Turing Machines

Our most powerful model of a computer is the Turing Machine. This is an FA with an infinite tape for storage.

A Turing Machine

A ***Turing Machine*** (TM) has three components:

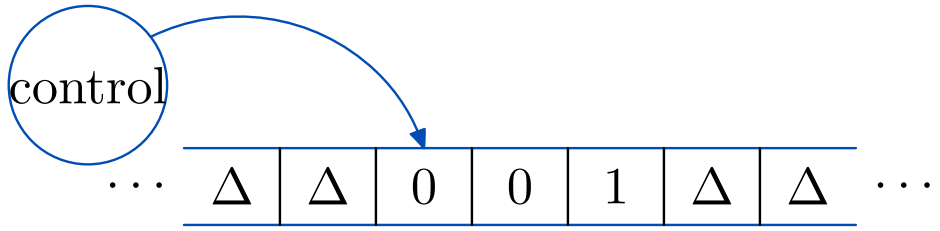
- An ***infinite tape*** divided into cells. Each ***cell*** contains one symbol.
- A ***head*** that accesses one cell at a time, and which can both read from and write on the tape, and can move both left and right.
- A ***memory*** that is in one of a fixed finite number of states.

The TM's Tape

We assume a ***two-way infinite*** tape that stretches to infinity in both directions.

Δ denotes an empty or ***blank*** cell.

The ***input*** starts on the tape surrounded by $\Delta$with the head at left-most symbol. (If input is ε, then tape is empty and head points to empty cell.)



The Program

The ***program*** of a TM is a transition function; depending on symbol under the head and state, the TM:

- writes a symbol,
- moves left or right or stays in place, and
- updates its state.

The ***language*** of TM is set of strings it accepts.

Like the PDA, once a TM enters an accept state it stops, and it terminates abnormally if there is no transition.

The Diagram

The TM is represented as a diagram, like for FA, except that each arrow is labeled with a triple:

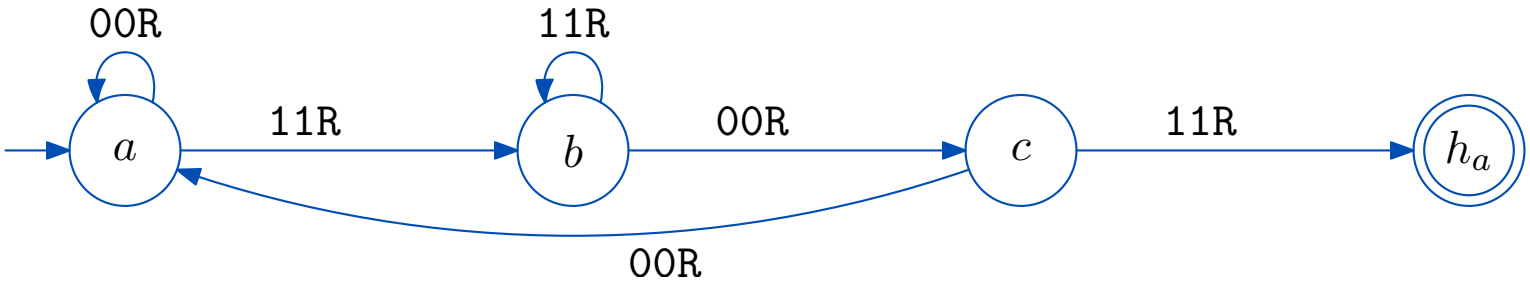
oldSymbol newSymbol moveDir ,

where "moveDir" is one of `L` (move left one cell), `R` (move right one cell), and `S` (stay in place).

For example, triple `01L` means "if reading a `0`, then write a `1` and move the head left."

Example TM: Strings Containing 101

Here is a simple TM that mimics an FA for the language of all binary strings that contain the substring 101.



