CS 208 : Automata Theory and Logic DATE 12/04/2024

Lecture - 37

Topic: Rice Theorem

Scribed by: Shresth Verma (22B2211)
Checked and compiled by:

1 Recap

In the previous lecture, we introduced two specific languages. The first one we discussed was the language L_d, defined as follows:

$$L_d = \{\langle M \rangle \mid M \text{ is a Turing machine and } M \text{ does not accept } \langle M \rangle\}$$

We observed that L_d is not a recursively enumerable language. However, the complement of L_d, denoted as $\overline{L_d}$, is recursively enumerable.

We also examined the language L_u, defined by:

$$L_u = \{\langle M, w \rangle \mid M \text{ is a Turing machine that accepts input } w\}$$

We noted that L_u is recursively enumerable but not recursive. This observation leads to an important conclusion regarding the complement of L_u, denoted $\overline{L_u}$. If both L_u and $\overline{L_u}$ were recursively enumerable, then L_u would also be recursive, according to the properties of recursively enumerable languages. Thus, since L_u is not recursive, $\overline{L_u}$ cannot be recursively enumerable.

2 Understanding Recursive Languages and Decidability

In this section, we will recap on what it means for a language to be recursive and discuss the concept of decidability in the context of formal languages.

2.1 Definition of Recursive Languages

Recursive languages are characterized by the presence of an algorithm that determines membership. Specifically, for a language to be considered recursive:

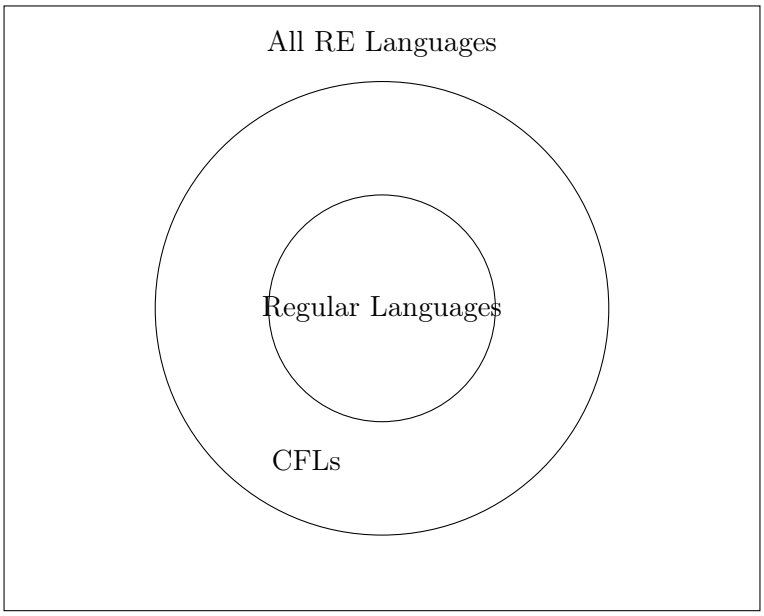
1. There must be an algorithm, by which we mean a definitive procedure, that decides whether any given input string belongs to the language or not.
2. There exists a Turing machine M for such languages where $H(M) = L$, implying that the Turing machine halts on every input. Furthermore, M halts in one of two states:
   - q_{accept} if the input is in the language,
   - q_{reject} if the input is not in the language.

This construction of Turing machines can be applied to the class of regular languages and context-free languages (CFLs). For example, given a regular language, one can construct a deterministic finite automaton (DFA) for it. Similarly, for a context-free language, it is possible to construct a corresponding grammar and a pushdown automaton (PDA) that recognizes it.

2.2 Decidability and Its Implications

Decidability is a fundamental concept in computability and formal language theory. A decision problem is considered decidable if there exists an algorithm that can provide a correct yes or no answer for every input instance of the problem within a finite amount of time or in other words a language is said to be decidable if there exists a halting turing machine M which halts on every input in the language.

2.3 Visualizing Language Classes



The above diagram represents the set of all recursively enumerable languages inside which you have the class of the regular languages , CFLS, recursive languages etc.

For further discussion , we will be referring the set of all recursively enumerable languages above as the Universal set(U).

Let's define what is meant by the property of a language. In the context of formal languages, a property of a language can be seen as a specific characteristic or feature that some languages possess. Specifically, a property of languages consists of a subset of languages within the universal set U (the set of all recursively enumerable languages, as depicted in the diagram above). Each language in this subset satisfies the property. This could involve specific structural features, computational properties, or limitations on what the languages can express or decide.

