Theory of Computation: Part 3- Limitations of Turing Machines

Michael Levet

January 16, 2014

1 Introduction

This tutorial will explore languages that Turing Machines can (and cannot) decide and accept. In essence, it takes the concepts from the last tutorials and begins to actually explore the limits of Turing Machines. Familiarity with Formal Languages is assumed.

2 Languages Turing Machines Can Decide

This section will introduce certain languages that Turing Machines can decide. Remember that in order for a language to be decidable, there must exist a Turing Machine that can determine correctly whether or not an arbitrary string is in the language.

Consider the first example, $L_{RX} = \{R, w :$ R is a regular expression, and $w = L(R)\}$. Remember that regular expressions are accepted by Finite State Automaton. Without loss of generality, let F be a non-deterministic FSA that accepts w. It is always possible to reduce a non-deterministic FSA to a deterministic FSA, F'. The transformation itself is not important in this section, but the result that follows is. That is, every regular language can be accepted by a deterministic finite state machine. Let T be a Turing Machine to simulate F'. Since FSA do not write anything, T will not write to the tape. Thus, T will generate $L(F')$. If $L(F') = w$, then T accepts R, w. Otherwise, T rejects R, w. Thus, the L_{RX} is decidable.

The intuition behind the comparison of $L(F')$ and w comes from using the symmetric difference set operation. That is, if $L(F') - w = \emptyset$, and $w - L(F') = \emptyset$, then $L(F') = w$. Regular languages also have an important property: A regular language L accepted by a FSA M is non-empty iff there exists a string in the language whose length is less than the number of states in M. Regular languages are also closed under intersection and complementation, so $L(F') - w$ and $w - L(F')$ are regular if and only if w and $L(F')$ are regular. Since F' is a FSA, $L(F')$ is regular. So if w is either non-regular or a language differing from $L(F')$, then the Turing Machine will reject the input R, w. Otherwise, it will accept the input string. Thus, there exists a Turing Machine to decide L_{RX}.

The next two languages to be introduced are $R = \{\rho(M) :$ M is a DFA and $L(M) = \emptyset\}$, and
$C = \{\rho(G)$: G is a Context-Free Grammar and $L(G) = \emptyset\}$. Both of these languages are decidable. Let's first explore how these languages are decidable, then discuss their importance.

To begin, let RE be the set of Recursively Enumerable Languages. The Chomsky Hierarchy depicts a relationship between various sets of languages. In particular, it describes regular languages as a subset of context-free languages. In turn, context-free languages are a subset of context-sensitive languages.

Finally, context-sensitive languages are a subset of RE.

It follows by this subset relationship that if C is decidable, then R is decidable, since $R \subset C$. Let's now explore why C is decidable. Remember that a CFG has Terminal Symbols, Non-Terminal Symbols, Rules, and a starting symbol S. The idea is similar to the construction of a Huffman Tree. That is, start by marking the terminal symbols. These are analogous to the leaf-nodes of a Huffman tree. The process is then to build the tree upwards. This is done by reviewing the non-terminals. If a given non-terminal has a rule which can be replaced by already marked symbols, that rule is marked and the symbols are replaced. If S can be reached, then $L(G)$ is non-empty and the Turing Machine rejects the string. Otherwise, $L(G)$ is empty and the input string is accepted.

Regular languages are generated by regular grammars, so a similar method of proof can be used to show that the language R is decidable. Regular languages are also generated by FSA. Thus, it is sufficient to execute a graph traversal on the graph representation of the FSA from the input string, $\rho(M)$. If it is possible to reach an accepting halt state from the initial state of the FSA, then $L(M)$ is non-empty, so the Turing Machine rejects $\rho(M)$. Otherwise, $\rho(M)$ is accepted. Thus, the language R is decidable.

