CS F351: Theory of Computation

06 – Turing Machine

BITS Pilani
Dubai Campus

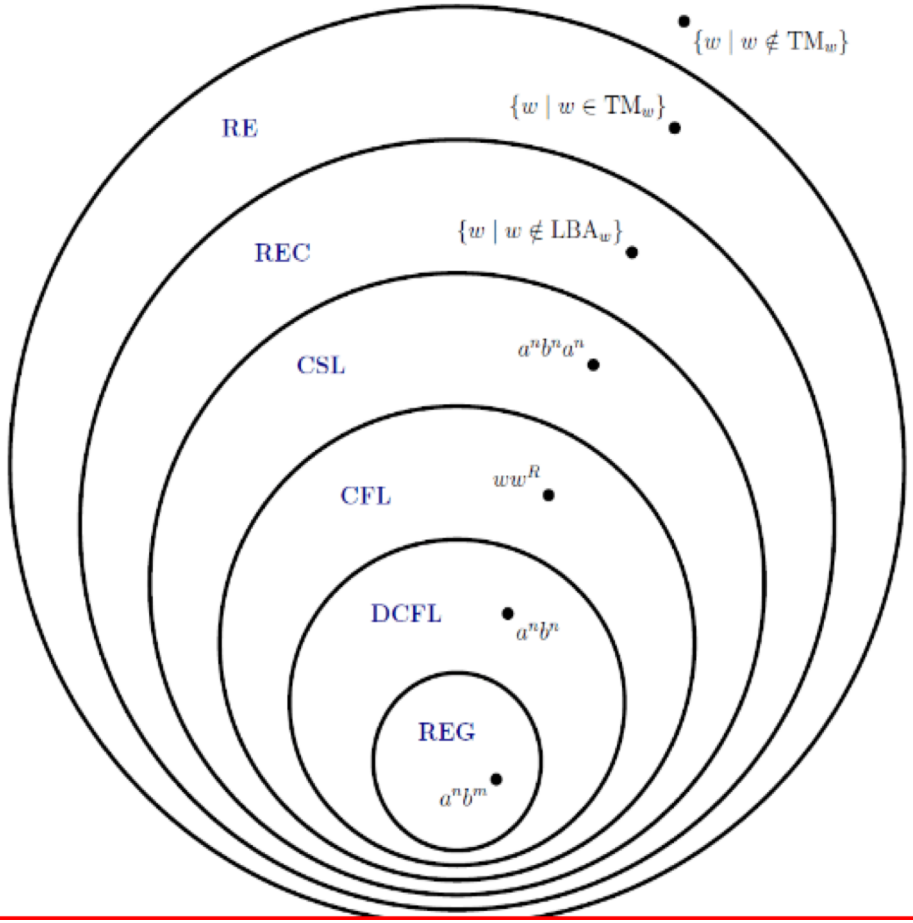
Dr Elakkiya R, AP/CS

Chomsky Hierarchy

innovate achieve lead

Chomsky Hierarchy

As a concise reference to the entire subject, we put some of the formal languages in the levels of the Chomsky hierarchy and summarize of the closure properties of the levels.



Turing Machine

1. Most Powerful model of computation (know to humans)
2. If there exits a turning machine for a problem, then the

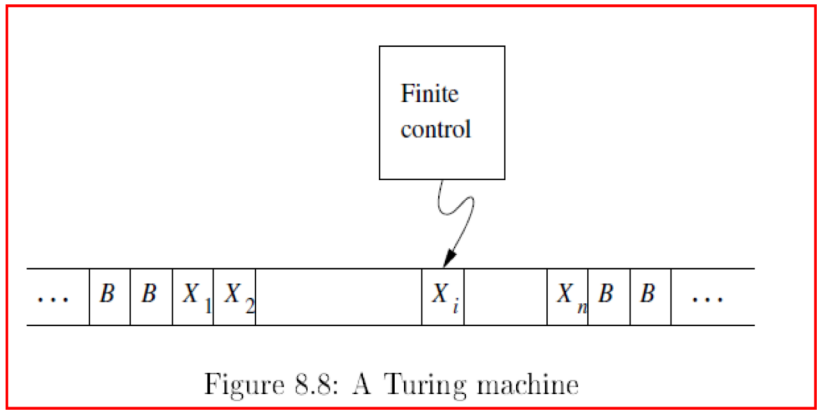
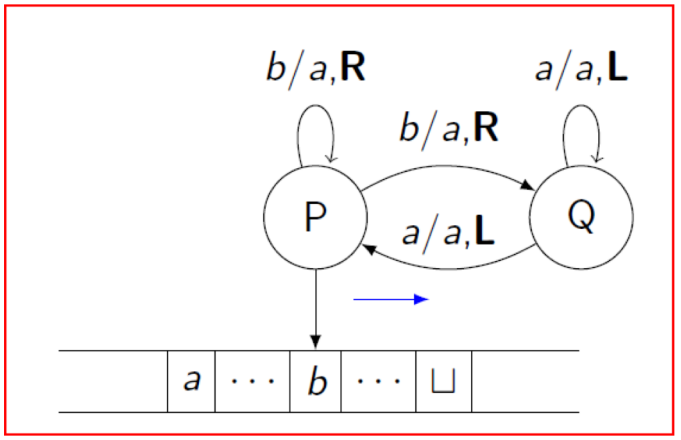
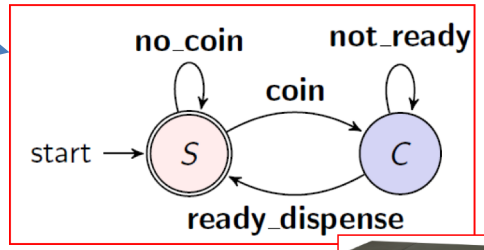


Figure 8.8: A Turing machine



1. PDA are obtained from NFA by adding a stack
2. Turing machines are obtained from NFA by adding an **infinite tape**.
3. A Turing machine can both **write** on the tape and **read** from it.
4. The **read-write head** can move both to **left** and **right**.
5. Initially all cells have special blank symbol, except where input is written.
6. Once accept/reject states are reached, computation terminates.
7. TMs can loop forever (equivalent to saying no algorithm exists to solve that problem)

Reg L



Formal Definition of TM

innovate achieve lead

Definition

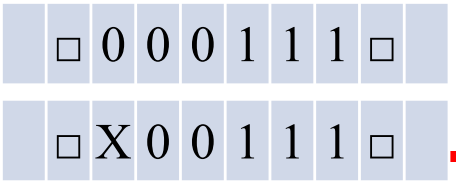
A Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

1. Q is a finite set of states,
2. Σ is a finite input alphabet,
3. Γ is a finite tape alphabet where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. q_0 is the start state,
5. q_{acc} is the accept state,
6. q_{rej} is the reject state,
7. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function.

For a $q \in Q$, $a \in \Gamma$ if $\delta(q, a) = (q', b, L)$, then

- q' is the new state of the machine,
- b is the letter which replaces a on the tape,
- the head moves to Left of the current position.

TM for 0^n1^n - *CFL*



1. Cross out the 1st 0 with X (so that we know, that 0 has been accounted for)
2. For every input which is a 0, move right (keeping the 0 intact), and till a 1 is reached
3. Replace the 1 with Y. (to indicate that we have found a matching 1), move left
4. Keep moving left, till a X is reached, and then move right.
5. if the input is a 0, then repeat step 1

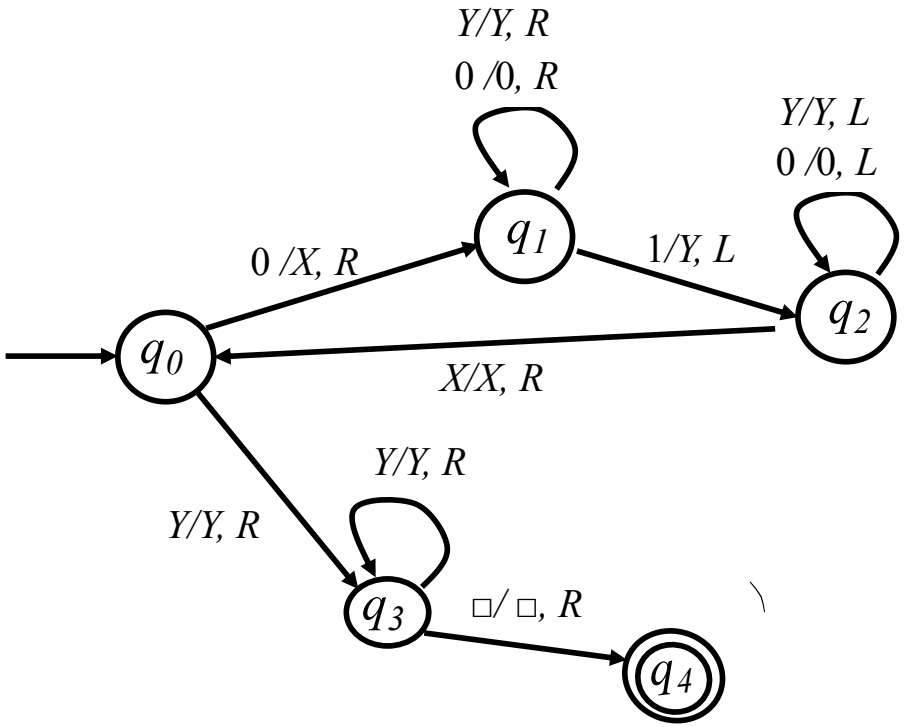
After few replacement the configuration would be as shown.