For example, one common property that is often studied is whether a language is *context-free*. A language is said to possess this property if there exists a context-free grammar that generates all and only the strings in the language. Similarly, properties such as being *regular* or *decidable* characterize other well-defined subsets of U.

2.4 Classification of Properties

In the study of formal languages, properties can be classified as either **trivial** or **non-trivial** based on how they partition the universal set U of all recursively enumerable languages.

Trivial properties are those for which either every language in U has the property or no language in U has it. Formally, a property P is trivial if $P = U$ or $P = \emptyset$, where P is the set of all languages in U that satisfy the property. These properties do not provide meaningful information for distinguishing among languages because they apply universally or not at all.

Non-trivial properties are characterized by the existence of at least one language $L_1 \in U$ that has the property and at least one language $L_2 \in U$ that does not have the property. Formally, a property P is non-trivial if $\exists L_1 \in P$ and $\exists L_2 \in U \setminus P$. This distinction means that non-trivial properties can effectively differentiate between languages, making them particularly valuable for theoretical studies and practical applications in computational linguistics and automata theory.

2.5 Decidability of Non-Trivial Properties

Having established the distinction between trivial and non-trivial properties of languages, we now turn to a fundamental question in the theory of computation: *Is a given non-trivial property P, which is a strict subset of U, decidable?*

It is important to note that for a property to be considered non-trivial, it must be a strict subset of U (i.e., $P \subset U$ and $P \neq U$, $P \neq \emptyset$). This distinction excludes trivial properties, as they do not provide meaningful information for computational decisions.

Question: Given a Turing machine M, is the language recognized by M, denoted $L(M)$, a member of the property P? This question is central to understanding the decidability of properties of languages.

For example, consider the non-trivial property of regularity. The question becomes: Given a Turing machine M, is $L(M)$ regular?

The question posed earlierâwhether the language $L(M)$ recognized by a given Turing machine M belongs to a non-trivial property Pâis, in general, undecidable. This implies that no universal algorithm exists that can determine, for every conceivable Turing machine M, whether its recognized language $L(M)$ exhibits the property P.

For instance, while we might be able to verify the regularity of languages generated by certain types of automata, extending this verification to languages recognized by arbitrary Turing machines introduces problems that transcend the capabilities of algorithmic decision-making.

2.6 Proof of Undecidability

Let's now see the proof of the above question being undecidable. Assume that the question is decidable, i.e., for a particular property P, we can construct a halting Turing machine M_p such that given any input Turing Machine M into it, it halts on the input if and only if $H(M)$ belongs to P, outputting "Yes" in this case and "No" otherwise.

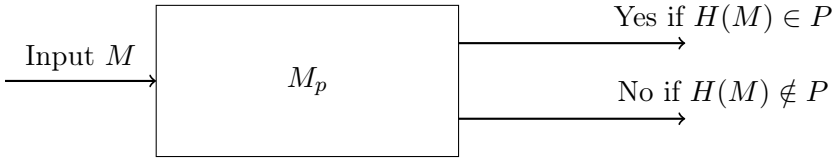


Figure 1: Operation of Turing Machine M_p

This theoretical construction aims to decide whether the language recognized by a given Turing machine M has the non-trivial property P. The diagram illustrates the decision process carried out by M_p.

Now the flow of the proof will be to use this Turing machine as a subroutine to construct another halting Turing machine for a langauge which is earlier proved to be undecidable and hence establish a contradiction. This is the most common idea taken while proving undecidability of langauges.

2.7 Constructing the Turing Machine for L_u

Consider the case that $\emptyset$ belongs to P. The other case, where $\emptyset$ does not belong to P, can be argued similarly by constructing another M_p' for the complement of P (outputs "Yes" if $H(M)$ does not belong to P and "No" otherwise). Now, I will construct a Turing machine for the language L_u using M_p as follows:

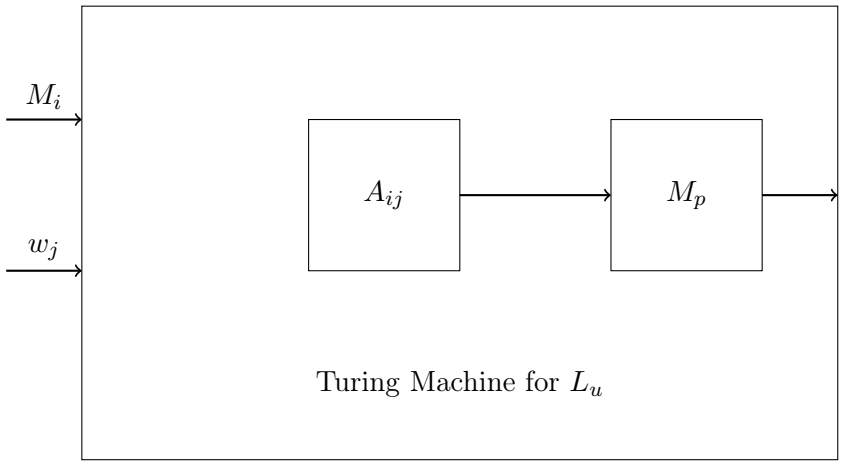


Figure 2: Construction of a Turing Machine for L_u using A_{ij} and M_p

This configuration demonstrates how the Turing machine for L_u processes inputs M_i and w_j, passing them through Turing machine A_{ij},here A_{ij} is a Turing machine defined for each pair of (M_i, w_j), whose output is subsequently fed into Turing machine M_p. This setup effectively uses A_{ij} and M_p as subroutines to decide whether M_i halts on w_j or not .

In the construction of the Turing machine A_{ij}, we consider it as a two-tape Turing machine where:

- The first tape T_1 already has w_j written on it. This is a part of the construction of A_{ij} for each pair (M_i, w_j).
- The second tape T_2 is used for computations involving another Turing machine M_2.

Now given that P is a non-trivial property, by definition, there exist two Turing machines M_1 and M_2 such that $H(M_1)$ belongs to P and $H(M_2)$ does not belong to P.Hence we will use this M_2 Turing machine as the one which A_{ij} mimics on the tape T_2 . The operational steps of A_{ij} are as follows:

1. A_{ij} mimics M_i on w_j on the first tape T_1.
2. If M_i halts on w_j, A_{ij} then proceeds to use the second tape T_2.
3. On tape T_2, A_{ij} mimics M_2 using the input provided on tape T_2.Note that the main input tape of A_{ij} is the tape T_2 as this is where the machine is given input by the user . On tape T_1, the encoding of w_j is already present as a part of the construction of A_{ij}.

From this construction, it follows that:

- If M_i halts on w_j, then A_{ij} moves from step 2 to step 3 and effectively behaves like M_2 on the second tape.Hence the language of A_{ij} and M_2 will be same in this case and since $H(M_2)$ does not belong to P, $H(A_{ij})$ also does not belong to P in this case.
- If M_i does not halt on w_j, then A_{ij} remains stuck on the first tape and never halts, meaning the language it accepts is empty ($\emptyset$). Since we assumed $\emptyset \in P$, $H(A_{ij})$ in this case belongs to P.

This setup allows the constructed A_{ij} to be fed into M_p. Depending on M_p's output:

- If M_p outputs "Yes", it indicates $H(A_{ij}) \in P$, meaning M_i did not halt on w_j.
- If M_p outputs "No", it signifies $H(A_{ij}) \notin P$, implying M_i did halt on w_j.

Thus, we establish a method to decide halting of M_i on w_j by M_p's output, which leads to a contradiction since L_u is known to be non-recursive. This contradiction shows that our assumption about the decidability of P is false, as construction of A_{ij} can be done for all possible pairs of (M_i, w_j), confirming that the property is indeed undecidable.

3 Conclusion and Application to Rice's Theorem

The undecidability result we have demonstrated in the previous section is a specific instance of a more general principle known as **Rice's Theorem**. Rice's Theorem asserts that any non-trivial property of the language recognized by a Turing machine is undecidable. This theorem is a cornerstone in the theory of computation as it broadly applies to any property that can distinguish between two Turing machines based on the languages they recognize.

3.1 Applications of Rice's Theorem

One of the many practical applications of Rice's Theorem is in the field of operating systems, particularly in the context of process management. Consider the problem of determining whether a process will lead to a crash in an operating system. According to Rice's Theorem, this is undecidable because the property P â whether the process will crash or not â is a non-trivial property of the computation performed by the process.

- **Property** P: A process crashes the operating system.
- **Implication**: It is undecidable to conclusively predict from the process code alone whether it will crash the system, as this involves predicting whether the execution trace of the process will enter a state that causes the crash.

This application is particularly significant because it highlights the limitations of static analysis tools in ensuring software reliability and security. Despite advances in software testing and verification, Rice's Theorem provides a fundamental limit on what can be achieved through purely algorithmic means.