The languages C and R are important in illustrating an important concept: solving a subset of a problem is not the same as solving a problem. Consider the language $L = \{\rho(M)$: M is a Turing Machine and $L(M) = \emptyset\}$. The language L is undecdiable; however, Turing Machines are capable of deciding all regular languages. Turing Machines are also capable of accepting CFG. Thus, there is the relationship that $R \subset C \subset L$. Clearly, deciding L would have implications about deciding C; and C being undecidable would imply that L is undecidable. Remember that Languages are descriptions of problems.

3 Non-Turing Acceptable Language

This section will introduce a language that is not recursively enumerable. Let $L_{diag} = \{\omega_i : \omega_i$ is not accepted by $M_i\}$, where ω_i is the ith string in lexicographical order over the given alphabet, and M_i is the ith Turing Machine. Note that this languages has no constraints about ω_i with respect to M_j, for $i! = j$.

Theorem 3.1. *L_{diag} is not recursively enumerable.*

Proof. This theorem is proven by contradiction. Suppose L_{diag} is recursively enumerable. Then there exists a Turing Machine that accepts L_{diag}. Let M_k be a Turing Machine to accept L_{diag}. By definition of language acceptance, M_k accepts all strings in the language. Suppose $\omega_k \in L_{diag}$. Then M_k accepts ω_k, which contradicts the definition of the language L_{diag}. Suppose ω_k is not in L_{diag}. Then ω_k is accepted by M_k. However, by definition of language acceptance, M_k cannot accept any string not in L_{diag}. Thus, $\omega_k \in L_{diag}$ if and only if $\omega_k \notin L_{diag}$. Therefore, no Turing Machine exists which accepts L_{diag}, so L_{diag} is not recursively enumerable. QED. □

For those familiar with a Cantor diagonal argument, the above proof may have a familiar feel to it. The idea is to look at pairs: (ω_i, M_i). If ω_i is not accepted by M_i, then w_i is in L_{diag}. In order to determine if L_{diag} is recursively enumerable, it suffices to show the existence of a Turing Machine that accepts L_{diag}. This is the crux of the proof. That is, understanding the definition of language acceptance will lead to a better understanding of why L_{diag} is not recursively enumerable. Remember that in order to be recursively enumerable, that there must exist a Turing Machine M which accepts L_{diag}. The definition of language acceptance states that $L(M) = L_{diag}$. That is, M only halts in the accepting state for strings in L_{diag}.

Remember that Turing Machines are countable, as well. That means that there is a natural number k, such that M_k (by assumption) accepts L_{diag}. Strings are enumerable as well. Thus, if the string ω_k over the given alphabet is accepted by M_k, then $\omega_k \notin L_{diag}$ and M_k accepts a string not in L_{diag}. This is a violation of the definition of language acceptance. Now if $w_k \in L_{diag}$, then M_k does not accept ω_k. Thus, $L(M_k) \neq L_{diag}$, so M_k does not accept L_{diag}.

4 Recursively Enumerable Languages and the Halting Problem

This section will introduce some recursively enumerable languages that are not decidable, as well as the Halting Problem. Recursively enumerable languages that are not decidable are used to describe problems whose solutions can be easily verified, but are difficult to find.

Consider the language: $\overline{L_{diag}} = \{\omega_i : \omega_i \text{ is accepted by } M_i\}$. This is the complement of L_{diag} in the last section. The language $\overline{L_{diag}}$ is recursively enumerable, but not decidable. In order for a language to be decidable, both it and its complement must be recursively enumerable. Since L_{diag} is not recursively enumerable, $\overline{L_{diag}}$ is clearly not decidable.

In order to show that $\overline{L_{diag}}$ is recursively enumerable, it is sufficient to show the existence of a Turing Machine that accepts it. The idea here is to have the Turing Machine, M_{accept}, invoke the Universal Turing Machine to find the Turing Machine, M_k. If the input string, ω_k, is accepted by M_k, then M_{accept} accepts ω_k.

