Variations of Turing Machines

The variants are

(i) Turing machines with two-way infinite tapes

(ii) multitape Turing machines

(iii) multihead Turing machines

(iv) nondeterministic Turing machines

(v) Turing machines with two-dimensional tapes

Two-Way Infinite Tape Turing Machine

A two-way infinite tape Turing machine (TTM) is a Turing machine with its input tape infinite in both directions, the other components being the same as that of the basic model. We see in the following theorem that the power of TTM is no way superior of that of the basic TM.

That a one-way TM M_0 can be simulated by a two-way TM M_D can be seen easily. M_D puts a # to the left of the leftmost nonblank and moves its head right and simulates M_0. If M_D reads # again, it halts rejecting the input as this means M_0 tries to move off the left end of the tape.

Proof The tape of the M_D at any instance is of the form

$\not b$	$\ldots$	a_{-2}	a_{-1}	a_0	a_1	a_2	$\ldots$	a_n	$\not b$

where a_0 is the symbol in the cell scanned by M_D initially. M_0 can represent this situation by two tracks

a_0	a_1	a_2	$\ldots$
$\#$	a_{-1}	a_{-2}	$\ldots$

When M_D is to the right of a_0, the simulation is done on the upper track. When M_D is to the left of a_0, the simulation is done in M_0 on the lower track.

The initial configuration would be

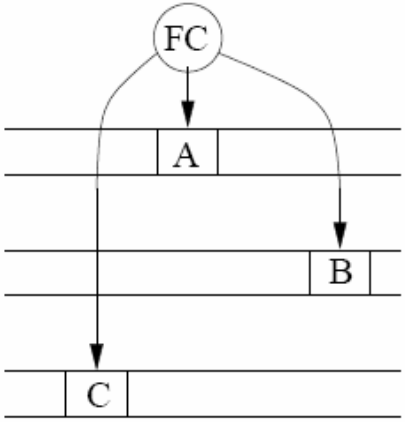
a_0	a_1	$\ldots$	a_n
$\not b$	$\not b$	$\not b$	$\not b$

$M_0 = (K', \Sigma', \Gamma', q_0', F')$ where
$K' = \{q_0'\} \cup (K \times \{1, 2\})$
$\Sigma' = \Sigma \times (\Sigma \cup \{\#\})$
$\Gamma' = \Gamma \times (\Gamma \cup \{\#\})$
$F' = \{[q, 1], [q, 2] | q \in F\}$
δ' is defined as follows:
If $\delta(q, a) = (q', c, L/R)$ and if the head of M_D is to the right of a_0 we have $\delta([q, 1], [a, b]) = ([q', 1], [c, b], L/R)$; simulation is done on the upper track 1.
If M_D is to the left of the initial position, simulation is done on the lower

track.
If $\delta(q, b) = (q', c, L/R)$
$\delta'([q, 2], [a, b]) = ([q', 2], [a, c], R/L)$
The initial move will be
$\delta'(q_0', [a_0, \not{b}]) = ([q, 1], [A, \#], R)$
if $\delta(q_0, a_0) = (q, A, R)$
$\delta'(q_0', [a_0, \not{b}]) = ([q, 2], [A, \#], R)$
if $\delta(q_0, a_0) = (q, A, L)$
When reading the leftmost symbol M_D behaves as follows:
If $\delta(q, a) = (p, A, R)$
$\delta'([q, 1/2], [a, \#]) = ([p, 1], [A, \#], R)$
If $\delta(q, a) = (p, A, L)$
$\delta'([q, 1/2], [a, \#]) = ([p, 2], [A, \#], R)$
while simulating a move when M_D is to the left of the initial position, M_0 does it in the lower track always moving in a direction opposite to that of M_D. If M_D reaches an accepting state q_f, M_0 reaches $[q_f, 1]$ or $[q_f, 2]$ and accepts the input.

Multitape Turing Machine

Suppose we have a 3-tape TM.



A multitape TM can be simulated by a single tape TM.

Proof Let $M = (K, \Sigma, \Gamma, \delta, q_0, F)$ be a k-tape Turing machine. It can be simulated by a single tape TM M' having $2k$ tracks. Odd numbered tracks contain the contents of M's tapes.

		...	A	...	
		...	X	...	
		...		...	B
		...		...	X
	C	...		...	
	X	...		...	

