ if $n > 2$, was made by Fermat 300 years ago, but no proof was found until quite recently. This statement is often referred to as "Fermat's last theorem."

8.1.2 Hypothetical tester:

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".



A hypothetical program H that is a hello-world detector

The hypothetical tester is a conceptual tool introduced in the study of undecidable problems to illustrate why certain problems cannot be solved by any algorithm. In this context, the "hypothetical tester" is designed to determine whether a given program, when executed with specific input, will output a particular result—in this case, whether it prints "hello, world" as its first output.

This program, denoted as H, takes two inputs:

1. A program P (the code to be tested).
2. An input I (the input to the program P).

The goal of H is to decide if the program P with input I will produce the output "hello, world" as the first set of characters it prints.

Expected Behavior of H:

• If P with input I prints "hello, world" as its first output, H should return "yes".

• Otherwise, it should return "no".

The challenge arises because:

• A program P may take an infinite amount of time to decide its output. For instance:

• If P is searching for a mathematical solution (like Fermat's Last Theorem) before it prints anything, H cannot predict whether it will eventually find a solution and print "hello, world" or loop indefinitely.

• Even for programs that terminate, analyzing the behavior of every possible program P with every possible input I is nontrivial and undecidable.

The hypothetical tester is a classic example used to demonstrate:
1. **Limits of Computation**: Some problems, even simple-sounding ones, are fundamentally unsolvable by any algorithm.
2. **Undecidability**: The inability to solve a problem algorithmically is a critical concept in theoretical computer science and is at the core of problems like the Halting Problem.

The **Halting Problem** is a fundamental concept in computer science and computational theory. It addresses whether it's possible to design an algorithm that can determine, for any given program and input, whether the program will eventually stop running (halt) or continue running forever.

8.1.3 Reducing one problem to another:

A problem that cannot be solved by computer is called undecidable. A **reduction technique** is used to prove the **undecidability of** a halting problem. 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. The strategy is suggested in the figure given below. The technique is called the reduction of P1 to P2.

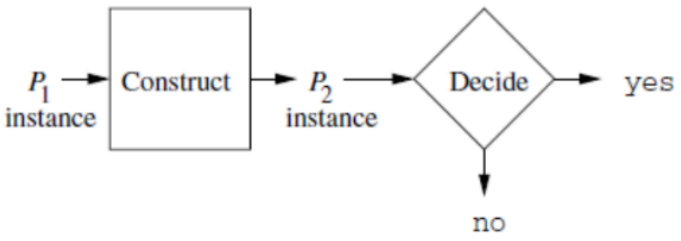


Figure 8.7: If we could solve problem P_2, then we could use its solution to solve problem P_1

Using this technique, a problem P1 is reducible to problem P2 if a solution to the problem P2 can be used to solve the problem P1. Thus,

➢ if P1 is reducible to P2 and P2 is decidable, then P1 is decidable

➢ If P1 is reducible to P2 and P2 is undecidable, then P1 is undecidable.

8.2What is Turing Machine?

In 1936, Alan M Turing proposed the Turing machine as a model of any possible computation. This model is computer like, rather than program like, even though true electronic, or even electromechanical computers were several years in the future.

A Turing machine is a computational model, like Finite Automata (FA), Pushdown automata (PDA), which works on unrestricted grammar. The Turing machine is the most powerful computation model when compared with FA and PDA. Instead of using a stack as in a PDA, the Turing Machine (TM) uses the tape to store the symbols.

8.2.1 Notation of Turing Machine:/Working of TM

The Turing machine consists of a finite control, which can be in any of a finite set of states. There is a tape divided into squares or cells, each cell can hold any one of a finite number of symbols.

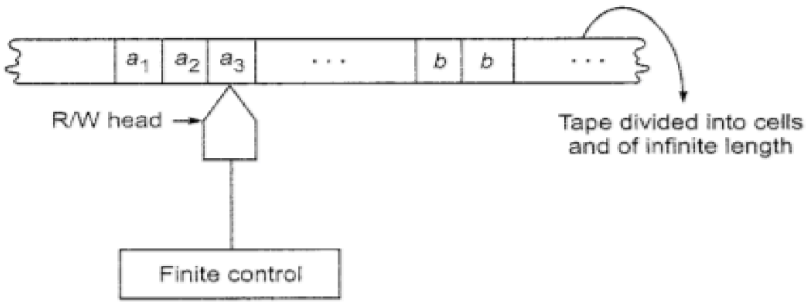


Fig. 9.1 Turing machine model.

Initially, the input, which is a finite length string of symbols chosen from the input alphabet is placed on the tape. All other tape cells, extending infinitely to the left and right, initially hold a special symbol called the blank. The blank is a tape symbol, but not an input symbol, and there may be other tape symbols besides the input symbols and the blank, as well.

There is a tape head that is always positioned at one of the tape cells. The Turing machine is said to be scanning that cell.Initially, the tape head is at the leftmost cell that holds the input.

A Move of the Turing machine is a function of the state of the finite control and the tape symbol scanned. In one move, the Turing machine will,
a) Change state: The next state optionally may be the same as the current state.
b) Write a tape symbol in the cell scanned. This tape symbol replaces whatever symbol was in that cell. Optionally, the symbol written may be the same as the symbol currently there.
c) Move the tape head left or right. In our formalism, we require a move, and do not allow the head to remain stationary.
This restriction does not constrain what a Turing machine can compute, since any sequence of moves with a stationary head could be condensed, along with the next tape head move, into a single state change, a new tape symbol, and a move left or right.

Definition of Turing Machine:

A Turing machine M is a 7-tuple, namely **(Q, Σ, Γ, δ, qo, B, F)**, where,

- Q: is the finite nonempty set of states.
- Γ is a finite nonempty set of tape symbols,
- B ∈ Γ is the blank.
- Σ is a nonempty set of input symbols and is a subset of Γ and B ∉ Σ.
- δ is the transition function

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

The arguments of δ (q, X) are a state q and a tape symbol X.
The value of δ(q, X) is a triple (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 the direction, either L or R, standing for "left" or "right" specifying the direction in which the R/W head moves.
- $q_o \in Q$ is the initial state, and
- F ⊆ Q is the set of final or accepting states.

