Theory of computation **Fall 2015**

Lecture 19: Decidability, and Church-Turing Thesis

Faculty: K.R. Chowdhary *: Professor of CS*

19.1 Decision Problems

We may be interested in following questions:

"Is a number perfect square?"

"Is a number prime?"

"Does a graph has cycle?"

"Does the computation of TM halt before 25th transition?"

Each of these general questions describe a decision problem. A decision problem P is a set of *related questions* p_i, each of which has Yes/No answer. For example:

p_0: Is 0 a perfect square?

p_1 Is 1 a perfect square?

p_2 Is 2 a perfect square?

...

Each of the p_i is an instance of the problem P. The solution of a decision problem P is an algorithm that determines the answer of every question "$p_i \in P$¿'. A decision problem is decidable, if it has a solution.

An algorithm that solves decision problem should be:

- Complete: correct answer is given for every problem instance,
- Mechanistic: finite sequence of instructions, each can be carried out without requirement of insight, ingenuity, or guesswork, and
- Deterministic: With identical input, the same computation is carried.

A procedure having the properties of complete, Mechanistic, and deterministic, is called *effective procedure*. A standard TM is an effective algorithm if it is, Mechanistic, deterministic, and complete. However, it is complete only if, it halts on every input.

Hence, TM can be used to solve decision problems. To solve the problem using TM, we need to transform the problem instances into strings, called, representation of the decision problem.

The *Church-Turing* thesis for decision problem says that, TM can be designed to solve the decision problem if it can be solvable by effective procedure.

Representation of problem in terms of acceptance of strings:

Problem instance	TM Input	Answer
p_1	w_1	Yes/No
p_2	w_2	Yes/No
...	...	...
p_i	w_i	Yes/No

For example, for "Whether a natural number is even?", when represented using unary representation, we have:

0	1	even
1	11	odd
2	111	even
...	...	...

In general, 1^i, such that i is odd, is an even number. The TMs for acceptance of odd numbers, can be constructed for unary (M_1) and binary numbers (M_2), as shown in figure 19.1.

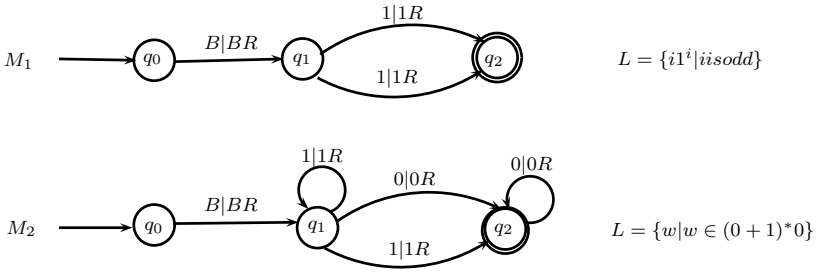

Figure 19.1: TMs to recognize odd numbers.

Decision problem vs Recursive languages: Since, completeness requires that TM shall terminate on every input string, therefore, the language recognized by this recursive. In other words, every TM solution to a decision problem defines a recursive language. The decision problem is membership problem: "Is $w \in L$?"

19.2 Church-Turing Thesis

The intuitive notion of an algorithm is that it is a mechanical procedure, which is unambiguous, with simple instructions for carrying out a computational task. This intuitive notion is adequate when we are required to design an algorithm to compute a problem. And, with this we are always sure about whether a procedure is an algorithm or not.

However, when we say that there exists no algorithm to compute a certain problem, the intuitive notion of algorithm is not sufficient. Here, we require precise definition of the algorithm. This is because when we say that no algorithm exists to solve it, we must be sure of all algorithms. That is impossible to answer unless, we have precise definition of algorithm.

The precise definition of an algorithm is given based on the Church-Turing thesis, which gives a connection between informal nature of algorithm and its formal (mathematical) counterpart of the precise definition.

The Church-Turing thesis claims that what an algorithm computes is exactly, what TM computes, i.e.,

"Being computed by an algorithm $\Leftrightarrow$ Being computed by TM."

