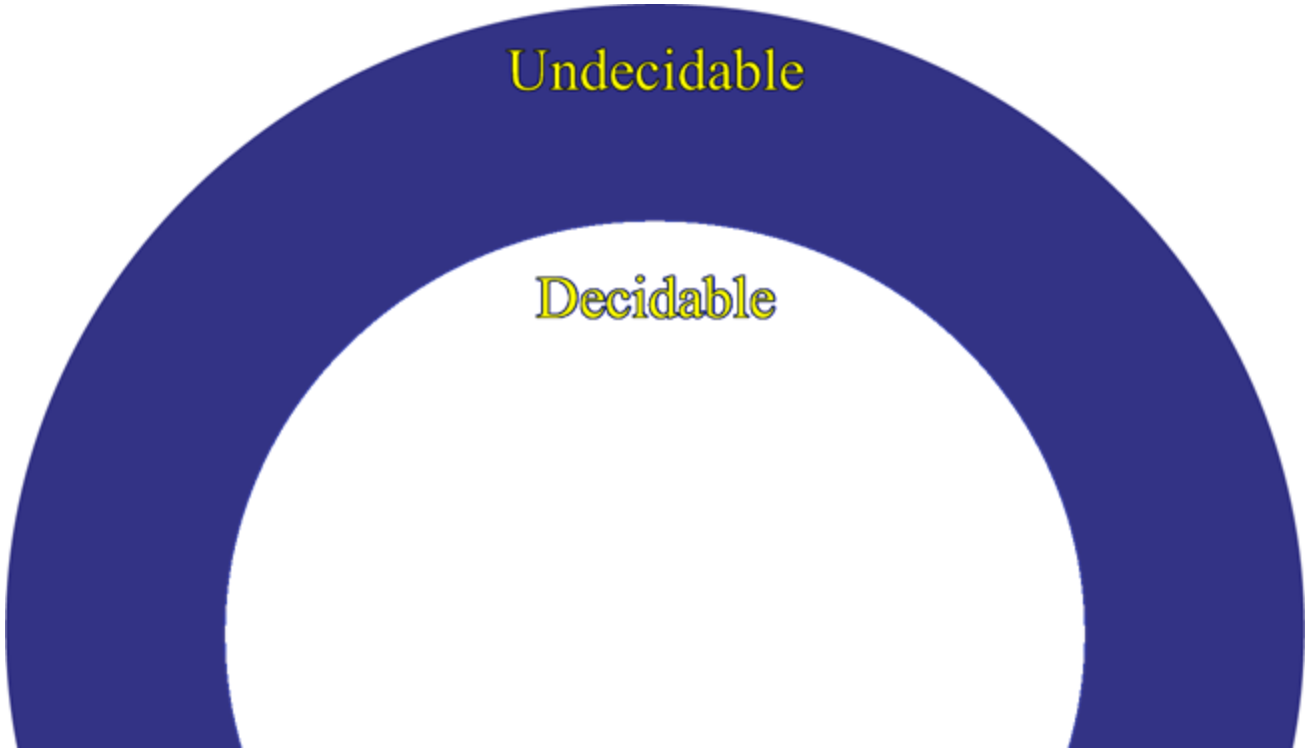


Theory of Computation

Lesson 11

Decidable & Undecidable Languages

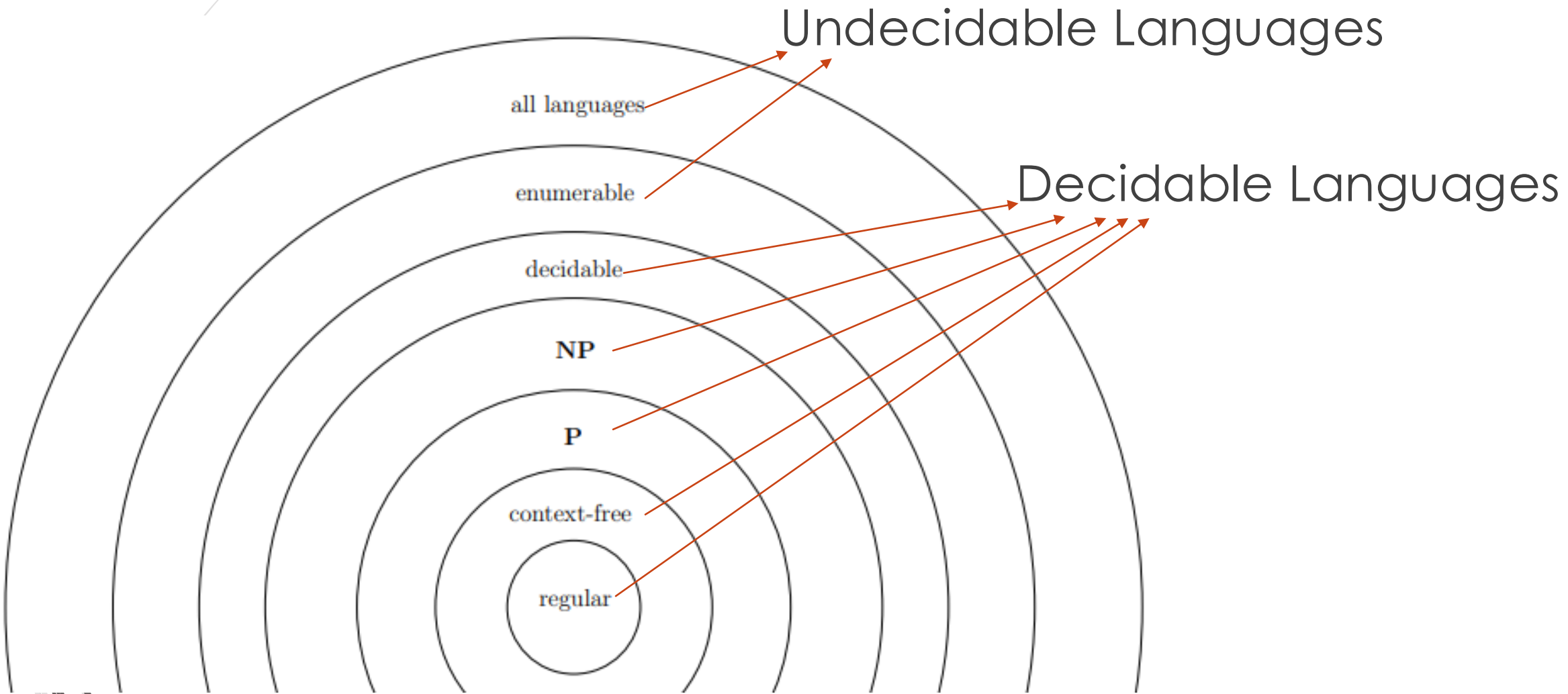
Decidability

The main titles you will hear within the scope of this chapter are as follows:

- Decidable Languages
- *Enumerable Languages*
- Undecidable Languages

In some sources, the terms *Recursively Enumerable* or *Turing Recognizable* is used instead of *Enumerable*. Don't be confused, they all represent the same thing.

Decidability



Decidable Languages

On a word entered for testing L-language, a Turing Machine (TM) can do one of following:

- stops and accepts,
- stops and rejects,
- or cannot stop.

If a TM prepared for an L-language guarantees to stop by making a decision of accept or reject on every word entered by a user, then the L-language is called **Decidable Language**.

Enumerable (Recognizable) Languages

L-language is called recursively enumerable or Turing recognizable, if a Turing Machine prepared for L

- **stops by accepting** words that are elements of L-language,
- **but cannot guarantee stopping** on words that are not elements (sometimes **rejecting** but sometimes not terminating)

Undecidable Languages

If a Turing Machine cannot terminate on L-language, in other words neither

- If any TM cannot guarantee terminate in words that are elements of the language,
- If any TM cannot guarantee stopping on words that are not elements of the language,

That L-language is called as **Undecidable**.

