1 Undecidability

Undecidability

Definition 1. A language L is *undecidable* if L is not decidable. Thus, there is no Turing machine M that halts on every input and $L(M) = L$.

- This means that either L is not recursively enumerable. That is there is no turing machine M such that $L(M) = L$, or
- L is recursively enumerable but not decidable. That is, any Turing machine M such that $L(M) = L$, M does not halt on some inputs.

Big Picture

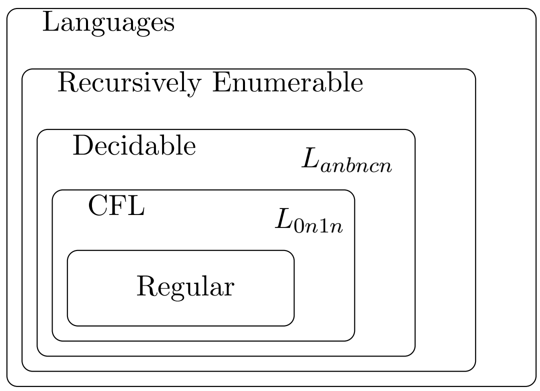


Figure 1: Relationship between classes of Languages

1.1 Diagonalization

The Diagonal Language

Definition 2. Define $L_d = \{\langle M \rangle \mid \langle M \rangle \notin \mathbf{L}(M)\}$. Thus, L_d is the collection of Turing machines (programs) M such that M does not halt and accept when given itself as input.

A non-Recursively Enumerable Language

Diagonalization: Cantor

Proposition 3. *L_d is not recursively enumerable.*

Proof. Recall that,

- Inputs are strings over $\{0, 1\}$
- Every Turing Machine can be described by a binary string and every binary string can be viewed as Turing Machine
- In what follows, we will denote the ith binary string (in lexicographic order) as the number i. Thus, we can say $j \in \mathbf{L}(i)$, which means that the Turing machine corresponding to ith binary string accepts the jth binary string.
- We can organize all programs and inputs as a (infinite) matrix, where the (i, j)th entry is Y if and only if $j \in \mathbf{L}(i)$.

Inputs $\longrightarrow$

TMs $\downarrow$	1	2	3	4	5	6	7	$\cdots$
1	[N]	N	N	N	N	N	N	
2	N	[N]	N	N	N	N	N	
3	Y	N	[Y]	N	Y	Y	Y	
4	N	Y	N	[Y]	Y	N	N	
5	N	Y	N	Y	[Y]	N	N	
6	N	N	Y	N	Y	[N]	Y	

- Suppose L_d is recognized by a Turing machine, which is the jth binary string. i.e., $L_d = \mathbf{L}(j)$. But $j \in L_d$ iff $j \notin \mathbf{L}(j)$!

$\square$

Acceptor for L_d?

Consider the following program

```
On input ⟨M⟩
    Run program M on ⟨M⟩
    Output ``yes'' if M does not accept ⟨M⟩
    Output ``no'' if M accepts ⟨M⟩
```

The above program does not recognize L_d because it may never output "yes" if M does not halt on $\langle M \rangle$.

Models for Decidable Languages

Question
Is there a machine model such that

- all programs in the model halt on all inputs, and
- for each problem decidable by a TM, there is a program in the model that decides it?

Answer

There is no such model! Suppose there is a programming language in which all programs always halt. Programs in this language can be described by binary strings, and can be simulated by TMs.

Consider the Turing Machine M_d

```
On input ⟨M⟩
    Run program M on ⟨M⟩
    Output ‘‘yes’’ if M does not accept ⟨M⟩
    Output ‘‘no’’ if M accepts ⟨M⟩
```

M_d always halts and solves a problem not solved by any program in our language! Inability to halt is *essential* to capture all computation.

