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.

- Test 2 / Quiz 7 return.

- Possible presentation topics (aiming for 15 minute presentations):
  1. Deterministic CFGs.
  2. Probabilistic automata.
  3. Circuit complexity.
  5. Information theory, entropy, and compression.

- Turing machines: automata with a tape.

- Quiz 8.

We now present a model of computation that’s as powerful as general computers. Recall that a finite automata reads symbols of an input, and moves between its intermediate states.

A Turing machine differs in that:

1. The tape head can move both left and right, while an automata only advances through the tape once.

2. It can both read and write on the tape, while an automata reads.

3. The tape is infinite, while an automata only has the input on the tape.

4. The accept/reject states take effect immediately, since there is no longer a notion of ‘consuming the input. In fact, we will see that checking whether a Turing machine ever finishes is quite difficult!
To see that such a machine is significantly more powerful than a finite automata / PDA, we will give a way to accept language of squares, 

\[ L = \{ w\#w \mid w \in \{0, 1\}^* \} \].

Recall that this language is neither regular nor context free.

We will discuss this formally, but due to the number of moving pieces in a Turing machine, it’s almost imperative to think about it intuitively.

1. We will have a symbol for denoting ‘we’ve already compared these characters’, \( x \).
2. Then we look at the first symbol, remember it (recall that a finite automata has constant memory, so can remember individual characters).
3. Then we mark this symbol with a \( x \), and advance until we see \#.
4. Then we advance past \# until the first symbol that’s not a \( x \).
5. We compare these two symbols: if they are different, we reject.
6. Otherwise these two do match, so we set the other symbol to \( x \) as well.
7. Then we ‘rewind’ the tape until the start, at which point we look for the first non \( x \) symbol again, that’s the next unchecked character.
8. When no more characters are left to check, we will see the \# symbol, which only appears once. At which point we verify that the rest of the input is entirely \( x \) (symbols that we’ve already checked), and quit.

We will do an example of this on the input

\[ 0011\#0011 \]

and there is another example of this in the text book.

1 Formal Definition of Turing Machine

A deterministic Turing machine differs from a DFA/NFA/PDA mostly its transition, which has 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.

Note that we do not have a ‘stay still’ command because the next state, \( r \), could:
1. keep whatever symbol on the tape there, and
2. move right by 1.

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 $q_0$, $q_{accept}$ and $q_{reject}$.

There is also a blank symbol, $\omega$ 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.

A configuration of a Turing machine then consists of

1. The current state.
2. The current tape contents.
3. The current head location.

Because the states are different than the tape symbols, we will use the shorthand $uqv$ for the configuration where the current state is $q$, the tape content is $uv$, and the head of the tape is the first symbol of $v$. This means:

1. The start configuration is $q_0w$.
2. Any configuration of the form $uq_{accept}v$ is an accepting configuration.
3. Any configuration of the form $uq_{reject}v$ is a rejecting configuration.

Accepting and rejecting configurations are known as halting configurations.

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.

## 2. 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 $\_$

2. It moves to a ‘regular start’ state $q_2$ after replacing the first 0 with a $\_$, 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 $\_$ 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 $\_\_$ denoting the left end of the tape, at which point it goes back to $q_2$. 