Theory of Automata
Turing Machine

Week-15-Lecture-01

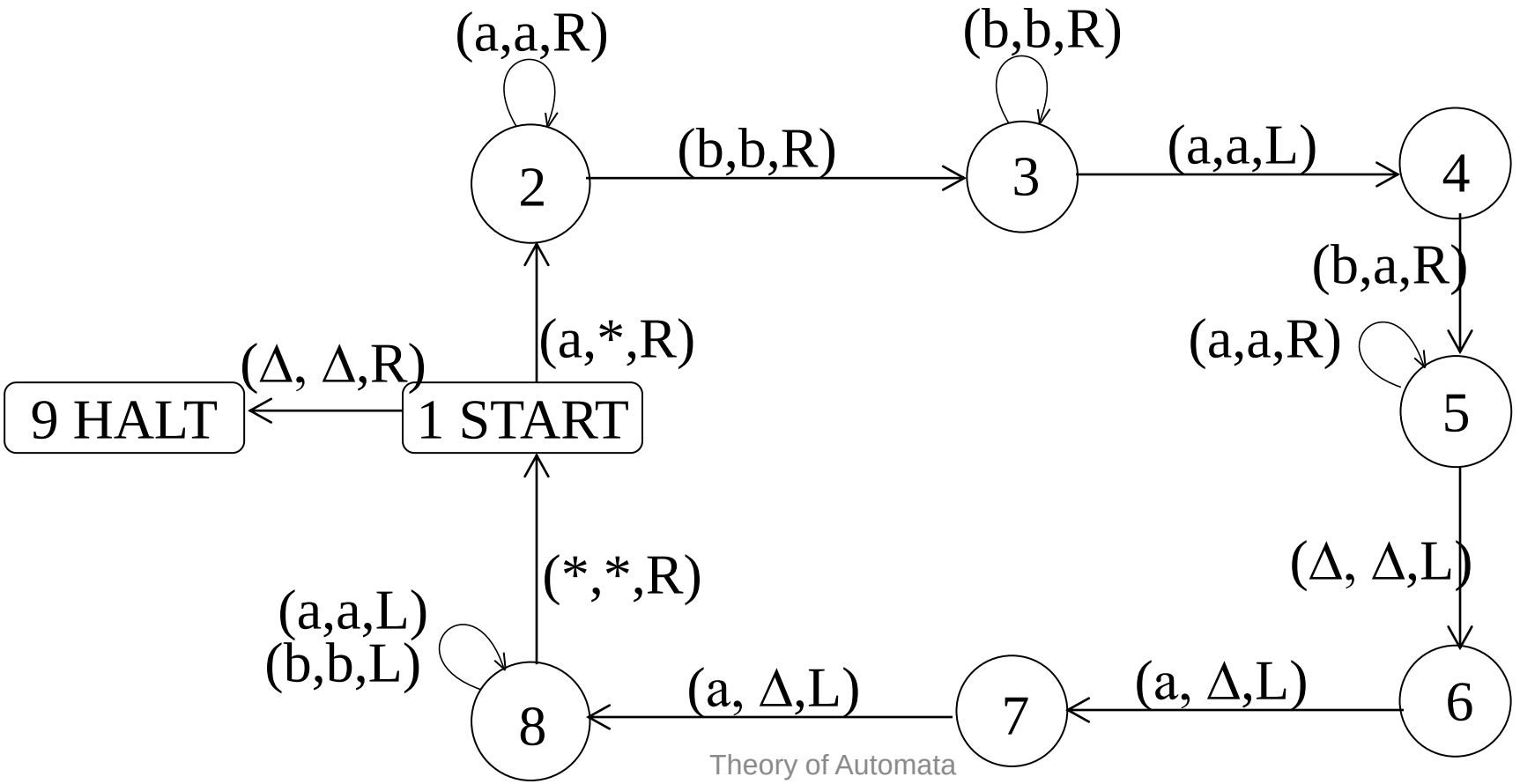
Hafiz Tayyeb Javed

Contents

- Definition
- Example
- Subprogram INSERT
- EQUAL
- Subprogram DELETE

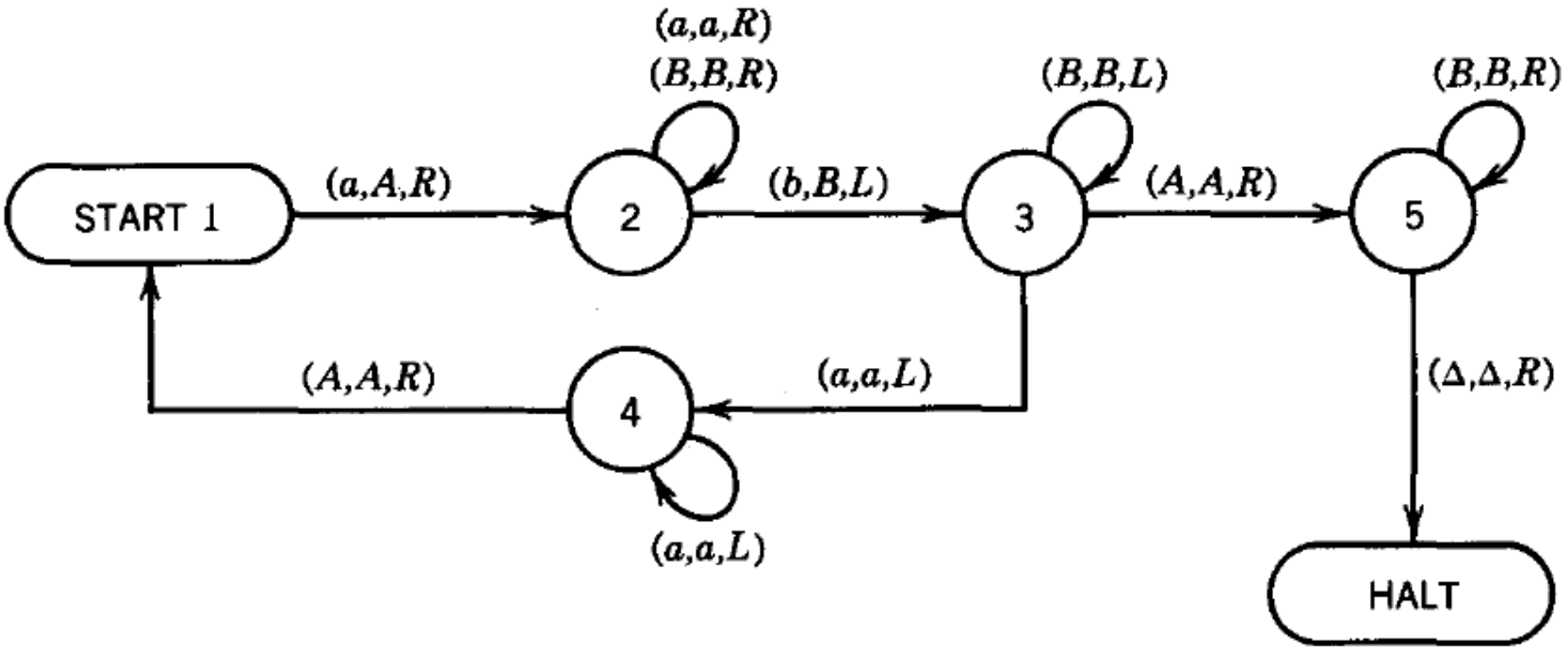
Example

Consider the following TM



Language Defined by	Corresponding Acceptor	Nondeterminism = determinism?	Language Closed Under	What Can be Decided	Example of Application
Regular expression	Finite automaton Transition graph	Yes	Union, product, Kleene star, intersection, complement	Equivalence, emptiness, finiteness, membership	Text editors, sequential circuits
Context-free grammar	Pushdown automaton	No	Union, product, Kleene star	Emptiness finiteness membership	Programming language statements, compilers
Type 0 grammar	Turing machine, Post machine, 2PDA, *n*PDA	Yes	Union, product, Kleene star	Not much	Computers