In order to handle all possible legal moves, the above steps are to be modified as follows.

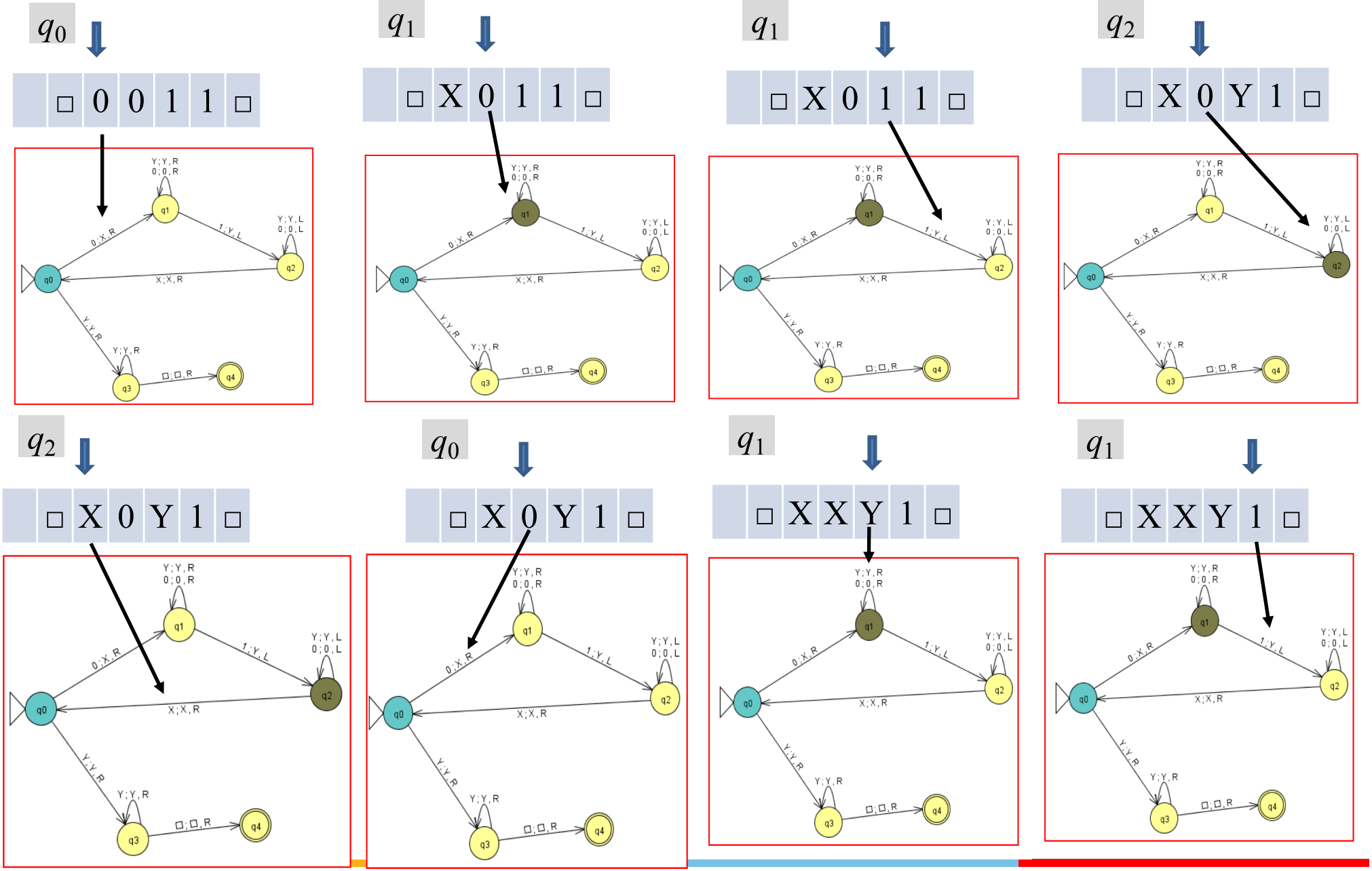
1. Cross out the next 0 with X (so that we know, that 0 has been accounted for)
2. For every input which is a 0/Y, move right (keeping the 0/Y intact), and till a 1 is reached
3. Replace the 1 with Y. (to indicate that we have found a matching 1), and move left
4. Keep moving left (Y/0), till a X is reached, and then move right.
5. if the input is a 0, then repeat step 1
6. if the input is a Y, then skip the inputs Y
7. if the input is a □, then accept and halt
8. Handle error conditions

TM for 0^n1^n - *CFL*

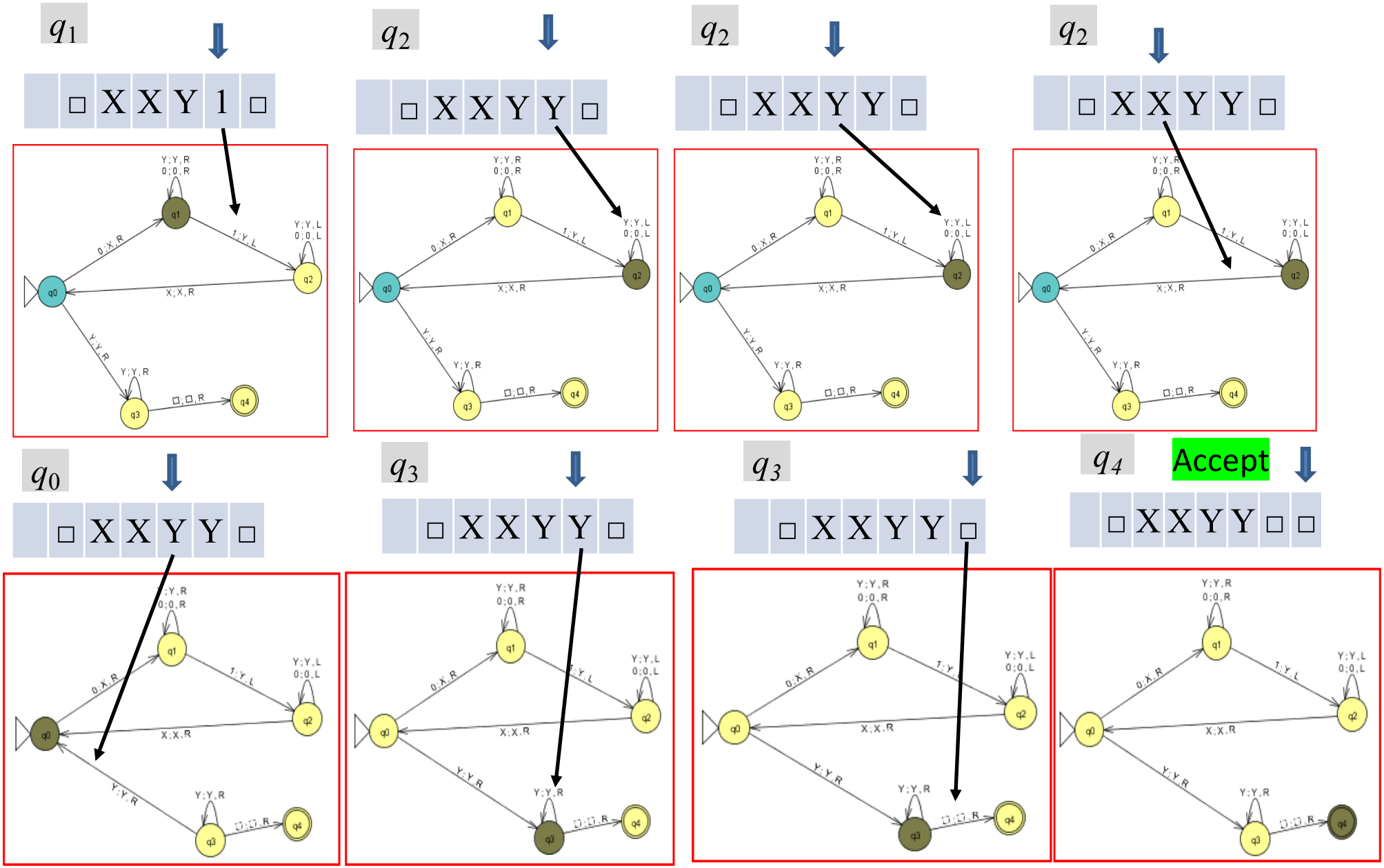
1. Cross out the next 0 with X (so that we know, that 0 has been accounted for), move right.
2. For every input which is a 0,Y, move right (keeping the 0/Y intact), and till a 1 is reached
3. Replace the 1 with Y. (to indicate that we have found a matching 1), and move left.
4. Keep moving left (Y/0), till a X is reached, and then move right.
5. if the input is a 0, then repeat step 1.
6. if the input is a Y, then skip the inputs Y, and move right.
7. if the input is a □, then accept and halt
8. Handle error conditions (Any non-accepting state which does not have a corresponding input transitions to an accept state).



