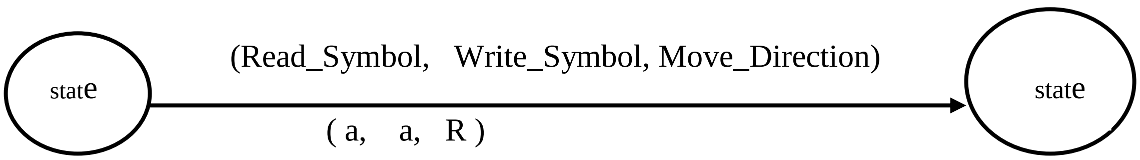
Coding Turing Machines:

A Turning Machine (TM) has a finite set of states with one START state and some (may be none) HALT states. We always mark the START state with 1 and the HALT state with 2, (when there is only one HALT state). There are transitions between states, each of which is marked by a triplet:



The above triplet shown on the arrow means read an **a** from the current cell of the READ/WRITE tape (see Figure 1), write an **a** on the same cell, and move right one cell on the tape. Each Turing Machine (TM) has an infinite Tape divided into a sequence of cells each containing a symbol or a blank. The input is presented to the machine one symbol per cell beginning the leftmost cell after a marker (#) as shown below (in Figure 1).

READ/WRITE TAPE

#	a	a	a	B	Δ	Δ	Δ	Δ	Δ	Δ	Δ

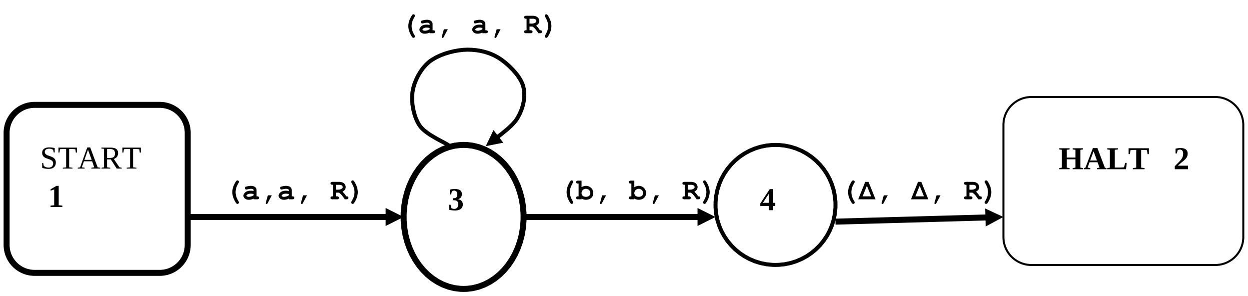


Figure 1: A Turing Machine for { aa*b }

The input is processed by the transitions as the TM goes from state to state starting from the start state. The TM given in Figure 1 accepts the language: { aa*b }, because for every string of the language, the machine starts from the START state and goes through the other states using the transitions, and finally reaches the HALT state.

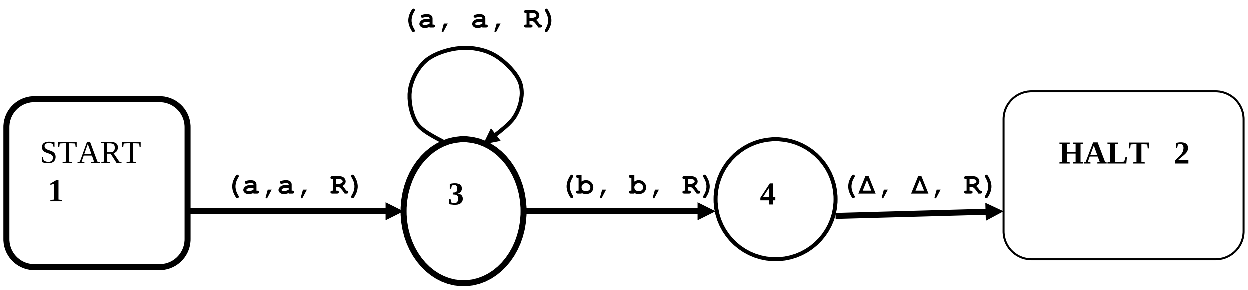
Encoding of Turing Machines:

Any Turing Machine (TM) can be represented in a table and then the table can be encoded into a string of **a**'s and **b**'s. The above TM is represented in the table below (Table 1):

FROM	TO	READ	WRITE	MOVE
1	3	a	a	R
3	3	a	a	R
3	4	b	b	R
4	2	Δ	Δ	R

Table 1. The TM of Figure 1 is represented in a tabular form

Since the START state is 1 and the HALT state is 2 all the information for operating the TM is available in the table. Any row of the table can be coded into a string of a's and b's.