Formal Definition

One can define a TM as a 7-tuple $(Q, \Sigma, \Gamma, q_0, h_a, h_r, \delta)$ where:

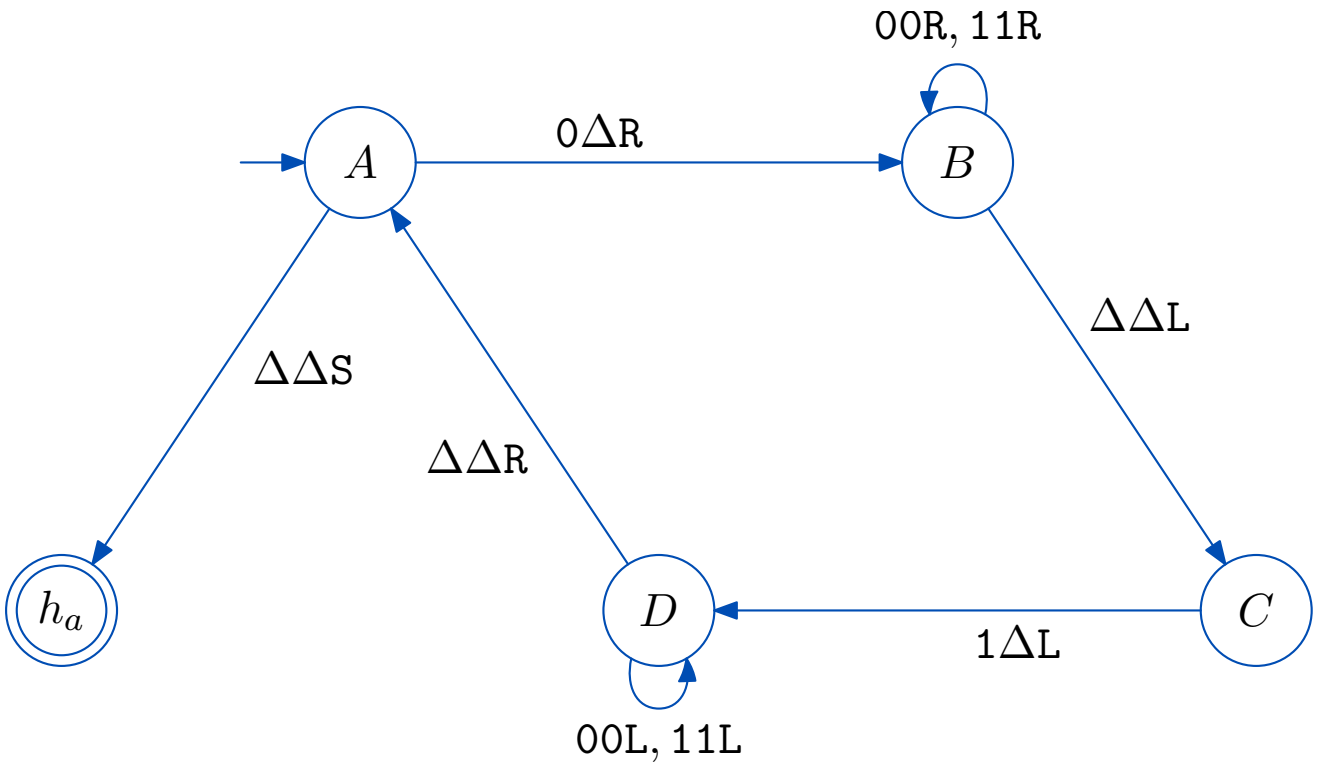
- Q is set of states.
- Σ is input alphabet.
- Γ is tape alphabet (more than Σ).
- q_0 is start state, h_a the unique halt-and-accept state, and h_r the (seldom drawn) unique halt-and-reject state.
- δ is the transition function $Q \times \Gamma \mapsto Q \times \Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\}$.

