**Procedures Versus Algorithms**

There are two senses in which a TM accepts a language.

1. The TM accepts the strings in the language (by final state), but does not halt on some of the strings not in the language.
   - Thus, we can never be sure whether those strings are rejected, or eventually will be accepted.
   - A language accepted in this way is called recursively enumerable (RE).
   - Note: this notion is the normal “accepted by a TM” notion.
   - The TM is sometimes referred to as a procedure.

2. The TM accepts by final state, but halts on every string, whether or not it is accepted.
   - A language accepted this way is called recursive.
   - As a problem, the question is called decidable.
   - The TM is called an algorithm.

**Plan**

1. Show a particular language not to be RE.
   - Like the “hello-world” argument, we show no TM can tell whether a given TM halts on a given input — the proof is by “diagonalization,” or self-reference.

2. Use the non-RE language from (1) to show another language to be RE, but not recursive.
   - Trick: if a language and its complement are both RE, then they are both recursive.
   - Thus, if a language $L$ is RE, but its complement is not, then $L$ is not recursive.

**TM’s as Integers**

We shall focus on TM’s whose input alphabet is \{0, 1\}. Each such TM can be represented by one or more integers, using the following code:

- Assume the states are \{q_1, q_2, \ldots\}. Represent $q_i$ by $0^i$.
- Assume the tape symbols are \{X_1, X_2, \ldots\}, where the first three of these are 0, 1, and $B$, in that order. Represent $X_i$ by $0^{|i|}$.
- Represent directions $L$ and $R$ by 0 and 00, respectively, and refer to them as $L = D_1$, $R = D_2$.
- Represent a rule of the TM $\delta(q_i, X_j) = (q_k, X_l, D_m)$ by $0010^{|i|}0^{|j|}10^{|k|}10^{|l|}10^{|m|}$.
- Represent the whole TM by $111C_111C_211 \cdots 11C_n111$, where $C_i$ is the code for one of the $\delta$ rules, in any order.
   - This string is some integer in binary, so we can call the TM $M_i$, where $i$ is that integer.
   - Conversely, every integer $i$ can be said to describe some TM $M_i$.
   - If $i$ in binary is not of the right form (111code\ldots), then $M_i$ is the TM with no moves. Thus, $H(M_i)$ is $L(0 + 1)^*$.
   - Note that many integers represent the same TM, but that is neither good nor bad.

**The Diagonalization Language**

Define $L_d$ to be the set of binary strings $w$ with the following properties:

1. First, let $i$ be the integer that is $1w$ in binary.
   - Refer to $w$ as the “ith string,” or $w_i$.
2. Then $w_i$ is in $L_d$ if and only if $w_i$ is not in $H(M_i)$.

**Proof $L_d$ is not RE**

Suppose $L_d$ is RE. Then $L_d = H(M)$ for some TM $M$.

- Since the input alphabet of $M$ is \{0, 1\}, $M$ is $M_j$ for at least one value of $j$.
- Let $x$ be the $j$th string; i.e., $1x$ is $j$ in binary.
- Question: is $x$ in $L_d$?
   - Suppose so. Then $x$ is not in $H(M_j)$, by definition of $L_d$. But $H(M_j) = H(M) = L_d$, so $x$ is not in $L_d$ (Contradiction).
   - Suppose not. Then $x$ is in $H(M_j)$ by definition of $L_d$. But $H(M_j) = H(M) = L_d$, so $x$ is in $L_d$ (Contradiction).
• Since we derive a contradiction in either case, we conclude that our assumption \(H(M) = L_d\) was wrong, and in fact, there is no such TM \(M\).

Rules About Complements
Let \(L\) and \(\overline{L}\) be a language and its complement with respect to alphabet \(\{0, 1\}\).
• If \(L\) is recursive, so is \(\overline{L}\).
  ♦ Proof: Find a TM \(M\) that accepts \(L\) by final state but always halt. Arrange for a TM \(M'\) to simulate \(M\), but accept if and only if \(M\) halts before accepting.
• If \(L\) and \(\overline{L}\) are RE, then both are recursive.
  ♦ Proof: Simulate TM's for both \(L\) and \(\overline{L}\) on separate tracks. One or the other is guaranteed to accept, so the simulating TM can always be made to halt.

