CS 4510 Automata and Complexity	3/17/2022
Lecture 13	
Lecturer: Frederic Faulkner	*Scribe(s):*

1 Unbounded memory

Recall from the beginning of the semester, that our goal in this class is to discuss the power and limitations of modern computing. To do so, we need some mathematical model to stand in for a computer. However, all of the models we have discussed so far (DFA's, NFA's, Regex, Context-Free Grammars, PDA's) are severely limited in the tasks they can perform and so are poor candidates for our ultimate choice of model.

For this reason, today we will introduce a new model of computing called the "Turing machine" (TM), invented by Alan Turing in the 1930's. We saw with pushdown automata, that adding memory to a DFA in the form of a stack greatly increases its power. With a Turing machine, we take this a step further and add effectively unbounded, unrestricted memory to a DFA in the form of a *tape*.

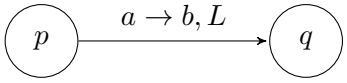
1.1 Performing calculations on a tape

A TM tape is a 1-dimensional line of blank cells stretching infinitely left and right. Each cell can hold a single character of the alphabet, and can be written to or read from. The TM uses this tape much the same way that you use scratch paper, performing calculations until it reaches its conclusion.

The TM *head* is a pointer pointing at a particular cell. This is the cell the TM is currently "looking at". At each step in a computation, the machine can overwrite the character currently under the head, and then move the head left or right one square.

The head of a TM is controlled by a DFA. (So a TM is effectively a DFA + a tape + the tape head.) However, unlike normal DFA's, there is exactly one instantaneous accept state and one instantaneous reject state. When the machine enters one of these states, the computation immediately ceases and the machine accepts or rejects the string as appropriate. If the TM is not in one of those two states, the computation continues.

A transition in a TM looks as follows:



This means "While in state p if there is a a under the tape head, replace the a by a b and move the tape head one square to the left." (Use R instead of L to move the head right.)

1.2 An example

Recall that checking two words for equality, i.e. the language $\{w\#w \mid w \in \Sigma^*\}$, is too hard for both DFA's and context-free grammars. We thus demonstrate that TM's are more powerful than both of these models by explaining how a TM would recognize this language.

1. Start by scanning the string from left to right to confirm that there is exactly one #.
2. Mark the first character of the string. (I.e after the previous step you must move the head back all the way to the left.)
3. Move to the right past the # until you come to an unmarked character. If this character is different than the one you marked in the previous step, reject. Else, mark this character, and move to the left past the #.
4. Move to the left until you get to the leftmost unmarked character. (I.e. move left until you see a marked character then move one step right.) If there were no remaining unmarked characters, go to step 5. Otherwise, return to step 2.
5. Scan to the right to check that there are no unmarked characters. If there are, reject; otherwise accept.

Let's examine the actions of this machine on the input 011#010.

- $\dot{0}11\#010$ After checking for the #, the machine marks the first character.
- $\dot{0}11\#\dot{0}10$ It moves right past the # and until it sees an unmarked character. In this case, the first unmarked character right of the # is a 0. Since this matches the previously marked character, it marks it and moves back left.
- $\dot{0}\dot{1}1\#\dot{0}10$ It continues zigzagging back and forth check for matches.
- $\dot{0}\dot{1}1\#\dot{0}\dot{1}0$
- $\dot{0}\dot{1}\dot{1}\#\dot{0}\dot{1}0$
- However, when it scans to that rightmost 0, it doesn't match the previous marked 1, so the string is rejected, as it should be.

Note that we did not give a formal "circle-and-arrow" diagram for this Turing machine. If you would like to see one, there is a diagram for this exact machine in the book. However, creating such a drawing quickly becomes tedious for even the simplest of Turing machines. Thus, we (and you) will typically use higher level language to describe the algorithm followed by the machine.

1.3 Getting formal

Definition 1 *A Turing machine is a 7-tuple* $(Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}, \delta)$ *where*

- Q *is the set of states*
- Σ *is the input alphabet. It doesn't contain* $\sqcup$ *(the empty cell or blank character).*
- Γ *is the tape alphabet. It automatically contains* $\sqcup$ *and every character in* Σ*, but it can also contain any other characters you need.*
- q_0 *is the start state*

- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ *is the transition function.*

Definition 2 *A Turing machine M accepts a string $x \in \Sigma^*$ if the machine reaches an accept state after being started with x on the tape and the head pointing to the leftmost character of x.*

Some notes:

- As defined, a Turing machine must write a character at each step. If you want to leave the current character unchanged, just write it. (I.e. if you are looking at a 0 and you "overwrite" it with a 0, the string effectively hasn't changed.)
- As defined, a Turing machine must move left or right exactly one square each step. If you want to stay in the same spot, move the head right and go to a special state, then move the head back to the left and go back to the state you would have gone to.
- There is no mention of marking a character. Really when we say "mark" a character, we just mean that if our input alphabet $\Sigma = \{0, 1\}$, then $\Gamma = \{0, 1, \dot{0}, \dot{1}, \textvisiblespace\}$. To the machine itself, it isn't "marking" anything, it is just overwriting a 0 with a $\dot{0}$, etc.

1.4 Another Example

Let $L = \{0^i 1^j 2^k \mid i * j = k\}$. This set contains strings like 012, 00112222, 00011222222, etc. We give a Turing machine that accepts this language.

1. Scan the string to make sure it has the correct format (0's followed by 1's followed by 2's).
2. Mark an unmarked 0. If you could not, go to step 5.
3. Zigzag to mark all of the 1's and an equal number of 2's. If you run out of 2's, reject.
4. Unmark all the 1's and go to step 2.
5. Check if there are any unmarked 2's. If there are, reject; otherwise accept.

And here is the machine processing 001122222:

- $\dot{0}01122222$
- $\dot{0}0\dot{1}\dot{1}2\dot{2}2222$
- $\dot{0}011\dot{2}\dot{2}222$
- $\dot{0}\dot{0}11\dot{2}\dot{2}222$
- $\dot{0}\dot{0}\dot{1}\dot{1}2\dot{2}\dot{2}\dot{2}2$
- $\dot{0}\dot{0}11\dot{2}\dot{2}\dot{2}\dot{2}$

At this point, there are no remaining unmarked 0's, but there are unmarked 2's, so this string is rejected.