Turing Machines

A PDA has a stack, which is a type of memory that can be written and read only at its end.

A Turing Machine (TM) is obtained by giving the automaton unrestricted access to memory.

Memory is organized as a tape over a finite alphabet Γ. The TM has a tape head that indicates a position on the tape. The TM can see what is on this position, and write to it. It can move the tape head left and right on the tape without restriction.

It will turn out that these automata are as powerful as any model of computation that has been invented.

Turing Machines

A Turing Machine M has form $(Q, \Sigma, \Gamma, \sqcup, \delta, q_s, F)$, where

- Q is a finite set of states.
- Σ is the input alphabet.
- Γ is the tape alphabet. We assume that $\Sigma \subset \Gamma$.
- $\sqcup \in \Gamma \backslash \Sigma$ is a special space (or blank) symbol.
- δ is the transition function, which has form

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}.$$

- $q_s \in Q$ is the starting state.
- $F \subseteq Q$ are the accepting states.

Explanation

In the beginning, the input word is on the tape. We assume that the tape is infinite. Left and right of the input word are infinitely many spaces ($\sqcup$).

The tape head starts at the position of the first letter of the input word, or a $\sqcup$ when the input word is empty.

The TM starts in state q_s. At each moment in time, the TM looks at its state, and the symbol at the head of the type. The transition function δ specifies

1. the next state
2. what should be written on the current position of the tape head.
3. in which direction the tape head should move: L means to the left, R means to the right, S means stay on the same place.

Turing Machines are introduced in Chapter 3 in Sipser.

Our definition is a bit different, because we want to be the same as in: `https://turingmachinesimulator.com/`

The differences are:

- We have no rejecting state. Input is rejected when the Turing machine gets stuck.
- We allow the tape head to stay on the same place (S).
- We assume infinite tape to the left of the input word.

Recognizing non-context free languages

Consider the language $\mathcal{L} = \{\ w^i \# w^i \mid i \geq 0, w \in \{a, b\}^*\ \}$. Is it context-free?

It can be recognized by a Turing Machine.

What about $\mathcal{L} = \{a^i b^i c^i \mid i \geq 0\}$?

Is it context free?

Can you define a Turing Machine for it?

Church-Turing Thesis

We have now seen Turing Machines.

If you want to use a programming language, it must be big. (Good type system, Powerful built-in Operations.)

If you want to explore borders of computability, it is better to use a small computational model.

Turing Machines are among the smallest models that we could invent. (Others are Tiling Problems, Recursive Functions, or Lambda Calculus.)

Turing thesis: All computational models are equivalent (can compute the same functions).

Concretely: One could build a compiler from Java (or C^{++}) into Turing Machines.