Consider the general form of the rows of Table 1:

FROM	TO	READ	WRITE	MOVE
X_1	X_2	X_3	X_4	X_5

Where X_1 and X_2 are numbers and X_3 and X_4 are characters from {a, b, #, **Δ** } and X_5 is a direction (either L or R). A separator, **b**, is used Between X_1 and X_2 , and between X_2 and X_3 The concatenated sequence of X_1 and X_2 with the separator has the form:

$$a^{X1} b a^{X2} b$$

which means a string of **a**'s of length X_1 concatenated to a **b** concatenated to a string of **a**'s X_2 long concatenated to a **b**. X_3 and X_4 are encoded by the following table:

X_3/ X_4	CODE
a	aa
b	ab
Δ	ba
#	bb

Next, X_5 is encoded as follows:

X_5	CODE
L	a
R	b

For the TM of Figure 1, the code for each row is given in the following table (Table 2):

	From	**Sep**.	To	**Sep**.	Read	Write	Move	Code for each Row
	1		3		a	a	R	
Code:	a	**b**	aaa	**b**	aa	aa	b	a**b**aaa**b**aaaab
	3		3		a	a	R	
Code:	aaa	**b**	aaa	**b**	aa	aa	b	aaa**b**aaa**b**aaaab
	3		4		b	b	R	
Code:	aaa	**b**	aaaa	**b**	ab	ab	b	aaa**b**aaaa**b**ababb
	4		2		Δ	Δ	R	
Code:	aaaa	**b**	aa	**b**	ba	ba	b	aaaa**b**aa**b**babab

Table 2. Each Row of Table 1 is coded in a's and b's immediately below that row

The code for the TM in Code Word Language (CWL) is the concatenation of the four encoded rows:

abaaabaaaabaaabaaabaaaabaaabaaaabababbaaaabaabbabab

Cohen (1997) explains how the coded TMs can be used to construct a Universal Turing Machine (UTM) (See http://www.asethome.org/mathfoundations/utm).

The Code Word Language (CWL) is characterized with following pattern:

CWL = the language defined by $(aa^*baa^*b(a+b)^5)^*$

Informally, $aa^* = a^+$ and, therefore, CWL = { $(a^+ba^+b(a+b)^5)^*$ }

Some TMs accept their own code; others do not accept their own code. The TM of Figure 1 does not accept its own code. The TM of Figure 2 accepts its own code.

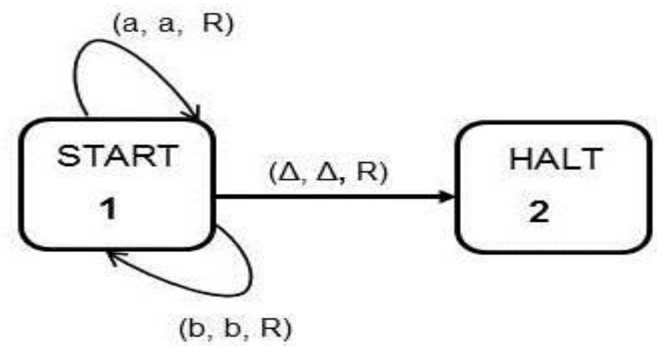


Figure 2: A Turing Machine for $(a+b)^*$

Please note that there are strings in CWL, that do not represent any TM. Thus, the string, **aabaabaaaab** is a valid string in CWL but it does not represent any TM. Now, we define the language ALAN as follows.

ALAN = { all the words in CWL that are not accepted by the TMs they represent or that do not represent any TM }

It can be proven that ALAN is not computable. The language ALAN is a strange one, so strange that it is not even recursively enumerable. We will prove this by contradiction. Assume that ALAN is r.e. Then there is a Turing machine that accepts ALAN. Call this Turing machine T. Now T can be encoded in CWL just as any Turing machine can be encoded. Call its encoding code(T). Consider the question, "Is code(T) in ALAN?" We shall consider the two possible answers separately.

Case 1: code(T) **is** in ALAN

Since T accepts words that are in ALAN, T accepts code(T). But ALAN contains no code word that is accepted by the machine it encodes, so this is a contradiction.

Case 2: code(T) **is not** in ALAN
Then T does not accept code(T). But then code(T) is in ALAN because any encoding that is not accepted by the machine it encodes is in ALAN. Again, this is a contradiction.

Since both cases lead to contradictions, ALAN is not a recursively enumerable language. This example proves the following theorem.

CASE 1: code(T) is in ALAN

CLAIM	REASON
1. T accepts ALAN	1. Definition of T
3. Code(T) is in ALAN	3. Hypothesis
4. T accepts code(T)	4. From 1 and 3

CLAIM	REASON
1.	
2. ALAN contains no code word that is accepted by the machine it represents.	2. Definition of ALAN
3.	
4. T accepts code(T)	4. From 1 and 3
5. Code(T) is not in ALAN	5. From 2 and 4
6. Contradiction	6. From 3 and 5

CASE 2: code(T) is NOT in ALAN

CLAIM	REASON
1. T accepts ALAN	1. Definition of T
2. If a word is not accepted by the machine it represents, it is in ALAN .	2. Definition of ALAN
3. Code(T) is NOT in ALAN	3. Hypothesis
4. code(T) is not accepted by T	4. From 1 and 3
5. Code(T) is in ALAN	5. From 2 and 4
6. Contradiction	6. From 3 and 5

The contradiction proves that ALAN is not computable.
Based on Cohen, D. (1997). Introduction to Computer Theory (2nd ed.), New York: John Wiley

Please visit:
http://www.asethome.org/mathfoundations/asynchronous/