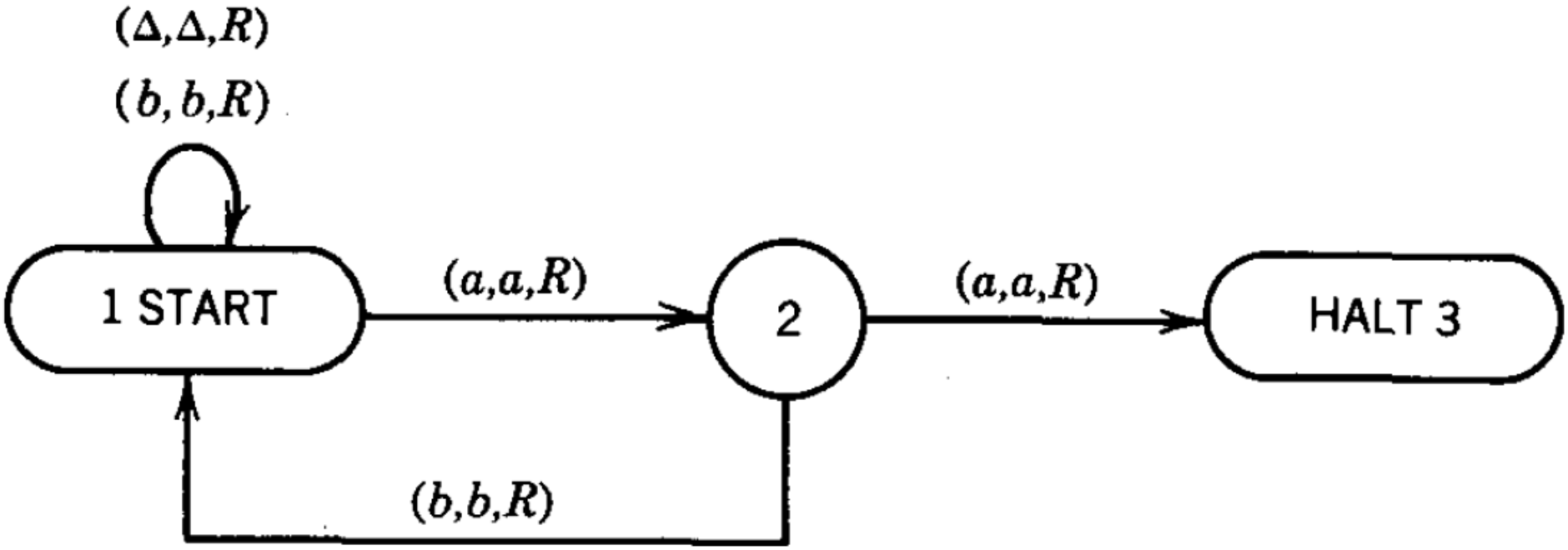
$a^n b^n$



Example continued …

The string aaabbbaaa can be observed to be accepted by the above TM. It can also be observed that the above TM accepts the non-CFL $\{a^n b^n a^n\}$.

Must have double ‘aa’



1. Those with a double a. They are accepted by the TM.
2. Those without aa that end in a. They crash.
3. Those without aa that end in b. They loop forever.

DEFINITION

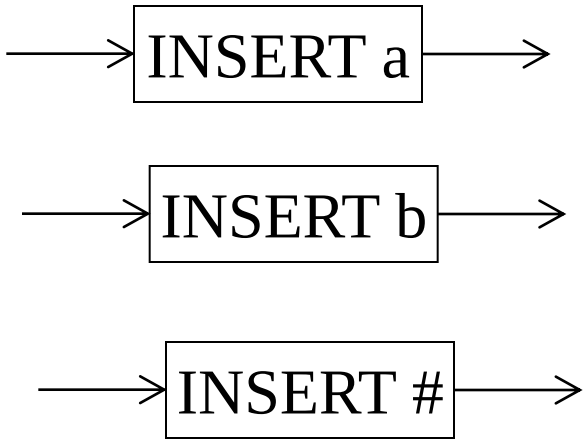
Every Turing machine T over the alphabet **divides the set of strings** into three classes:

1. ACCEPT(T) is the set of all strings leading to a HALT state. This is also called the *language accepted by T.*
2. REJECT(T) is the set of all strings that crash during execution by moving left from cell i or by being in a state that has no exit edge that wants to read the character the **TAPE HEAD is reading.**
3. LOOP(T) is the set of all other strings, that is, strings that loop forever while running on *T.*

INSERT subprogram

Sometimes, a character is required to be inserted on the TAPE exactly at the spot where the TAPE Head is pointing, so that the character occupies the required cell and the other characters on the TAPE are moved one cell right. The characters to the left of the pointed cell are also required to remain as such.

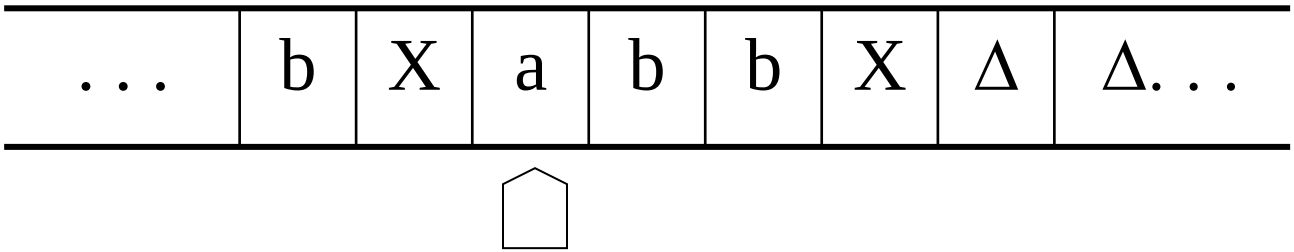
In the situation stated above, the part of TM program that executes the process of insertion does not affect the function that the TM is performing. The subprogram of insertion is independent and can be incorporated at any time with any TM program specifying what character to be inserted at what location. The subprogram of insertion can be expressed as



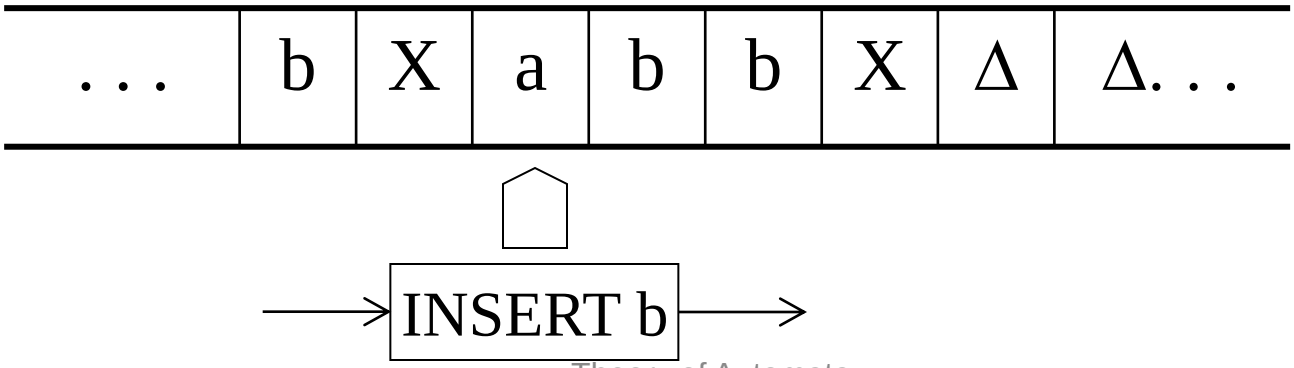
The above diagrams show that the characters a,b and # are to be inserted, respectively. Following is an example showing how does the subprogram INSERT perform its function