8.2.2 Instantaneous Description (IDs) of Turing Machine:

In Turing Machine, the instantaneous description is defined on the whole string and the current state of the machine.
Thus, we shall use the string $X_1X_2,\ldots\ldots X_{i-1}\,\mathbf{q}X_iX_{i+1}\ldots\ldots X_n$ to represent an ID in which,

1. q is the state of the Turing machine.
2. The tape head is scanning the ith symbol from the left.
3. $X_1X_2,\ldots X_n$ is the portion of the tape between the leftmost and the right most nonblank. As an exception, if the head is to the left of the leftmost nonblank or to the right of the rightmost nonblank, then some prefix or suffix of $X_1X_2,\ldots X_n$ will be blank.

- We use $|\text{-- }_M$ to designate a move of a Turing machine M from one ID to another.
- If $\delta(q, X_i) = (p, Y, L)$, then:

  $X_1X_2 \cdots X_{i-1}qX_iX_{i+1} \cdots X_n \; |\text{-- }_M$
  $X_1X_2 \cdots X_{i-2}pX_{i-1}Y\,X_{i+1} \cdots X_n$
- If $\delta(q, X_i) = (p, Y, R)$, then:

  $X_1X_2 \cdots X_{i-1}qX_iX_{i+1} \cdots X_n \; |\text{-- }_M$
  $X_1X_2 \cdots X_{i-1}Y\,pX_{i+1} \cdots X_n$

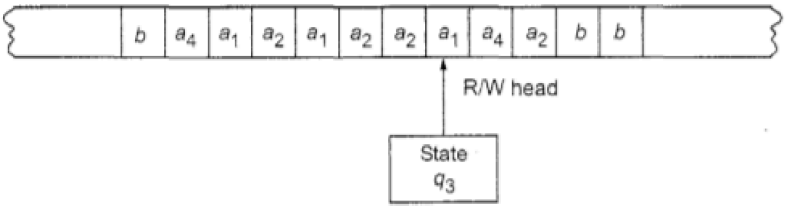


Fig. 9.2 A snapshot of Turing machine.

Thus, the ID is as given in Fig. 9.3.

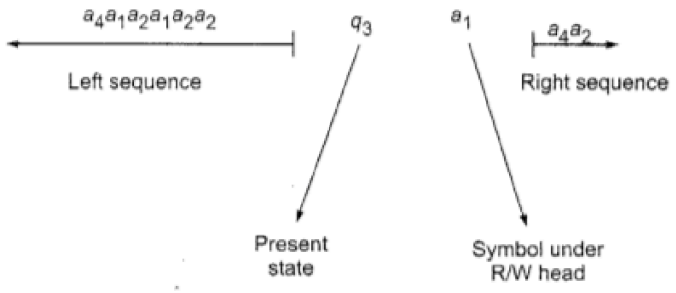


Fig. 9.3 Representation of ID.

8.2.3 Transition Table of Turing Machine:

Let us design a Turing machine and see how it behaves on a typical input. The TM we construct will accept the language
L={0ⁿ1ⁿ:n>=1}
Initially, it is given a finite sequence of 0's and 1's on its tape, preceded and followed by an infinity of blanks. Alternately, the TM will change a 0 to an X and then a 1 to a Y, until all 0's and 1's has been matched.
The formal specification of the TM M is,

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

where δ is given by the table in Fig. 8.9.

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	–	–	–	–	–

Figure 8.9: A Turing machine to accept $\{0^n1^n \mid n \geq 1\}$

Above table is called as transition table.

8.2.4 Transition Diagrams for Turing Machines:

A transition diagram for TM consists of a set of nodes corresponding to the states of the TM. An arc from state q to state p is labeled by one or more items of the form X/Y, D where X and Y are tape symbols, and D is a direction, either L or R. That is, whenever δ(q,X)=(p, Y, D), we find the label X/Y, D on the arc from q to p.

Examples 1:

Let $L = \{0^n 1^n : n \geq 1\}$, which is a CFL. We will design a Turing machine M accepting L. Let

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

where $Q = \{q_0, q_1, \ldots, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, X, Y, B\}$, $F = \{q_4\}$, and δ is shown in the following figure.

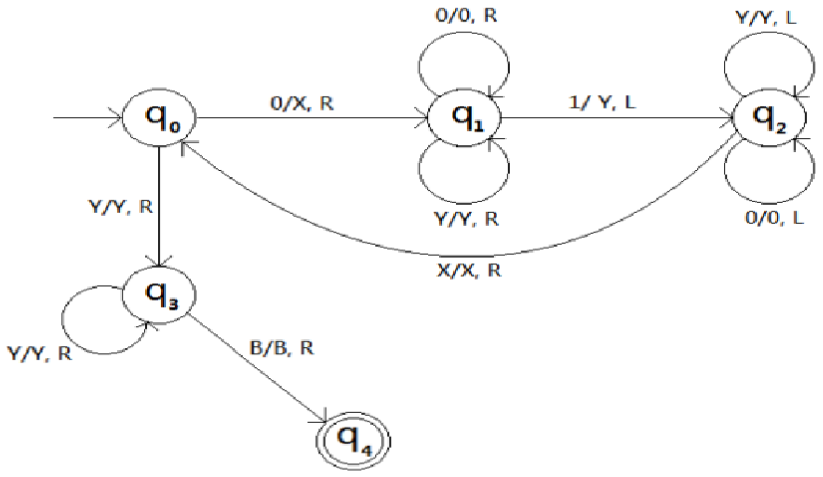


Figure 3: Turing machine accepts $\{0^n 1^n : n \geq 1\}$

Let show the IDs for the input strings 0011 and 0010.
Example

Here is an example of an accepting computation by M. Its input is 0011. Initially, M is in state q_0, scanning the first 0, i.e., M's initial ID is $q_0 0011$. The entire sequence of moves of M is:

$$q_0 0011 \vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y 1 \vdash q_2 X 0 Y 1 \vdash$$
$$X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash X X q_2 Y Y \vdash X q_2 X Y Y \vdash$$
$$X X q_0 Y Y \vdash X X Y q_3 Y \vdash X X Y Y q_3 B \vdash X X Y Y B q_4 B$$