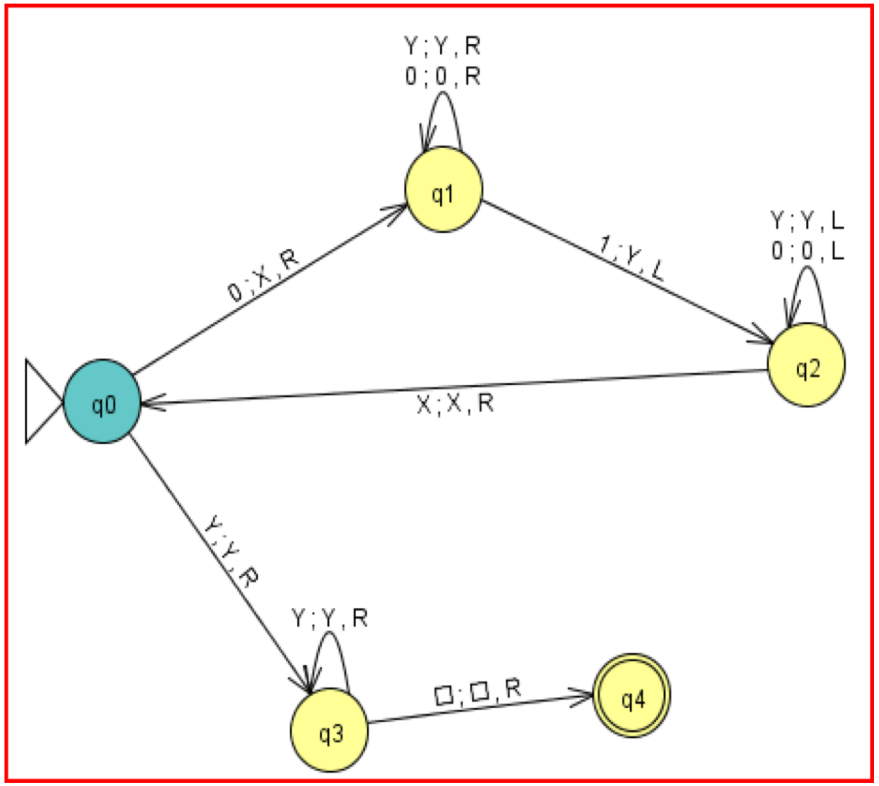
0^n1^n ,n>=1 *CFL*

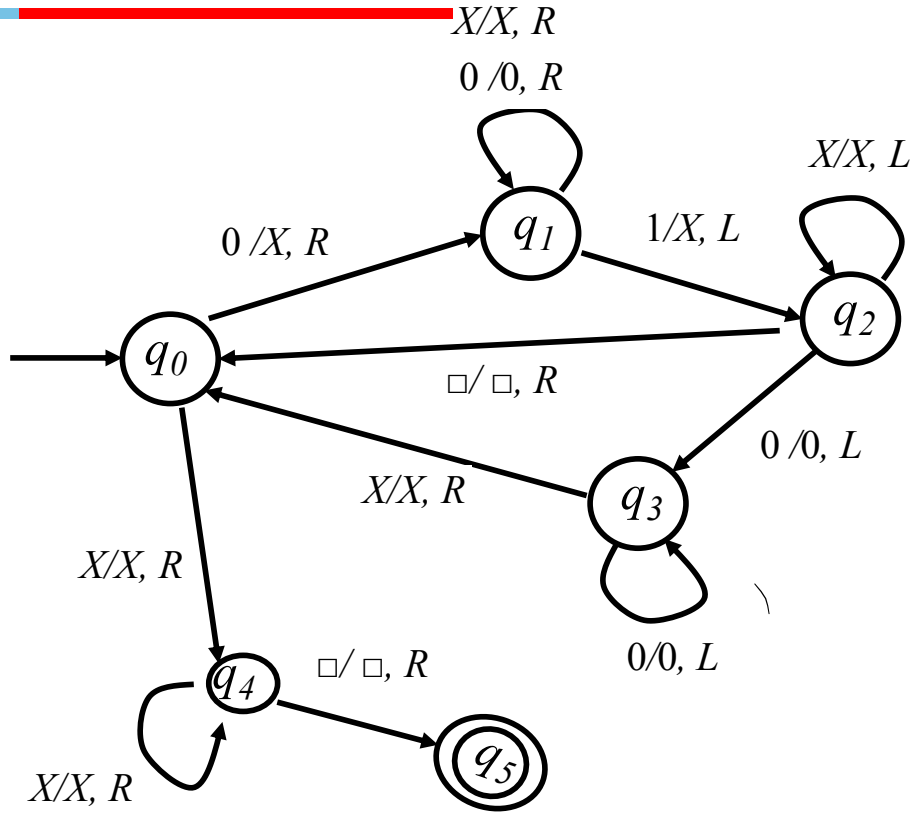

q_0
□ 0 0 1 1 □
q_1
□ X 0 1 1 □
q_1
□ X 0 1 1 □
q_2
□ X 0 Y 1 □
q_2
□ X 0 Y 1 □
q_0
□ X 0 Y 1 □
q_1
□ X X Y 1 □
q_1
□ X X Y 1 □
Y;Y,R 0;0,R
0;X,R
1;Y,L
Y;Y,L 0;0,L
X;X,R
Y;Y,R
□;□,R
q0
q1
q2
q3
q4

0^n1^n ,n>=1 *CFL*


q1
□ X X Y 1 □
q2
□ X X Y Y □
q2
□ X X Y Y □
q2
□ X X Y Y □
q0
□ X X Y Y □
q3
□ X X Y Y □
q3
□ X X Y Y □
q4
Accept
□ X X Y Y □ □
Y;Y,R 0;0,R
0;X,R
1;Y,L
Y;Y,L 0;0,L
X;X,R
Y;Y,R
□;□,R
q0
q1
q2
q3
q4

0^n1^n - CFL





TM which accepts, 0^n1^n

0's are replace by X's

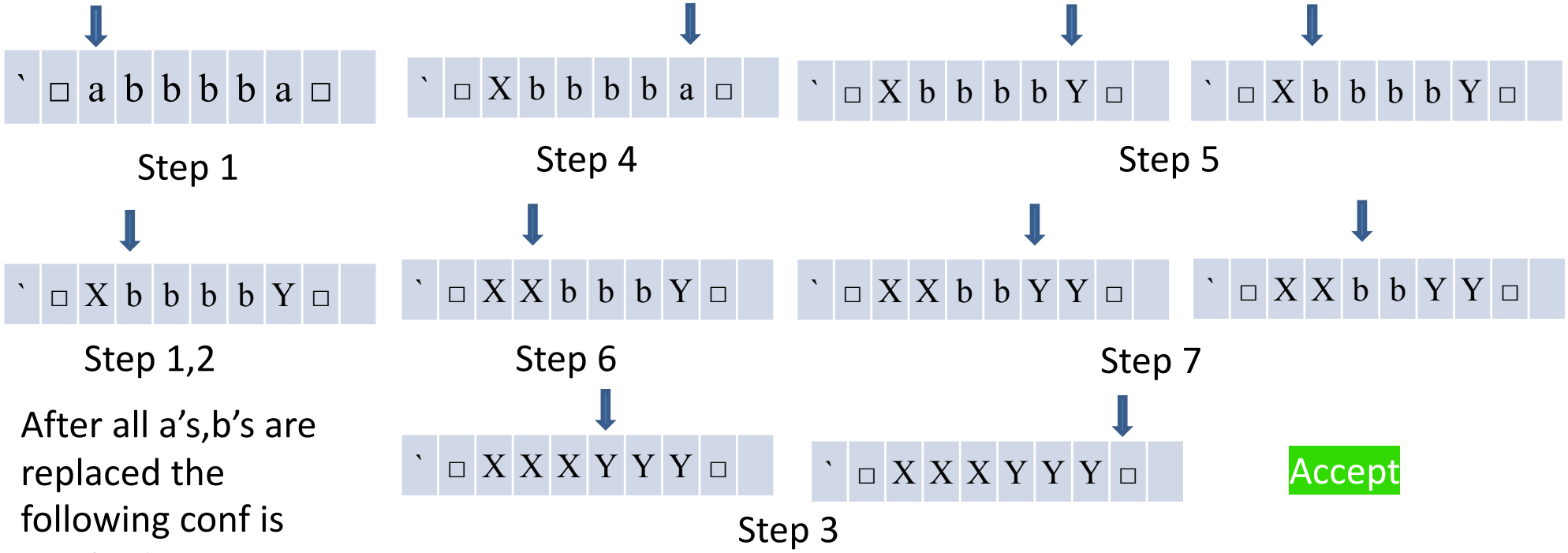
1's, are replaced by Y's.