Example

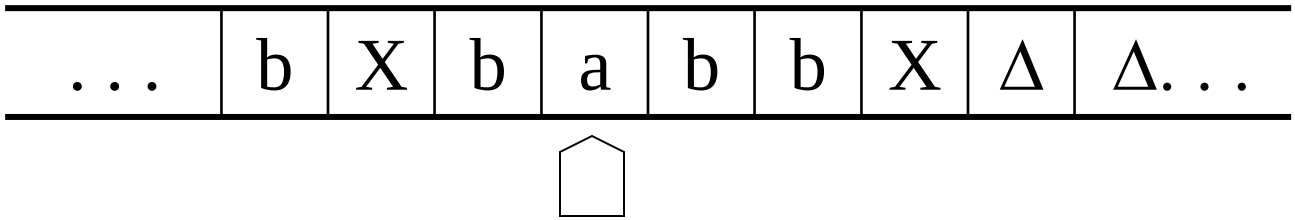
If the letter b is inserted at the cell where the TAPE Head is pointing as shown below



then, it is expressed as



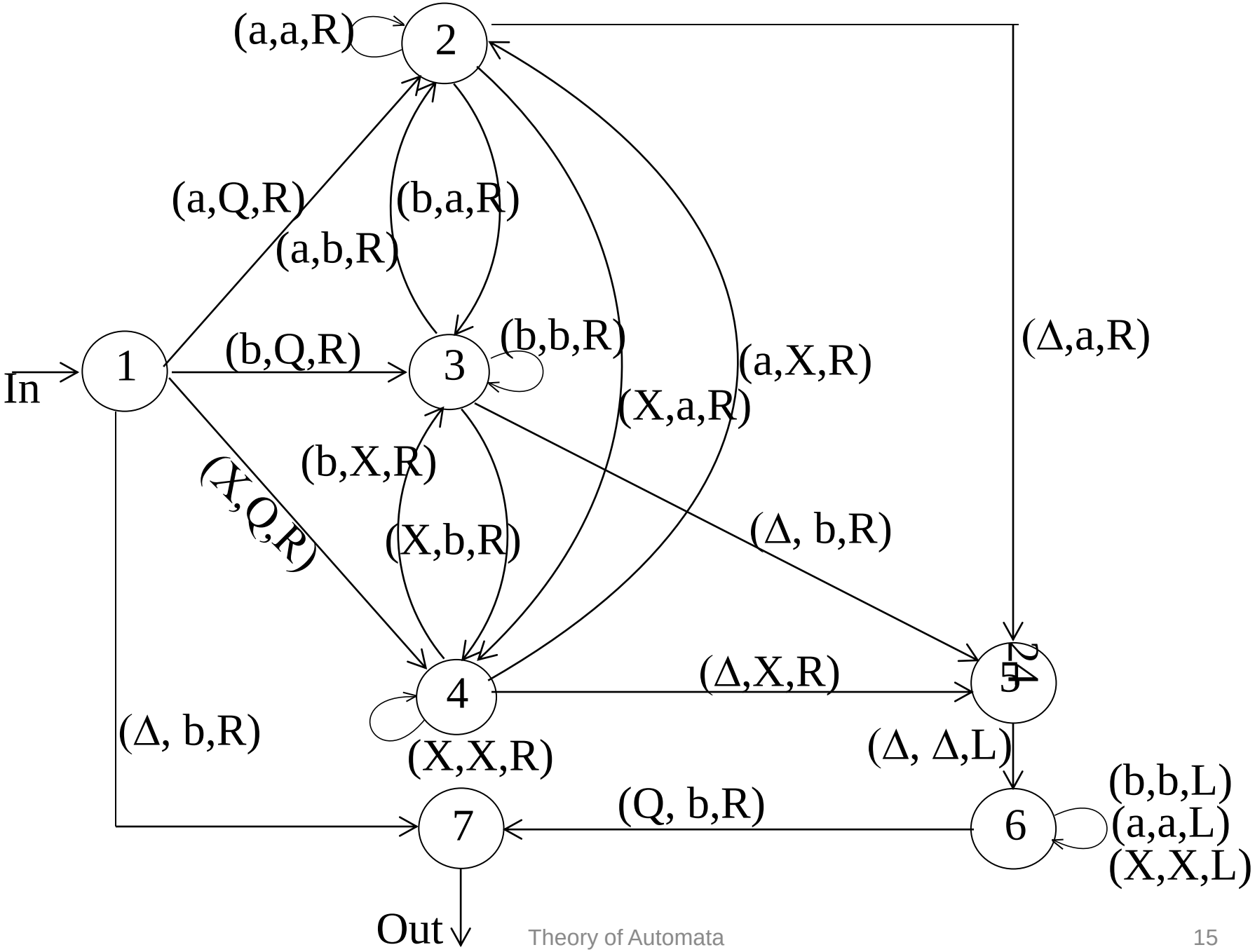
The function of subprogram INSERT b can be observed from the following diagram



Following is the INSERT subprogram

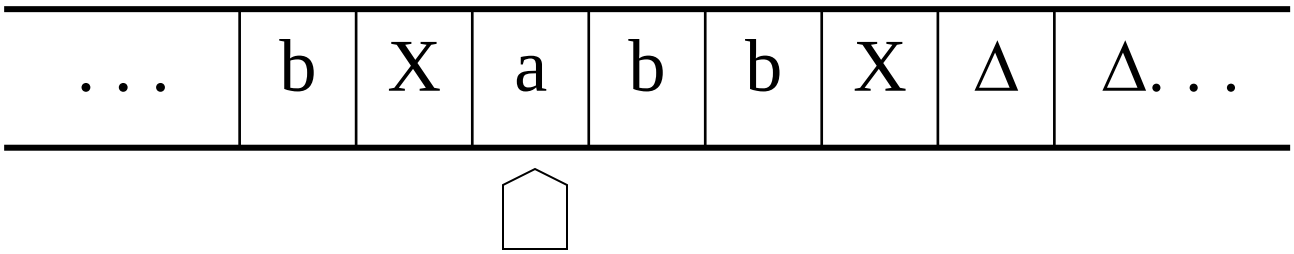
The subprogram INSERT

Keeping in view the same example of inserting b at specified location, to determine the required subprogram, first Q will be inserted as marker at the required location, so that the TAPE Head must be able to locate the proper cell to the right of the insertion cell. The whole subprogram INSERT is given as


(a,a,R)
2
(a,Q,R)
(b,a,R)
(a,b,R)
(b,Q,R)
(b,b,R)
(a,X,R)
(Δ,a,R)
In
1
3
(X,a,R)
(b,X,R)
(X,Q,R)
(Δ, b,R)
(X,b,R)
(Δ,X,R)
4
5
(Δ, b,R)
(X,X,R)
(Δ, Δ,L)
(b,b,L)
(Q, b,R)
7
6
(a,a,L)
(X,X,L)
Out

It is supposed that machine is at state 1, when b is to be inserted. All three possibilities of reading a, b or X are considered by introducing the states 2, 3 and 4 respectively. These states remember what letter displaced during the insertion of Q.

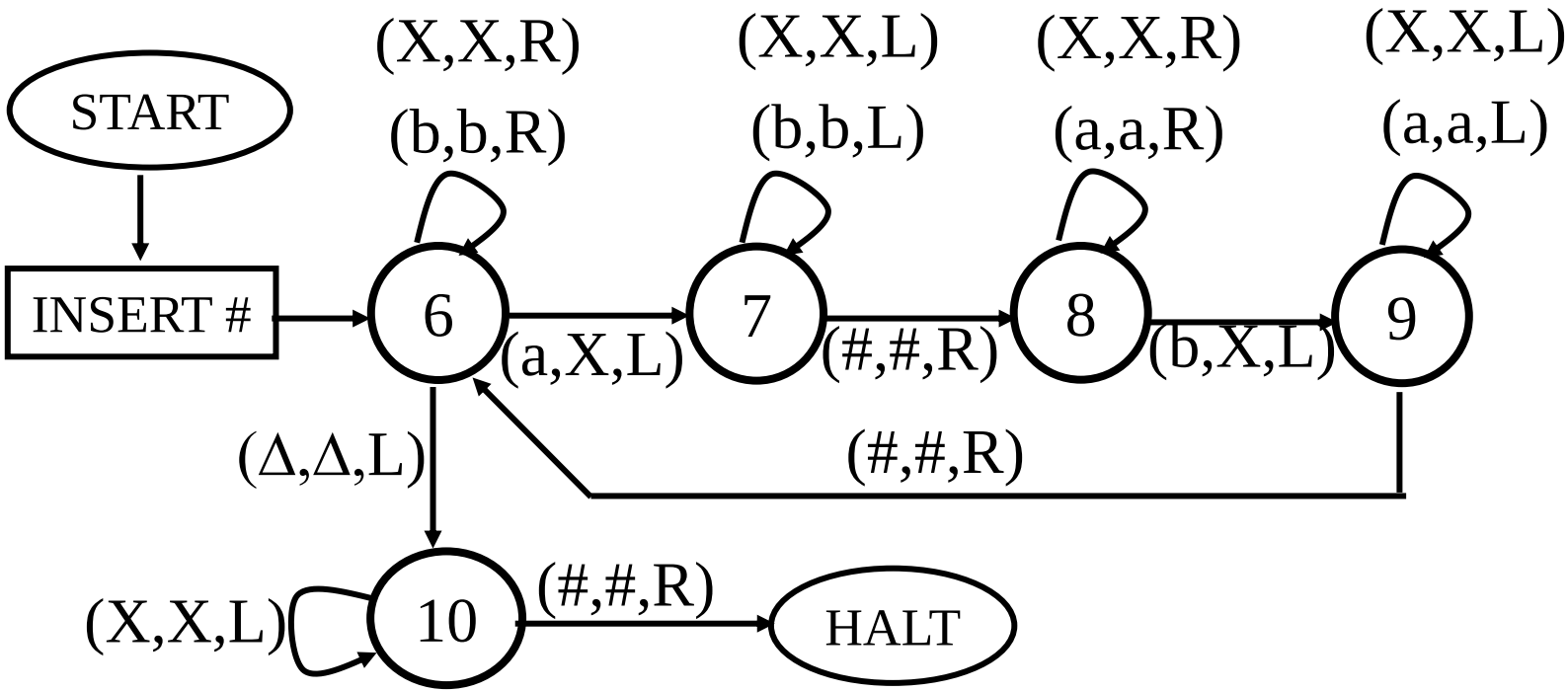
Consider the same location where b is to be inserted