To simulate a move of the multitape TM M, the single tape TM M' makes two sweeps, one from left to right and another from right to left. It starts on the leftmost cell which contains a X in one of the even tracks. While moving from left to right, when it encounter a X, it stores the symbol above it in its finite control. It keeps a counter as one of the component of the state to check whether it has read the symbols from all the tapes. After the left to right move is over, depending on the move of M determined by the symbols read, the single tape TM M' makes a right to left move changing the corresponding symbols on the odd tracks and positioning the X's properly in the even numbered tracks. To simulate one move of M, M' roughly takes (it may be slightly more depending on the left or right shift of X) a number of steps equal to twice the distance between the leftmost and the rightmost cells containing a X in an even numbered track. When M' starts all X's will be in one cell. After i moves the distance between the leftmost and rightmost X can be at most $2i$. Hence to simulate n moves of M, M' roughly takes $\sum_{i=1}^{n} 2(2i) = \sum_{i=1}^{n} 4i = O(n^2)$ steps. If M reaches a final state, M' accepts and halts. $\square$

Multihead Turing Machine

A multihead TM is a single tape TM having k heads reading symbols on the same tape. In one step all the heads sense the scanned symbols and move or write independently.

A multihead TM M can be simulated by a single head TM M'.

Proof Let M have k heads. Then M' will have $k+1$ tracks on a single tape. One track will contain the contents of the tape of M and the other tracks are used to mark the head positions. One move of M is simulated by M' by making a left to right sweep followed by a right to left sweep. The simulation is similar to the one given in Theorem 10.1.2. One fact about which one has to be careful here is the time when two heads scan the same symbol and try to change it differently. In this case some priority among heads has to be used.

Nondeterministic Turing Machine $\delta : K \times \Gamma \to \mathbb{P}(K \times \Gamma \times \{L, R\})$

Every NTM can be simulated by a deterministic TM (basic model).

Computation of a NTM 'N' on any input w is represented as a tree. Each branch is a branch of nondeterminism. Each node is a configuration of N. Root will be the start configuration. One has to traverse the whole tree in a 'breath-first' manner to search for a successful path. One cannot proceed by 'depth-first' search as the tracing may lead to an infinite branch while missing the accepting configurations of some other branches.

Using a multi-tape TM one can simulate N on a given input. For each path of the tree, simulation is done on a separate tape. The paths are considered one-by-one in the increasing order of depth and among paths of equal length, the paths are considered from left to right.

Let us see how the implementation works on a DTM with the tapes. There is an input tape containing input which is never altered. Second tape will be a simulation tape which contains a copy of N's tape content on some branch of its nondeterministic computation. The third tape keeps track of the location of the DTM in NTM's computation tree. The three tapes may be called as input tape, simulation tape and address tape.

Suppose every node in the tree has at most b children. Let every node in the tree have address which is a string over the alphabet $\Sigma_b = \{1, 2, \ldots, b\}$ (say). To obtain a node with address 145, start at the root going to its child numbered 1, move to its 4th child and then move to the nodes that corresponds to its 5th child. Ignore addresses that are meaningless. Then in a breath-first manner check the nodes (configurations) in lexicographic order as $\epsilon, 1, 2, 3, \ldots, b, 11, 12, 13, \ldots, 1b, 21, 22, \ldots, 2b, \ldots, 111, 112, \ldots$ (if exist). Then DTM on input $w = a_1 \ldots a_n$ works as follows. Place w on the input tape and the others are empty. Copy the contents of the input tape to simulation tape. Then simulate NTM's one nondeterministic branch on the simulation tape. On each choice, consult the address tape for the next move. Accept if the accepting configuration is reached. Otherwise abort this branch of simulation. The abortion will take place for the following reasons.

- symbols on address tape are all used.
- rejecting configurations encountered
- nondeterministic choice is not a valid choice.

Once the present branch is aborted, replace the string on the address tape with the next string in lexicographic ordering. Simulate this branch of the NTM as before. □

Two Dimensional Turing Machine

The Turing machine can have two dimensional tapes. When the head is scanning a symbol, it can move left, right, up or down. The smallest rectangle containing the nonblank portion is $m \times n$, then it has m rows and n columns. A one dimensional TM which tries to simulate this two dimensional TM will have 2 tapes. On one tape this m rows of n symbols each will be represented as m blocks of size n each separated by markers. The second tape is used as scratch tape. When the two dimensional TM's head moves left or right, it is simulated in a block of the one dimensional TM. When the two dimensional TM's head moves up or down, the one dimensional TM's head moves to the previous block or the next block. To move to the correct position in that block, the second tape is used. If m or n increases, number of blocks or the size of the blocks is increased.