Another TM which accepts, 0^n1^n

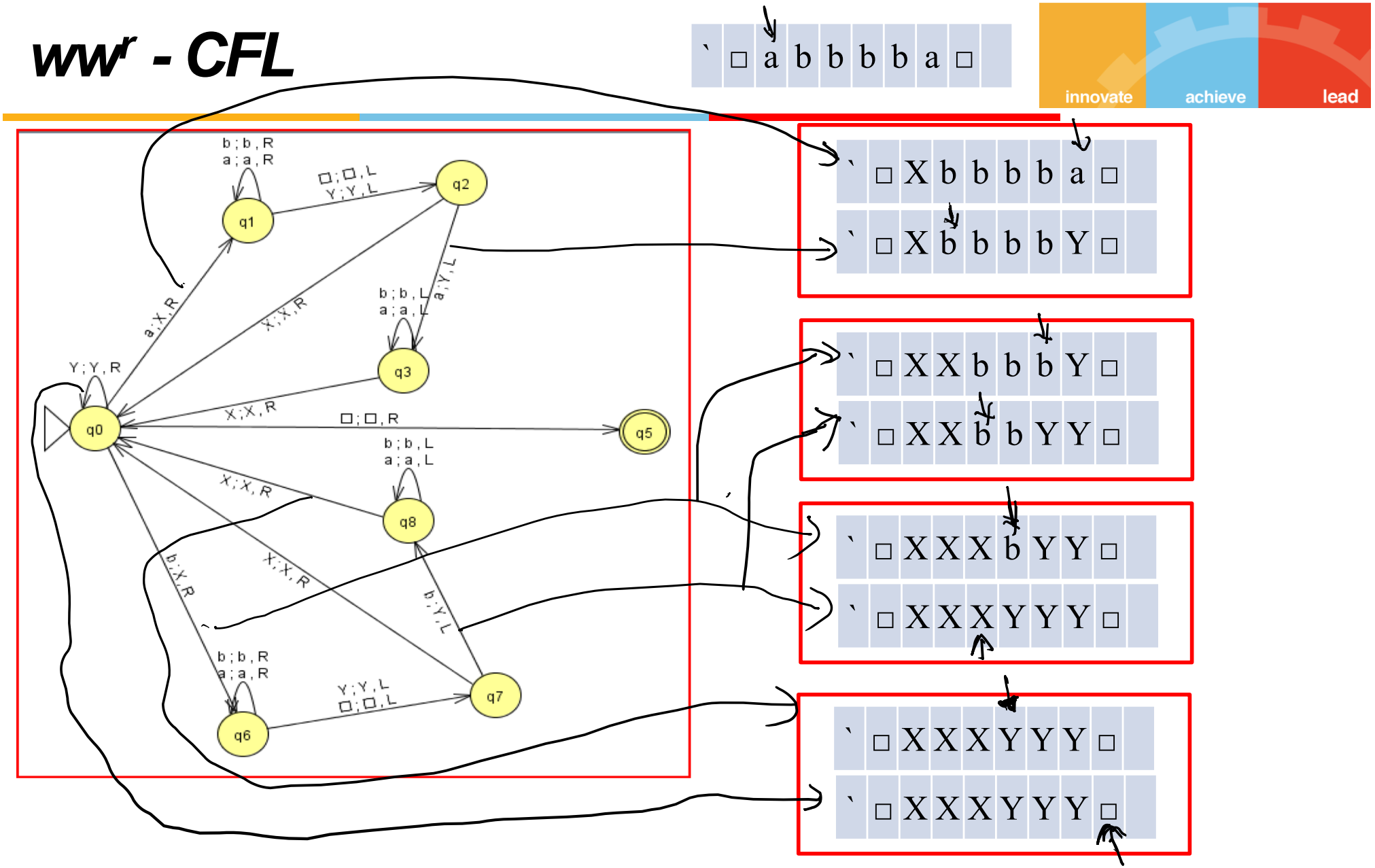
Both 0's and 1's, are replaced by X's.

ww^r (consider both odd & even) - CFL

1. if the next input is a, then replace with X, move right, goto 4
2. if the next input is b, then replace with X, move right, goto 6
3. if the next input in Y, then skip Y's (move right), till a □ is reached, Accept and Halt.
4. Skip a's, b's (move right) and till a □/Y is reached, move left.
5. if the input is a, replace with Y, move left skipping a's, b's, till X is reached, move right, goto 1
6. Skip a's, b's (move right) and till a □/Y is reached, move left.
7. if the input is b, replace with Y, move left skipping a's, b's, till X is reached, move right, goto 1
8. Handle error conditions (Any non-accepting state which does not have a corresponding input transitions to an accept state).



ww^r - CFL

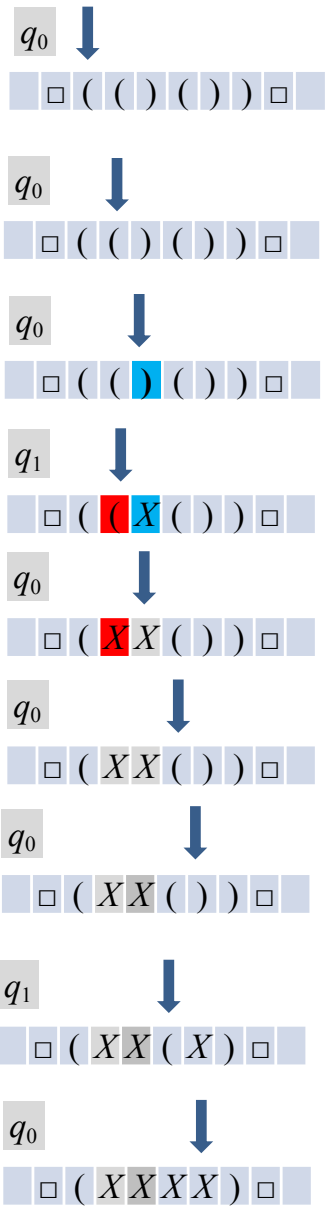

` □ a b b b b a □
q0
q1
q2
q3
q5
q6
q7
q8
b ; b , R
a ; a , R
□ ; □ , L
Y ; Y , L
a ; X , R
X ; X , R
b ; b , L
a ; a , L
a ; Y , L
Y ; Y , R
□ ; □ , R
b ; X , R
b ; Y , L
Y ; Y , L
□ ; □ , L
` □ X b b b b a □
` □ X b b b b Y □
` □ X X b b b Y □
` □ X X b b Y Y □
` □ X X X b Y Y □
` □ X X X Y Y Y □
` □ X X X Y Y Y □
` □ X X X Y Y Y □