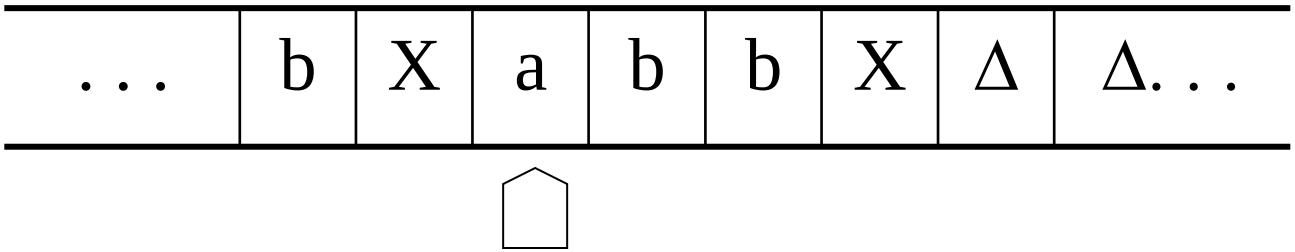
After reading a from the TAPE, the program replaces a by Q and the TAPE Head will be moved one step right. Here the state 2 is entered. Reading b at state 2, b will be replaced by a and state 3 will be entered. At state 3 b is read which is not replaced by any character and the state 3 will not be left.

At state 3, the next letter to be read is X, which will be replaced by b and the state 4 will be entered. At state 4, Δ will be read, which will be replaced by X and state 5 will be entered. At state 5 Δ will be read and without any change state 6 will be entered, while TAPE Head will be moved one step left. The state 6 makes no change whatever (except Q) is read at that state. However at each step, the TAPE Head is moved one step left. Finally, Q is read which is replaced by b and the TAPE Head is moved to one step right.

EQUAL


START
INSERT #
6
7
8
9
10
HALT
(X,X,R)
(b,b,R)
(X,X,L)
(b,b,L)
(X,X,R)
(a,a,R)
(X,X,L)
(a,a,L)
(a,X,L)
(#,#,R)
(b,X,L)
(Δ,Δ,L)
(#,#,R)
(X,X,L)
(#,#,R)

Hence, the required situation of the TAPE can be shown as



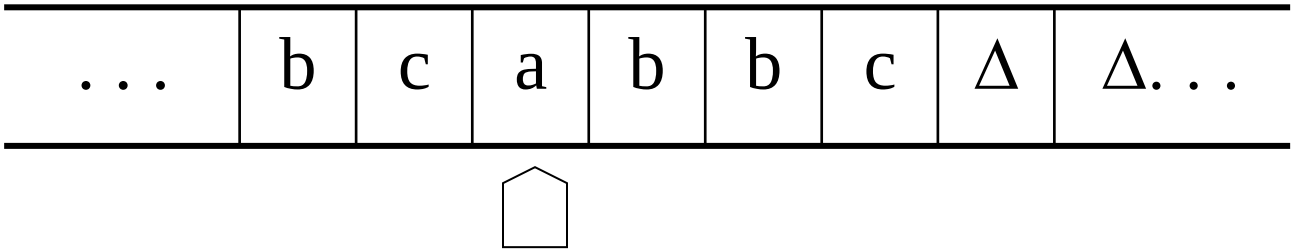
DELETE subprogram

Sometimes, a character is required to be DELETED on the TAPE exactly at the spot where the TAPE Head is pointing, so that the other characters on the right of the TAPE Head are moved one cell left. The characters to the left of the pointed cell are also required to remain as such.

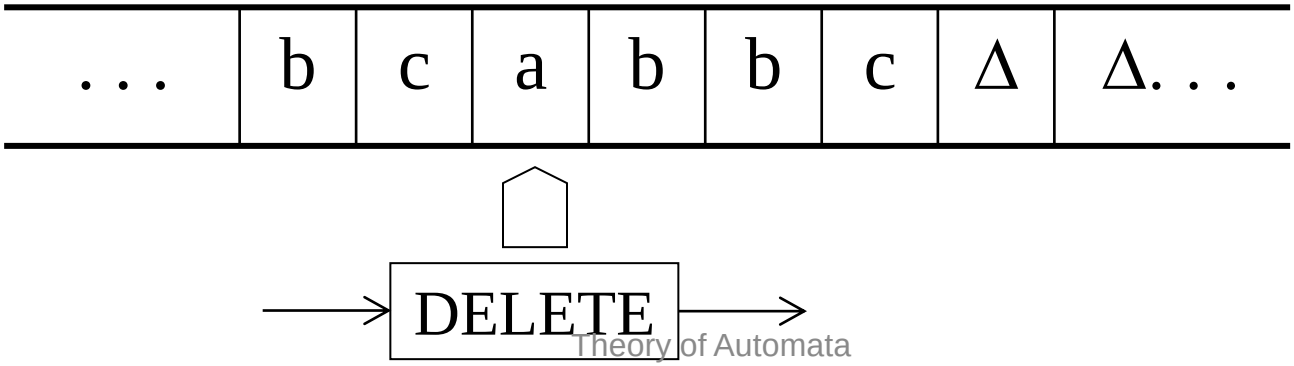
In the situation stated above, the part of TM program that executes the process of deletion does not affect the function that the TM is performing. The subprogram of deletion is independent and can be incorporated at any time with any TM program specifying what character to be deleted at what location. The subprogram of deletion can be expressed as

Example

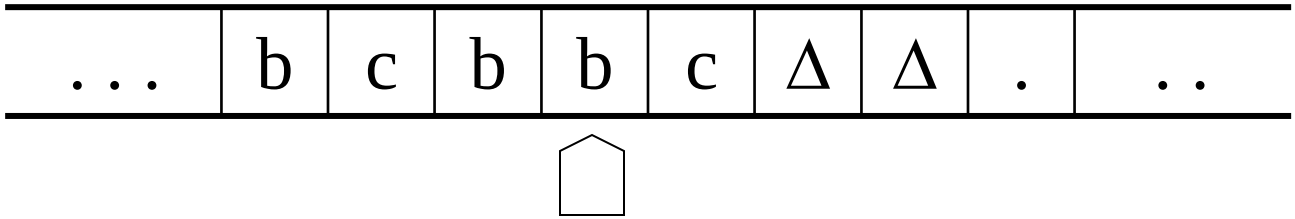
If the letter a is to be deleted from the string bcabbc, shown below