This statement has two parts: "algorithm $\Rightarrow$ TM", and " algorithm $\Leftarrow$ TM". We can intuitively accept $\Leftarrow$ of the Church-Turing thesis, because no matter what Turing-machine is given, we can compute mechanically, to obtain the output.

The direction "$\Rightarrow$" means, what ever algorithm is given we can describe TM to infer of this. But this is impossible to prove this direction, because we do not have precise definition of algorithm. In other words, we cannot give counter examples, i.e., there exists procedures as algorithms but cannot be described as Turing-machines.

The Church-Turing Thesis, takes the direction "$\Rightarrow$" as granted on the assumption that if it can be computed by effective algorithm, then Turing-machine can compute it. Thus, Church-Turing thesis is not what we can prove, but what we can admit.

However, there are evidences:

- no algorithm exists, which cannot be described by TM,
- any algorithm can be described by TM,
- several models have been defined independently, to define algorithms, which have been proved equivalent in power to TM, and
- All the independent models reveal that these models are natural, and robust, in the sense that their power is in variant.

Since the termination property of the TM requires the termination of TM for every input string, the language accepted by TM is *recursive*. Thus every TM solution of a decision problem defines recursive a language. Thus, every TM solution of a decision problem defines a recursive language. In other words, every recursive language can be considered as a solution to a decision problem. The decision problem is a membership problem, and consists of a question like, "Is $w \in L$?", for a given alphabet $w \in \Sigma^*$, and language over Σ.

Accordingly, we can say that, solvable decision problem has dual nature - one as recursive language, and other as TM.

Investigation into properties of computability has resulted into number of approaches:

- String transformations : Markov chains, unrestricted grammars,
- Evaluation of functions: recursive functions, lambda calculus,
- Abstract computing machines: TM, register machine, and
- Programming languages.

All the above methods have not resulted in any different problem solving capability, the reason being that the problem solution part is inherent feature of every problem itself and not of some algorithm system. Church-Turing thesis validates this intuition.

The only difference between the above four types of computations is that they perform different types of operations on different types of data, but there are equivalences in operations and data both in one system of computation to other system. Like unary vs decimal, and addition vs concatenation of unary stings.

In fact there is no single definition of algorithm and no single system for performing effective computation. However, there are well defined bounds that can be bounds on what can be accomplished in each system. The Church-Turing thesis formalizes this belief in a general statement about the capabilities and limitations of algorithmic computations.

The Church-Turing thesis is defined in three different platforms:

(a) *Church-Turing thesis for Decision problem:* There is effective procedure to solve a decision problem if and only if, for this, the TM halts for all input strings. Hence, the language accepted by TM is recursive.

A variant of this exists - TM may halt for all instances of decision problem for which the solution is Yes, and when No, it may halt or continue for ever. Thus, it is partial solution. Such a procedure is effective but not *complete*. In such case, the solution is formulated as membership to *recursively enumerable* language.

(b) *Church-Turing Thesis for Recognition problems:* A decision problem is partially solvable if and only if there is a TM that accepts instances of the problem whose answer is Yes.

(c) *Church-Turing thesis of Computable functions:* A function is effectively computable (Turing computable) if and only if there is a TM available that computes it.

The Church-Turing thesis is not a mathematical theorem, hence it cannot be proved. This would require a formal definition of the intuitive notion of effective procedure: a procedure which is mechanistic, complete, and deterministic.

The claim could however be disproved. This could be done by finding an effective procedure that cannot be computed by TM. However due to the following reasons, such procedure cannot found:

- The equivalence ot TM to other algorithmic systems,
- Robustness of TM architecture, and
- the lack of counter-example.

A proof (solution) by Church-Turing thesis is often shortcut, in evidence of decision algorithm. Rather than constructing a TM solution to a decision problem, we describe intuitively effective procedure that solves the problem.

For complicated machines, we simply give description of the actions of a computation of a machine, and we assume that complete machine could be explicitly constructed.