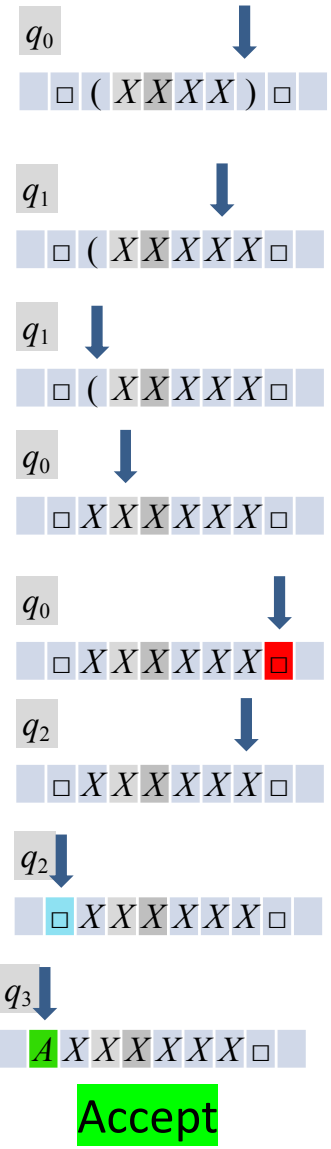
Balanced Parenthesis

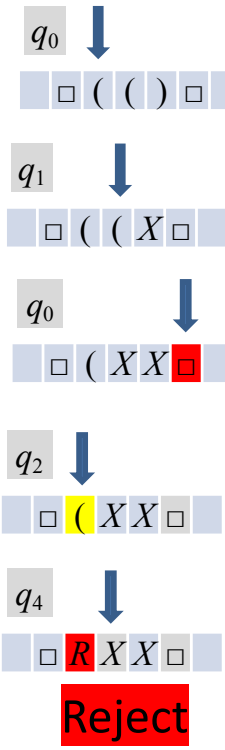
Parenthesis

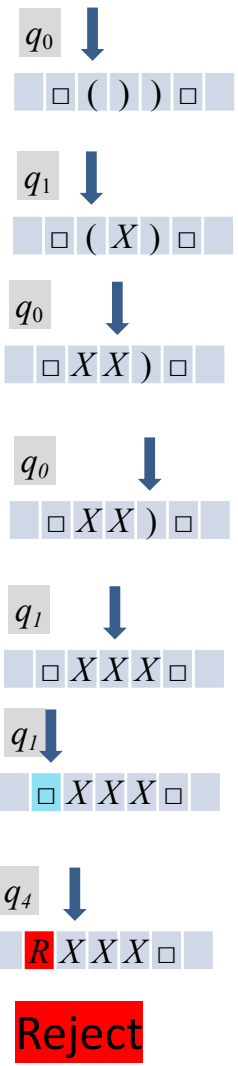
q0: move right looking for a),

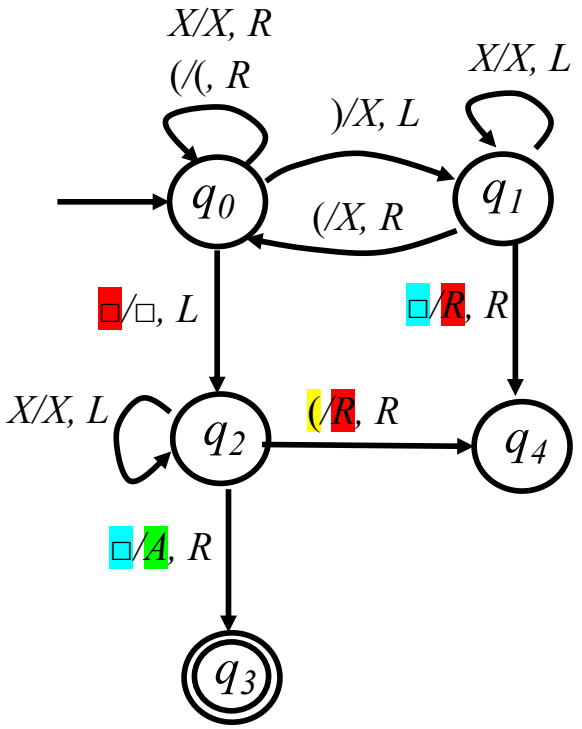
q1: Got), move left look for matching (



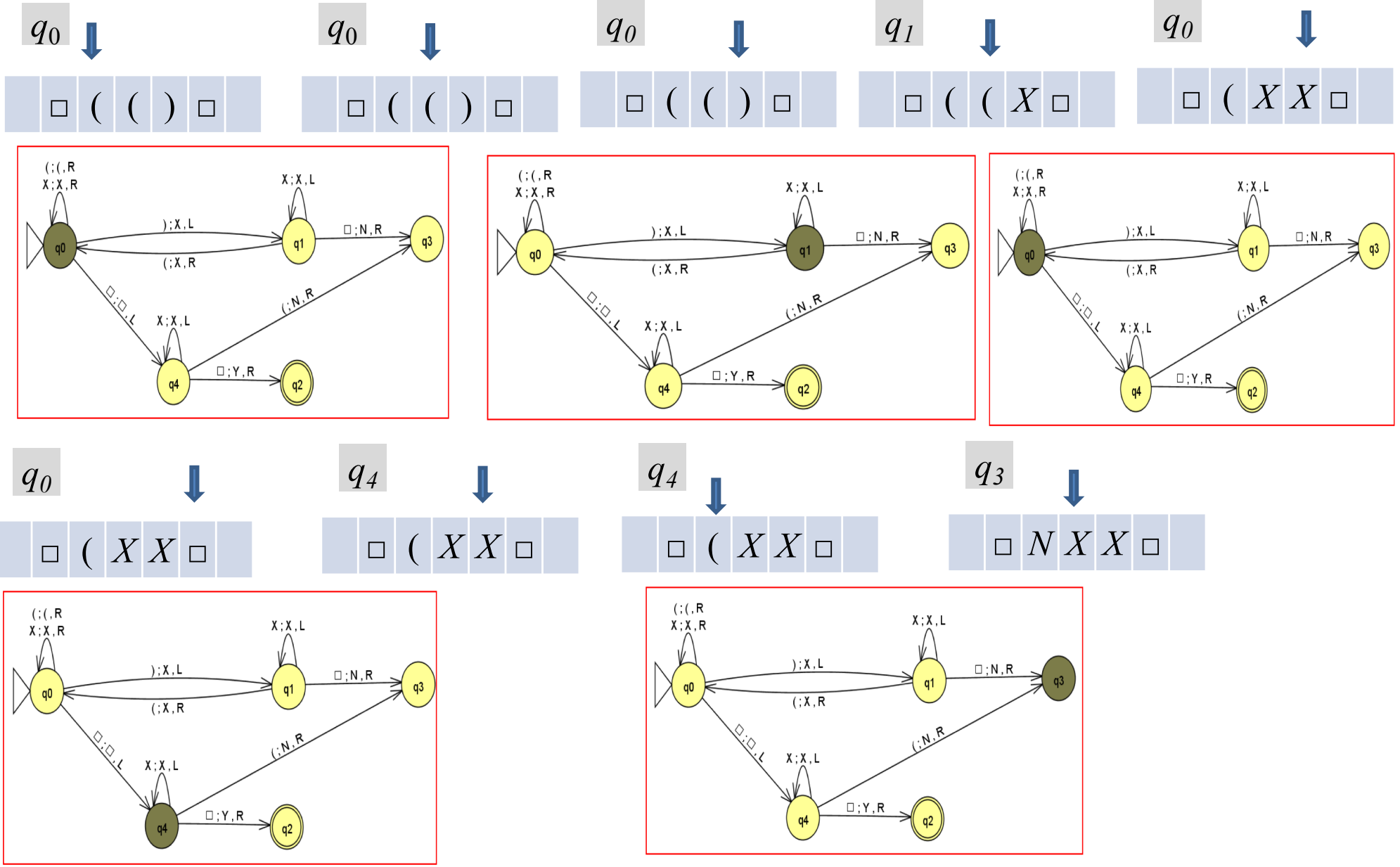


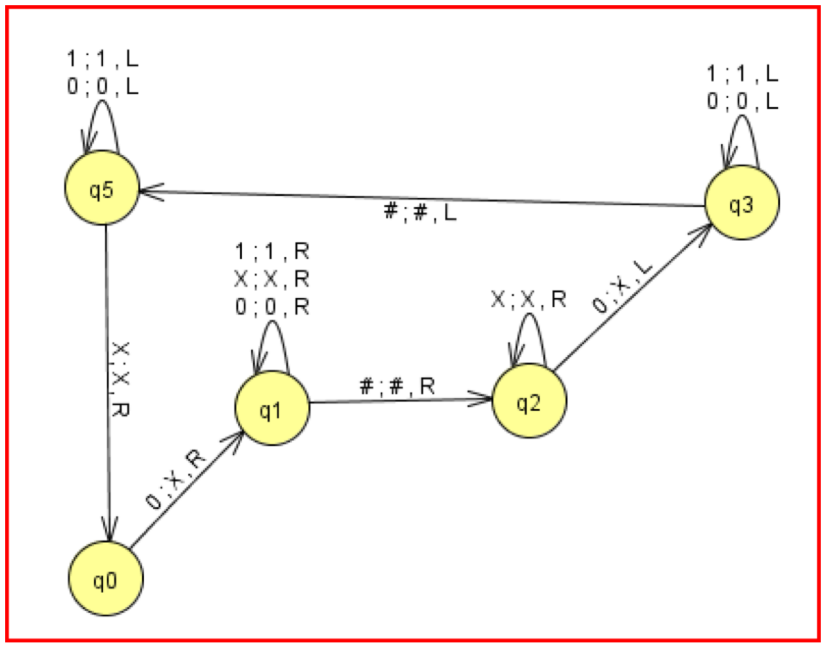




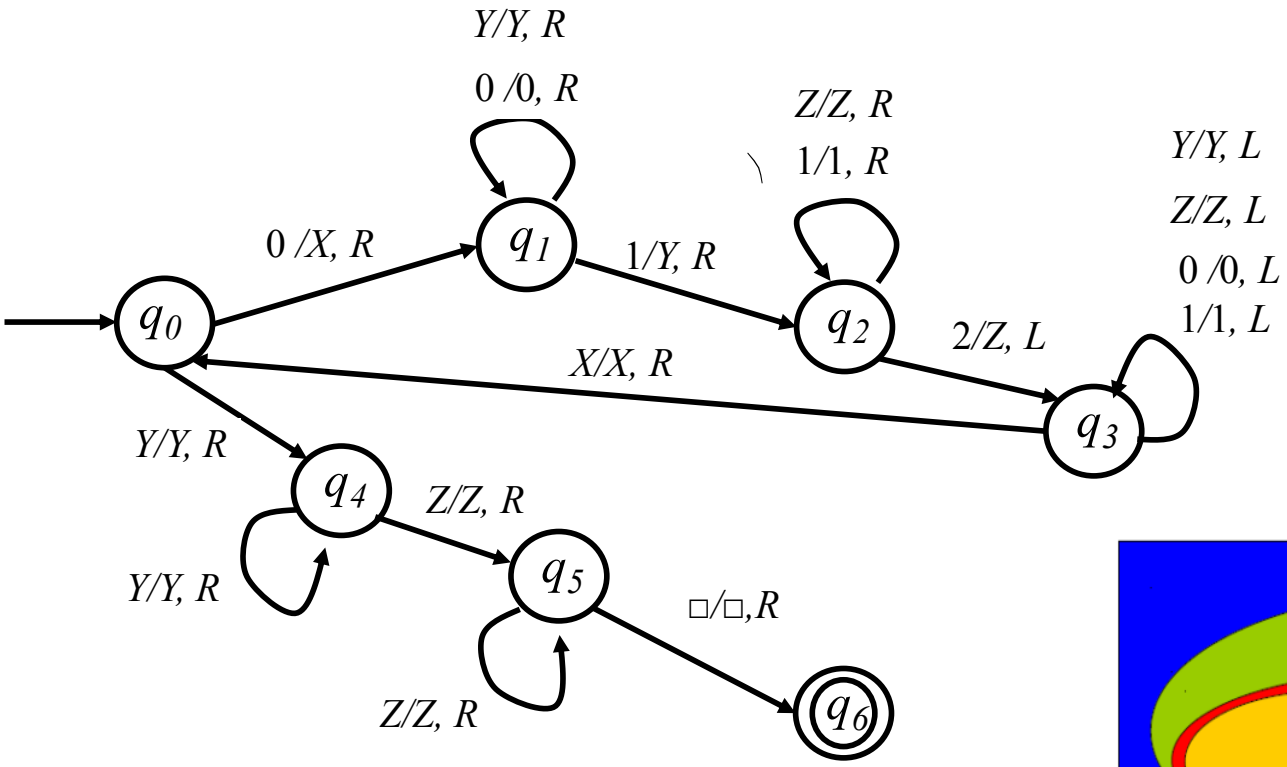


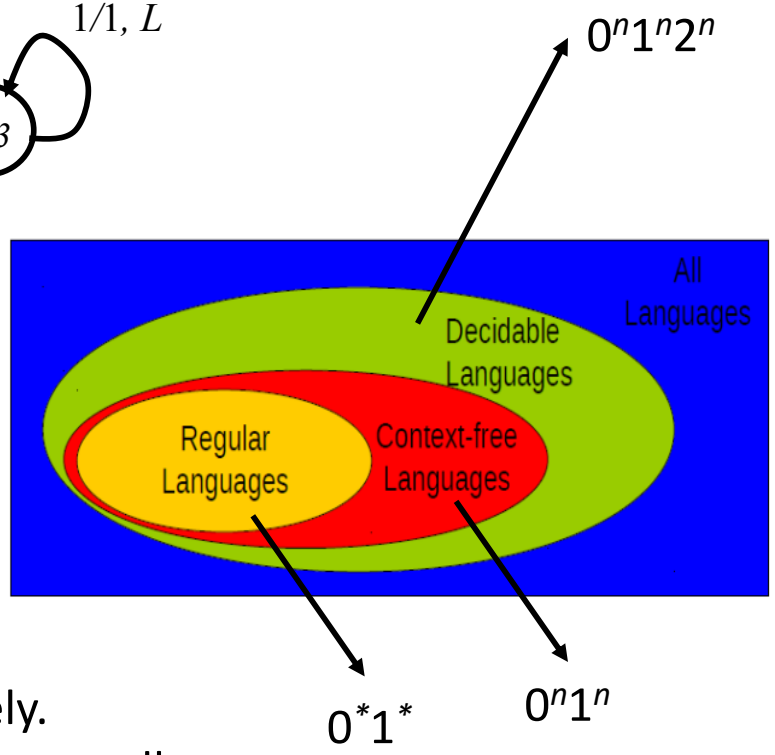
Balanced Parenthesis




1 ; 1 , L
0 ; 0 , L
q5
1 ; 1 , L
0 ; 0 , L
q3
#;#,L
1 ; 1 , R
X ; X , R
0 ; 0 , R
q1
#;#,R
q2
X ; X , R
0 ; X , L
X ; X , R
0 ; X , R
q0