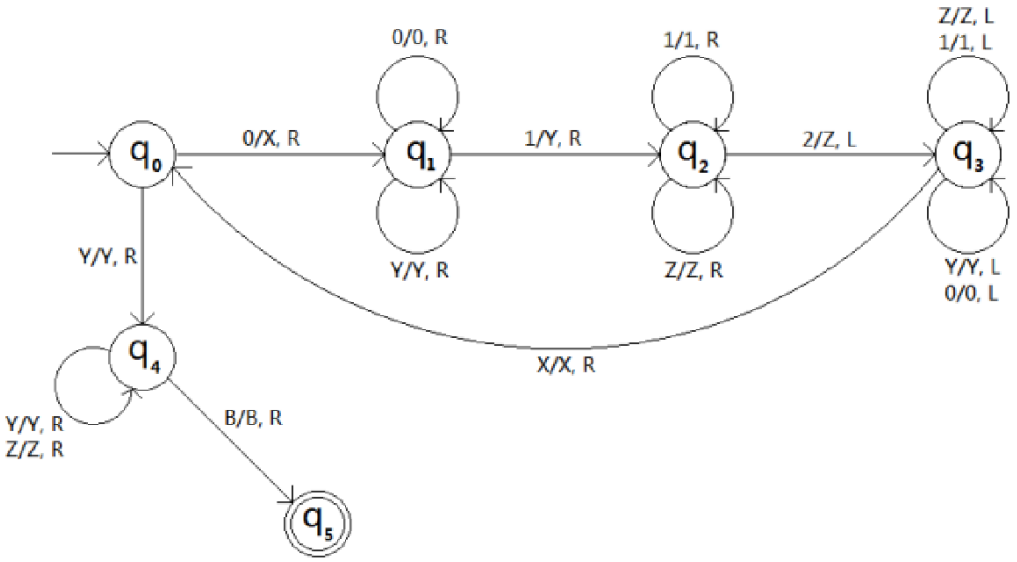
For another example, consider what M does on the input 0010, which is not in the language accepted.

$$q_0 0010 \vdash X q_1 010 \vdash X 0 q_1 10 \vdash X q_2 0 Y 0 \vdash q_2 X 0 Y 0 \vdash$$
$$X q_0 0 Y 0 \vdash X X q_1 Y 0 \vdash X X Y q_1 0 \vdash X X Y 0 q_1 B$$

The behavior of M on 0010 resembles the behavior on 0011, until in ID $X X Y q_1 0$ M scans the final 0 for the first time. M must move right, staying in state q_1, which takes it to the ID $X X Y 0 q_1 B$. However, in state q_1 M has no move on tape symbol B; thus M dies and does not accept its input. □

Example 2:
Construct a T.M. to accept the language L= $\{0^n 1^n 2^n : n>=1\}$.
Ans:

The Turing machine works similarly with the above example: it marks the first 0 as X, the first 1 as Y, the first 2 as Z; then it goes back to the second 0, and marks it as X, the second 2 as Y, the second 2 as Z, and so on.



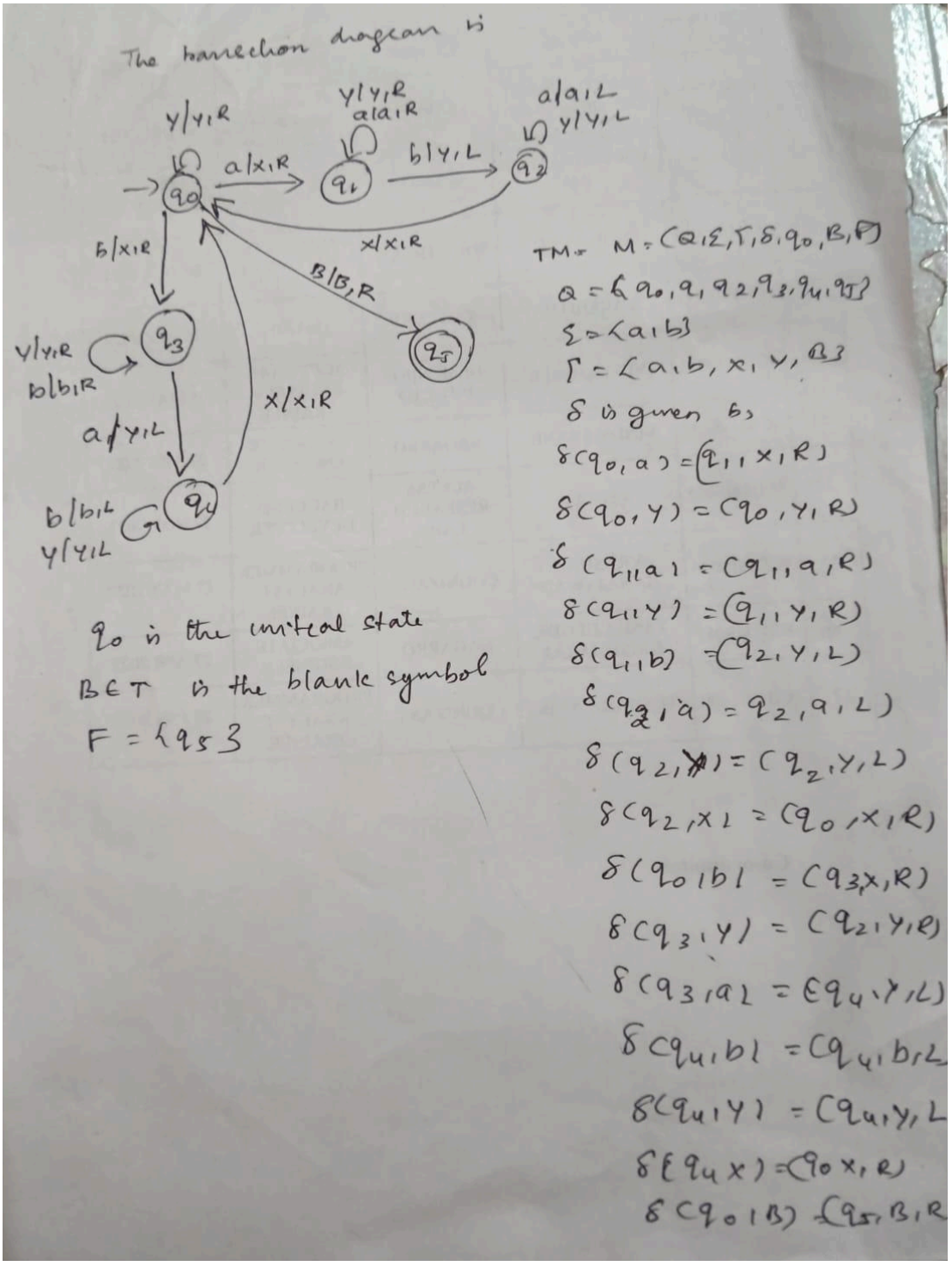
Mention 7 tuple representation.