1.2 The Universal Language

Recursively Enumerable but not Decidable

- L_d not recursively enumerable, and therefore not decidable. Are there languages that are recursively enumerable but not decidable?
- Yes, $A_{\rm TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

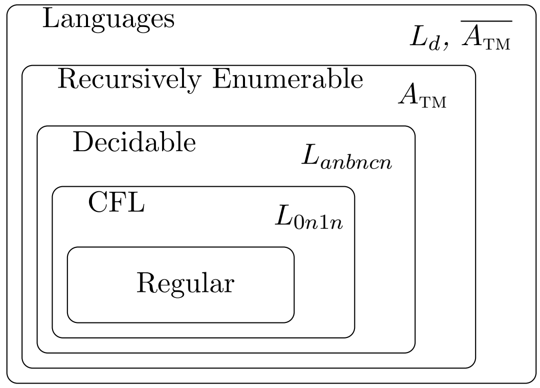
Proposition 4. *$A_{\rm TM}$ is r.e. but not decidable.*

Proof. We have already seen that $A_{\rm TM}$ is r.e. Suppose (for contradiction) $A_{\rm TM}$ is decidable. Then there is a TM M that always halts and $\mathbf{L}(M) = A_{\rm TM}$. Consider a TM D as follows:

```
On input ⟨N⟩
    Run M on input ⟨N,⟨N⟩⟩
    Output ‘‘yes’’ if M rejects ⟨N,⟨N⟩⟩
    Output ‘‘no’’ if M accepts ⟨N,⟨N⟩⟩
```

Observe that $\mathbf{L}(D) = L_d$! But, L_d is not r.e. which gives us the contradiction. □

A more complete Big Picture



---

2 Reductions

Reductions

A *reduction* is a way of converting one problem into another problem such that a solution to the second problem can be used to solve the first problem. We say the first problem *reduces* to the second problem.

- Informal Examples: Measuring the area of rectangle reduces to measuring the length of the sides; Solving a system of linear equations reduces to inverting a matrix
- The problem L_d reduces to the problem A_{TM} as follows: "To see if $\langle M \rangle \in L_d$ check if $\langle M, \langle M \rangle \rangle \in A_{\text{TM}}$."

Undecidability using Reductions

Proposition 5. *Suppose L_1 reduces to L_2 and L_1 is undecidable. Then L_2 is undecidable.*

Proof Sketch.
Suppose for contradiction L_2 is decidable. Then there is a M that always halts and decides L_2. Then the following algorithm decides L_1

- On input w, apply reduction to transform w into an input w' for problem 2
- Run M on w', and use its answer.

This can be seen Pictorially as follows.

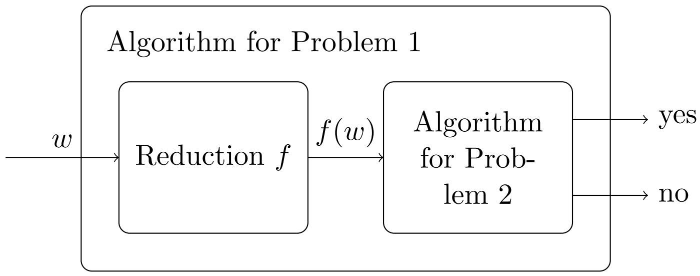


Figure 2: Reductions schematically

The Halting Problem

Proposition 6. *The language HALT* $= \{\langle M, w\rangle \mid M$ *halts on input* $w\}$ *is undecidable.*

Proof. We will reduce A_{TM} to HALT. Based on a machine M, let us consider a new machine $f(M)$ as follows:

```
On input x
    Run M on x
    If M accepts then halt and accept
    If M rejects then go into an infinite loop
```

Observe that $f(M)$ halts on input w if and only if M accepts w

Suppose HALT is decidable. Then there is a Turing machine H that always halts and $\mathbf{L}(H) =$ HALT. Consider the following program T

```
On input ⟨M, w⟩
    Construct program f(M)
    Run H on ⟨f(M), w⟩
    Accept if H accepts and reject if H rejects
```

T decides A_{TM}. But, A_{TM} is undecidable, which gives us the contradiction. □