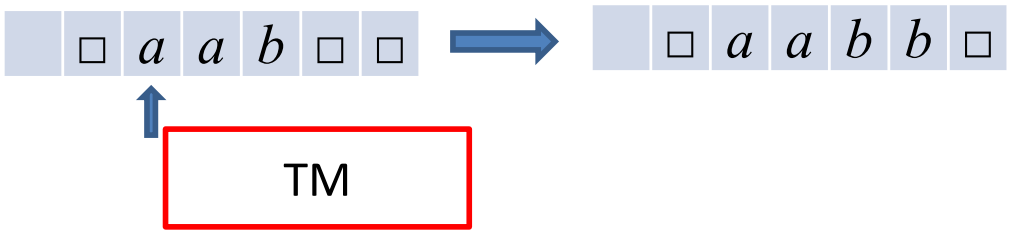
TM for $0^n1^n2^n$ (Non-CFL)





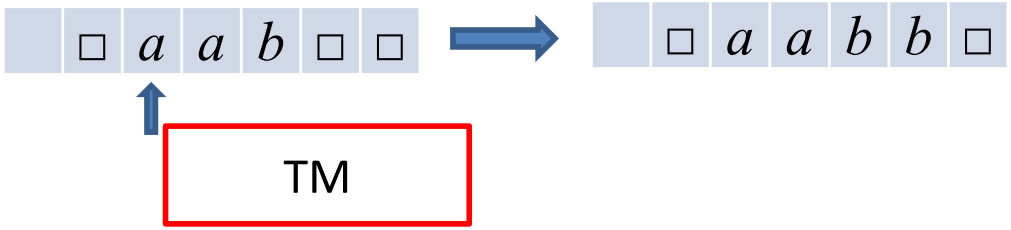
- Every matching 0,1,2 is replaced by X,Y,Z respectively.
- Machine goes back and forth do the above steps repeatedly.

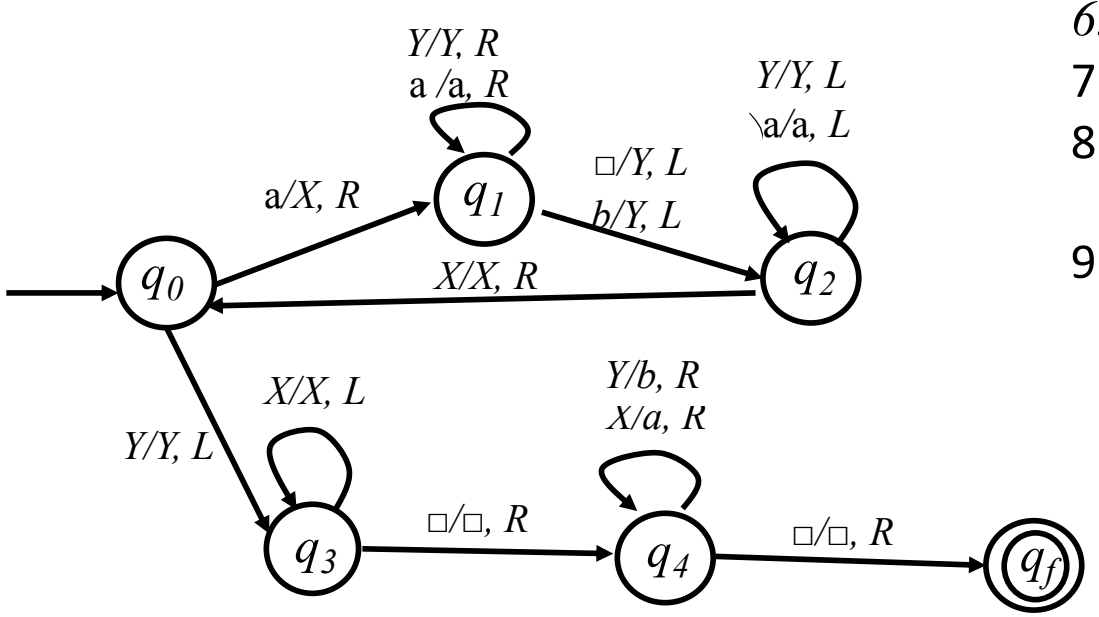
TM: Add b's to Match the a's $a^i b^j, 0 \leq j \leq i$



1. loop:
2. Mark a with X
3. Scan right, check first b or blank(□)
4. If b, mark with Y scan left
5. if blank (□), mark with Y scan left
6. if no more a's , exit loop
7. Scan left for a $blank(□)$
8. Scan right replace X with a, Y with b till blank..
9. Handle error cases.

