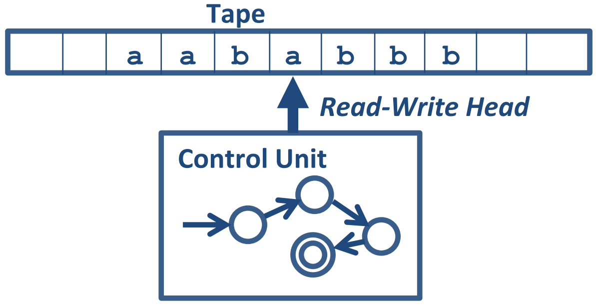
12 - Turing Machines

Let's briefly review the types of machines we've examined so far:

- FA – can only remember via its states, limited to regular languages,
- PDA – can remember via its stack, limited to context-free languages.

A <u>Turing Machine</u> simplifies the idea of the PDA by replacing the stack and the input string with a single mechanism called the **tape**. The control unit can move left and right on the tape, and either read or write on it:



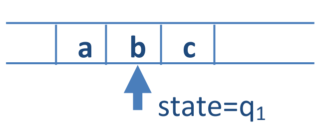
Depending on the symbol read from the tape, and the state the control unit is currently in, the Turing machine does the following:

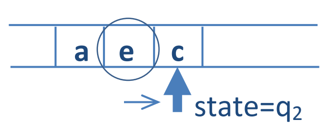
1. Changes state (or stays in the current state),
2. Writes a symbol on the tape, overwriting what was in that cell, and
3. Moves the read/write head LEFT or RIGHT, one cell.

Formally, a Turing Machine is defined by the septuple:

- Q – a finite set of states
- Σ – an input alphabet
- Γ – a tape alphabet, which includes Σ as a subset
- δ – transitions $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$
- q_0 – initial state ϵ Q
- □ – a special "blank" symbol, where $\square \epsilon \Gamma$ and $\square \notin \Sigma$
- F – set of final states $\subseteq$ Q

For example, transition $\delta\,(q_1,b) = (q_2,e,R)$ would mean:

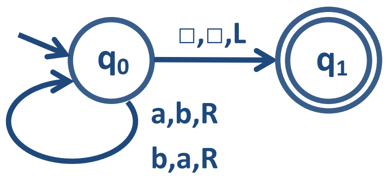




Example 1 – What does the following Turing machine do?

$Q = \{q_0, q_1\}$
$\Sigma = \{a,b\}$
$\Gamma = \{a,b,\square\}$
$F = \{q_1\}$

$\delta(q_0,a) = (q_0,b,R)$
$\delta(q_0,b) = (q_0,a,R)$
$\delta(q_1,\square) = (q_1,\square,L)$

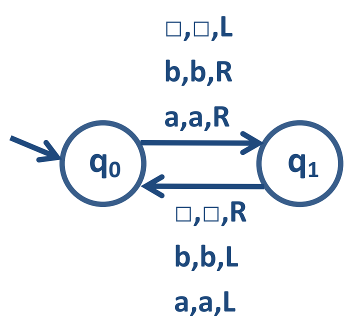


answer: it changes all the b's on the input tape into a's, and all the a's into b's

Example 2 – What does the following Turing machine do?

$Q = \{q_0, q_1\}$
$\Sigma = \{a,b\}$
$\Gamma = \{a,b,\square\}$
$F = \{ \}$

$\delta(q_0,a) = (q_1,a,R)$
$\delta(q_0,b) = (q_1,b,R)$
$\delta(q_0,\square) = (q_1,\square,R)$
$\delta(q_1,a) = (q_0,a,L)$
$\delta(q_1,b) = (q_0,b,L)$
$\delta(q_1,\square) = (q_0,\square,L)$



answer: it oscillates between two characters on the tape.

Instantaneous Description (ID) ⊢

As for PDAs, there is an ID notation for showing a series of transitions in a Turing Machine. For each move, we show the entire tape, with the current control unit state immediately preceding the symbol where the read/write head is positioned. In example #1 above, the moves on string aa would be:

$$q_0aa \vdash bq_0a \vdash bbq_0\square \vdash bq_1b$$

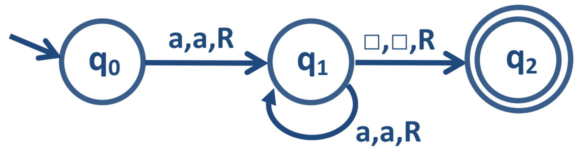
It isn't generally required to show occurrances of the blank symbol (□). However, it is often useful to do so, to make the operation of the Turing Machine more clear. The sequence of moves starting from some initial configuration, until it halts, is called a ***computation***.