Example TM: 0^n1^n

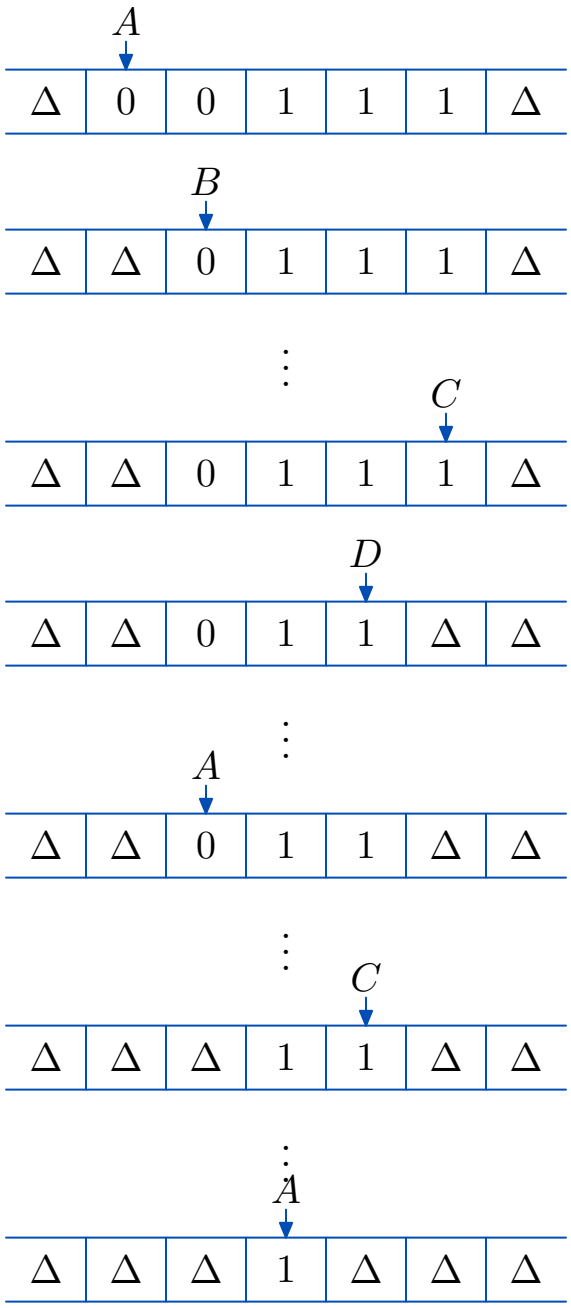
For a TM that accepts $\{ 0^n1^n \}$, pair off the 0's and 1's—repeatedly erase first 0 and last 1 until ε reached. In pseudocode:

(1) If HeadSymbol=0, then Write(Δ) else Reject.
(2) Move head right until HeadSymbol=Δ.
(3) Move head left.
(4) If HeadSymbol=1, then Write(Δ) else Reject.
(5) Move head left until HeadSymbol=Δ.
(6) Move head right.
(7) If HeadSymbol=Δ, then Accept.
(8) Goto (1).

Example Diagram: 0^n1^n



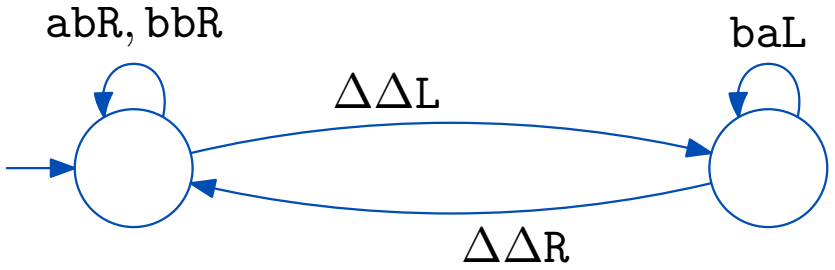
Here is what happens on input 00111...


