Course: Automata Theory

Lecture 13: Problem Computability and the Halting Problem

Lecturer: Martha Gichuki

Course description

- The course begins with an introduction to logic and formal grammar where learners will do a recap on sets, logic and truth tables, sequences, relations and functions
- A coverage of finite state machines, Push Down automata and Turing Machines (The Church's thesis) will culminate the study of various models of computation.
- Formal language and grammar will then follow to enable learners differentiate regular and context free languages.
- An evaluation of the computability and complexity of practical computational problems which are the foundations of automata theory will then be done and the outcome will be problem description.

Learning outcomes:

Lecture 13: Problem Computability and the Halting Problem

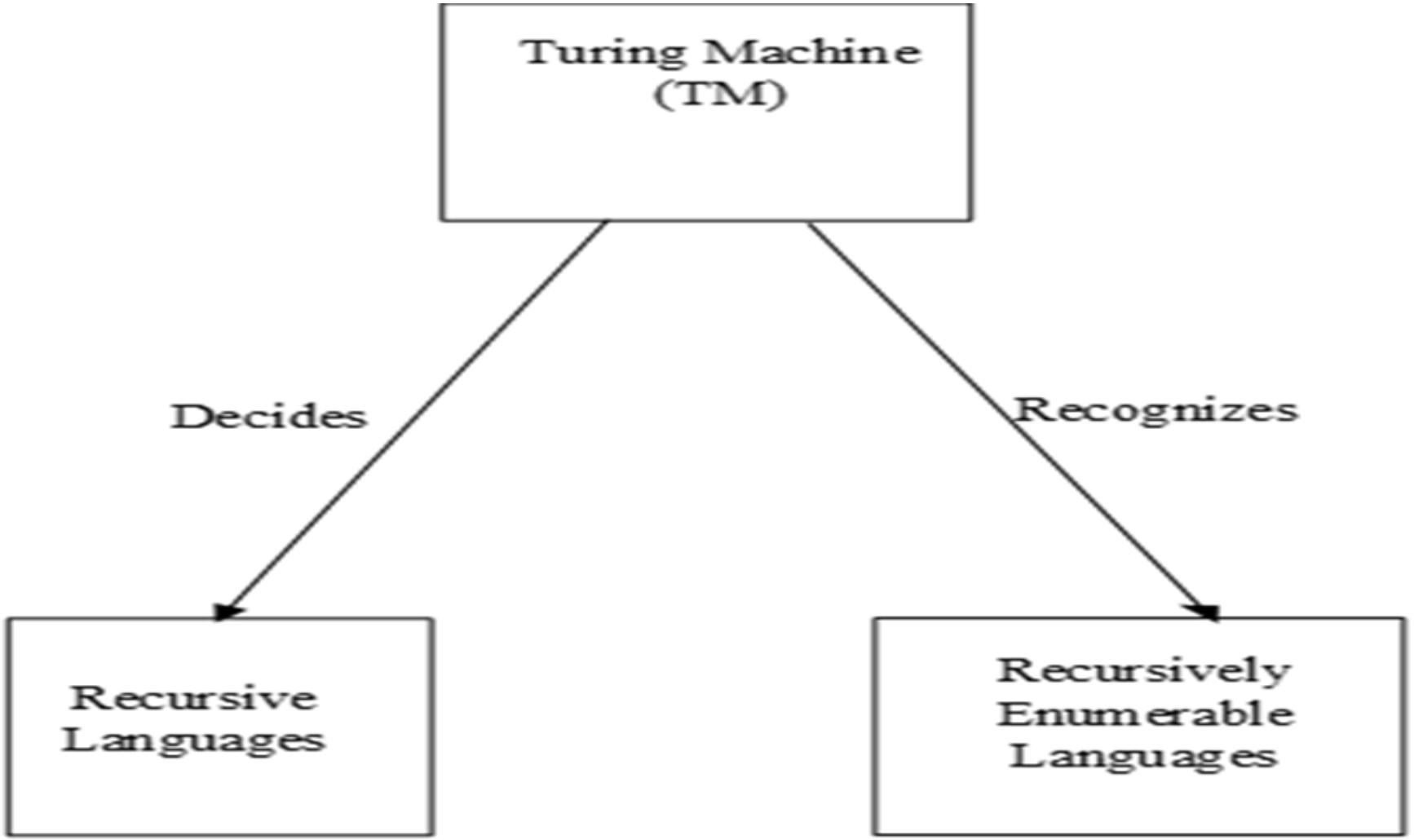
At the end of the lecture the learner will be able to:

- Describe Problem computability.
- Describe the Halting Problem
- Differentiate decidable and undecidable problems

Introduction

- Recall that Turing machines models are very powerful and they are known to be built for every problem that is intuitively computable.
- However, Turing Machines have limitations since in real life not every problem is computable.
- Therefore, most problems are not solvable by Turing machines and, therefore, not solvable by computers.

Turing machines decide recursive languages and recognize the recursively enumerable languages.



Equivalence of Turing Machine with Other Models

- Researchers have proposed several models of computation with time with all of them exhibiting the characteristic of unrestricted access to unlimited memory.
- Some of these models are similar to the Turing machines, while others are different e.g. the lambda calculus by Alonzo Church.
- This points at the fact that **computational power depends on the computer model** in use.
- Whatever can be handled by a powerful super-computer can also be handled by a basic Turing machine.

Computability Theory

- In computability theory, the halting problem is the problem of checking whether a program will finish running or continue to run forever upon an input.
- According to Anil & Michiel (2019), Alan Turing proved that a general algorithm that can solve the halting problem for all possible program input-pairs is not practical and cannot possibly exist.

The Halting Problem

- The halting problem is undecidable over Turing machines and it is among the first cases of decision problems that are unsolvable.
- Consider a program meant to check whether a program halts, another uncontrolled program takes in some input and can pass its own source and inputs to this program and do the reverse of what it predicts the uncontrolled program does.

- In computability theory and computational complexity theory, an *undecidable problem is a decision problem*
- *For undecidable problems, i*t has been proved impossible to construct an algorithm that is always leading to a correct definite answer of Yes or No.
- Therefore, the halting problem can be put as *given an arbitrary program and finite input, decide whether the program finishes or runs forever*.

- Not every set of natural numbers is computable.
- Consider a set of natural numbers $\mathbb{N}$, The halting problem in itself is a set of Turing Machine descriptions that halt upon input 0 and is a popular example of non-computable set.

- There are many Turing Machines in existence meaning that many non-computable sets exist.
- In addition, there are many sets of natural numbers that are possibly uncountable.

Undecidable Problems

The terminology "undecidable" has two distinct meanings in contemporary use.

i. That a statement is neither provable nor refutable in a specified deductive system

ii. In relation to computability theory, that which applies not to statements but to decision problems, which are countably infinite sets of questions each requiring a Yes or No answer.

Such a problem is said to be undecidable if there is no computable function that precisely answers every question in the problem set, Anil & Michiel (2019).