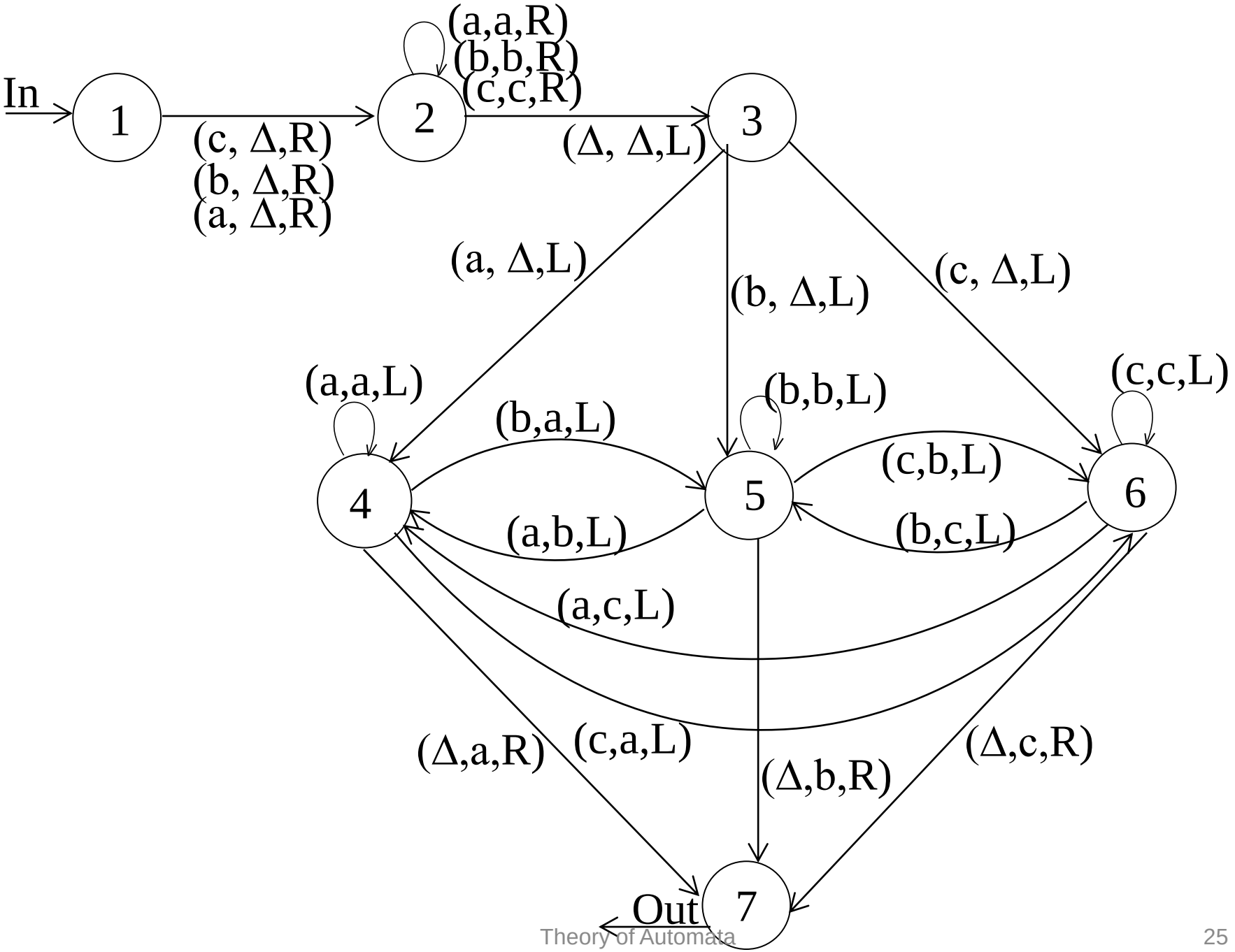
then, it is expressed as



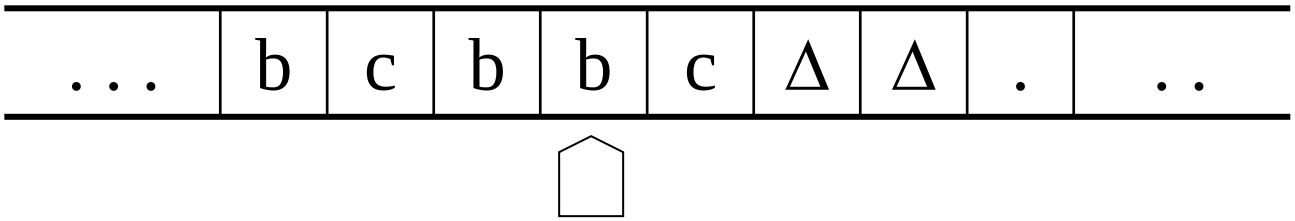
The function of subprogram DELETE can be observed from the following diagram



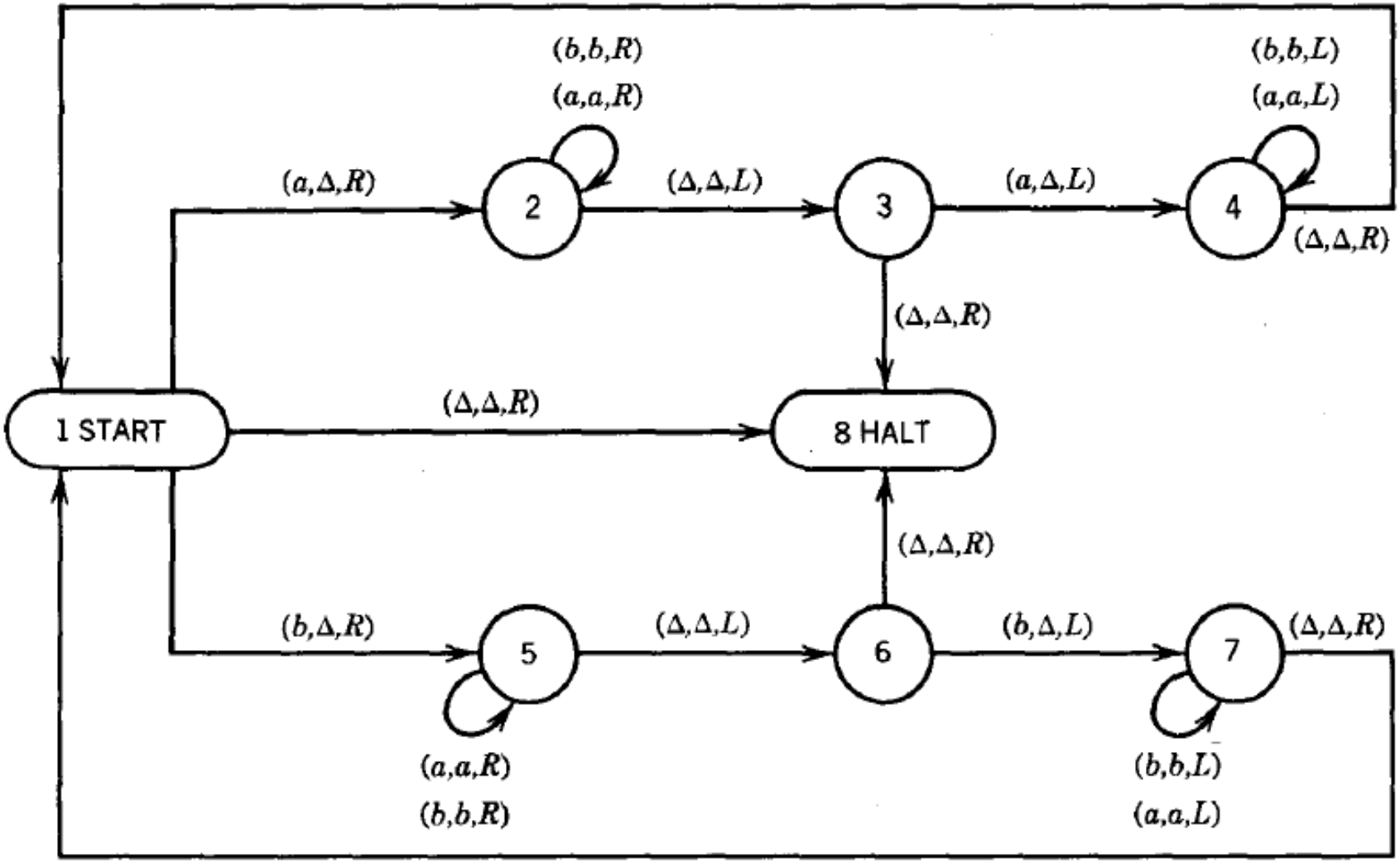
Following is the DELETE subprogram


In
1
2
3
4
5
6
7
Out
(a,a,R)
(b,b,R)
(c,c,R)
(c, Δ,R)
(b, Δ,R)
(a, Δ,R)
(Δ, Δ,L)
(a, Δ,L)
(b, Δ,L)
(c, Δ,L)
(a,a,L)
(b,a,L)
(b,b,L)
(c,c,L)
(c,b,L)
(a,b,L)
(b,c,L)
(a,c,L)
(Δ,a,R)
(c,a,L)
(Δ,b,R)
(Δ,c,R)

