Automata Theory, Languages, and Computation

Name: _____ Date: _____

Note: The purpose of the following questions is:

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

Class 16: Turing Thesis

1. [Slide 2] What is Turing's thesis?

Turing's thesis is a combined hypothesis ("thesis") about the nature of functions whose values are effectively calculable; or, in more modern terms, functions whose values are algorithmically computable. In simple terms, the Turing thesis states that a function is algorithmically computable if and only if it is computable by a Turing machine.

Some arguments for accepting the Turing thesis as the definition of a mechanical computation are:

1. Anything that can be done or any existing digital computer can also be done by a Turing machine.
2. No one has yet been able to suggest a problem, solvable by what we intuitively consider an algorithm, for which a Turing machine program cannot be written.
3. Alternative models have been proposed for mechanical computation,, but none of them is more powerful than the Turing machine model,

In some sense, Turing's thesis plays the same role in computer science as do the basis laws of physics and chemistry. Classical physics, for example, is based largely on Newton's laws of motion.

2. [Slide 7] Define the standard Turing machine.
3. [Slide 8] What are the variations of the Standard model of Turing machine?

Other Models of Turing Machines

Our definition of standard Turing machine is not the only possible one; thee are alternative definitions that could serve equally well. The conclusions we can draw about the power of Turing machine are largely independent of the specific structure chosen for it. In this class we look at several variations, showing that the standard Turing machine is equivalent, in a sense we will define, to others, more complicated models.

If we accept Turing's thesis, we expect that complicating the standard Turing machine by giving it a more complex storage device will not have any effect on the power of the automaton. Any computation that can be performed on such a new arrangement will still fall under the category of a mechanical computation and, therefore can be done by a standard model. It is nevertheless instructive to study more complex models, if for no other reason than that an explicit demonstration of the expected result will demonstrate the power of the Turing machine and thereby increase our confidence in Turing's thesis. Many variations on the basic model [class 15, slide 55] are possible. For example, we can consider Turing machines with more than one tape or with tapes that extend in several dimensions.

We also look at nondeterministic Turing machine and show that they are no more powerful than deterministic ones. This is unexpected, since Turing's thesis covers only mechanical computations and does not address the clever guessing implicit in nondeterminism. Another issue that is not immediately resolved by Turing's thesis is that of one machine executing different programs at different times. This leads to the idea of a "reprogrammable" or "universal" Turing machine.

4. [Slide 9] What do we mean by an essential difference between one class of automata and another?

Minor Variations on Turing Machine Theme

Whenever we change a definition, we introduce a new type of automata and raise the question whether these new automata are in any real sense different from those we have already encountered. Although there may be clear differences in their definitions, these differences may not have any interesting consequences. We have seen an example of this in the case of deterministic and nondeterministic finite automata. These have quite different definitions, but they are equivalent in the sense that they both are identified exactly with the family of regular languages. Extrapolating from this, we can define equivalence of nonequivalence for classes of automata in general.

5. [Slide 10] Define equivalence or nonequivalence for classes of automata in general?
6. [Slide 12] Show how the simulation technique is used for demonstrating the equivalence in Turing's machine.

Turing Machines with a Stay-Option

In our definition of a standard Turing machine, the read-write head must move either to the right or to the left. Sometimes it is convenient to provide a third option, to have the read-write head stay in place after rewriting the cell content. Thus, we can define a Turing machine with a stay-option by replacing δ in [class 15, slide 55] by

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

with the interpretation that δ signifies no movement of the read-write head. This option does not extend the power of the automaton.

7. [Slide 17] Prove that, the class of Turing machines with stay-option is equivalent to the class of standard Turing machines.

Standard Machine – Multiple Track Tape: In the picture [slide 23] we have divided each cell of the tape into three parts, called **tracks**, each containing one member of the triplet. Based on this visualization, such an automaton is sometimes called a Turing machine with **multiple tracks**, but such view in no way extends the definition in [class 15, slide 55], since all we need to do is make Γ an alphabet in which each symbol is composed of several parts.

However, other Turing machine models involve a change of definition, so the equivalence with the standard machine has to be demonstrated. Here we look at such models, which are sometimes used as the standard definition.

