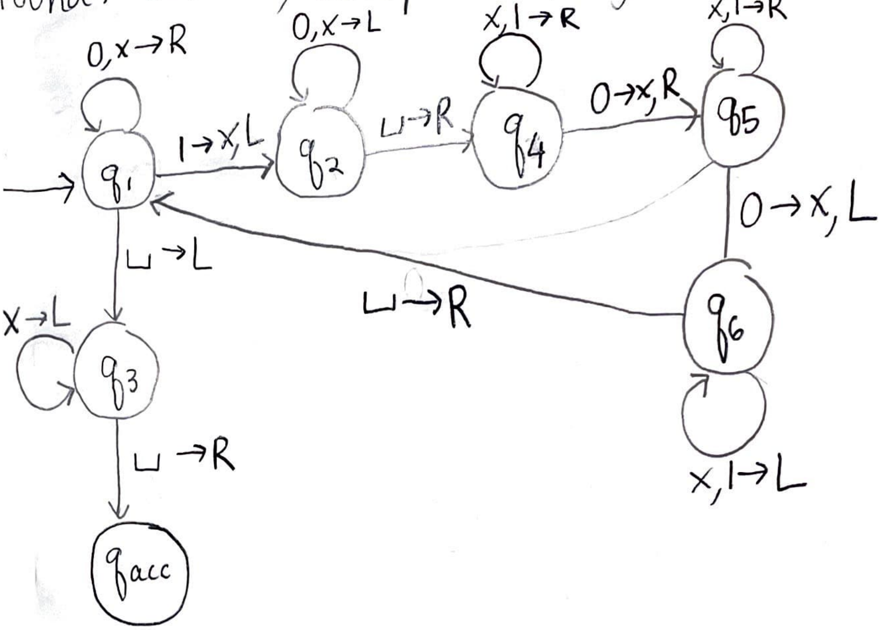
ring Machine that decides language: $\{w \mid w$ contains twice as many 0s as 1s$\}$

Scan the tape and X off the first 1 found. If no 1s found, go to stage 3. Else, return head to the front of the tape.

) Scan and mark the first two 0s found. If less than two are found, reject. Else, return head to front and go to stage 1.

3) Scan to check that no extra 0s are found. If not, accept. Else, reject.



$M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{acc}, q_{rej})$

$Q = \{q_1, q_2, \dots, q_6, q_{acc}, q_{rej}\}$

$\Sigma = \{0, 1\}$

$\Gamma = \{0, 1, X, \sqcup\}$

We describe δ with state diagram

Start, accept, and reject states are q_1, q_{acc}, q_{rej} respectively

Enumerators

Infinite tape, finite state control (like TM)
+ a printer (special state)

Tape begins empty (takes no input)

Printer outputs (prints) all of the strings in a language (aka "enumerates" a language) VS. "TM decides" to accept/reject strings based on whether they are in the language

* Can repeat strings, and can print in any order

THM

A language is Turing-recognizable iff some enumerator enumerates it

E → TM

On input w ...

Run E

Compare each printed string to w, if printed, accept

TM → E

Construct E using TM "M" as a subroutine

Run M on all possible strings, if M accepts then print it out

HOWEVER: If we run sequentially, M might loop forever on some string → Must run all in parallel

SOLUTION

Run TM on all strings simultaneously:

List out all possible strings: for example, if language over Σ^*

$$\Sigma^* = \{ \underset{\varepsilon}{\underset{\uparrow}{s_1}}, \underset{0}{\underset{\uparrow}{s_2}}, \underset{1}{\underset{\uparrow}{s_3}}, \underset{00}{\underset{\uparrow}{s_4}}, \dots \}$$

* Computation on any of these could be infinite - must not get stuck on any. Ex, TM may loop on 1, but accept 00

Algorithm

```
For $i = 1, 2, 3 \dots \infty$ (infinite loop)
    For $j = 1$ to $i$
        Run M on $s_j$, only for $i$ steps

        If M accepts $s_j$ in first $i$ steps;
            print $s_j$
    END
END
```

Example :

s1 accepted after 2 steps
s3 accepted after 4 steps

	1	2	3	4	5	6	...
s1		A	A	A	A	A	
s2							
s3				A	A	A	
s4							
s5							
s6							

Print output

```
s1
s1
s1
s3
s1
s3
...
```