A
Δ 0 0 1 1 1 Δ
B
Δ Δ 0 1 1 1 Δ
C
Δ Δ 0 1 1 1 Δ
D
Δ Δ 0 1 1 Δ Δ
A
Δ Δ 0 1 1 Δ Δ
C
Δ Δ Δ 1 1 Δ Δ
A
Δ Δ Δ 1 Δ Δ Δ

Reject

TMs might not halt

Here is a particularly unhelpful TM. It does not halt.

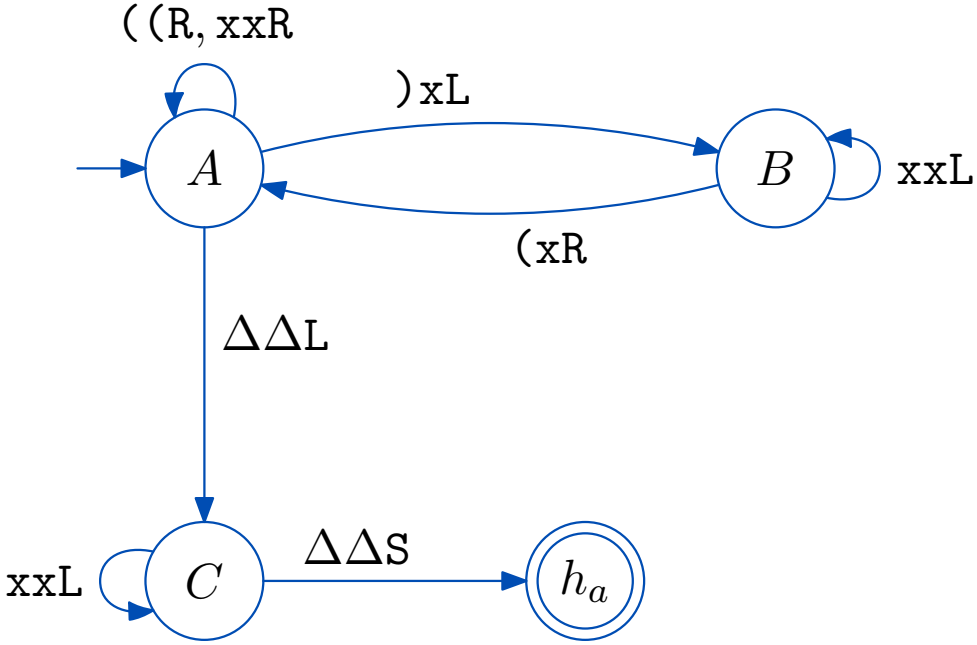


Example TM: Balanced Brackets

For a TM for balanced brackets, one idea is to find the innermost matching pair of brackets, erase them, and repeat the process.