The Universal Language
\(L_u\) is the set of binary strings consisting of a code for some TM \(M\) followed by some binary string \(w\), such that \(w\) is in \(H(M)\).
• Proof in reader that \(L_u\) is RE.
  ♦ In essence: a TM can be treated as a stored-program device, just like a real computer.
  ♦ Hard part of proof: Since \(M\) may have any number of states and tape symbols, one multitape TM \(M\) cannot simulate these states and symbols directly. Rather, it represents them as strings of 0's (as in the code we developed) and compares using scratch tapes.
• Proof \(L_u\) is not recursive: show \(\overline{L_u}\) is not RE.
  ♦ Remember, if \(L_u\) were recursive, then \(\overline{L_u}\) would be recursive, and therefore RE.
• Proof that \(\overline{L_u}\) is not RE:
  ♦ A reduction from \(L_u\) to \(\overline{L_u}\): Show that if there is a TM for \(\overline{L_u}\), then there is a TM for \(L_u\) (which we know there isn't).
  ♦ Transform \(w\) by first checking that \(1w\) represents some TM \(M\) (i.e., it is of the form 111<code>11111). If so, produce \(1ww\) as input to a hypothetical \(L_u\) TM. If not, reject \(w\), since \(1w\) represents a TM that accepts everything.
• If \(1ww\) is produced, simulate the \(L_u\) TM on this input. If it accepts, then TM \(M\) (represented by \(1w\)) does not accept the \(w\)th string, \(w\), so \(w\) is in \(L_u\).
• If \(1ww\) is not in \(\overline{L_u}\), then \(M\) does accept \(w\), so \(w\) is not in \(L_u\).

Summary:
• \(L_u\) is undecidable (not recursive), and in fact not RE.
• \(L_u\) is undecidable, but RE.
• \(\overline{L_u}\) is like \(L_u\), not RE.
• \(\overline{L_u}\) is like \(L_u\), RE, although we did not prove this.

Rice's Theorem
Essentially, any nontrivial property of the language of a TM is undecidable.
• Note the difference between a property of \(L(M)\) from a property about \(M\):
  ♦ Example: \(L(M) = \emptyset\) is a property of the language.
  ♦ Example: “\(M\) has at least 100 states” is a property of the TM itself.
  ♦ “\(= \emptyset\)” is undecidable; “has 100 states” is easily decidable, just look at the code for \(M\) and count.

Properties
A property of the RE languages is a set of strings, those that represent TM’s in a certain class.
• Example: the property “is context-free” is the set of codes for all TM’s \(M\) such that \(L(M)\) is a CFL.
• The property is “of languages” if TM’s whose languages are the same either all have the property or none do.

Proof of Rice’s Theorem
Let \(P\) be any nontrivial property of the RE languages; i.e., at least one RE language has the property, and at least one does not.
• We shall prove that \(P\) (as a language, i.e., a set of TM codes) is undecidable.
• Assume \( \emptyset \) does not have property \( P \).
  ♦ If it does, consider \( \overline{P} \). \( P \) is decidable if and only if \( \overline{P} \) is.

• Suppose \( P \) is decidable. Assume \( L \) is a language with property \( P \), and \( \emptyset \) is a language without property \( P \). We can decide \( L_m \) (something we know is impossible) as follows.
  ♦ Given \((M, w)\), test if \( w \) is in \( H(M) \) as follows. First, we shall construct a TM \( N \) to accept either \( \emptyset \) or \( L \), depending on whether \( M \) accepts \( w \).
  ♦ \( N \) simulates \( M \) on \( w \). Note that \( w \) is not input to \( N \); rather \( N \) writes \( w \) on a scratch tape and simulates \( M \) which is part of \( N \)'s own states.
  ♦ If \( M \) accepts \( w \), \( N \) then simulates a TM \( M_L \) for language \( L \) on \( N \)'s own input \( x \). If \( M_L \) accepts \( x \) then \( N \) accepts \( x \).
  ♦ If \( M \) never accepts \( w \), \( N \) never gets to simulate \( M_L \), and therefore accepts \( \emptyset \).
  ♦ Feed the constructed \( N \) to the hypothetical \( P \) tester. Accept \((M, w)\) if and only if \( N \) has property \( P \).

Consequences of Rice's Theorem
We cannot tell if a TM:
• Accepts \( \emptyset \).
• Accepts a finite language.
• Accepts a regular language, a context free language, etc., etc.

Reductions
To prove a problem \( P_1 \) to be hard in some sense (e.g., undecidable), we can reduce \( P_2 \), a known hard problem, to \( P_1 \).
• For each instance \( w \) (string in) \( P_2 \), we construct an instance \( x \) of \( P_2 \), using some fixed algorithm.
  ♦ The same algorithm must also turn a string \( w \) that is not in \( P_2 \) into a string \( x \) that is not in \( P_1 \).
• We can then argue that if \( P_1 \) were decidable, we could use the algorithm in which we transformed \( w \) to \( x \) and then tested \( x \) for membership in \( P_1 \) as a way to decide \( P_2 \).
  ♦ Since \( P_2 \) is undecidable, we have a contradiction of the assumption \( P_1 \) is decidable.

• The same idea works for showing \( P_1 \) not to be RE, but now \( P_2 \) must be non-RE, and the transformation from instances of \( P_2 \) to instances of \( P_1 \) may be a procedure, not necessarily an algorithm.

• Common error: trying to do the reduction in the wrong direction.