Theory of Formal Languages (Decidable properties of Regular Languages)

Lecture 14: April 13, 2021

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

14.1 Decision Problems and Recursive Languages

A decision problem is membership problem, like, "Is $w \in L$?", whose answer is Yes/no. Since, *completeness* requires that the corresponding TM should terminate on every input string, hence, the language recognized by this is *Recursive* (R). In other words, every TM solution to a decision problem defines a recursive language. The duality between solvable decision problems and recursive languages can be exploited to broaden the techniques available for establishing the decidability of a decision problem. For example, the decision problem of determining whether any given number is perfect square, is *decidable*.

Based on the above conclusions, now we construct the following definition for Turing Recognizable and decidable languages.

Definition 14.1 Turing recognizable. *A language L is Turing recognizable (also called, Recursively Enumerable (RE)) if there exists a Turing machine M such that $L(M) = L$. It is possible for a TM to never reach a halting configuration, and on some given input w, and it might instead loop for ever.* □

Definition 14.2 Turing Decidable. *A language L is Turing-decidable or simply* decidable*, if there is a Turing machine M that decides L. Such TMs halts on every input $w \in L$.* □

Definition 14.3 Semi-decidable. *Let $M = (Q, \Sigma, \Gamma, \delta, s, H)$ be a Turing machine and let $L \subseteq \Sigma^*$ be a language. In this case, M* semi-decides *L if for any string $w \in \Sigma^*$ the following holds: $w \in L$ if and only if M halts on input w. In addition, L is recursively enumerable if and only if there is a Turing machine M that semi-decides L.* □

Theorem 14.4 *If a language L is recursive then its complement $\overline{L}$ is also recursive.*

Proof: Let the language L is decided by the Turing machine $M = (Q, \Sigma, \Gamma, \delta, s, \{y, n\})$. Let M' be another Turing machine, such that,

$$M' = (Q, \Sigma, \Gamma, \delta', s, \{n, y\}).$$

The steps for recognition through M' (Fig. 14.1) are as follows:

1. The accepting states of M are made non-accepting states of M' with no transitions, i.e., here M' will halt without accepting also.

2. If s is new accepting state in M', then there is no transition from this state.

3. If L is recursive, then $L = L(M)$ for some TM M that always halts. Transform M into M' so that M' accept when M does not and vice-versa. So M' always halts and accepts $\overline{L}$. Hence $\overline{L}$ is recursive.

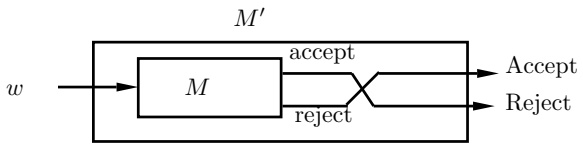

Figure 14.1: Co-Recursive is Recursive

The transition function δ' can be represented by,

$$\delta'(q,a) = \begin{cases} n, \text{ if } \delta(q,a) = y, \\ y, \text{ if } \delta(q,a) = n, \\ \delta(q,a), \text{ otherwise .} \end{cases} \tag{14.1}$$

It is clear that $M(w)$ yields y iff $M'(w)$ yields n. Thus, M decides the language $\overline{L}$, and, this proves the theorem.□

14.2 Decidable Languages

In this section we give some examples of languages that are decidable by algorithms. We focus on languages concerning automata and grammars. For example, we can present an algorithm that tests whether a string is a member of a context-free language (CFL). This is a parsing algorithm. These languages are interesting for several reasons. First, certain problems of this kind are related to applications. The problem of testing whether a CFL generates a string, is related to the problem of recognizing and compiling programs in a programming language. Second, certain other problems concerning automata and grammars are not decidable by algorithms. Starting with examples where decidability is possible helps us to appreciate the undecidable problems.

14.2.1 Decidable Properties of Regular Languages

We begin with certain computational problems concerning finite automata. We give algorithms for testing,

i. Whether a finite automaton accepts a string,

ii. Whether the language of a finite automaton is empty, and

iii. Whether two finite automata are equivalent.

Note that we chose to represent various computational problems by languages. Doing so is convenient because we have already set up terminology for dealing with languages. For example, the acceptance problem for DFAs is of testing whether a particular deterministic finite automaton B accepts a given string w. It can be expressed as a language, we call as A_{DFA}. This language contains the encoding of all DFAs together with strings that the DFAs accept [hopc68].