Turing Machines with Semi-Infinite Tape [slide 25-35] we can visualize this as a tape that has a left boundary [slide 27]. This Turing machine is otherwise identical to our standard model, except that no left move is permitted when the read-write head is at the boundary.

It is not difficult to see that this restriction does not affect the power of the machine. To simulate a standard Turing machine M_1 by a machine M_2 with a semi-infinite tape, we use arrangement shown in slide 28.

8. [Slide 29] Prove that, the class of Turing machines with Semi-Infinite Tape simulates standard Turing machines.

The simulating machine M_2 has a tape with two tracks. On the upper one, we keep the information to the right of some reference point on M's tape. The reference point could be, for example, the position of the read-write head at the start of the computation. The lower track contains the left part of M_2 tape in reverse order. M_2 is programmed so that it will use the information on the upper track only as long as M_2 read-write head is to the right of the reference point, and work on the lower track as M_1 moves into the left part of the tape. The distinction can be made by partitioning the state of M_2 into two parts. Special end markers # are put on the left boundary of the tape to facilitate switching from one track to the other.

The Off-Line Turing Machine [slide 36-43] If we put the input file back into the picture, we get what is known as an **off-line Turing machine**. In such a machine, each move is governed by the internal state, which is currently read from the input file, and what is seen by the read-write head. A schematic representation of an off-line machine is shown in slide 36.

9. [Slide 37] Briefly indicate why the class of off-line Turing machines is equivalent to the class of standard machines.

First, the behavior on any standard Turing machine can be simulated by some off-line model. All that needs to be done by the simulating machine is to copy the input from the input file to the tape. Then it can proceed in the same way as the standard machine.

A standard machine can simulate the computation of an off-line machine by using four-track arrangement shown in slide 41. In that picture, the tape contents shown represent the specific configuration of slide 36. Each of the four tracks of M_2 plays a specific role in the simulation. The first track has the input, the second marks the position at which the input is read, the third represents the tape of M_1, and the fourth shows the position of M_1's read-write head.

The simulation of each move of M_1 requires a number of moves of M_2. Starting from some standard position, say the left end, and with the relevant information marked by special end markers, M_2 reaches track 2 to locate the position at which the input file of M_1 is read. The symbol found in the corresponding cell on track 1 is remembered by putting the control unit of M_2 into state chosen for this purpose. Next, track 4 is searched for the position of the read-write head of M_1. With the remembered input and the symbol on track 3, we know that M_1 is

to do. This information is again remembered by M_2 with an appropriate internal state. Next, all four tracks of M_2's tape are modified to reflect the move of M_1. Finally, the read-write head of M_2 returns to the standard position for the simulation of the next move.

Turing Machines with More Complex Storage

The storage device of standard Turing machine is so simple that one might think it possible to gain power by using more complicated storage devices. But this is not the case, as we now illustrate with example.

Multitape Turing Machine [slide 44-50] A multitape Turing machine is a Turing machine with several tapes, each with it is own independently controlled read-write head [slide 44].

10. [slide 46] Show the equivalence between multitape and standard Turing machine.

To show the equivalence between multitape and standard Turing machines, we argue that given multitape Turing machine M_1 can be simulated by standard Turing machine M_2 and, conversely, that any standard Turing machine can be simulated by a multitape one. The second part of this claim needs no elaboration, since we can always elect to run a multitape machine with only one of its tapes doing useful work [slide 46]. The simulation of a multitape machine by one with a single tape is a little more complicated, but conceptually straightforward [slide 47].

Consider for example, the two-tape machine in the configuration depicted in slide 45. The simulating single-tape machine will have four tracks [slide 48]. The first track represents the contents of tape 1 of M_1. The nonblank part of the second track has all zeros, except for a single 1 marking the position on M_1's read-write head. Track 3 and 4 play similar role for tape 2 of M_1. Slide 48 make it clear that, for relevant configurations of M_2 (that is, the ones that have the indicated form), there is a unique corresponding configuration of M_1.

The representation of multitape machine by a single-tape machine is similar to that used in the simulation of an off-line machine. The actual steps in the simulation are also much the same, the only difference being that there are more tapes to consider.

It is important to keep in mind the following point. When we claim that a Turing machine with multiple tapes is no more powerful than a standard one, we are making statement only about what can be done by these machines, particularly, what languages can be accepted.