Example 3:

Obtain a TM to accept a string of a's and b's such that $n_a(w)=n_b(w)$.

Ans:

The general procedure: When TM is in initial state q0, first symbol of the input string can be either a or b. Replace the first symbol a (or b) by X and search for next symbol b (or a) and replace it by symbol Y. Repeat this until blank symbol B is scanned



Example 4: Let x and y are two positive integers. Obtain a turing machine to perform x+y. (Addition operation on TM)

Ans: Let us represent the numbers on the tape using unary representation 1'. Let 0 be the separator.

Ex: x=5 can be represented as 11111 and y=3 can be represented as 111

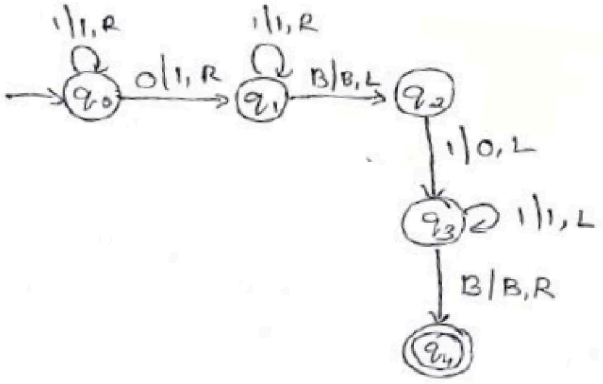
Then 5+3 can be represented on the tape together with 0 as a separated,

1 1 1 1 1 0 1 1 1

Then output should be of the form:

1 1 1 1 1 1 1 1 0

ic, read all 1's until we encounter the 0. Then change 0 to 1 and keep moving right until blank symbol is encounter. Once found move left and change 1 to 0 and keep moving right read sum from leftmost blank symbol.
The transition diagram is,



EXAMPLE 3: Design a turing Maxhine for proper subtraction function:

Turing machine might compute the function which is called minus or proper subtraction and is defined by m -n =max(m-n,0) =if m>=n then m-n else 0(if m<n)

State	Symbol 0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	—
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	—
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
q_6	—	—	—

Figure 8.11: A Turing machine that computes the proper-subtraction function

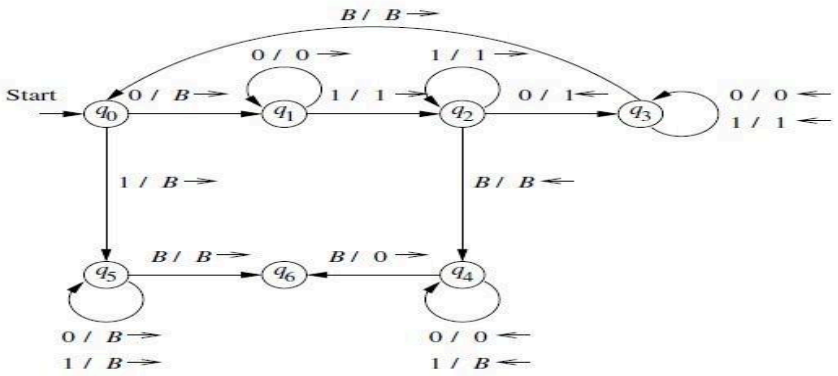
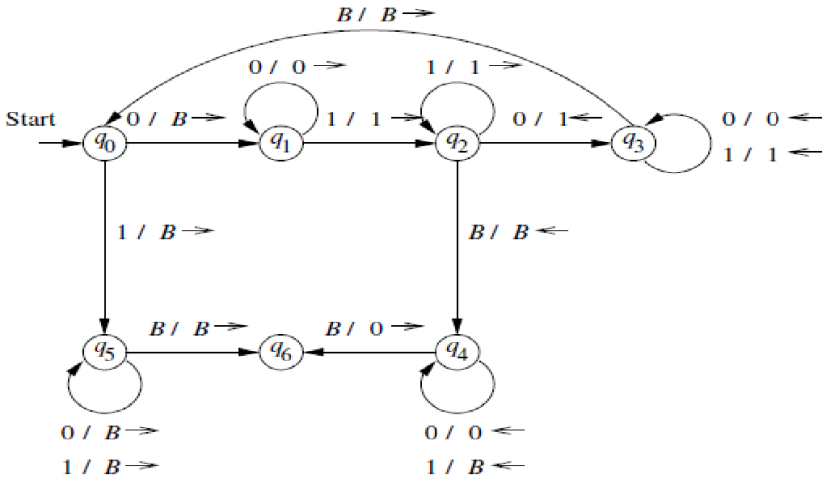


Figure 8.12: Transition diagram for the TM of Example 8.4

Example 7: Monus operation or propoer subtraction on TM.
Let us,

show how a Turing machine might compute the function $\dot{-}$, which is called *monus* or *proper subtraction* and is defined by $m \dot{-} n = \max(m - n, 0)$. That is, $m \dot{-} n$ is $m - n$ if $m \geq n$ and 0 if $m < n$.