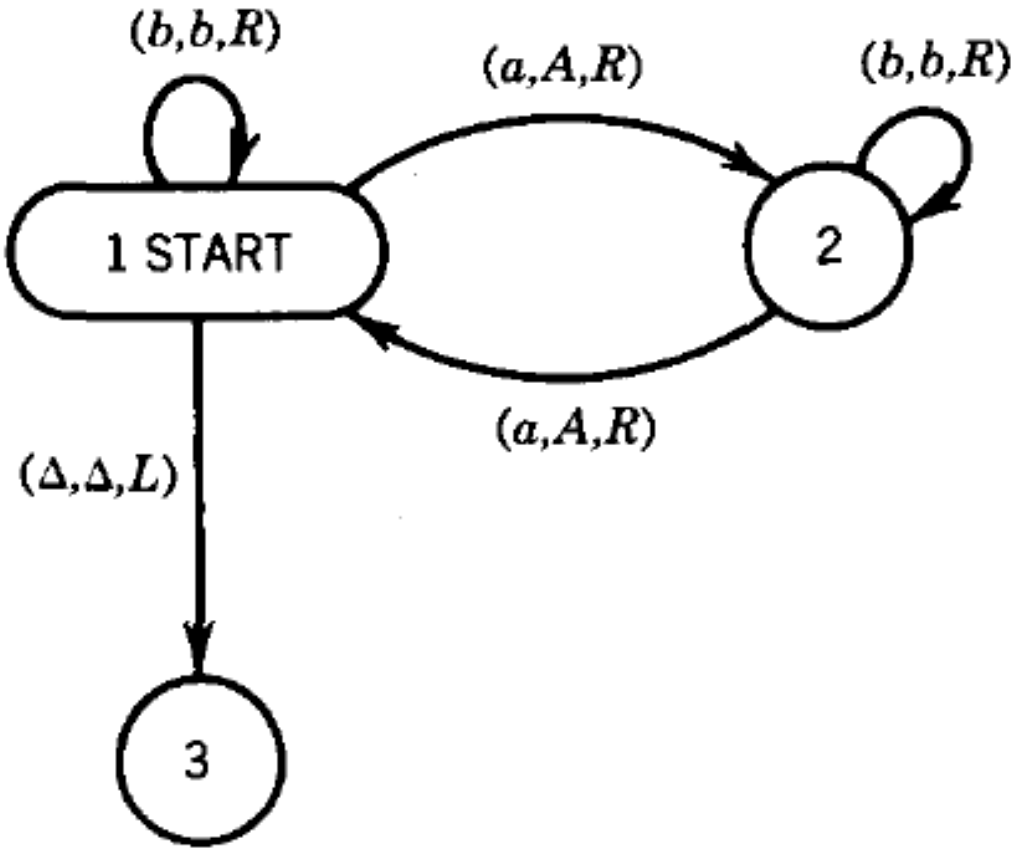
The process of deletion of letter a from the string bcabbc can easily be checked, giving the TAPE situation as shown below



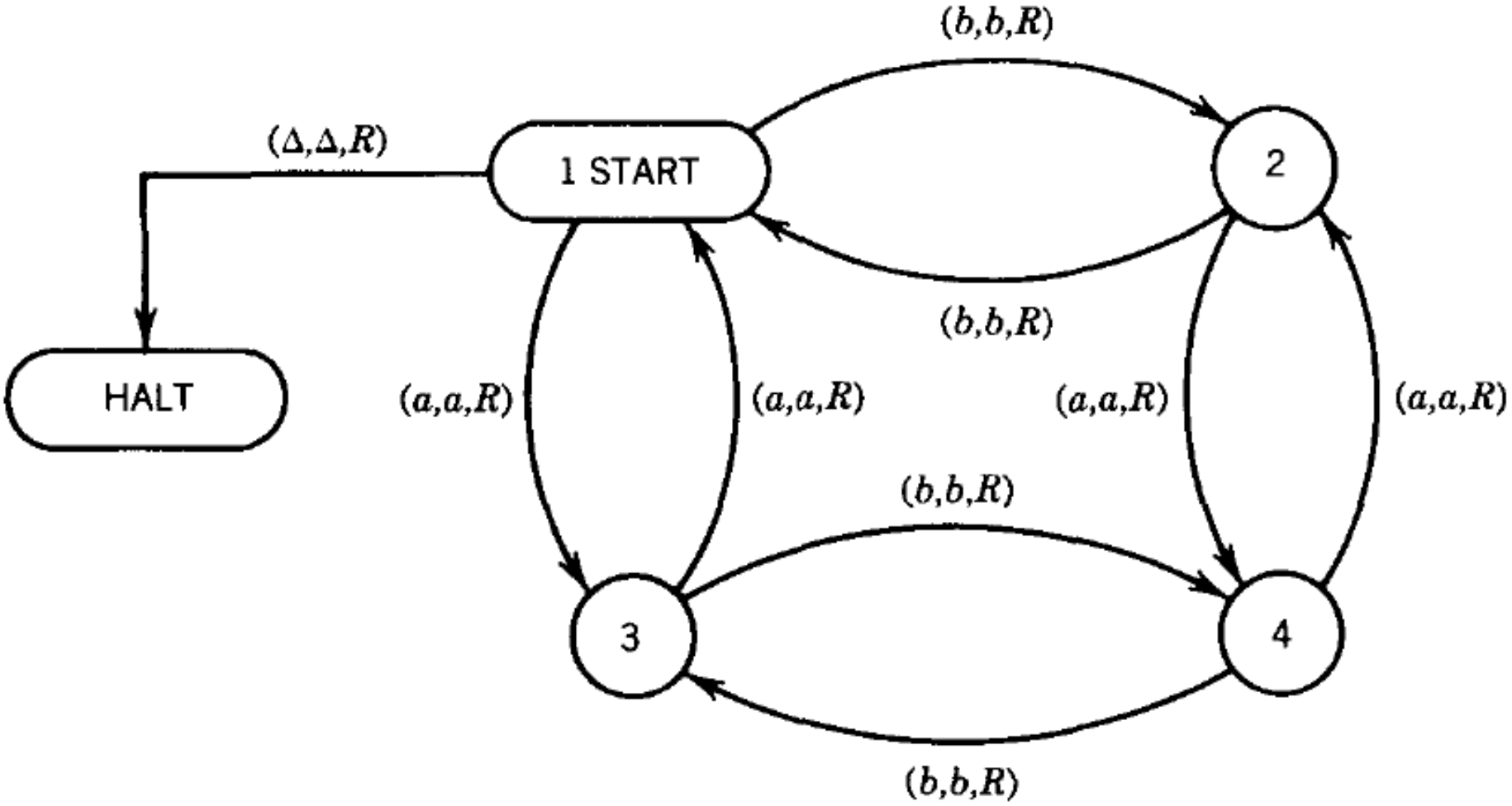
OddPalindrome / EvenPalindrome



Even a's



Even-Even



Double Word Problem

The language this TM accepts is DOUBLEWORD, the set of all words of the form ww where w is a nonnull string in *{a, b}*.*
DOUBLEWORD **= {aa *bb aaaa abab baba bbbb*** *aaaaaa aabaab . . }*