A_{DFA} Language with TM

We can implement any DFA language on JFLAP, and the code can decide a word is element of that language or not. We can be sure that our code will terminate absolutely. Because we know that, there is no loop-forever in DFA. It has states and each state means accept or reject.

DFA language is decidable.

A_{NFA} Language with TM

We learned that DFA and NFA are equivalent to each other. With this information alone, you can interpret that, if DFA is decidable, then NFA must be decidable too. Because every NFA can be transformed into a DFA, we can prove this theoretically.

NFA language is decidable.

A_{CFG} Language with TM

In order to find out if it is decidable, we can consider Chomsky normal form. If we transform a CFG into Chomsky normal form, for a word with n length, it is guaranteed that it terminates in 2n-1 steps. Thus, we have proved the following result:

CFG language is decidable.

A_{TM} Language with TM

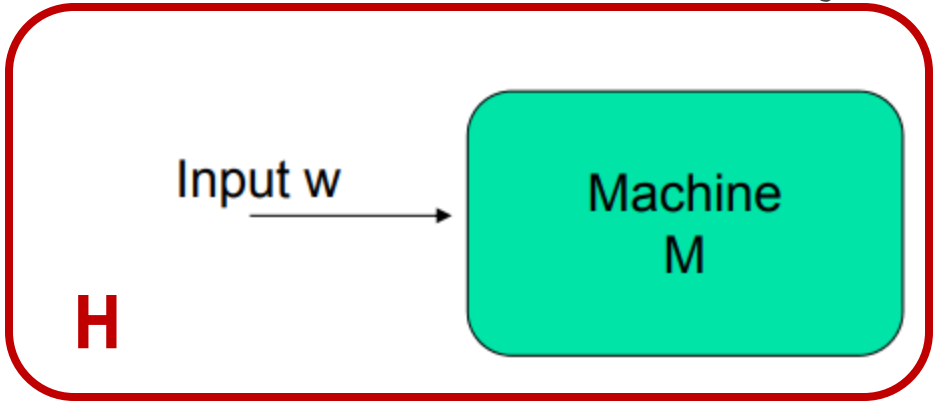
We can start by considering the JFLAP implementation again. When a code written with a TM is loaded into JFLAP,

- Can JFLAP detect that the code has an infinite-loop?
- Can JFLAP **guarantee terminating** on every TM code?

A_{TM} Language with TM

We assume that A_{TM} is decidable. Then there exists a Turing machine H that has the following property. For every input string <M, w> for H:

- If <M, w> $\in A_{TM}$ (i.e., M accepts w), then H terminates in its accept state.
- If <M, w> $\notin A_{TM}$ (i.e., M rejects w or M does not terminate on input w), then H terminates in its reject state.



A_{TM} Language with TM

We create a new Turing machine D to run on <M>. You can think of D as a compiler and M is just another TM code to run on D. The D compiler is executed as follows:

