DISCLAIMER: These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

- Quiz 8 return.
- Definition of Turing machines and another example.
- About the next 3 quizzes:
  1. Quiz 8 average: 0.99, 58 2s.
  2. Will ensure to have questions that are solvable from definitions that have been repeatedly covered. Won’t require knowing particular theorems / tools.
- Variants of Turing Machines:
  1. Multi-Tape.
  2. Non-Deterministic.

Last time we introduced Turing machines, which are finite automatas augmented with tapes, instead of stacks or read-only input.

Formally, a Turing machine (TM) is a 7-tuple consisting of states $Q$, alphabet $\Sigma$, tape alphabet $\Gamma$, transition $\delta$, and starting/accept/reject states $q_0$, $q_{\text{accept}}$ and $q_{\text{reject}}$. Its transitions have the form:

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

Specifically

$$\delta(q, a) = (r, b, L)$$

means that if the machine is at state $q$ and it reads $a$ off the tape, then it writes the symbol $q$, goes to state $r$, and moves leftwards by one on the tape.

There is also a blank symbol, $\_\_$ that denotes the rest of the tape that does not contain the input. This is how we can assume we’re dealing with an infinite tape, and that we can always tell when we ‘went off’ of the input.

Furthermore, we say that a configuration $C$ yields a configuration $\hat{C}$ if the transition takes us from $C$ to $\hat{C}$.

We will use the language of $M$, $L(M)$, to denote the set of strings accepted by $M$. However, because Turing machines may no longer halt, it has one more possible outcome, loop. So we also have the notion of a TM that never loops, known as a decider. We say that a language $L$ is Turing-decidable if there is a Turing machine that accepts $L$, and never infinite loops.
1 Example of a Turing Machine

We will finish with a formal definition of a Turing machine that decides the language on the alphabet $\Sigma = \{0\}$

$$L = \{0^{2^n} \mid n \geq 0\}.$$

The main idea is to repeatedly sweep left to right from the tape, and cross out any 0s. Except if the tape contains a single 0, give accept. However, if the tape consists of an odd number of 0s, give reject.

Formalizing this Turing machine requires four key components:

1. A component that skips every $x$ encountered.
2. A DFA that remembers whether an odd, or even number of 0s have been encountered.
3. Distinguishing in this DFA between the first 0 and every subsequent 0s.
4. Rewinding the tape after erasing every other 0 from it.

Once again we use $x$ to denote a stack symbol for a 0 that has been erased, and implement these operations by:

1. a starting state $q_1$ that moves to reject if it encounters $x$ or $\omega$.
2. It moves to a ‘regular start’ state $q_2$ after replacing the first 0 with a $\omega$, thus removing the leftmost character.
3. Then it keeps on moving to the right as long as it sees $x$s, until it sees a 0, at which point it goes to $q_3$.
4. Then it needs loop and turn every other 0 into a $x$. This is done by bouncing between $q_3$ and $q_4$.
5. If the right end of the string, denoted by a $\omega$ is encountered, then the machine moves to reject.
6. Otherwise, the machine rewinds the tape via state $q_5$ that only moves leftwards, until it encountered $\omega$ denoting the left end of the tape, at which point it goes back to $q_2$.

2 Variants of Turing Machines

Turing machines are some ways the most powerful computing devices. In that as long as we are not worried about the number of operations taken, they can simulate most other forms of communication. In fact, not having to worry about the cost of moving one at a time on a tape is a major reason for introducing the tape as a simplification to random access memory.
2.1 Multi-Tape

A simple augmentation is what happens if we have two, or $k$, tapes. This leads to transitions of the form

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k.$$ 

Here $S$ means being stationary.

To fit everything onto one tape, we place some division symbols $\#$, one per tape, and use another symbol ($\cdot$) as a virtual tapehead.

We then create a finite state machine with enough states to store $\Gamma^k$, and simulates one step of the multi-tape TM by:

1. Scanning through the tape once to find all the tapehead symbols.
2. Goto a state corresponding to the action (new state, new symbols to write, and new movements).
3. Scans through the entire tape again to update the states of each of the tapes.
4. Moves the tape content around so that there are empty spaces behind each tape.

This is basically the Turing machine equivalent of a memory manager. However, note that the DFA is augmented sufficiently to store $\Gamma^k$, which in some sense is not cheating since the original transition function $\delta$ also stores these info.

2.2 Universal Turing Machine

Conceptually the most interesting is the notion of a universal Turing machine: that is, a machine that reads as input a Turing machine $M$ and its input, $w$ and simulates the result of running $M$ on $w$. We will spend most of next week discussing the applications of this idea. For now, just imagine a multi-tape machine containing:

1. $w$.
2. the transition function of $M$.
3. the current state that $M$ is in.
4. the rest of the description of $M$, aka. $q_{accept}$ and $q_{reject}$.

Then the first tape is basically the tape state of $M$, with the tape head at the current character too.

Then at that character, we go lookup in the transition function the entry corresponding to that symbol and the current state. Based on this, we move the first tape, and update the state stored on the third tape.

If there is time, I will also briefly mention the notion of an enumerator: a Turing machine that outputs every string in a language, and how that is equivalent to a language having a Turing machine recognizing it.