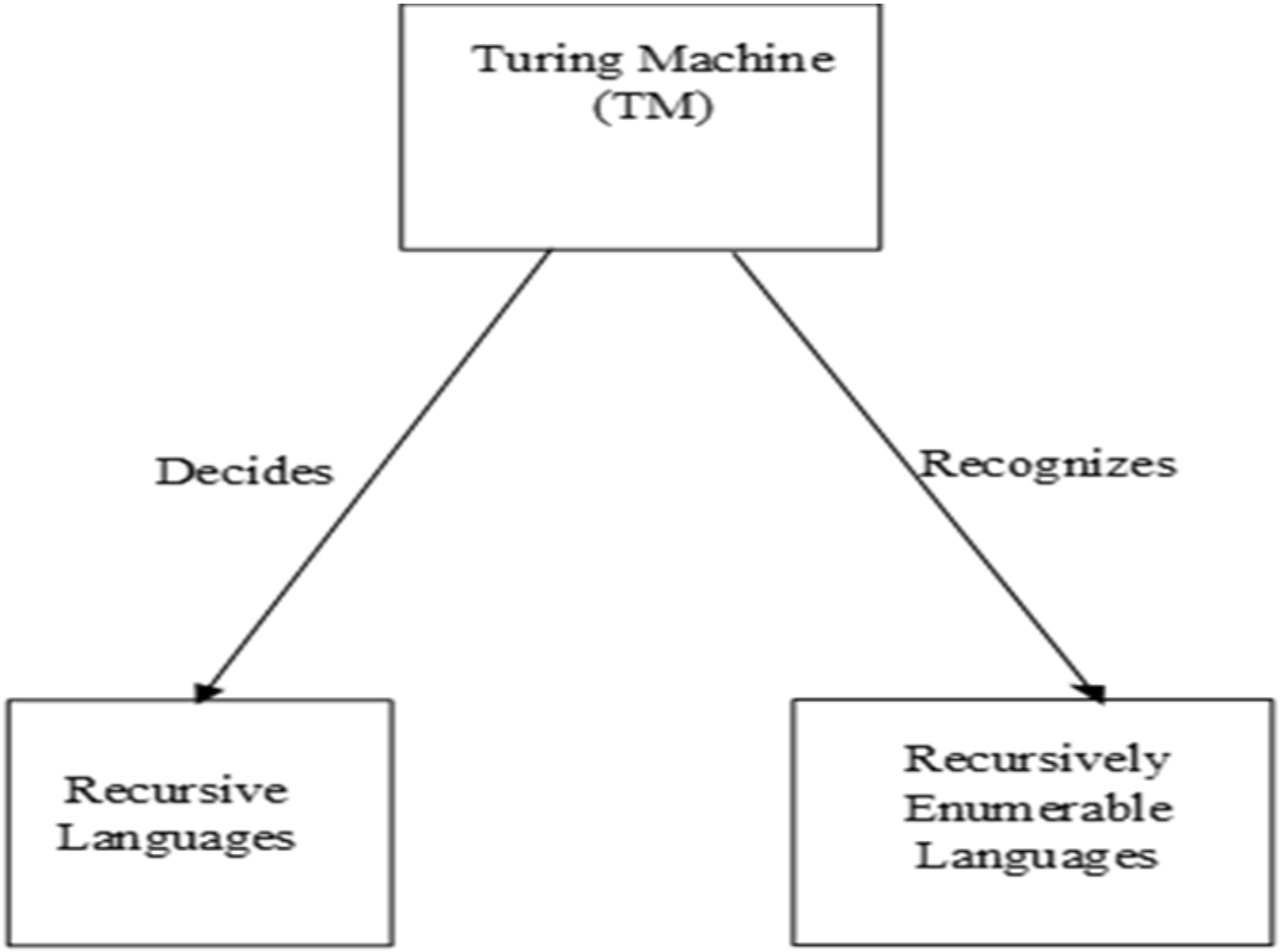
- The connection between these two meanings is that if a decision problem is undecidable then there is no consistent, effective formal system proving that for every question Q in the problem, the answer is either Yes or No.

Decidability

Which language does a Turing Machine (TMs) recognize/decide?

- Turing machines are equivalent to algorithms and are the theoretical basis for modern computers.
- Turing machines decide recursive languages and recognize the recursively enumerable languages.

Turing Machine decides Recursive Languages



Turing Machines and Recursive Languages….

- In Lecture 9 we indicated that Turing machines accept or reject input strings from which we can define the following class of languages:

Decidable Languages…

- Let Σ be an alphabet and let $A \subseteq \Sigma^*$ be a language.
- We say that A is decidable, if there exists a Turing machine M, such that for every string $w \in \Sigma^*$, the following two conditions hold:

Decidable Language A

1. If w ∈ A, then the computation of the Turing machine M, on the input string w, ***terminates in the accept state***.
2. If w ∉A, then the computation of the Turing machine M, on the input string w, ***terminates in the reject state***.

Decidable languages

In other words, the language A is decidable, if there exists an algorithm that:

i. terminates on every input string w, and
ii. correctly tells us whether $w \in A$ or $w \notin A$.

A language A that is not decidable is called **undecidable**. For such a language, there does not exist an algorithm that satisfies (i) and (ii) above.