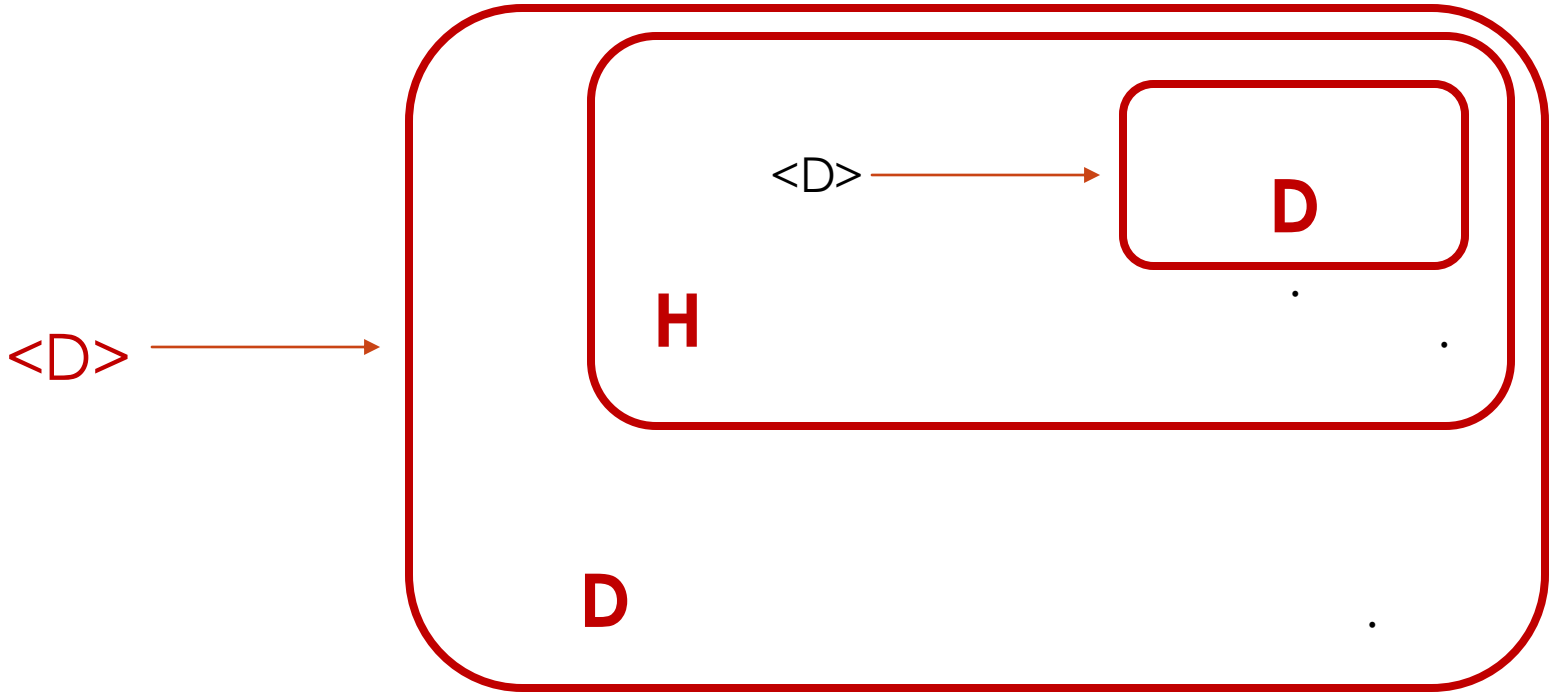
- Run the Turing machine H on the input <M,<M>>.
- If H terminates in its accept state, then D terminates in its reject state.
- If H terminates in its reject state, then D terminates in its accept state.

A_{TM} Language with TM

Assume that we load D itself as code into the D compiler again. The D compiler was already testing the loaded code by sending its own codes as strings via H. So there is a **contradiction** here.

- If D accepts <D>, then H terminates in its accept state, thus D rejects <D>.
- If D rejects <D> or does not terminate, then H terminates in its reject state, thus D accepts <D>.

A_{TM} Language with TM



If D accepts <D>, then H terminates in its accept state, thus D rejects <D>.
If D rejects <D> or does not terminate, then H terminates in its reject state, thus D accepts <D>.

Halting Problem

Based on the Church-Turing thesis, we can say that the same problem can occur in a Java program. We define the following language:

- Halt = {<P, w>: P is a Java program that terminates on the input string w}.