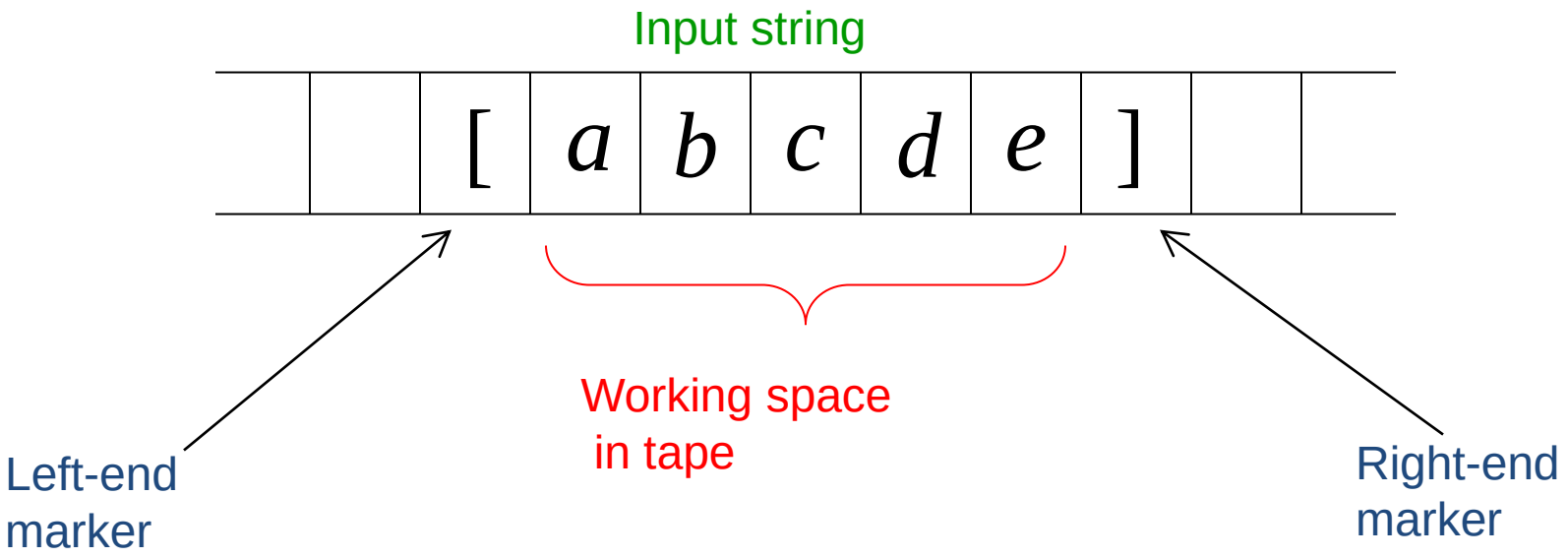
Chomsky Hierarchy

- **Type-0** grammars (unrestricted grammars) include all formal grammars.
- **Type-1** grammars (context-sensitive grammars) generate the context-sensitive languages.
- **Type-2** grammars (context-free grammars) generate the context-free languages.
  - Context free languages are the theoretical basis for the syntax of most programming languages.
- **Type-3** grammars (regular grammars) generate the regular languages.
  - These languages are exactly all languages that can be decided by a finite state automaton. Additionally, this family of formal languages can be obtained by regular expressions.

Linear-Bounded Automata:

Same as Turing Machines with one difference:

the input string tape space is the only tape space allowed to use



All computation is done between end markers

We define LBA's as NonDeterministic

Open Problem:

Non-Deterministic LBA's have same power as Deterministic LBA's ?

Example languages accepted by LBAs:

$$L = \{a^n b^n c^n\} \qquad L = \{a^{n!}\}$$

LBA's have more power than PDA's

LBA's have less power than Turing Machines

Unrestricted Grammars:

Productions

$$u \rightarrow v$$

String of variables and terminals

String of variables and terminals

Type-0 grammar (Unrestricted Grammar)

- They generate exactly all languages that can be recognized by a Turing machine.
- These languages are also known as the recursively enumerable languages.
- This is different from the recursive languages which can be *decided* by an always halting Turing machine.

Example unrestricted grammar: $S \to aBc$

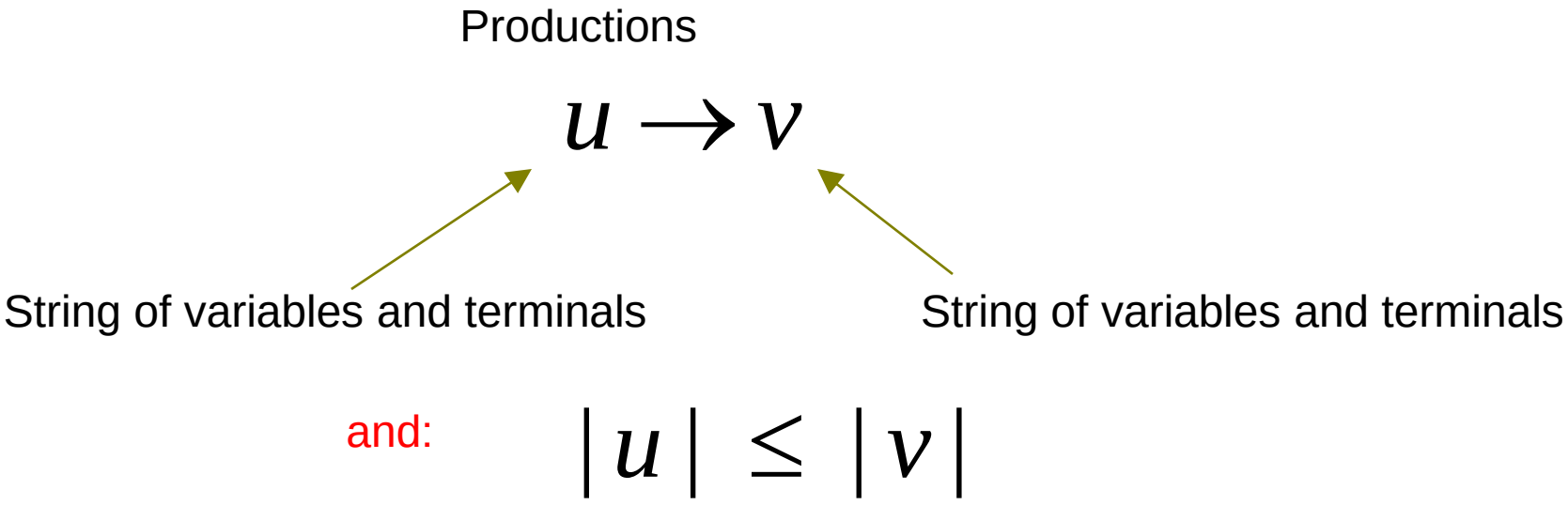
$$aB \to cA$$

$$Ac \to d$$

Theorem:

A language L is Turing-Acceptable if and only if L is generated by an unrestricted grammar

Context-Sensitive Grammars:



- Type-1 grammar
  - The rule is allowed if *S* does not appear on the right side of any rule.
  - The languages described by these grammars are exactly all languages that can be recognized by a non-deterministic Turing machine whose tape is bounded by a constant times the length of the input.

The language $\{a^n b^n c^n\}$

is context-sensitive:

$$S \to abc \mid aAbc$$

$$Ab \to bA$$

$$Ac \to Bbcc$$

$$bB \to Bb$$

$$aB \to aa \mid aaA$$

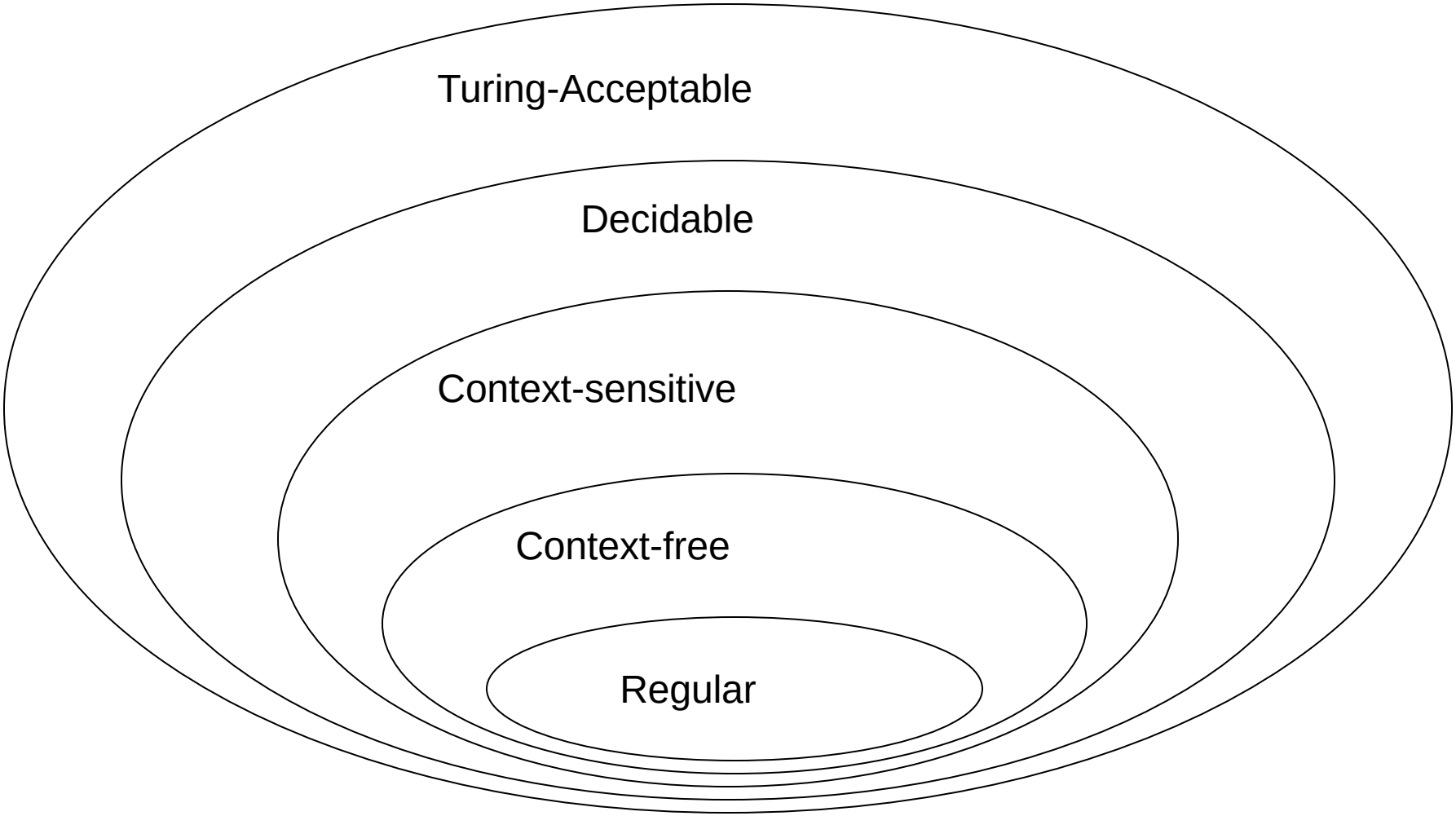
Theorem:

A language L is context sensitive iff it is accepted by a Linear-Bounded automaton

Observation:

There is a language which is context-sensitive but not decidable

The Chomsky Hierarchy



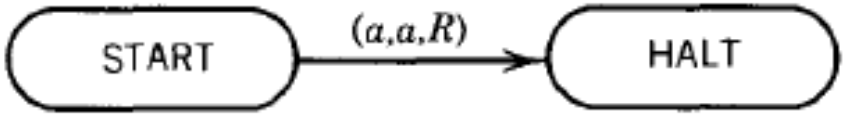
Summary

Automata theory: formal languages and formal grammars			
Chomsky hierarchy	**Grammars**	**Languages**	**Minimal automaton**
Type-0	(unrestricted)	Recursively enumerable	Turing machine
	(unrestricted)	Recursive	Decider
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown
Type-3	Regular	Regular	Finite
Each category of languages or grammars is a proper superset of the category directly beneath it.			

- The Halting Problem for Turing machines is *unsolvable, which means that* there does not exist any such algorithm

Recursive Language

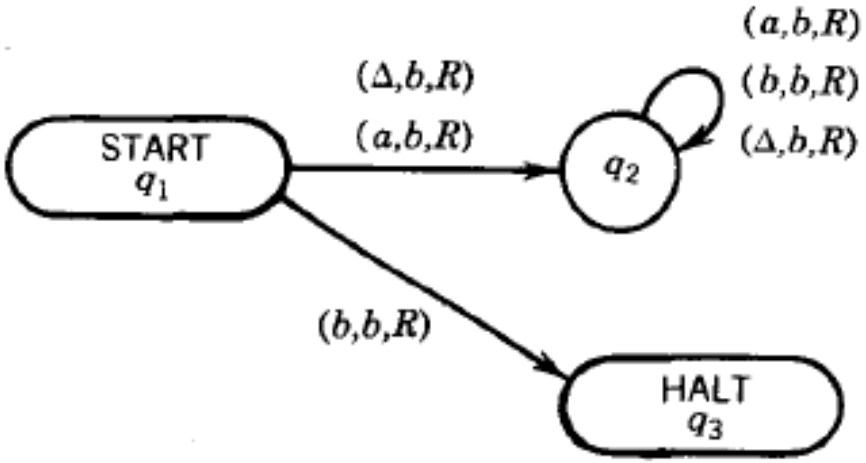
The following TM accepts the language of all words over $\{a,b\}$ that start with a and crashes on (rejects) all words that do not.



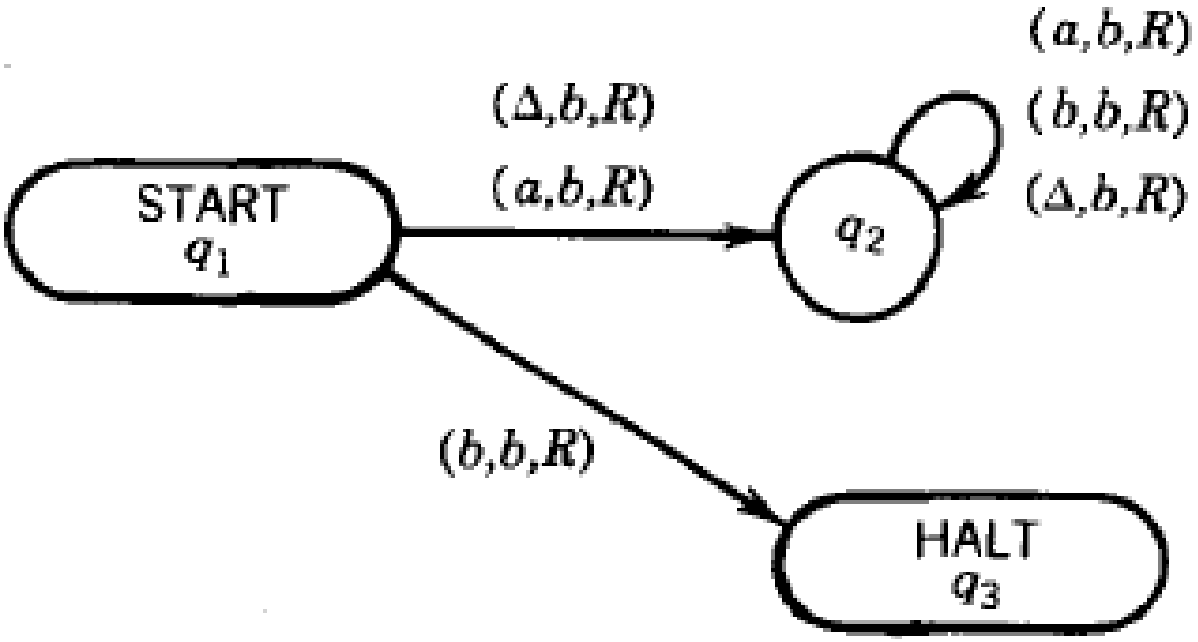
- Accept(TM) = Accept(2PDA) = Accept(PM)
- Reject(TM) = Reject(2PDA) = Reject(PM)
- Loop(TM) = Loop(2PDA) = Loop (PM)

L={all words starting with b}

- This silly machine tries to turn the whole tape into b if the input string does not belong to the L`.



$$\text{accept}(T_1) = L$$
$$\text{loop}(T_1) = L'$$
$$\text{reject}(T_1) = \phi$$



$$\text{accept}(T_1) = L$$

$$\text{loop}(T_1) = L'$$

$$\text{reject}(T_1) = \phi$$