Recursive and Recursively Enumerable Languages

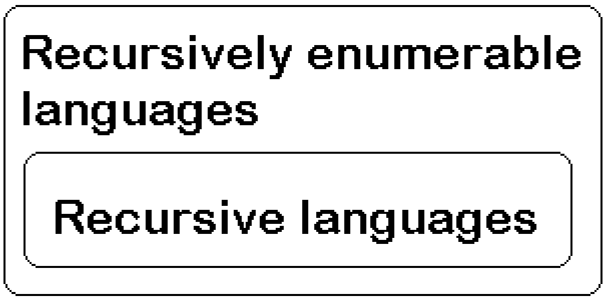
Remember that there are *three* possible outcomes of executing a Turing machine over a given input. The Turing machine may

- Halt and accept the input;
- Halt and reject the input; or
- Never halt.

A language is *recursive* if there exists a Turing machine that accepts every string of the language and rejects every string (over the same alphabet) that is not in the language.

Note that, if a language L is recursive, then its complement -L must also be recursive. (Why?)

A language is *recursively enumerable* if there exists a Turing machine that accepts every string of the language, and does not accept strings that are not in the language. (Strings that are not in the language may be rejected or may cause the Turing machine to go into an infinite loop.)



Clearly, every recursive language is also recursively enumerable. It is not obvious whether every recursively enumerable language is also recursive.

Non-Recursively Enumerable Languages

A language is a subset of Σ*. A language is *any* subset of Σ*.

We have shown that Turing machines are enumerable. Since recursively enumerable languages are those whose strings are accepted by a Turing machine, the set of recursively enumerable languages is also enumerable.

We have shown that the powerset of an infinite set is *not* enumerable -- that it has more than $\aleph_0$ subsets. Each of these subsets represents a language. Therefore, there must be languages that are not computable by a Turing machine.

According to Turing's thesis, a Turing machine can compute any effective procedure. Therefore, there are languages that cannot be defined by any effective procedure.

We can find a non-recursively enumerable language X by diagonalization. Using the enumerations described earlier, let string i belong to language X if and only if it does *not* belong to language i.

Problem. I've just defined a procedure for defining a non-recursively enumerable language. Isn't this a contradiction?

When Recursively Enumerable Implies Recursive

Suppose a language L is recursively enumerable. That means there exists a Turing machine T_1 that, given any string of the language, halts and accepts that string. (We don't know what it will do for strings *not* in the language -- it could reject them, or it could simply never halt.)

Now let's also suppose that the complement of L, -L = {w: w $\notin$L}, is recursively enumerable. That means there is some other Turing machine T_2 that, given any string of -L, halts and accepts that string.

Clearly, any string (over the appropriate alphabet Σ) belongs to either L or -L. Hence, any string will cause either T_1 or T_2 (or both) to halt. We construct a new Turing machine that emulates both T_1 and T_2, alternating moves between them. When either one stops, we can tell (by whether it accepted or rejected the string) to which language the string belongs. Thus, we have constructed a Turing machine that, for each input, halts with an answer *whether or not* the string belongs to L. Therefore L and -L are recursive languages.

We have just proved the following **theorem:** If a language and its complement are both recursively enumerable, then both are recursive.

Recursively Enumerable But Not Recursive

We know that the recursive languages are a subset of the recursively enumerable languages, We will now show that they are a *proper* subset.

We have shown how to enumerate strings for a given alphabet, w_1, w_2, w_3, ... We have also shown how to enumerate Turing machines, T_1, T_2, T_3, ... (Recall that each Turing machine defines a recursively enumerable language.) Consider the language

$L = \{w_i: w_i \in L(T_i)\}$

A little thought will show that L is itself recursively enumerable. But now consider its complement:

$-L = \{w_i: w_i \notin L(T_i)\}$

If -L is recursively enumerable, then there must exist a Turing machine that recognizes it. This Turing machine must be in the enumeration somewhere -- call it T_k.

