Turing Machines

Key Definition (Turing Machine): A Turing Machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, where:

1. Q is a **finite** set of states,
2. Σ is a **finite** set of input symbols not including $\sqcup$,
3. Γ is a **finite** set of tape symbols including $\sqcup$ and all of Σ,
4. δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times$ {L,R} is the transition function,
5. q_0 is the start state,
6. q_a is the accept state, and
7. q_r is the reject state ($q_a \neq q_r$).

The intuition is that a Turing Machine is an automata that, similar to a PDA, has access to some memory. Rather than a stack, TMs use an infinite tape and the automata can read, write, and move left and right on the tape. This model of computation is the closest formal model we have to computers today.

The **Church-Turing Thesis** says that our intuitive notion of algorithms equals the Turing Machine model of algorithms.

Key Definition (Turing-recognizable): a language is called Turing-recognizable if some Turing Machine recognizes it. These are also referred to as the **recursively enumerable** languages.

Key Definition (Turing-decidable): a language is called Turing-decidable (or just decidable) if some Turing Machine decides it. These are also referred to as the **recursive** languages.

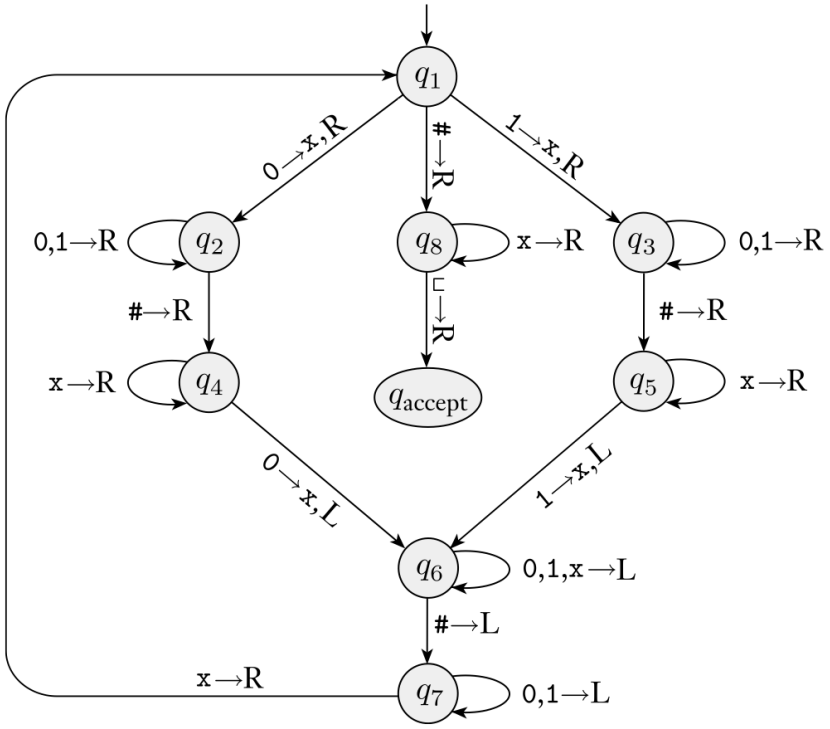
Intuition: Unlike regular languages and context-free languages, which are always decidable (they always halt and return an answer, accept or reject), Turing machines can run indefinitely and may never halt. If a language is recognizable, then a recognizing Turing machine will halt and accept; but if it is NOT recognizable, then the Turing machine will either reject, or it will answer nothing at all (run for ever). Decidable languages, by contrast, always halt and return an answer (accept or reject). Every decidable language is Turing-recognizable.

Example 1. We know that language **{w#w | w $\in${0,1}*}** is not regular and not context-free. It is, however, Turing-decidable. Here's how the Turing Machine operates:

M_1 = "On input string w:

1. Scan left to right, looking for a single #. If none, reject. If more than one, reject. (Simple states can easily keep track.) Reset head back to beginning.
2. Read the next 0 or 1 symbol, and scan right, past the # for a matching symbol. If no match, reject. As they are read, replace the symbol with an x to mark it off.
3. Reset head past # to last x symbol.
4. Repeat step 2-3 above until # is reached. If right side has fewer symbols at any time, reject. If right side still has symbols after # is reached, reject. If all checks off and the next symbol is $\sqcup$, accept."

The state diagram for M_1 would look like so:



Example 2. The language **{**$0^{2^n} | n \geq 0$ **}** is not regular and not context-free. Yet, it is Turing-decidable. Here is the TM:

M_2 = "On input string w: ... [details left to you]"

The gist of this machine will be that it sweeps left to right, marking-off every other 0. It resets and repeats this process until no more zeros are left. Each time reducing by half, means powers of two... (watch out for cases that fail and reject). Details left to you.

Example 3. Is **{**$a^n b^n c^n | n \geq 0$ **}** Turing-decidable? Sure – The algorithm is not much different than Example 1 above! Details are left to you. **This would make a great final exam question!**