Restricted Turing Machines

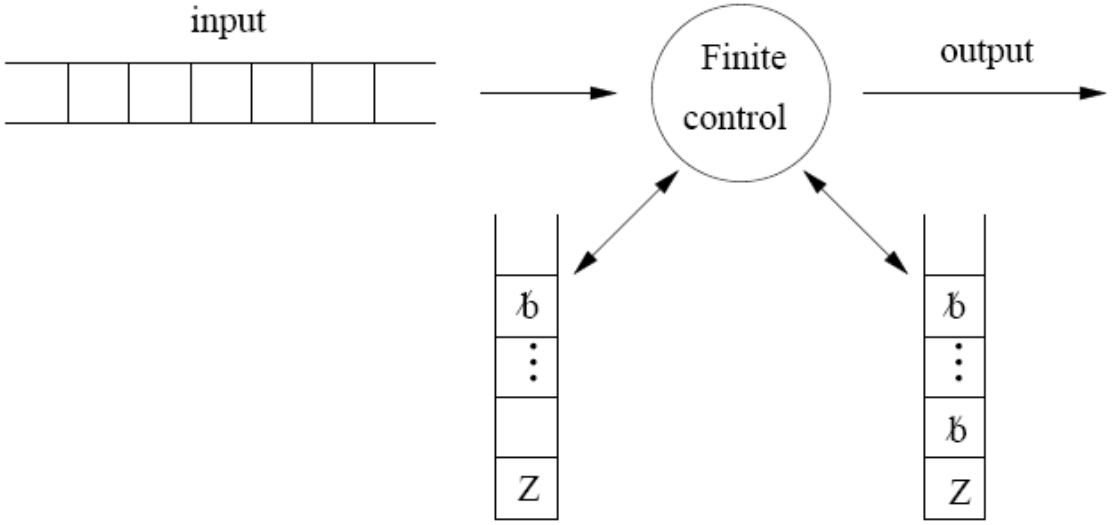
Turing machine can be restricted leading to multi-stack Turing machines, counter machines, etc, without losing out on accepting power.

A deterministic Turing machine with read only input and two storage tape is called a Deterministic Two way Stack Turing Machine (DTSTM). When the head of the DTSTM tries to move left on the tapes, a blank symbol ß will be printed.

There exists a DTSTM that simulates the basic TM on any given input.

One can easily simulate a TM with a DTSTM. At any point of time one can see the symbol being scanned by the head of the TM, placed on the top of one stack, the symbols to its left on this stack below the symbols scanned, placing the symbols closer to the present head position, closer to the top of the stack. Similar exercise is done for the symbols to the right of the present head position by placing them in the second stack. Hence clearly a move of the Turing machine has a corresponding action on the input and stacks of the DTSTM and the simulation can be done.

The next variant is a 'counter' machine. A 'counter' machine can store finite number of integers each counter storing a number. The counters can be either increased or decreased and cannot cross a 'stack' or 'counter' symbol 'Z'. In other words it is a machine with stacks having only two stack symbols Z and $\not b$ (blank). Every stack will have Z as its initial symbol. A stack may hold a string of the form $\not b^i Z$, $i \geq 0$, indicating that the stack holds an integer i in it. This stack can be increased or decreased by moving the stack head up or down. A counter machine with 2-stacks is illustrated in the following figure.



For a basic TM, there exists an equivalent 4-counter Turing machine.

Proof The equivalence is shown between a two-stack TM (DTSTM) and a 4-counter machine. We have already seen that a DTSTM and basic TM are equivalent.

We now see how to simulate each stack with two counters. Let X_1, X_2, $\ldots$, X_{t-1} be the $(t-1)$ stack symbols. Each stack content can be uniquely represented by an integer in base 't'. Suppose $X_{i_1}X_{i_2}\ldots X_{i_n}$ is the present stack content with X_{i_n} on the top in DTSTM. Then the integer count will be

$$k = i_n + ti_{n-1} + t^2 i_{n-2} + \cdots + t^{n-1} i_1.$$

For example if the number of stack symbols used is 3, and an integer for the stack content $X_2X_3X_1X_2$ will be

$$k = 2 + 4 + 4^2.3 + 4^3.2 = 172.$$

Suppose X_r is to be put on the top of the stack, then the new integer counter has to be $kt+r$. The first counter contains k and the second counter contains 0 at this point. To get $kt + r$ in the second counter, the counter machine has to move the first counter head to the left by one cell and move the head of the second counter t cells to the right. Thus when the first counter head reaches 'Z' the second counter contains kt. Now add r to the second counter to get $kt + r$.