Does w_k belong to L?

- If w_k belongs to L then (by the way we have defined L) T_k accepts this string. But T_k accepts only strings that do not belong to L, so we have a contradiction.
- If w_k does not belong to L, then it belongs to -L and is accepted by T_k. But since T_k accepts w_k, w_k must belong to L. Again, a contradiction.

We have now defined a recursively enumerable language L and shown by contradiction that -L is not recursively enumerable.

We mentioned earlier that if a language is recursive, its complement must also be recursive. If language L above were recursive, then -L would also be recursive, hence recursively enumerable. But -L is not recursively enumerable; therefore L must not be recursive.

We have therefore shown that L is recursively enumerable but not recursive, therefore the set of recursive languages is a proper subset of the set of recursively enumerable languages.

Decidable Languages about DFA

A_{DFA} is a decidable language.

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

```
M = "On input $\langle B, w\rangle$, where $B$ is a DFA and $w$ is a string:
    1. Simulate $B$ on input $w$.
    2. If the simulation ends in an accept state, accept. If it ends in a
       nonaccepting state, reject."
```

A_{NFA} is a decidable language.

$A_{\text{NFA}} = \{\langle B, w\rangle \mid B \text{ is an NFA that accepts input string } w\}$.

```
N = "On input $\langle B, w\rangle$ where $B$ is an NFA, and $w$ is a string:
    1. Convert NFA $B$ to an equivalent DFA $C$, using the
       last procedure .
    2. Run TM $M$ on input $\langle C, w\rangle$.
    3. If $M$ accepts, accept; otherwise, reject."
```

A_{REX} is a decidable language.

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

```
P = "On input $\langle R, w\rangle$ where $R$ is a regular expression and $w$ is a string:
    1. Convert regular expression $R$ to an equivalent DFA $A$
    2. Run TM $N$ on input $\langle A, w\rangle$.
    3. If $N$ accepts, accept; if $N$ rejects, reject."
```

Haltin

Accep

Does a Turing machine accept an input string?

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}.$

A_{TM} is recursively enumerable.

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w. \end{cases}$$

$D =$ "On input $\langle M\rangle$, where M is a TM:
1. Run H on input $\langle M, \langle M\rangle\rangle$.
2. Output the opposite of what H outputs; that is, if H accepts, *reject* and if H rejects, *accept*."

$$D(\langle M\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M\rangle. \end{cases}$$

$$D(\langle D\rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D\rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D\rangle. \end{cases}$$

The Halting Problem:

Review: Turing machines can be encoded as strings, and other Turing machines can read those strings to perform \simulations".

Definition 1. *A language is* **Turing-recognizable** *if there exists a Turing machine which halts in an accepting state iff its input is in the language.*

Definition 2. *A language is* **Turing-decidable** *if it halits in an accepting state for every input in the language, and halts in a rejecting state for every other input.*

Intuitively, for recognizability we allow our TM to run forever on inputs that are not in the language, while for decidability we require that the TM halt on every input.

Now recall our two most important results:

Theorem 1. *There exist (uncountably many!) languages which are not Turing-recognizable.*

Proof. (intuitive) There are as many strings as natural numbers, because every (finite) string over a finite alphabet can be encoded as a binary number. There are as many TMs as strings, because every TM can be encoded as a string. Thus, both the strings and the TMs are countably infinite.

There are as many languages as real numbers. Every language is a subset of the set of strings; we can think of this subset as being encoded by an infinite binary sequence (i.e. a real number) with 1s at indices corresponding to strings in the language and 0s everywhere else.

Thus there are more languages than Turing machines; we conclude that some (indeed, "most") languages are not recognized by any TM.

Now we will construct a specific undecidable language.