TM, M will start with a tape consisting of 0m 1 0n surrounded by blanks.
M halts with 0m-n on its tape, surrounded by blanks.
for example 5-3 can be represented as,
B000001000B
The result 5-3=2 will be represented on the tape as,
BBBBB00BBBB
The transition diagram is,Here the symbol→ indicates R(right)
and ≈ indicates L (left)



8.2.5 The Language of a Turing Machine

In TM the input string is placed on the tape, and the tape head begins at the leftmost input symbol. If the TM eventually enters an accepting state, then the input is accepted, and otherwise not.
More formally, let M= (Q, Σ, Γ, δ, qo, B, F) be a Turing machine. Then L(M) is the set of strings w in Σ* such that $q_0 w \vdash^* \alpha p \beta$ for some state p in F and any tape strings α and β.
The set of languages we can accept using a Turing machine is often called the recursively enumerable languages or RE languages.

8.2.6 Turing Machines and Halting Problem:

There is another notion of "acceptance" that is commonly used for Turing machines "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, δ(q,X) is undefined.

8.3Programming Techniques for TM's

TM's may be used as a computer as well, not just a language recognizer.

Techniques are:

1. Storage in the State
2. Multiple Tracks

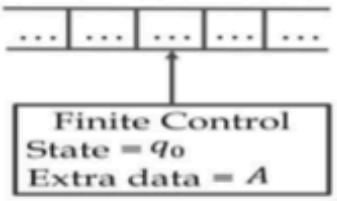
3. Subroutines

8.3.1 Storage in the State:

A state that consists of a fixed number of fixed-size components can be made into a tuple. The behaviour of a TM programme can be made simpler by allowing the tuple's components to store a predetermined amount of data.

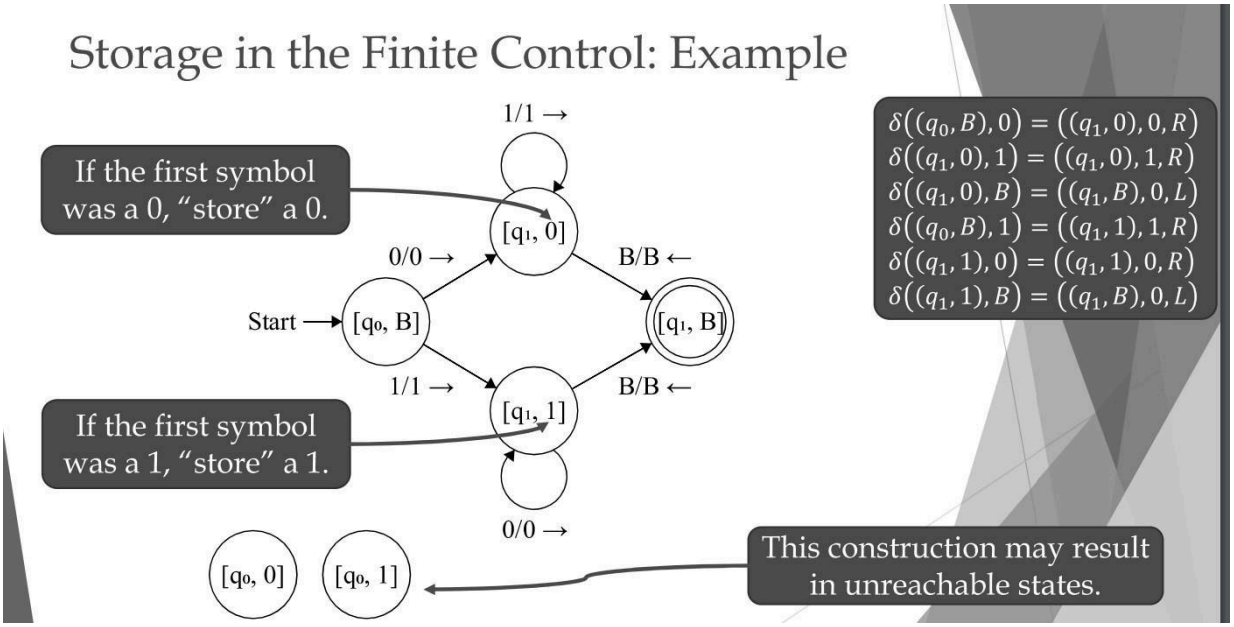
Storage in the finite control:

- You can keep track of a *finite* amount of extra data by incorporating it into the *state* of a TM.
- Example: Keep track of an additional symbol:
- If the "extra data" can be A or B, and the "state" can be q_0 or q_1, then the actual states of the TM can be $\{[q_0, A], [q_1, A], [q_0, B], [q_1, B]\}$.



- Example: Scan the input and ensure the first symbol doesn't appear a second time.
- $M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, [q_0, B], B, F)$
- $Q = \{q_0, q_1\} \times \{0, 1, B\}$
- $F = \{[q_1, B]\}$
- $\delta([q_0, B], 0) = ([q_1, 0], 0, R)$
- $\delta([q_1, 0], 1) = ([q_1, 0], 1, R)$
- $\delta([q_1, 0], B) = ([q_1, B], 0, L)$
- $\delta([q_0, B], 1) = ([q_1, 1], 1, R)$
- $\delta([q_1, 1], 0) = ([q_1, 1], 0, R)$
- $\delta([q_1, 1], B) = ([q_1, B], 0, L)$
- Note: This doesn't change the fact that the TM has a finite number of states!