If it is a clear move of the stack, X_{i_k} is to be cleared. Then k has to be reduced to $\left[\frac{k}{t}\right]$, the integer part of $\frac{k}{t}$. Now the adjustment is decrementing the count of the first counter in steps and increment the second counter by one. This repeats till the first counter becomes zero.

Now by the above exercise, one is able to identify the stack symbol on the stack from the two counters thus designed. That is k mod t is the index i_n and hence X_{i_n} is the top symbol of the stack. □

For a basic TM, there exists an equivalent 3-counter TM.

Proof The idea of the simulation is similar to the previous theorem. Instead of having two counters for adjusting the two counters that correspond to two stacks, one common counter is used to adjust the operation of pop, push or change of the stack symbols. □

For a basic TM, there exists an equivalent 2-counter TM.

Proof The simulation is now done using the previous theorem. One has to simulate the 3-counters by using 2-counters to get an equivalent result. Let i, j and k be the numbers in the three counters. We have to represent these by a unique integer. Let $m = 2^i 3^j 5^k$ be the integer. Put this in one counter.

To increment say i, one has to multiply m by 2. This can be done using the second counter as we have done earlier. Similarly for j and k increment can be done using the second counter. Any of i, j, k will be zero whenever m is not divisible by 2, 3 or 5 respectively. For example to say whether $j = 0$, copy m to the second counter and while copying store in finite control whether m is divisible by 2, 3 or 5. If it is not divisible by 4, then $j = 0$.

Finally to decrease i, j, k divide m by 2, 3, 5 respectively. This exercise is also similar to the previous one except that the machine will halt whenever m is not divisible by a constant by which we are dividing. □

There exists a TM with one tape and tape alphabet $\{0, 1, \not b\}$ *to recognize any recursively enumerable language* L *over* $\{0, 1\}$.

Proof Let $M = (K, \{0, 1\}, \Gamma, \delta, q_0, F)$ be a TM recognizing L. Now the tape alphabet Γ can be anything. Our aim is to construct an equivalent TM with $\Gamma = \{0, 1, \not b\}$. For that we encode each symbol of Γ. Suppose Γ has 't' symbols. We use binary codes to code each symbol of Γ by 'k' bits where $2^{k-1} < t < 2^k$.

We now have to design another TM M' with $\Gamma' = \{0, 1, \not b\}$. The tape of M' will consists of coded symbols of Γ and the input over $\{0, 1\}$. The simulation of one move of M by k moves of M' is as follows. The tape head of M' is initially at the leftmost symbol of the coded input. M' has to scan the next $k - 1$ symbols to its right to make a decision of change of state or overwrite or move left or right as per M. The TM M' stores in its finite control the state of M and the head position of M' which is a number between 0 to $k-1$. Hence M' clearly indicates at the end of a block of moves whether one move of M is made or not. Once the finite control indicates '0' as head position, it means this is time for the change on the tape, state as per M's instruction. If the thus changed state is an accepting state of M, M' accepts.

One observation is that on the tape of M', there has to be code for the blank symbol of M. This is essential for simulating the blank of M by M'. Second observation is that any tape symbol of M is directly coded in terms of 0s and 1s and as any string w is placed on the tape of M, the codes for each symbol of w is concatenated and placed on the tape of M'. □

Any Turing machine can be simulated by an offline TM having one storage tape with two symbols 0 *and* 1 *where* 0 *indicates blank. A blank* (0) *can be retained as* 0 *or replaced by* 1 *but a '*0*' cannot be rewritten at a cell with '*1*'.*

One can see from the previous result that even if the input for L is not over $\{0, 1\}$, the above TM construction will work, because the input of L over some other alphabet will be now coded and placed as input for a TM with tape alphabet $\{0, 1, \not b\}$.

Also one can see that one can construct a multi-tape TM that uses only two symbols $0, 1$ as tape alphabet to simulate any TM. One has to keep the input tape fixed with the input. There will be a second tape with tape symbols coded as binary symbols. This simulates moves of the original TM. The newly constructed TM must have positions on the tape to indicate the present head position, cells to indicate that the binary representation of the symbol under scan is already copied. Each ID is copied on the third tape after simulation of one move. If we take the input alphabet $\{0, 1\}$ we start with the second tape (first tape is not necessary). In the third tape IDs are copied one by one without erasing.