Halting Problem

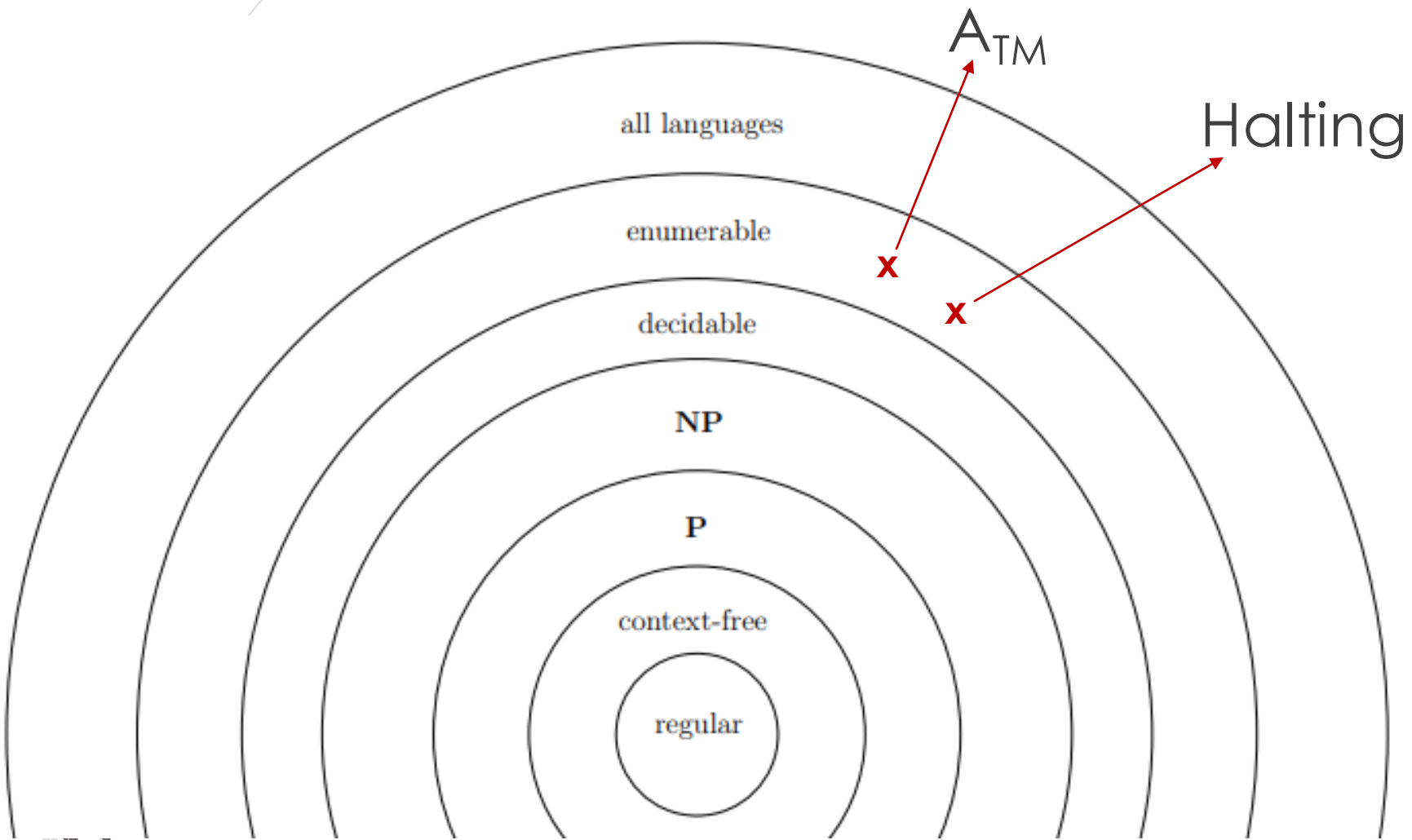
Let the H and Q programs below be prepared with the same logic as in the previous example.

Algorithm $Q(\langle P \rangle)$**:**

while $H(P, \langle P \rangle) = true$
do have a beer
endwhile

$$H(P, w) = \begin{cases} true & \text{if } P(w) \text{ terminates,} \\ false & \text{if } P(w) \text{ does not terminate.} \end{cases}$$

A_{TM} and Halting are Enumerable



Decidability vs. Enumerability

If you are in doubt about a problem, you can comment using the rules below.

1. P is Decidable, if and only if ~P is Decidable.

- $(P \in D) \leftrightarrow (\sim P \in D)$

2. P is Decidable, if and only if P and ~P are Enumerable.

- $(P \in D) \leftrightarrow (\sim P \in E)$ and $(P \in E)$

3. If P is Decidable, then P is Enumerable.

- $(P \in D) \rightarrow (P \in E)$

- That’s all.
- Thanks for listening.