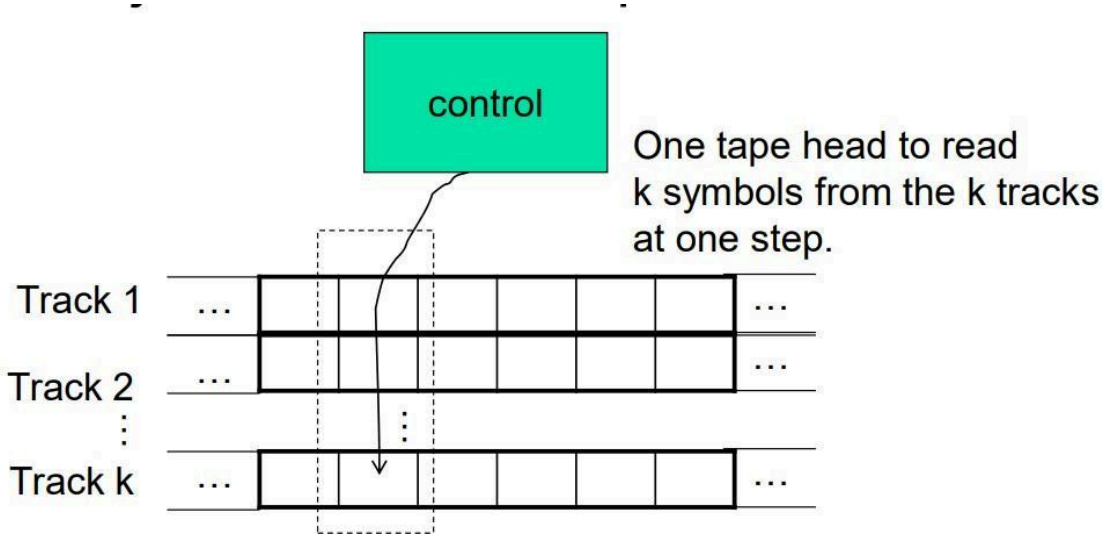
Example: Design a TM 01*+10* ; Keep track of an additional symbol. The actual states of the TM can be [q0, A], [q1, A], [q0, B], or [q1, B]



8.3.2 Multiple Tracks:

TM with multiple tracks, but just one unified tape head

A particular kind of multi-tape Turing machine called a multi-track Turing machine has numerous tracks, but only one tape head can read and write on each track. One tape head reads n symbols from n tracks in this instance. Recursively enumerable languages are accepted, just like they are for singletrack, single-tape Turing machines.



A Multi-track Turing machine can be formally described as a 6-tuple (Q, X, ∑, δ, q0, F) where

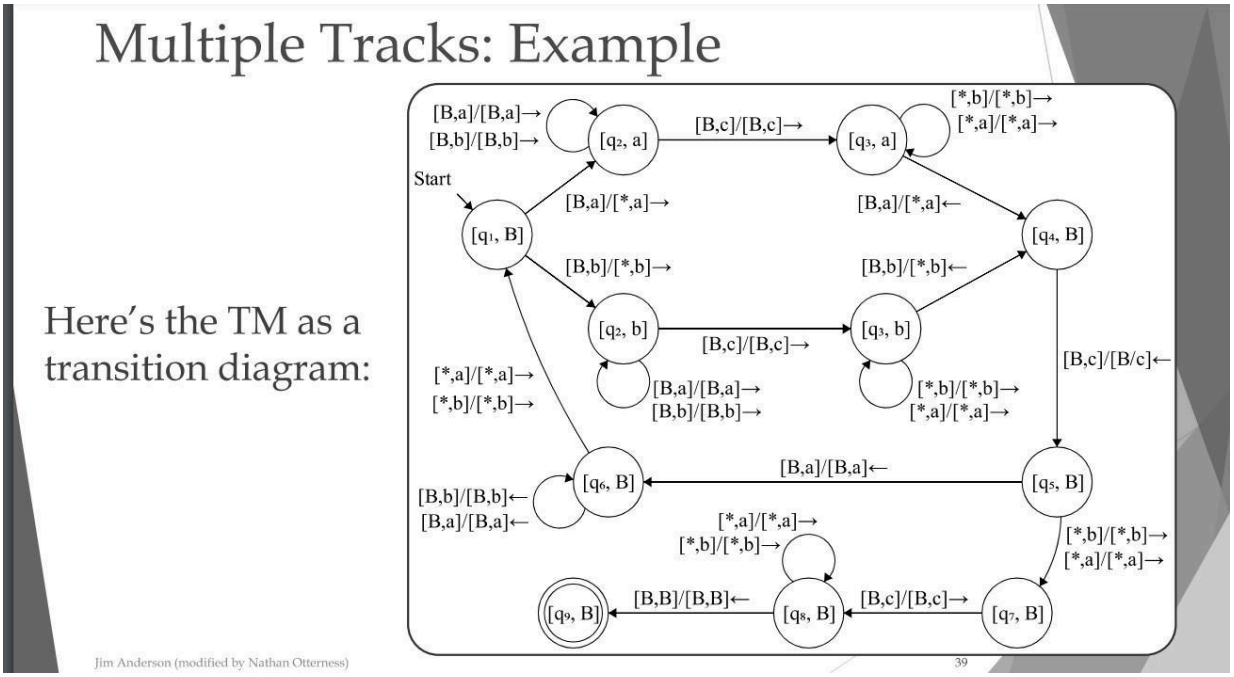
Q is a finite set of states

X is the tape alphabet ∑ is the input alphabet δ is a relation on

states and symbols where δ(Qi, [a1, a2, a3,….]) = (Qj, [b1, b2,

b3,….], Left_shift or Right_shift) q0 is the initial state

F is the set of final states



8.3.3. Subroutines

TMs can emulate subroutines, including those that send parameters and use recursion.