Turing Machines as Enumerators

The languages accepted by the Turing machines are recursively enumerable sets. One can think of a Turing machine as generating a language. An enumerator is a Turing machine variant which generates a recursively enumerable language. One can think of such a Turing machine to have its output on a printer (an output tape). That is the output strings are printed by the output device printer. Thus every string that is processed freshly, is added to the list, thereby printing it also.

The enumerator machine has an input tape, which is blank initially, an output device which may be a printer. If such a machine does not halt, it may perform printing of a list of strings infinitely. Let M be an enumerator machine and $G(M)$ be the list of strings appearing on output tape. We have the following theorem for M.

Theorem 10.2.7 *A language L is recursively enumerable if and only if there exists an enumerator M such that $G(M) = L$.*

Proof Let M be an enumerator such that $L = G(M)$. To show that there exists a Turing machine $\overline{M}$ recognizing L. Let w be an input for $\overline{M}$. Perform the following two steps on w.

1. Run M and compare each output string of M with w.

2. If w appears on the output tape of M, accept w.

That is $\overline{M}$ accepts only those string that appear on the output tape of M. Hence $T(\overline{M}) = G(M)$.

Conversely let $\overline{M}$ be a Turing machine such that $L = T(\overline{M})$. We construct an enumerator M that prints every string of L as follows.

Let Σ be the alphabet of L and $w_1, w_2, w_3, w_4, \ldots$ be all possible strings over Σ.

The enumerator machine M will do the following for any input from Σ^*.

3. If $\overline{M}$ accepts any string w_i print the corresponding string w_i.

Clearly if any string w is accepted by $\overline{M}$, it will be output by the enumerator M. In the above procedure one can see that there will be repeated printing of a string w. It is straightforward to see $G(M) = L = T(M)$.

If L is a recursive set (a set accepted by a TM which halts on all inputs), then there exists an enumerator M for L which will print the strings in L in canonical order.

Equivalence Between Turing Machines and Type 0 Languages

If L is the language generated by an unrestricted grammar $G = (N, T, P, S)$, then L is recognised by a TM.

Proof For G, we construct a Turing machine M with two tapes such that on one tape we put the input w and the other tape is used to derive w using P. Each time a rule from P is applied, compare the two tapes for acceptance or rejection of w. Initially put w on one tape. Then M initially places S on the second tape. Nondeterministically select a rule $S \rightarrow \alpha$ from P, replace S by α on the second tape. Now compare the tapes, if they agree accept w. Otherwise, from the present string α, choose a location 'i' nondeterministically such that β is a subword occurring in α from position i. Choose a rule $\beta \rightarrow \gamma$ again nondeterministically. Apply to α, by inserting γ at the position of β. Now let the present tape content be α_1. If $\alpha_1 = w$, then accept w, otherwise continue the procedure. □

If L is accepted by a TM M, then there exists an unrestricted grammar generating L.

Proof Let L be accepted by a TM $M = (K, \Sigma, \Gamma, q_0, \delta, F)$. Then G is constructed as follows. Let $G = (N, \Sigma, P, S_1)$ where $N = ((\Sigma \cup \{\epsilon\} \times \Gamma) \cup \{S_1, S_2, S_3\})$. P consists of the following rules:

1. $S_1 \rightarrow q_0 S_2$

2. $S_2 \rightarrow (a, a)S_2$ for each $a \in T$.

That is G produces every time two copies of symbols from Σ.

3. $S_2 \rightarrow S_3$

4. $S_3 \rightarrow (\epsilon, \epsilon)S_3$

5. $S_3 \rightarrow \epsilon$

6. $q(a, X) \rightarrow (a, Y)p$

if $\delta(q, X) = (p, Y, R)$ for every a in $\Sigma \cup \{\epsilon\}$, each $q \in Q$, $X, Y \in \Gamma$. This rule simulates the action of M on the second component of the symbols (α, β).

7. $(b, Z)q(a, X) \rightarrow p(b, Z)(a, Y)$

if $\delta(q, X) = (p, Y, L)$ for each $a, b \in \Sigma \cup \{\epsilon\}$, each $q \in Q$, $X, Y, Z \in \Gamma$. This rule does the same job as rule 6.

8. $[a, X]q \rightarrow qaq$, $q[a, X] \rightarrow qaq$ and $q \rightarrow \epsilon$ for each $a \in \Sigma \cup \{\epsilon\}$, $X \in \Gamma$ and $q \in F$.