The Halting Problem, which is an incredibly important problem in computer science, comes to the forefront in determining limits of computational models. The idea is to determine if a given program will terminate on an arbitrary input string. That is, is it possible to determine if for all programs and for all input strings, a given program will terminate on a given input string? Note the universal quantifiers. The answer is that it is impossible to determine if a given program will halt an an arbitrary input. However, there are examples of programs where it is possible to determine if they halt on given input. Examples of these, and associated proofs, are reviewed in the context of Algorithm Analysis.

The consequences of the Halting Problem are far reaching. An important goal of software engineering is to show correctness and halting of a piece of industrial grade software. The Halting Problem implies that it is impossible to do this. There are practices in software engineering to improve one's ability to verify a program's correctness, but it is impossible to do this universally.

Now let's formalize the Halting Problem. Let $L_H = \{\rho(M), \omega : \text{M halts on } \omega\}$. The language L_H is not decidable, but it is recursively enumerable. To show that L_H is recursively enumerable, the same approach is used as in showing that $\overline{L_{diag}}$ is recursively enumerable. That is, let H be the Turing Machine to accept L_H. Given an input string, H invokes the UTM to simulate M on ω. H accepts the input string if and only if M halts on ω. Thus, H accepts L_H, so L_H is recursively enumerable.

Showing that L_H is not decidable is a bit more subtle. It is done by contradiction. That is, assume there exists a Turing Machine T to decide L_H. Since decidable languages are closed under complementation, $\overline{L_H} = \{\rho(M), \omega : \text{M does not halt on } \omega\}$ is also decidable. Consider L_{diag}. For each string $x_k \in L_{diag}$, compose strings $\rho(M_k), x_k$ and let $L'_{diag} = \{\rho(M_k), x_k : M_k \text{ does not halt on } x_k\}$, which is a subset of $\overline{L_H}$. It follows that since L_H can be decided by assumption, L'_{diag} can be decided too. This implies that L_{diag} can be decided, which is a contradiction, since L_{diag} is not recursively enumerable. Thus, L_H is undecidable. QED.

The Halting Problem is commonly used in proofs showing certain other languages are undecidable. The idea is to prove such languages are undecidable, using proof by contradiction. It is then shown that the given language being decidable implies the existence of a Turing Machine to decide L_H, which is known to be a contradiction. A common mistake when reducing one language (problem) to another language (problem) is simply reducing one problem to a known hard problem. This proves nothing about the original problem. Instead, it must be shown that the reduction yields a solution to the known hard problem, which produces the necessary contradiction.

Some of these problems will now be explored.

- Consider the language $L_{ES} = \{\rho(M) : \epsilon \in L(M)\}$. L_{ES} is a recursively enumerable, but not decidable language. It is easy to show that L_{ES} is recursively enumerable. Let T be a Turing Machine to accept L_{ES}. T operates by invoking the Universal Turing Machine to run M, from the input $\rho(M)$, on an empty tape. T accepts $\rho(M)$ if and only if M halts on the empty tape.

  Showing that L_{ES} is undecidable is a bit trickier. It is done by contradiction, with reduction to the halting problem. The idea is to assume the existence of a TM, M_{ES}, that decides L_{ES}. Then, it suffices to find a Turing Machine such that it is impossible to decide if it contains the empty string. Thus, L_{ES} is undecidable.

  Consider an arbitrary Turing Machine M, and input string ω, and construct a Turing Machine M_ω. The TM M_ω, when given the empty string as input, writes ω to the tape and simulates M on ω. M_ω accepts the empty string if and only if M halts on ω. It has already been shown that it is impossible to decide if M halts on ω, as the Halting Problem is undecidable. Thus, it cannot be decided if $L(M)$ contains the empty string, and $\rho(M) \in L_{ES}$. It follows that since $L(M)$ is undecidable that L_{ES} is undecidable.