Example TM: Balanced Brackets

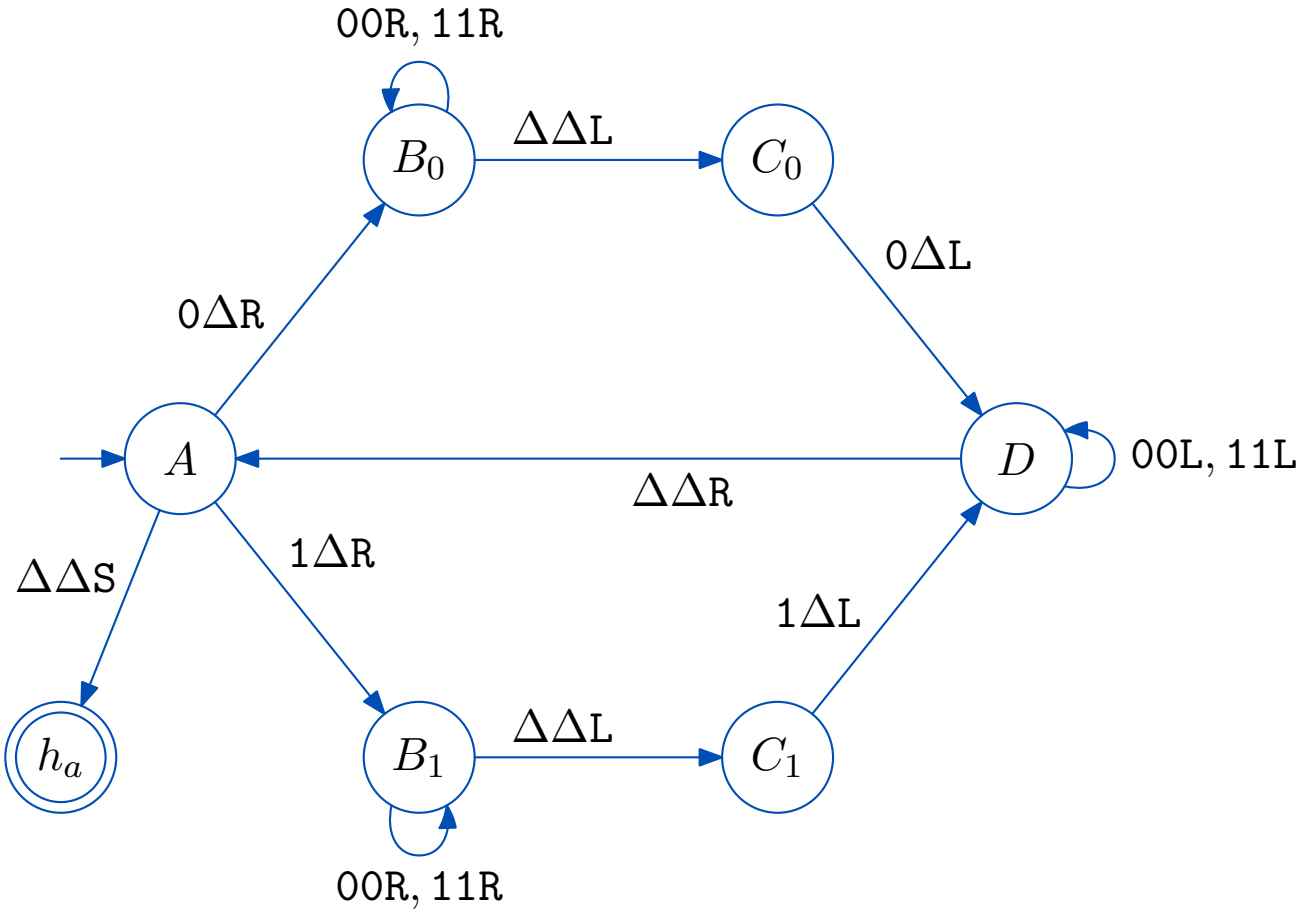
For a TM for balanced brackets, one idea is to find the innermost matching pair of brackets, erase them, and repeat the process. We use `x` to indicate an erased bracket.



Example TM: Palindromes

For even-length palindromes, we match first and last symbols and erase; then repeat. If reach ε without mismatch, then string was palindrome.

Example TM: Even-length Palindromes



Informal TM

We often present TM as pseudocode or English.

Example: A TM that recognizes $\{ w\#w : w \in \Sigma^* \}$.

Informal TM

Example: A TM that recognizes $\{ w\#w : w \in \Sigma^* \}$.

The TM crosses off first entry and remembers it. It then marches right until first entry after the hash mark. It checks that that is correct and crosses that off. The TM returns to left-most uncrossed entry and repeats the process, ignoring the crossed-out symbols. The TM accepts if it manages to cross out all the symbols without encountering a problem.

TM Subroutines

At this stage it looks like a TM is a simple computer perhaps with a limited instruction set. However, we will see that it can perform complex tasks. One aid to this is to design TM subroutines: basic tasks that are useful.