Example: Multiplication using a "copy" subroutine

Given 0^m10^n1 as input, produce 0^{mn} as output. This can be done by "copying" n 0's m times Here's how the overall TM will work:

1. The tape will, in general, have one nonblank string of the form $0^i10^n10^{kn}$ for some k.
2. In one basic step, we change a 0 in the first group to B and add n 0's to the last group, giving us a string of the form $0^{i-1}10^n10^{(k+1)n}$.
3. As a result, we copy the group of n 0's to the end m times, once each time we change a 0 in the first group to B. When the first group of 0's is completely changed to blanks, there will be mn 0's in the last group.
4. The final step is to change the leading 10^n1 to blanks, and we are done.

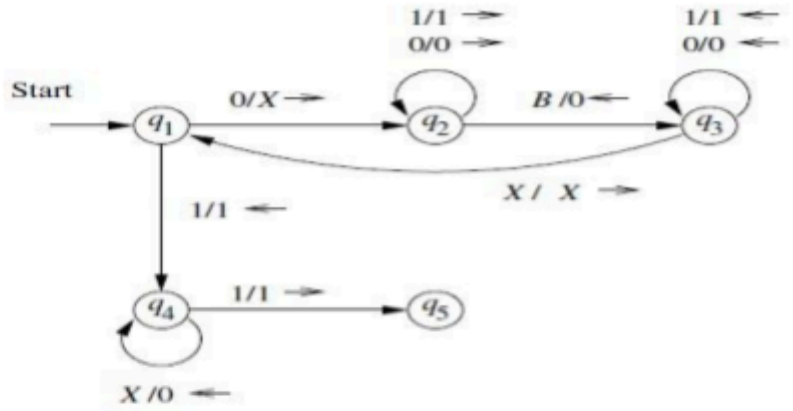
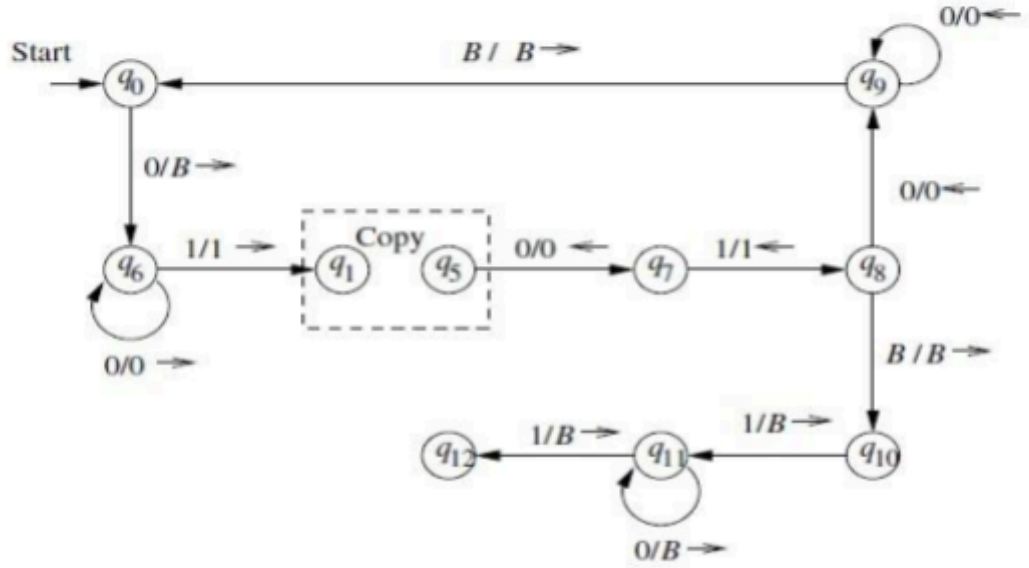


Figure 8.14: The subroutine Copy



gure 8.15: The complete multiplication program uses the subroutine Copy

8.4 Extensions to the TM:

1. Multi Tape TM
2. Equivalence of One Tape & Multi Tape TMs

3.Running Time & Many Tapes to One Construction'

4. Non Deterministic TMs

8.4.1MultiTape TM:

A multitape TM is as suggested by Fig. 8.16. The device has a finite control (state), and some finite number of tapes. Each tape is divided into cells, and each cell can hold any symbol of the finite tape alphabet. As in the single-tape TM, the set of tape symbols includes a blank, and has a subset called the input symbols, of which the blank is not a member. The set of states includes an initial state and some accepting states. Initially:

1. The input, a finite sequence of input symbols, is placed on the first tape.
2. All other cells of all the tapes hold the blank.
3. The finite control is in the initial state.
4. The head of the first tape is at the left end of the input.
5. All other tape heads are at some arbitrary cell. Since tapes other than the first tape are completely blank, it does not matter where the head is placed initially; all cells of these tapes "look" the same.

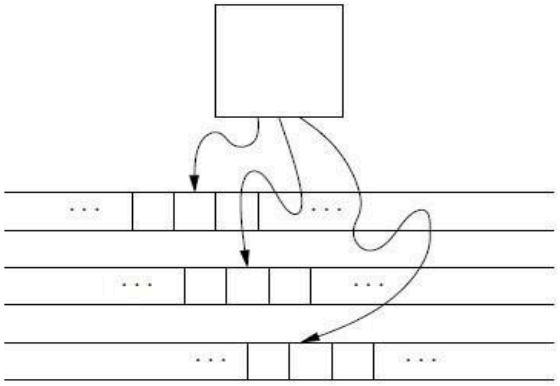


Figure 8.16: A multitape Turing machine

A move of the multitape TM depends on the state and the symbol scanned by each of the tape heads. In one move, the multitape TM does the following:

1. The control enters a new state, which could be the same as the previous state.
2. On each tape, a new tape symbol is written on the cell scanned. Any of these symbols may be the same as the symbol previously there.
3. Each of the tape heads makes a move, which can be either left, right, or stationary. The heads move independently, so different heads may move in different directions, and some may not move at all.