- Let $L_{empty} = \{\rho(M) : L(M) = \emptyset\}$. This language is undecidable. The idea in proving L_{empty} undecidable is similar to showing L_{ES} to be undecidable. That is, it suffices to show the existence of a Turing Machine where deciding if its language is empty will produce a solution to the Halting Problem. The clever bit is in designing such a Turing Machine. Consider an arbitrary Turing Machine M, and an arbitrary string ω. Now construct a Turing Machine M_ω. Given an input string x, it ceases to halt if $x \neq \omega$. Otherwise, if $x = \omega$, then it simulates M on ω. Thus, $L(M_\omega)$ is empty if and only if M does not accept ω. Thus, L_{empty} is decidable if and only if the Halting Problem is decidable. It has already been shown that the Halting Problem is undecidable; and thus, the contradiction.

  It follows that since L_{empty} is undecidable that determining if two languages of Turing Machines are equal is undecidable. It simply reduces to the case showing that $L(M_1) = \emptyset$ is undecidable, where M_1 is a Turing Machine.

- Let $L_{reg} = \{\rho(M) : L(M) \text{ is regular }\}$. This language is undecidable. This proof of this is another proof by contradiction. The idea is to find a Turing Machine whose language is regular, but that it cannot be decided as such. The construction will be a Turing Machine whose language is conditionally regular, based on whether or not another Turing Machine accepts an arbitrary input string. This condition is undecidable, thus L_{reg} is undecidable.

  Let T be a Turing Machine and let ω be an input string. Now construct a second Turing Machine, T', which will work in the following way. If the input string x is of the form 0^n1^n, for $n \in \mathbb{N} \cup \{0\}$, then T' accepts it. Otherwise, T' simulates T on ω. T' accepts x if M accepts ω. Thus, if T accepts ω, then $L(T') = \Sigma^*$, which is regular. Otherwise, $L(T') = \{0^n1^n : n \geq 0\}$, which is strictly a context-free grammar. Determining if T accepts ω is undecidable, determining if $L(T')$ is regular is also undecidable. Thus, L_{reg} is undecidable.

The language L_{reg} is a special case of Rice's Theorem. In many texts, Rice's Theorem is stated as "any non-trivial property is undecidable," where a trivial property is one that applies to either all languages in RE or no languages in RE. This formalizes quite well to the following: Let RE be the set of Recursively Enumerable Languages, and let $C \subset RE$, $C \neq \emptyset$. Then C is undecidable.

The idea in the proof of Rice's Theorem is quite similar to the proof that L_{reg}, by constructing a Turing Machine T whose language is conditional on a secondary Turing Machine. That condition relies on the Halting Problem being decidable, which is known not to be the case. Hence, the contradiction.

It is important as well to remember that the set of decidable languages is closed under complementation. So if C is decidable, then $\overline{C}$ is decidable as well. Without loss of generality, let the language $L_{null} = \emptyset \in C$. If $L_{null} \in \overline{C}$, the Turing Machine to decide C will still have to decide if L_{null} is (not) present in C.

Since C is a proper subset of RE, $\overline{C}$ is non-empty. Consider a language L' in $\overline{C}$, with $L' \neq \emptyset$. Since both L_{null} and L' are recursively enumerable, there exist Turing Machines M_{null} and M' such that $L(M_{null}) = L_{null}$, and $L(M') = L'$.

The idea now is to construct a Turing Machine T, such that $L(T) = \emptyset$ or $L(T) = L'$, based on another Turing Machine, M. Given M and an arbitrary string, ω, T will operate in the following way. Given an input x, T first simulate M on ω. If M rejects ω, then M' rejects ω. If M accepts ω, then T simulates M' on x, accepting x if and only if M' accepts x. Thus, $L(T) = \emptyset$ if M does not accept ω, and $L(T) = L'$ otherwise.

In order to decide if $L(T)$ is in C, it is necessary to decide if M halts on ω and accepts it. Given that the Halting Problem is undecidable, it follows that C is undecidable.