This rule brings out w from the pair if the second component of the input pair is properly accepted by M.

Hence we see from the rules that the constructed grammar "nondeterministically" generates two copies of w in Σ^* using rules (1) and (2) and simulates M through the rules 6, 7. Rule 8 brings out w if it is accepted by M. The equivalence that if $w \in L(G)$ then $w \in L(M)$ and conversely can be proved by induction on the number of derivation steps and on the number of moves of the TM. Hence the theorem. □

Linear Bounded Automata

A Linear Bounded Automata (LBA) is a nondeterministic Turing machine with a bounded finite input tape. That is input is placed between two special symbols ¢ and \$.

¢	a_1	a_2	$\cdots$	a_n	\$

But all the other actions of a TM are allowed except that the read / write head cannot fall off on left of ¢ and right of \$. Also ¢ and \$ are not altered. One can say that this is a restricted version of TM.

Definition 10.4.1 *A LBA is a 8-tuple* $M = (K, \Sigma, \Gamma, \delta, q_0, ¢, \$, F)$ *where* $K, \Sigma, \Gamma, q_0, F$ *and* δ *are as in any TM. The language recognized by* M *is* $L(M) = \{w / w \in \Sigma^* \text{ and } q_0 ¢ w \$ \overset{*}{\vdash} \alpha p \beta \text{ for some } p \in F\}$.

One can show that the family of languages accepted by a LBA is exactly CSL.

Theorem 10.4.1 *If L is a CSL, then L is accepted by a LBA.*

Proof For L, one can construct a LBA with 2-track tape. The simulation is done as in Theorem 10.3.2 where we place w on the first track and produce sentential forms on the second track, every time comparing with contents on the first track. If $w = \epsilon$, the LBA halts without accepting. □

If L is recognized by a LBA, then L is generated by a context-sensitive grammar.

Proof Let $M = (K, \Sigma, \Gamma, q_0, \delta, \not\in, \$, F)$ be a LBA such that $L(M) = L$. Then one can construct a CSG, $G = (N, \Sigma, P, S_1)$ as below.
N consists of nonterminals of the form (a, β) where $a \in \Sigma$ and β is of the form x or qx or $q\not\in x$ or $x\$$ or $qx\$$ where $q \in K, x \in \Gamma$.
P consists of the following productions:

1. $S_1 \rightarrow (a, q_0 \not\in a)S_2$
2. $S_1 \rightarrow (a, q_0 \not\in a\$)$
3. $S_2 \rightarrow (a, a)S_2$
4. $S_2 \rightarrow (a, a\$)$ for all $a \in \Sigma$.
   The above four rules generate a sequence of pairs whose first components form a terminal string $a_1 a_2 \ldots a_t$ and the second components form the LBA initial ID.
   The moves of the LBA are simulated by the following rules in the second component.
5. If $\delta(q, X) = (p, Y, R)$ we have rules of the form $(a, qX)(b, Z) \rightarrow (a, Y)(b, pZ)$
   $(a, q\not\in X)(b, Z) \rightarrow (a, \not\in Y)(b, pZ)$ where $a, b \in \Sigma$, $p, q \in K$, $X, Y, Z \in \Gamma$.

6. If $\delta(q, X) = (p, Y, L)$ we have rules of the form $(b, Z)(a, qX) \rightarrow (b, pZ)(a, Y)$
   $(b, Z)(a, qX\$) \rightarrow (b, pZ)(a, Y\$)$ where $a, b \in \Sigma$, $p, q \in K$, $X, Y, Z \in \Gamma$.

7. $(a, q\beta) \rightarrow a$ if q is final, for all $a \in \Sigma$.

8. $(a, \alpha)b \rightarrow ab$
   $b(a, \alpha) \rightarrow ba$ for any $a \in \Sigma$ and all possible α.

Clearly all the productions are context-sensitive. The simulation leads to a stage where the first components emerge as the string generated if the second components representing LBA ID has a final state.

ϵ will not be generated by the grammar whether or not it is in $T(M)$. $\square$

We have already seen that $\epsilon \notin L$, if L is context-sensitive by definition. To include ϵ we must have a new start symbol S' and include $S' \rightarrow \epsilon$, making sure S' does not appear on the right-hand side of any production by adding $S' \rightarrow \alpha$ where $S \rightarrow \alpha$ is a rule in the original CSG with S as the start symbol.