Consider the language,

$$A_{DFA} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}. \tag{14.2}$$

The problem of testing whether a DFA B accepts an input w is the same as the problem of testing whether $\langle B, w\rangle$ is a member of the language A_{DFA}. Similarly, we can formulate other computational problems in terms of testing membership in a language. Hence, showing that the language is decidable is the same as showing that the computational problem is decidable.

In the following theorem we show that A_{DFA} is in fact decidable. Hence this theorem shows that the problem of testing whether a given finite automaton accepts a given string is decidable.

Theorem 14.5 *A_{DFA} is decidable.*

Proof: We simply need to present a TM M that decides A_{DFA}, where M is described as follows:

The TM M receives input $\langle B, w\rangle$, where B is description/representation of a DFA and w is a string. Then TM M simulate DFA B with w as input to B. If in this simulation B accepts w then M accepts, else M rejects. The A_{DFA} (acceptability of DFA) is language whose elements are all such pairs $\langle B, w\rangle$. □ ■

In the similar line, it can be easily shown that A_{NFA} is decidable language. For this, first it is necessary to convert NFA into DFA, then run the above algorithm, such that if decidability of $\langle N, w\rangle$ is to be decided, where N is an NFA, and whether w is acceptable by N or not. Turing machine T_1 translates the NFA N into an equivalent DFA D, and a Turing machine T_2 decides membership of $\langle D, w\rangle$ by simulating DFA D on T_2. If DFA D accepts w then T_2 accepts w, then NFA accepts. Otherwise, if DFA D simulated on T_2 does not accept w, then T_2 does not accept, which imlies that NFA does not accept w. □

Similarly, we can determine whether a regular expression R generates a given string w. Accordingly, we may define a language $\langle R, w\rangle$ such that R is a regular expression, and it generates string w, as,

$$A_{REX} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}. \tag{14.3}$$

Following theorem proves that A_{REX} is decidable language.

Theorem 14.6 *A_{REX} is a decidable language.*

Proof: Let T_1 be a TM that decides A_{REX}. Let $\langle R, w\rangle$ is input to T_1, where R is regular expression, and w is some string. Follow these steps to decide about A_{REX}.

1. Run T_1 to convert R into equivalent NFA A,
2. Input $\langle A, w\rangle$ to a TM T_2,
3. Run TM T_2 on input $\langle A, w\rangle$,
4. If T_2 accepts, then T_1 *accepts*, else *rejects*.□

■

Theorem 14.7 *E_{DFA} is decidable language.*

Proof: A DFA accepts some string iff reaching an accept state from the start state by traveling along the edges of a transition graph is possible. To test this condition, we can design a TM T that uses a marking algorithm.

Let T has input $\langle A\rangle$, and A is a DFA. The steps are:

1. Mark the start of A.
2. Mark every state that has a transition coming into it from any state that is already marked.
3. If no accept state is marked, *accept*; otherwise, *reject*.□

The next theorem states that determining whether two DFAs recognize the same language is decidable. The language of all pairs $\langle A, B\rangle$, where A and B are representation of two DFAs, which recognize the identical language is expressed as,

$$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are } DFAs \text{ and } L(A) = L(B)\}. \tag{14.4}$$

Theorem 14.8 *EQ_{DFA} is decidable language.*

Proof. To prove this, we construct a DFA D from DFAs A and B, where D accepts only those strings that are accepted either by A or B, but not by both. Thus, if A and B recognize the same language, D will accept nothing[1]. The language D is,

$$L(D) = (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) \tag{14.5}$$

Once we have constructed DFA D, we can use theorem 14.7 to test whether $L(D)$ is empty. If empty, then $L(A)$ and $L(B)$ are equal. Let there is a TM T as defined below:

1. Input $\langle A, B\rangle$ to T,
2. Construct DFA D from these two DFAs A, B as per equation (14.5),
3. Run TM T (as defined in theorem 14.7) on input D,
4. If T accepts, then *accept*, else *reject*.□

[1]If two sets are P and Q, then $P \oplus Q = (\overline{P} \cap Q) \cup (P \cap \overline{Q})$. If P and Q are equal, then as per this formula, $P \oplus Q = \phi$. Note that $\oplus$ is exclusive-OR operation on sets.