Definition 3. *The language XT M = {(M) : M does not accept (M)}*

Theorem 2. *XT M is not Turing-decidable.*

Question. *Give the "paradox" proof.*

Proof 1. (by Epimenides' paradox) Suppose we had a Turing machine *M* deciding this lan- guage. What happens when we run *M* with input *(M)*? If we claim it accepts, then by definition it ought to reject; if it rejects, then it ought to accept! Either way, we have a contradiction. □

Alternatively, here's a proof that it's not even Turing-recognizable.

Question. *Give the "diagonalization" proof.*

Proof 2. (by diagonalization) Suppose we constructed an (infinite) chart listing all the Turing machines and all encodings of Turing machines. We write 1 in cell [*i, j*] if *Mi* accepts *(Mj)* and 0 otherwise:

	(M1)	*(M2)*	*(M3)*	· · ·
M_1	0	1	1	
M_2	1	1	1	
M_3	1	0	1	
⋱				⋱

(The particular arrangement of 1s and 0s is unimportant—it will depend on our encoding scheme.) Note that by reading the 1s from the *i*th row, we get the language decided by *Mi*. Does *XT M* appear in any row of this list? Not the first row: *M*1 rejects *(M*1*)*, so *(M*1*)* **must be in** *XT M* . Not the second row: *M*2 accepts *(M*2*)*, so *(M*2*)* **must not be in** *XT M* . Continuing with this procedure, we can show that *XT M* is not the same as any row of this enumeration of TMs, and as a consequence is not decided by any TM. □

The Accepting Problem

XT M is admittedly a rather strange language, and it's not obvious why we should really care that it's not recognizable. Let's look at a different one:

Definition 4. *The language AT M* = (*{(M), w*) : *M accepts w}*

The question of whether *AT M* is decidable is precisely the question of whether one TM can predict the output of another TM!

Question. *Is this language Turing-recognizable?*

Yes. Given *(M), w* as input, we can just simulate *M* on *w*.

Question. *Is this language Turing-decidable?*

No.

Decidability:

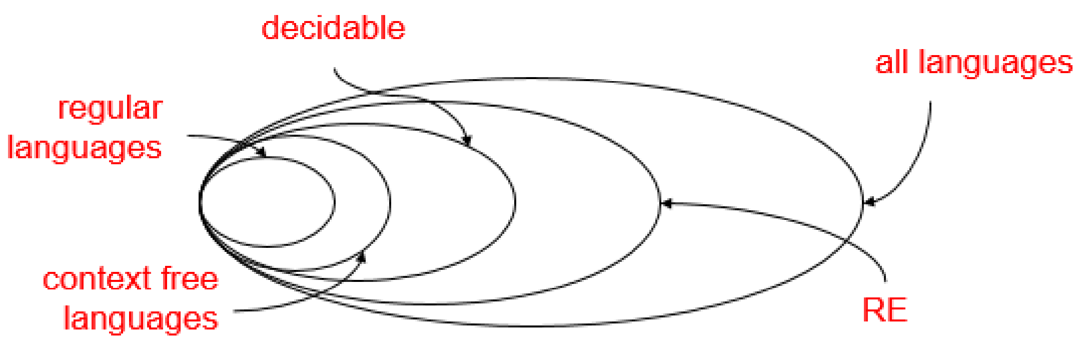
Turing decidability

L is *Turing decidable* (or just decidable) if there exists a Turing machine M that accepts all strings in L and rejects all strings not in L. Note that by rejection we mean that the machine halts after a finite number of steps and announces that the input string is not acceptable. Acceptance, as usual, also requires a decision after a finite number of steps.

Turing Recognizability

L is *Turing recognizable* if there is a Turing machine M that recognizes L, that is, M should accept all strings in L and M should not accept any strings not in L. This is not the same as decidability because recognizability does not require that M actually reject strings not in L. M may reject some strings not in L but it is also possible that M will simply "remain undecided" on some strings not in L; for such strings, M's computation never halts.

- Recursion and Recursive Functions
- Enumerable Sets
- Recursively Enumerable Languages
- Recursive Languages
- Non-Recursively Enumerable Languages



decidable ⊂ RE ⊂ all languages