Turing Machines as Language Acceptors

Example 3 – Design a Turing Machine that accepts the language L=aa*

$Q = \{q_0, q_1, q_2\}$
$F = \{q_2\}$

$\delta(q_0,a) = (q_1,a,R)$
$\delta(q_1,a) = (q_1,a,R)$
$\delta(q_1,\square) = (q_2,\square,R)$

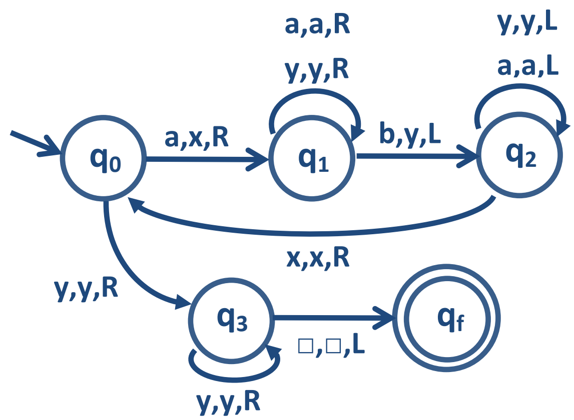


Note that three states are needed to avoid accepting λ

Example 4 – Design a Turing Machine that accepts $L=a^nb^n$

$Q = \{q_0,q_1,q_2,q_3,q_4,q_f\}$
$F = \{q_f\}$

$\delta(q_0,a) = (q_1,x,R)$
$\delta(q_1,a) = (q_1,a,R)$
$\delta(q_1,y) = (q_1,y,R)$
$\delta(q_1,b) = (q_2,y,L)$
$\delta(q_2,y) = (q_2,y,L)$
$\delta(q_2,a) = (q_2,a,L)$
$\delta(q_2,x) = (q_0,x,R)$
$\delta(q_0,y) = (q_3,y,R)$
$\delta(q_3,\square) = (q_f,\square,L)$



Note the strategy is to mark off the leftmost "a" by replacing with "x", then move to the leftmost "b" and replace it with "y". Then, move to the left to the newly leftmost "a" and repeat until there is nothing but x's and y's.

The ID sequence that recognizes the string **aabb** *is as follows:*

$q_0aabb \vdash xq_1abb \vdash xaq_1bb \vdash xq_2ayb \vdash q_2xayb$
$\vdash xq_0ayb \vdash xxq_1yb \vdash xxyq_1b \vdash xxq_2yy \vdash xq_2xyy$
$\vdash xxq_0yy \vdash xxyq_3y \vdash xxyyq_3\square \vdash xxyq_fy$

The ID sequence that rejects the string **abb** *is as follows:*

$q_0abb \vdash xq_1bb \vdash q_2xyb \vdash xq_0yb \vdash xyq_3b$ and halts

The ID sequence that rejects the string **aab** *is as follows:*

$q_0aab \vdash xq_1ab \vdash xaq_1b \vdash xq_2ay \vdash q_2xay$
$\vdash xq_0ay \vdash xxq_1y \vdash xxyq_1\square$ and halts

Turing Machines as Transducers

Turing Machines not only accept languages, they can also compute.

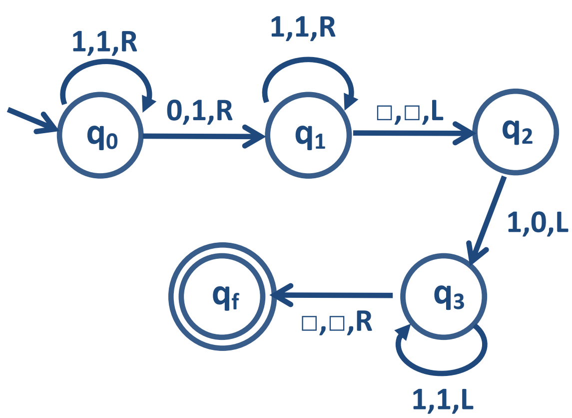
Example 5 – Design a Turing Machine that computes x + y

Approach: Let's limit x and y to positive integers, and use unary notation (each value is a string of 1's) to encode them on the tape, separated by a single 0. We will produce the result on the tape as a series of 1's terminated by a 0, with the head positioned at the left.

Solution: One way is to swap the rightmost "1" with the separating 0. So, starting at x, move to the right until we encounter the 0, replacing it with a 1. Then move to the far right, replacing the final 1 with a 0. Finally, move to the far left, switch to the accept state, and halt:

$Q = \{q_0,q_1,q_2,q_3,q_f\}$
$F = \{q_f\}$

$\delta(q_0,1) = (q_0,1,R)$
$\delta(q_0,0) = (q_1,1,R)$
$\delta(q_1,1) = (q_1,1,R)$
$\delta(q_1,\square) = (q_2,\square,L)$
$\delta(q_2,1) = (q_3,0,L)$
$\delta(q_3,1) = (q_3,1,L)$
$\delta(q_3,\square) = (q_f,\square,R)$



Church-Turing Thesis

<u>Any computation that can be done mechanically can be performed by some Turing Machine</u>. It can't really be proved or disproved without a precise definition of "mechanical". But it is generally accepted, because:

1. Turing Machines can do anything that existing computers can do, and
2. No algorithm has been found that can't be stated as a Turing Machine.

One example of a problem that isn't computable – or more accurately stated isn't *decidable* – is the "Halting Problem": *<u>Given a description of an arbitrary computer program, decide whether the program eventually halts, or whether it will run forever</u>*. Turing showed in 1936 that a Turing Machine cannot be built to solve the halting problem, so it is believed unsolvable.