Example Subroutine

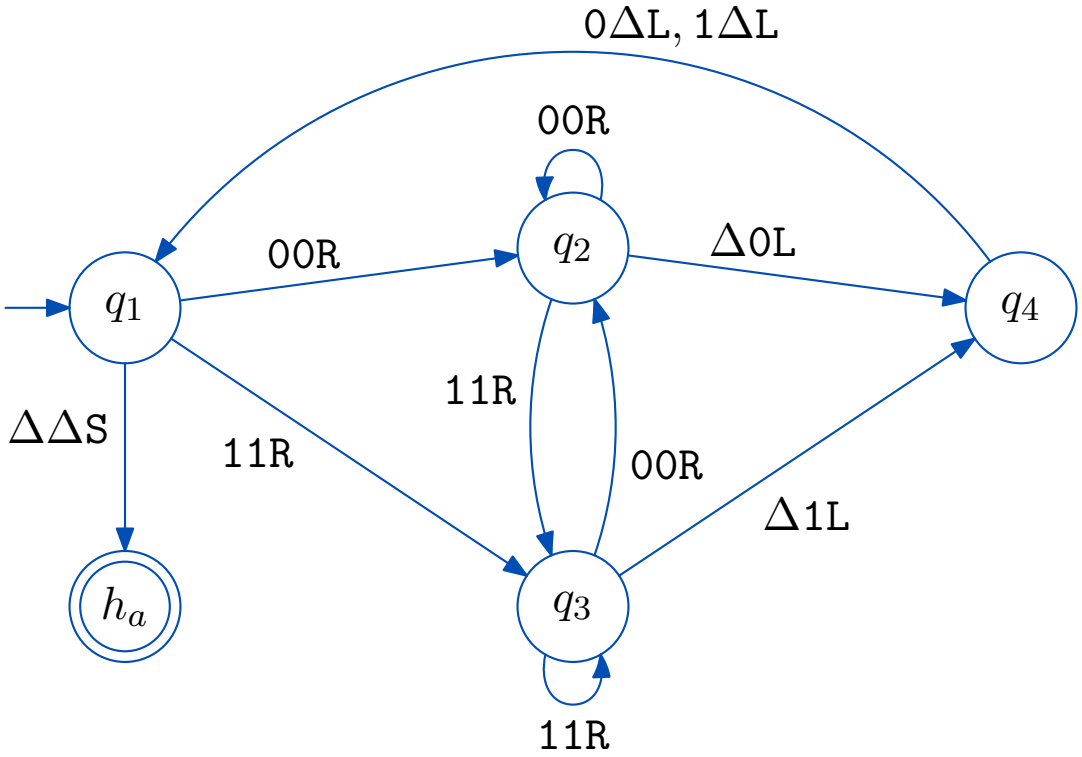
Example. A subroutine that shifts the entire input string one cell to the right.

Example Subroutine

Example. A subroutine that shifts the entire input string one cell to the right.

An inefficient TM could move right-most symbol over, then next one, and so on. That is, it moves to end of string, remembering the last symbol seen as it goes. When it reaches the right-hand end, it writes down the remembered symbol, then backs up and erases the symbol. Then repeats. . .