8.4.2 Equivalence of One-Tape and Multitape TM's

Recall that the recursively enumerable languages are defined to be those accepted by a one-tape TM. Surely, multitape TM's accept all the recursively enumerable languages, since a one-tape TM *is* a multitape TM. However, are there languages that are not recursively enumerable, yet are accepted by multitape TM's? The answer is "no," and we prove this fact by showing how to simulate a multitape TM by a one-tape TM.

Theorem 8.9: Every language accepted by a multitape TM is recursively enumerable.

PROOF: The proof is suggested by Fig. 8.17. Suppose language L is accepted by a k-tape TM M. We simulate M with a one-tape TM N whose tape we think of as having $2k$ tracks. Half these tracks hold the tapes of M, and the other half of the tracks each hold only a single marker that indicates where the head for the corresponding tape of M is currently located. Figure 8.17 assumes $k = 2$. The second and fourth tracks hold the contents of the first and second tapes of M, track 1 holds the position of the head of tape 1, and track 3 holds the position of the second tape head.

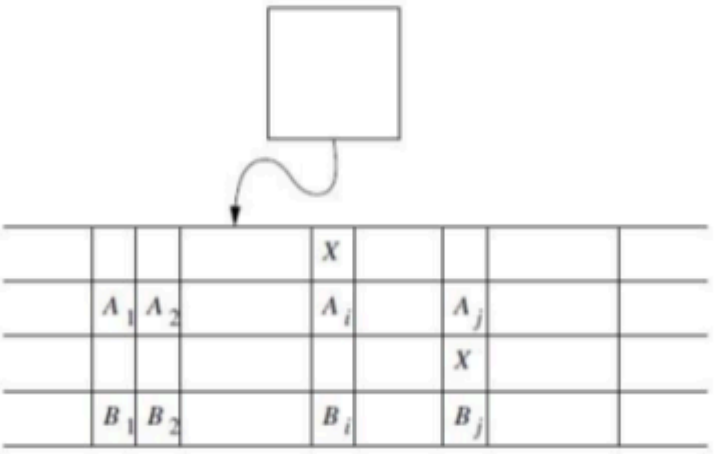


Figure 8.17: Simulation of a two-tape Turing machine by a one-tape Turing machine

8.4.3 Running Time and the Many-Tapes-to-One Construction

Let us now introduce a concept that will become quite important later: the "time complexity" or "running time" of a Turing machine. We say the *running time* of TM M on input w is the number of steps that M makes before halting. If M doesn't halt on w, then the running time of M on w is infinite. The *time complexity* of TM M is the function $T(n)$ that is the maximum, over all inputs

Theorem 8.10: The time taken by the one-tape TM N of Theorem 8.9 to simulate n moves of the k-tape TM M is $O(n^2)$.

PROOF: After n moves of M, the tape head markers cannot have separated by more than $2n$ cells. Thus, if N starts at the leftmost marker, it has to move no more than $2n$ cells right, to find all the head markers. It can then make an excursion leftward, changing the contents of the simulated tapes of M, and moving head markers left or right as needed. Doing so requires no more than $2n$ moves left, plus at most $2k$ moves to reverse direction and write a marker X in the cell to the right (in the case that a tape head of M moves right).

Thus, the number of moves by N needed to simulate one of the first n moves is no more than $4n + 2k$. Since k is a constant, independent of the number of moves simulated, this number of moves is $O(n)$. To simulate n moves requires no more than n times this amount, or $O(n^2)$. □

8.4.4. Non Deterministic TM

- **8.4.4 Nondeterministic TM's**
  - A nondeterministic TM (NTM) has multiple choices of next moves, i.e.,
    $\delta(q, X) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$.
  - The NTM is not more 'powerful' than a deterministic TM (DTM), as said by the following theorem.
  - **Theorem 8.11**
    If M_N is NTM, then there is a DTM M_D such that $L(M_N) = L(M_D)$. (for proof, see the textbook)

56

- The equivalent DTM constructed for an NTM in the last theorem may take exponentially more time than the DTM.
- It is unknown whether or not this *exponential slowdown* is necessary!

Theorem 8.11 : If M_N is a nondeterministic Turing machine, then there is a deterministic Turing machine M_D such that $L(M_N) = L(M_D)$.

PROOF: M_D will be designed as a multitape TM, sketched in Fig. 8.18. The first tape of M_D holds a sequence of ID's of M_N, including the state of M_N. One ID of M_N is marked as the "current" ID, whose successor ID's are in the process of being discovered. In Fig. 8.18, the third ID is marked by an x along with the inter-ID separator, which is the $*$. All ID's to the left of the current one have been explored and can be ignored subsequently.

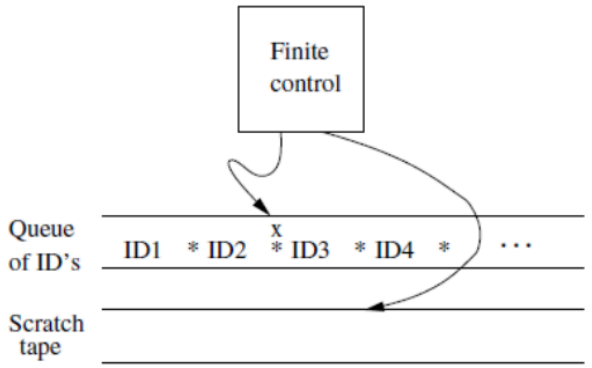


Figure 8.18: Simulation of an NTM by a DTM

8.5 Decidability -