The Halting Problem

- Turing machines were designed to represent all possible computations, and therefore they have one limitation: - some Turing machines never halt while subjected to certain inputs.

Non-halting

- Non-halting happens due to various reasons like mis-programming a Turing machine so that it gets into a tight loop, for example, in state q_1 the read write head may read a 1 it transits to state q_1, write a 1 and replace its head by 0. If we reach a situation with only blank symbols to the right, we will keep looping in the same state, moving one step to the right, and looking for a "1".
- A decent compiler can easily detect and solve these cases of non-halting.

Decidable and undecidable languages

- Several examples of decidable and undecidable languages exist.
- These examples involve languages A whose elements are pairs of the form (M, w), where M is some computation model (for example, a Deterministic Finite Automaton - DFA) and w is a string over the alphabet Σ.

- The pair (M, w) is in the language A if and only if the string w is in the language of the computation model M.

- For different computation models M, we will ask the question whether A is decidable, i.e., whether an algorithm exists that decidesA , i.e. for any input (M, w), whether or not this input belongs to the language A.

- Since the input to any algorithm is a string over some alphabet, we must encode the pair (M, w) as a string.

- In all cases that we consider, such a pair can be described using a finite amount of text.

- Therefore, we assume, without loss of generality, that binary strings are used for these encodings.

Theorem: language Halt is undecidable.

- Proofing this theorem by contradiction entails assuming that the language Halt is decidable.
- Then there exists a Java program H that takes as input a string of the form (P, *w*), where P is an arbitrary Java program and *w* is an arbitrary input for P.

The program H has the following property:

- f (P, *w*) ∈ Halt (i.e., program P ***terminates on input*** *w*), then H outputs **true.**
- If (P, *w*) ∉ Halt (i.e., program P ***does not terminate on input*** *w*), then H outputs **false**.
- In particular, H terminates on any input (P, *w*).

- The output of H is denoted by H (P, w) and we denote the computation obtained by running the program P on the input w as P(w).

- Hence, H(P, w) = true if P(w) terminates and false if P(w) does not terminate.

- Looking at the halting problem differently, we assume Halt = {(P, w): P is a Java program that terminates on the input string {w} is undecidable.
- Let all sets of all Java programs be countable, meaning that all Java programs P can be described by a finite amount of text.
- To denote this description with a binary string we use ⟨P⟩.

For any integer $n \geq 0$, there are at most 2^n (i.e., finitely many) Java programs P whose description (P) has length n.

Therefore, to obtain a list of all Java programs, we perform the following tasks:

- List all Java programs P with length = 0.
- List all Java programs P with length = 1.
- List all Java programs P with length = 2.
- List all Java programs P with length = 3.
- List all Java programs P with length = 4.
- List all Java programs P with length = 5.
- … etc.

- This is an infinite list, where every Java program occurs exactly once meaning, the set of all Java programs is countable as P1, P2, P3, . . . in which every Java program occurs exactly once.
- Assume that the language Halt is decidable, then there exists a Java program H that decides this language.
- We may assume that, on input (P, w), H returns true if P terminates on input w, and false if P does not terminate on input w.

Summary

- Turing Machines are the **most powerful computational machines**.
- They possess an **infinite memory** in the form of **a tape**, and **a Read/Write** head that changes the tape
- The halting problem is undecidable over Turing machines and it is among the first cases of decision problems that are unsolvable.

References

- Rowan G. & John T., (2009), *Discrete Mathematics: Proofs, Structures and Applications*, CRC Press, ISBN: 9781439812808.
- W. D. Wallis (2003), *A Beginners Guide to Discrete Mathematics*, Springer Science & Business Media, ISBN: 978-0817642693.
- Introduction to the theory of computation (3rd ed.), Michael, S. Boston, Cengage Learning. ISBN-13: 978-1133187790, (2012).
- Introduction to languages and the theory of computation (3rd ed.), Martin, J., New York: McGraw-Hill. ISBN-13: 978-0072322002, (2002)
- Introduction to Theory of Computation, Anil Maheshwari, Michiel Smild: School of Computer Science – Carleton University Ottawa, Canada. (2019)