11. [Slide 51] Consider the language $\{a^n b^n\}$. In [class 15 – Slide 36], we described a laborious method by which this language can be accepted by a Turing machine with one tape. Repeat the example using a two-tape machine.

Using a two-tape machine makes the job much easier. Assume that an initial string $a^n b^m$ is written on tape 1 at the beginning of the computation. We then read all the a's, copying them onto tape 2. When we reach the end of the a's, we match the b's on tape 1 against the copied a's on tape 2. This way, we can determine whether there are an equal number of

a's and *b*'s without repeated back-and forth movement on the read-write head.

Remember that that various models of Turing machine are considered equivalent only with respect to their ability to do things, not with respect to ease of programming or any other efficiency measure we might consider.

Multidimensional Turing Machine [slide 53-58]: A multidimensional Turing machine is one in which the tape can be viewed as extending infinitely in ore than one dimension. A diagram of a two-dimensional Turing machine is shown in slide 53.

12. [slide 54] Show the equivalence between multidimensional and standard Turing machine.

To simulate this machine on a standard Turing machine, we can use the two-track model depicted in slide 56. First, we associate an ordering or address with the cells of the two-dimensional tape. This can be done in a number of ways, for example, in the two-dimensional fashion indicated in slide 53. The two-track tape of the simulating machine will use one track to store cell contents and the other one to keep the associated address. In the scheme of slide 53, the configuration in which cell (?, ?) contain *a* and cell (?, ?) contains *b* is shown in slide 56. Note one complication: The cell address can involve arbitrarily large integers, so the address track cannot use a fixed-size field to store addresses. Instead, we must use a variable field-size arrangement, using some special symbols to delimit the fields, as shown in the picture.

Let us assume that, at the start of the simulation of each move, the read-write head of the two-dimensional machine M_1 and the read-write head of the simulating machine M_2 are always on corresponding cells. To simulate a move, the simulating machine M_2 first computes the address of the cell to which M_1 is to move. Using the two-dimensional address scheme, this is a simple computation. Once the address is computed, M_2 finds the cell with this address o track 2 and then changes the cell contents to count for the move of M_1. Again, given M_1, there is a straightforward construction for M_2.

Nondeterministic Turing Machines [slide 59-71] since nondeterminism involves an element of choice and so has a nondeterminism falvor, an appeal to Turing Thesis is inappropriate. We must look at the effect of nondeterminism in more details if we want to argue that nondeterminism adds nothing to the power of a Turing machine. Again we resort to simulation, showing that nondeterministic behavior can be handled deterministically.

13. [slide 61] Nondeterministic automata are usually viewed as accepters. How this applies to a nondeterministic Turing machine?
14. [slide 62] Prove the Theorem: The class of deterministic Turing Machines and the class of nondeterministic Turing machine are equivalent.

To show that nondertministic Turing machine is no more powerful than a deterministic one,

we need to provide a deterministic equivalent for the nondeterminism. We already alluded to one. Nondterminism can be viewed as deterministic backtracking algorithm [slide 62], and deterministic machine can simulate one as long as it can handle the bookkeeping involved in the backtracking [slide 63]. To see how this can be done [slide 66-69] simply, let us consider an alternative view of nondeterminism, one which is useful in many arguments: A nondeterminism machine can be seen as one that has the ability to replicate itself whenever necessary. When more tha one move is possible, the machine produces as many replicas as needed and gives each replica the task carrying out one of the alternatives. This view of nondeterminism may seem particularly nonmechanistic, since unlimited replication is certainly not within the power of present-day computers. Nevertheless, a simulation is possible.

One way to visualize the simulation is to use a standard Turing machine, keeping all possible instantaneous description of the nondeterministic machine on its tape, separated by some convention [slide 67-68]. The details are certainly tedious, but not hard to visualize. Based on this simulation, we conclude that for every nondeterministic Turing machine there exists an equivalent deterministic standard machine [slide 70].

15. [slide 64] As always, nondeterminism can be seen as a choice between alternatives. How can you visualize nondeterminism as decision tree?
16. [slide 71] The simulation in the Deterministic machine takes time exponential time compared to the Nondeterministic machine. Explain?

The width of such configuration tree depends on the branching factor, that is, the number of options available on each move. If k denotes the maximum branching, then $M=k^n$ is the maximum number of configurations that can exist after n moves.