TM: Add b's to Match the a's $a^i b^j, 0 \le j \le i$





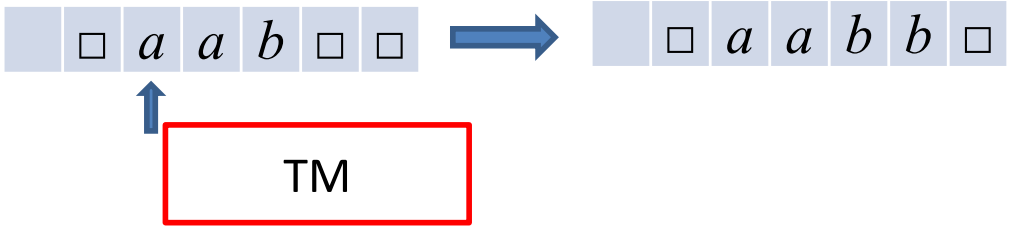
1. loop:
2. Mark a with X
3. Scan right, check first b or blank(□)
4. If b, mark with Y scan left
5. if blank (□), mark with Y scan left
6. if no more a's , exit loop
7. Scan left for a $blank(□)$
8. Scan right replace X with a, Y with b till blank..
9. Handle error cases.

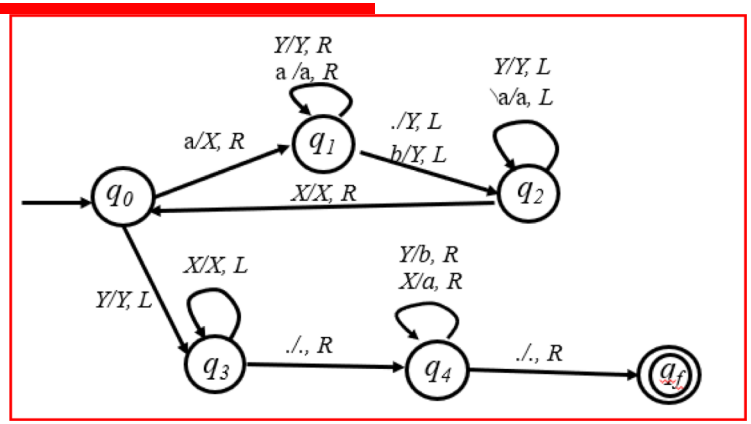
$$q_0aab \vdash Xq_1ab \vdash Xaq_1b \vdash Xq_2aY \vdash q_2XaY \vdash Xq_0aY \vdash XXq_1Y \vdash XXYq_1\square$$

$$\vdash XXq_2YY \vdash Xq_2XYY \vdash Xq_3XYY \vdash q_3XXYY \vdash q_3\square XXYY \vdash q_4XXYY$$

$$\vdash aq_4XYY \vdash aaq_4YY \vdash aabq_4Y \vdash aabbq_4\square \vdash aabb\square q_f$$

TM: Add *b*'s to Match the *a*'s $a^i b^j, 0 \le j \le i$





$\square\ q_0\ a\ \ a\ \ b\ \square$ q_0aab

$\vdash \square\ X\ q_1\ a\ \ b\ \square$ $\vdash Xq_1ab$

$\vdash \square\ X\ \ a\ q_1\ b\ \square$ $\vdash Xaq_1b$

$\vdash \square\ X\ q_2\ a\ \ Y\ \square$ $\vdash Xq_2aY$

$\vdash \square\ q_2\ X\ \ a\ \ Y\ \square$ $\vdash q_2XaY$

$\vdash \square\ X\ q_0\ a\ \ Y\ \square$ $\vdash Xq_0aY$

$\vdash \square\ X\ \ X\ q_1\ Y\ \square$ $\vdash XXq_1Y$

$\vdash \square\ X\ \ X\ \ Y\ q_1\ \square$ $\vdash XXYq_1\square$

$\vdash \square\ X\ \ X\ q_2\ Y\ \ Y$ $\vdash XXq_2YY$

$\vdash \square\ X\ q_2\ X\ \ Y\ \ Y$ $\vdash Xq_2XYY$

$\vdash \square\ X\ \ X\ q_0\ Y\ \ Y$ $\vdash XXq_0YY$

$\square\ X\ q_3\ X\ \ Y\ \ Y$ $\vdash Xq_3XYY$

$\square\ q_3\ X\ \ X\ \ Y\ \ Y$ $\vdash q_3XXYY$

$q_3\ \square\ X\ \ X\ \ Y\ \ Y$ $\vdash q_3\square XXYY$

$\square\ q_4\ X\ \ X\ \ Y\ \ Y$ $\vdash q_4XXYY$

$\square\ a\ q_4\ X\ \ Y\ \ Y$ $\vdash aq_4XYY$

$\square\ a\ \ a\ q_4\ Y\ \ Y$ $\vdash aaq_4YY$

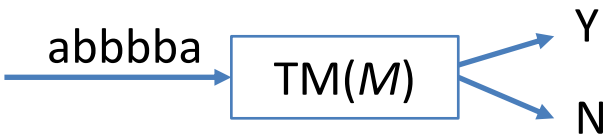
$\square\ a\ \ a\ \ b\ \ Y$ $\vdash aabq_4Y$

$\square\ a\ \ a\ \ b\ \ b\ q_4\ \square$ $\vdash aabbq_4\square$

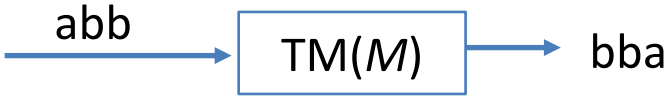
$\square\ a\ \ a\ \ b\ \ b\ \square\ q_f$ $\vdash aabb\square q_f$

Turing Machine use cases

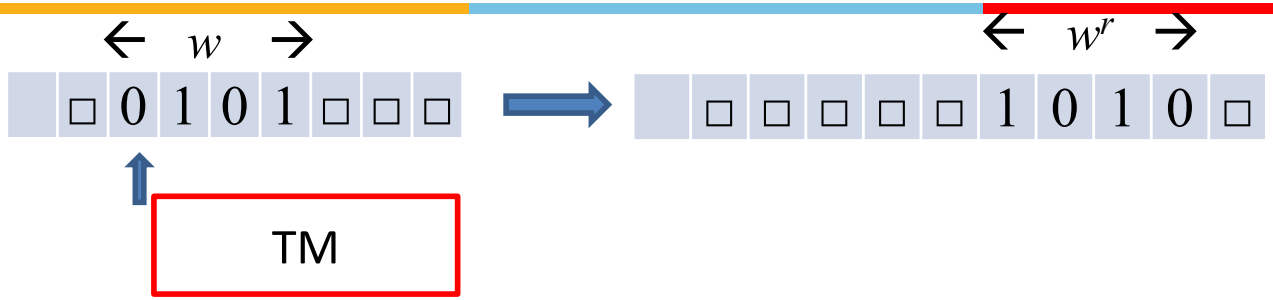
- TM as language decider/recognizer.
  - Deciders TMs always halt with Y/N answer.
  - Recognizer: TM halts for correct inputs with Y answer
    for wrong inputs it may never halt.



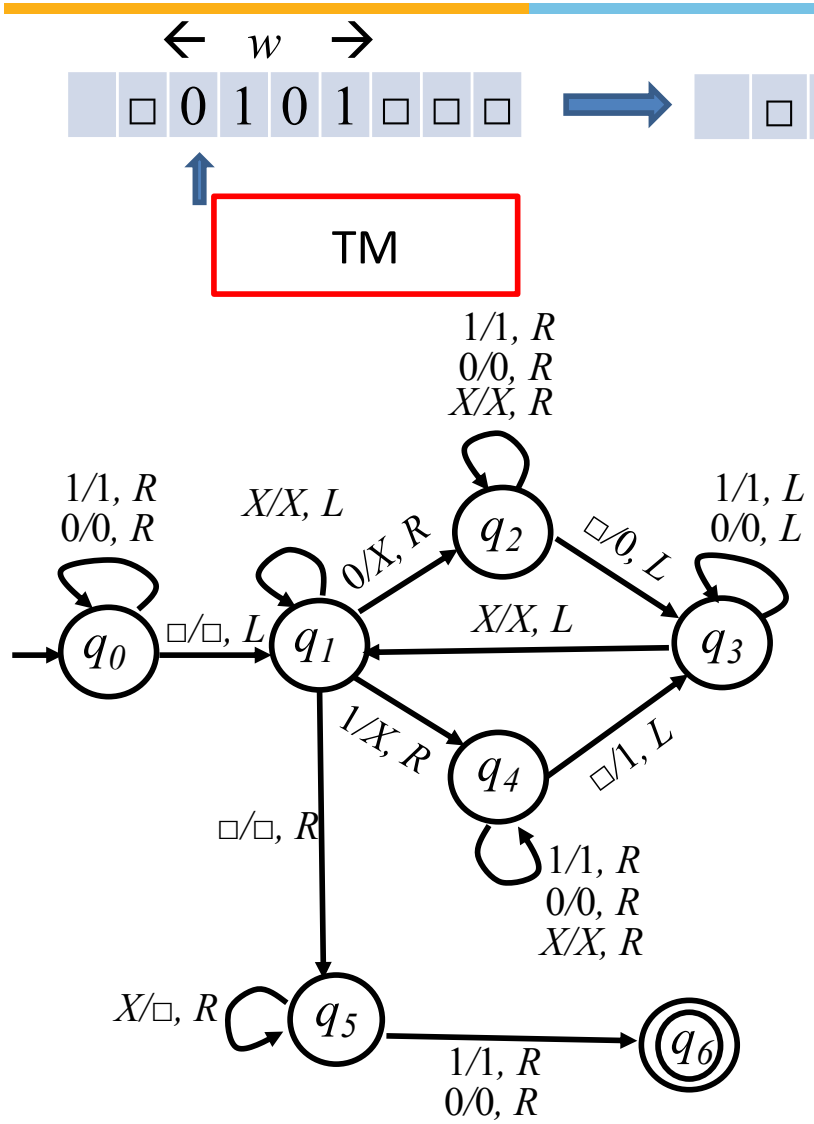
- TM use to compute functions
  - eg. TM outputs the reversed string
- More close to “computing systems”