TM for Shifting Right



TMs That Do Not Halt

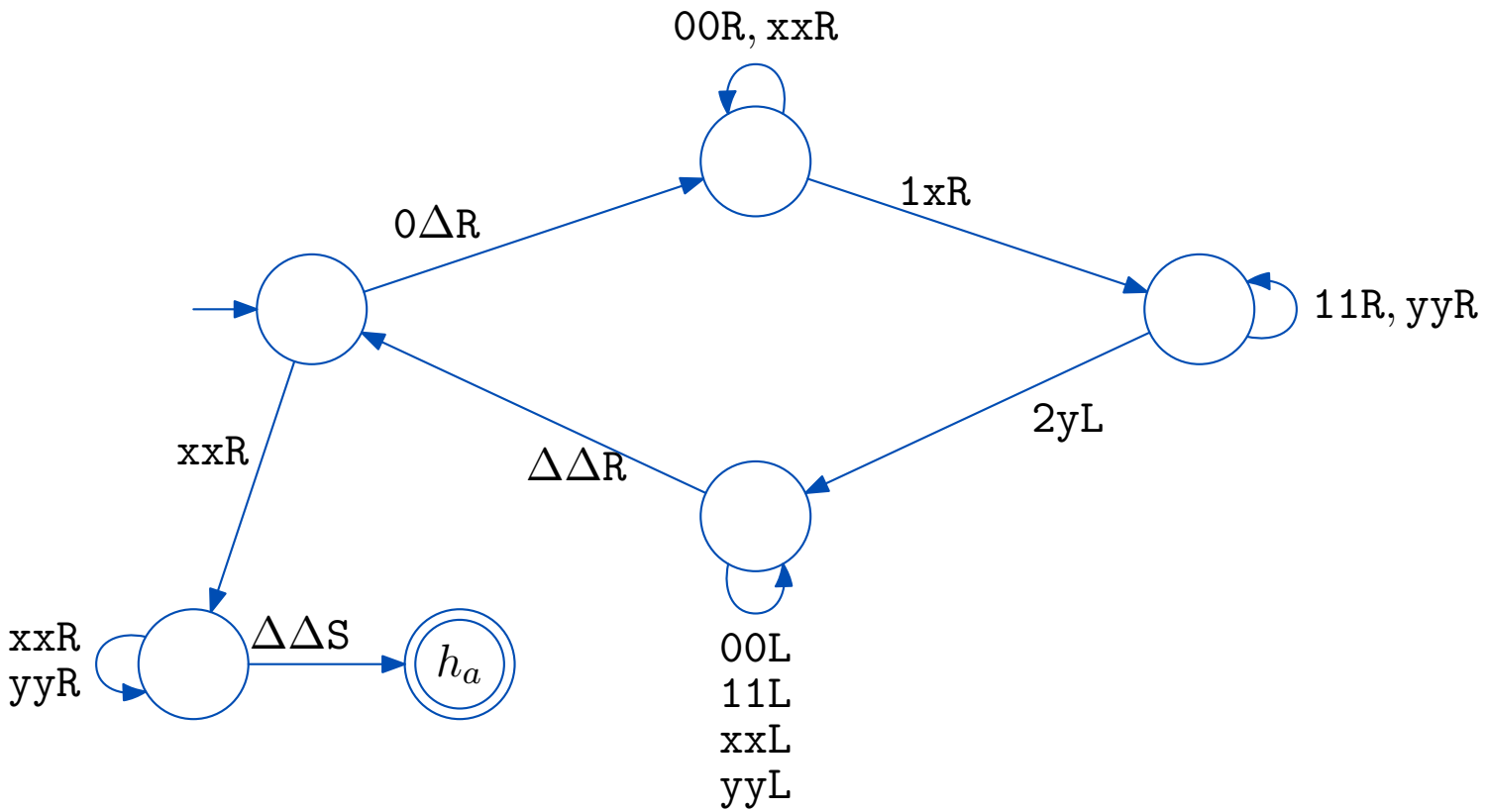
A TM does not necessarily halt. (And if not, it is not necessarily stuck in a loop.) And, we might not know beforehand whether it will halt.

Consider, for example, a TM that tries to find a counterexample to Goldbach's conjecture that every even number at least 4 is sum of two primes. The TM tries every value of even n in increasing order. For each n, it checks whether there is i such that both i and $n - i$ are prime. If not, it stops. Otherwise it continues forever. So we have built a TM that we humans don't know whether halts.

Practice

Draw the diagram for a TM that accepts the language $\{0^n 1^n 2^n\}$.

Solution to Practice



Summary

A Turing Machine (TM) is like an FA, but it has an infinite tape. The input starts on the tape surrounded by blank cells denoted Δ. The program of a TM is represented as a diagram: depending on the symbol under the head and the state, the machine writes a symbol, moves left or right or stays in place, and/or changes state. Once a TM enters the accept state it stops.