Automata Theory, Languages, and Computation

Name: _____ Date: _____

Note: The purpose of the following questions is:

• *Enhance learning*	• *Summarized points*	• *Analyze abstract ideas*

Class 15: Turing Machines

So far we have encountered some fundamental ideas, in particular the concepts of regular and context-free languages and their association with finite automata and pushdown accepters. Our study has revealed that the regular languages form a proper subset of the context-free languages, and therefore, that pushdown automata are more powerful than finite automata. We also saw that context-free languages, while fundamental to the study of programming languages, are limited in scope.This was made clear in the last class, where our results showed that some simple languages, such as $\{a^n b^n c^n\}$ and $\{ww\}$, are not context-free. This prompts us to look beyond context-free Languages and investigate how one might define new Language families that include these examples. To do so, we return to the general picture of an automaton. If we compare finite automata with pushdown automata, we see that the nature of the temporary storage creates the difference between them. If there is no storage, we have a finite automaton; if the storage is a stack, we have the more powerful pushdown automaton. Extrapolating from this observation, we can expect to discover even more powerful language families if we give the automaton more flexible storage. For example, what would happen if, in the general scheme of class 0, we used two stacks, three stacks, a queue, or some other storage device? Does each storage device define a new kind of automaton and through it a new langrage family? This approach raises a large number of questions, most of which turn out to be uninteresting. It is more instructive to ask a more ambitious question and consider how far the concept of an automaton can be pushed. What can we say about the most powerful of automata and the limits of computation? This leads to the fundamental concepts of a Turing machine and, in turn, to a precise definition of the idea of a mechanical or algorithmic computation.

1. What can we say about the most powerful of automata and the limits of computation?
2. What are the fundamental concepts of Turing machine?

The Standard Turing Machine: Turing machine storage is actually quite simple. It can be visualized as a single, one-dimensional array of cells, each of which can hold single symbol. This array extends indefinitely in both directions and is therefore capable of holding unlimited amount of information. This information can be read and changed in any order. We will call such storage device a tape because it is analogous to the magnetic tapes used in older computers.

Definition of a Turing Machine: A Turing machine is an automaton whose temporary storage is a tape. This tape is divided into cells, each of which is capable of holding one symbol. Associated with the tape is read-write head that can travel right or left on the tape and can read and write a single symbol on each move. To deviate slightly from the general scheme in Class 0, the automaton that we can use as a Turing machine will have neither an input file nor any special output mechanism. Whatever input and output is necessary will be done on the machine's tape. A diagram giving an intuitive visualization of a Turing machine is shown in slide 4.

3. Construct Turing machine that will accept the Language $L = L(aa^*)$. Show the action of the machine for accepting the string aaa. Show the action of the machine for rejecting the string aba.
4. Construct Turing machine that will accept the Language $L = L(aa^* + b(a+b)^*)$.
5. For $\sum$={a, b}, design a Turing machine that accepts $L = \{a^n b^n : n \geq 1\}$
6. Design a Turing machine that accepts $L = \{a^n b^n C^n : n \geq 1\}$

Turing Machines as Language Accepters: Turing machines can be viewed as accepters in the following sense. A string w is written on the tape, with blanks filling out the unused portions. The machine is started in the initial state q_0 with read-write head positioned on the leftmost symbol of w. If, after a sequence of moves, the Turing machine enters a final state and halts, then w is considered to be accepted.

Formal Definition of Turing Machines:

Computing Functions with Turing Machines

7. Given two positive integers x and y, design a Turing machine that computes $x+y$.
8. Design a Turing Machine that copies strings of 1's. More precisely, find a machine that performs the computation

$$q_0 \vdash^* q_f ww$$

Combining Turing Machines for Complicated Tasks

In the previous examples we have shown explicitly how some important operations found in all computers can be done on a Turing machine. Since in digital computers, such primitive operations are the building blocks for more complex instructions. To demonstrate how Turing machines can be combined, we follow a practice common in programming. We start with high-level description, and then refine it successively until the program is in the actual language with which we are working. We can describe Turing machines several ways at a high level; block diagrams or pseudocode are the two approaches we will use most frequently. In a block diagram, we encapsulate computations in boxes whose function is described, but whose interior details are not shown. By using such boxes, we implicitly claim that they can actually be constructed.

9. Design a Turing machine that computes the function.

$$f(x,y) = x+y \quad \text{if } x \geq y,$$
$$= 0 \quad \text{if } x < y.$$