TM to compute the reverse (ww^r)

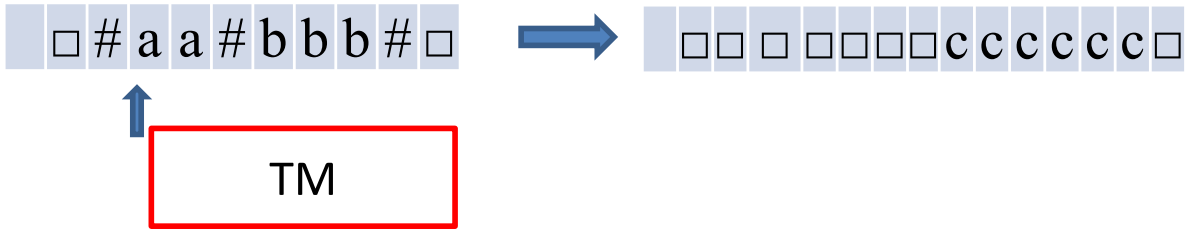


1. Skip the 0/1's till the first blank (□).
2. Go left. (now we point to the last 1/0)
3. *loop:*
4. if its 0, replace with X, Skip (go right) to the next blank(□), replace with 0
5. if its 1, replace with X, Skip (go right) to the next blank(□), replace with 1
6. go left, skip 0/1 till we reach X,
7. go left, skip Xs, if its 0/1 goto 3 (loop)
8. if it's a blank(□), then start going right and replace all X/s with blanks(□).
9. halt at the first 0/1.



1. Skip the 0/1's till the first blank (□).
2. Go left. (now we point to the last 1/0)
3. *loop:*
4. if its 0, replace with X,
   go right, replace next blank(□) with 0
5. if its 1, replace with X,
   go right, replace next blank(□) with 1
6. go left, skip 0/1 till we reach X,
7. go left, skip Xs, if its 0/1 goto 3 (loop)
8. if it's a blank(□),
   go right, replace all X/s with blanks(□).
9. halt at the first 0,1

Multiplication



1. skip #
2. if cur_ch == ‘a’, a -> X, skip a’s till we get @
3. if cur_ch == ‘b’, b -> Y, skip b’s till we get $
4. Skip c’s , till next blank(□). □ -> c, Start moving left
5. *Skip, c,b,$, if cur_ch == Y, loop 3; (go left check for b’s)*
6. *if cur_ch == $, keep going left*
7. *Replace Y -> b, Skip, a,@. if cur_ch == X, loop 2*
8. *if cur_ch == @, keep going left*
9. *if cur_ch = #, replace with* blank(□), go right
10. *Replace X’s,@.$, b, with* blank(□)

□ # a a@b b b $ □ □ □ □ □

□ # X a@b b b $ □ □ □ □ □ □

□ # X a@Y b b $ □ □ □ □ □ □

□ # X a@Y b b $ c □ □ □ □ □

□ # X a@Y Y Y $ c c c □ □ □

□ # X a@b b b $ c c c □ □ □

□ # X X@b b b $ c c c □ □ □

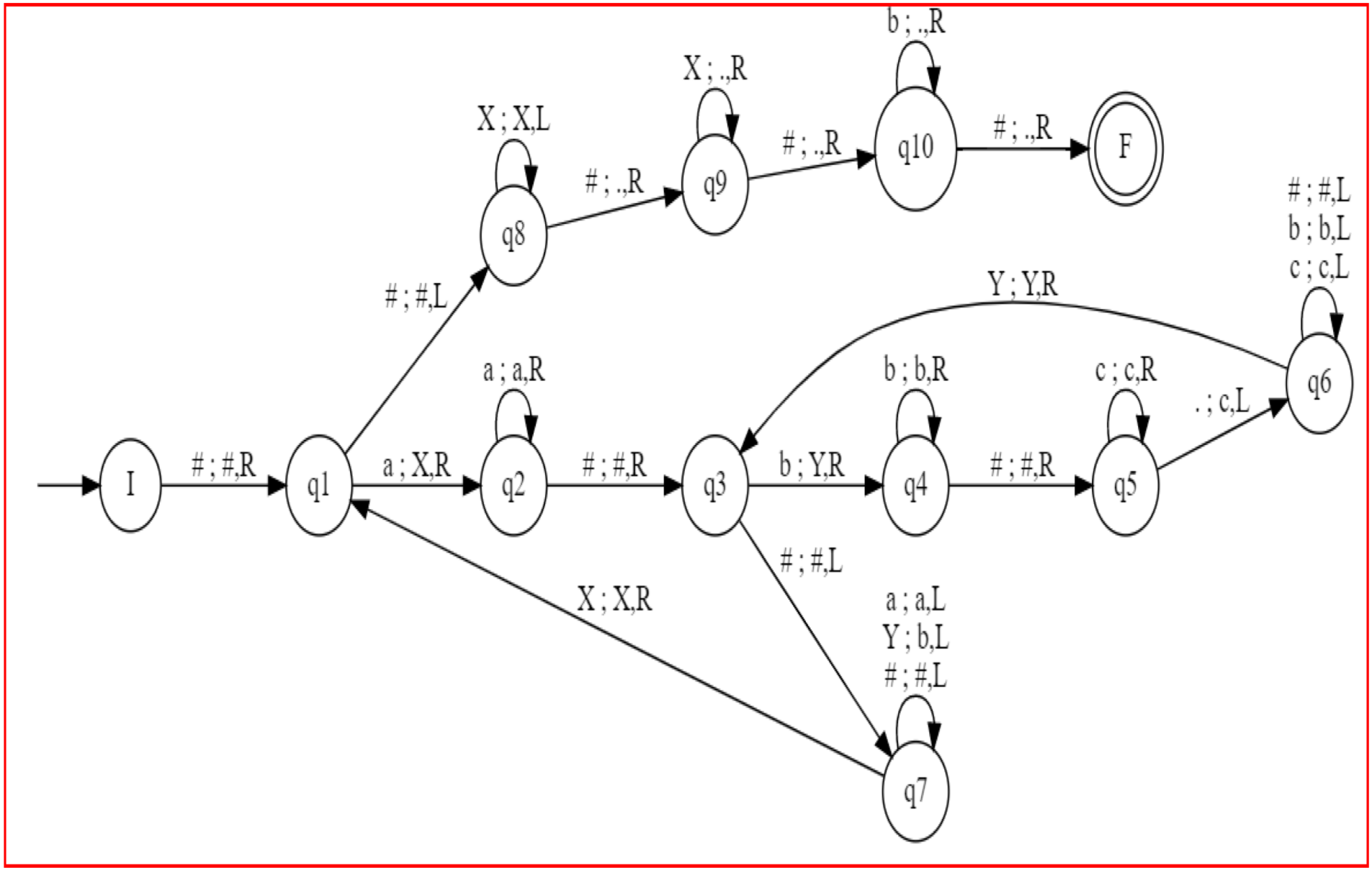
□ # X X@Y b b $ c c c c □ □

□ # X X@Y Y Y $ c c c c c c □

□ # X X@b b b $ c c c c c c □

□□ □ □□□□□□c c c c c c □

Multiplication



Hailstone Sequence

```
// Online C compiler to run C program online
#include <stdio.h>
int main()
{    unsigned int n;
     printf("Pl. enter no :");
     scanf("%d", &n);
     while ( n > 0 ) {
       printf( "%d ", n);
       if ( n == 1)  break;
       if (!(n & 1)) n/= 2;
       else n = 3 * n + 1;
     }
     printf("Done \n");
     return 0;
 }
```

Is there any n for which the program does not terminate. ? (converge to 1)

Ans:
We don't know.

1 Done
2 1 Done
3 10 5 16 8 4 2 1 Done
4 2 1 Done
5 16 8 4 2 1 Done
6 3 10 5 16 8 4 2 1 Done
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 Done
8 4 2 1 Done
9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 Done