Proposition EQ_{CFG} is not a Turing-decidable language.

This proposition is proven using a technique called reducibility.

An even more general result is true, the equivalence problem for Turing machines is undecidable:

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are Turing machines and } L(M_1) = L(M_2) \}.$$

Proposition EQ_{TM} is not a Turing-decidable language

This proposition follows from another result, namely that the emptiness testing problem for Turing machines is undecidable:

$$E_{TM} = \{ \langle M \rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset \}.$$

Proposition E_{TM} is not a Turing-decidable language.

Returning to context-free grammars, we now know that L_{CFG} and E_{CFG} are Turing-decidable, but EQ_{CFG} is not.
Recall that a language is called context-free if it can be generated by a context-free grammar.

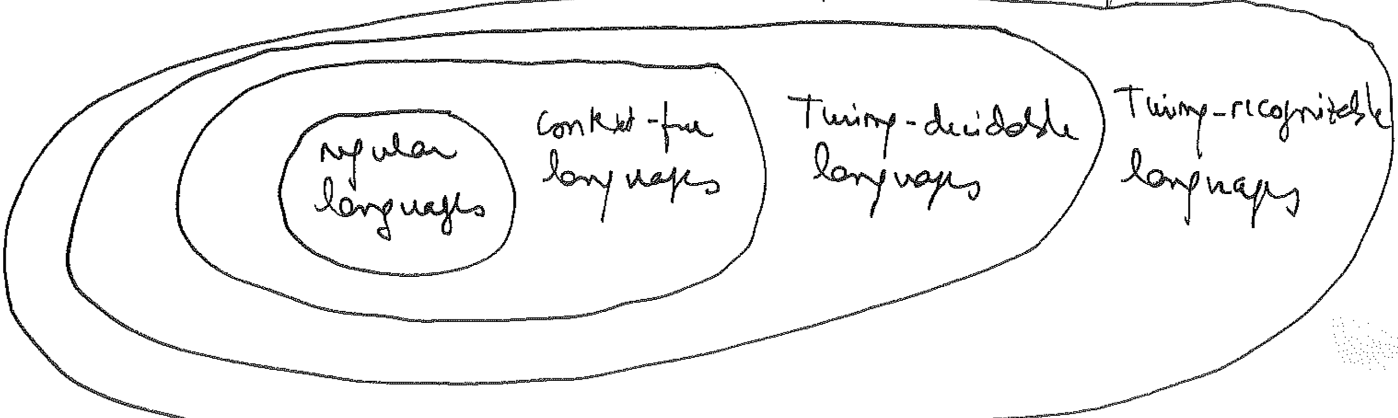
Moral of the story

We now know how the main types of languages relate to each other

$$\{\text{regular languages}\} \subset \{\text{context-free languages}\} \subset \{\text{Turing-decidable languages}\}$$
$$\cap$$
$$\{\text{Turing-recognizable languages}\}$$

this set includes nonregular languages

Visually, we represent the relationship using a Venn diagram:



So Turing machines provide a very powerful computational model. What is surprising is that once we have built a Turing machine to recognize a language, we do not know whether there is a simpler computational model such as a DFA that recognizes the same language. Define

$$REGULAR_{TM} = \{\langle M\rangle \mid M \text{ is a Turing machine and } L(M) \text{ is a regular language}\}.$$

<u>Theorem</u> $REGULAR_{TM}$ is not a Turing-decidable language.

This theorem is proven using reducibility.
In fact, even more is true:

<u>Rice's Theorem</u> Any property of the languages recognized by Turing

machines is not Turing-decidable.

$\boxed{\text{Undecidability}}$

<u>Task</u> Understand why certain problems are algorithmically unsolvable.

Recall that a Turing machine M is defined as a 7-tuple

$$(S, A, \tilde{A}, t, i, s_{accept}, s_{reject})$$

set of states — input alphabet — tape alphabet — transition mapping $t: S \times \tilde{A} \to S \times \tilde{A} \times \{L, R\}$ — initial state — accept state — reject state

<u>Def</u> An <u>encoding</u> $\langle M \rangle$ of a Turing machine M refers to the 7-tuple $(S, A, \tilde{A}, t, i, s_{accept}, s_{reject})$ that defines M, and is therefore a finite string.

Recall that earlier in the module we proved the following results:

<u>Theorem</u> If A is a finite alphabet, then the set of all words over A $A^* = \bigcup_{j=0}^{\infty} A^j$ is countably infinite.

<u>Corollary I</u> If A is a finite alphabet, then the set of all languages over A is uncountably infinite.

<u>Corollary II</u> The set of all programs in any programming language is countably infinite.

Recall that we proved Corollary II by realizing that for any programming language, a program is a finite string over the finite alphabet of all allowable characters in that programming language.

<u>Corollary III</u> Given a finite alphabet A, the set of all Turing-recognizable languages over A is countably infinite.

<u>Proof</u> An encoding $\langle M \rangle$ of a Turing machine M is the 7-tuple $(S, A, \tilde{A}, t, i, s_{accept}, s_{reject})$, which is a finite string over a language

B that contains A and is finite. By the Theorem, $B^* = \bigcup_{j=0}^{\infty} B^j$ is countably infinite. Since $\langle M \rangle \in B^*$, there are at most countably infinitely many Turing machines M that recognize languages over A $\Rightarrow$ there are at most countably infinitely many Turing-recognizable languages over A. We know we can build Turing machines with as large a set of states S as we want $\Rightarrow$ the set of Turing machines that recognize languages over A cannot be finite $\Rightarrow$ it is countably infinite.

(q.e.d.)

<u>Proposition</u> Let A be a finite alphabet. Not all languages over A are Turing-recognizable.

<u>Proof</u> By Corollary I, the set of all languages over A is uncountably infinite. By Corollary III, the set of all Turing-recognizable languages over A is countably infinite $\Rightarrow$ there are many more languages over A than can be recognized by a Turing machine.

(q.e.d.)

<u>Remark</u> This result makes a lot of sense because while we normally look at simpler, well-structured problems where there is a pattern, most languages over A have no pattern to them.

To understand more on the set of all Turing machines, we define the language $L_{TM} = \{\langle M, w \rangle \mid M \text{ is a Turing machine and } M \text{ accepts } w\}$. Here w is a string over the input alphabet A.

We will prove that L_{TM} is a Turing-recognizable language, but L_{TM} is <u>NOT</u> Turing decidable.

<u>Proposition</u> L_{TM} is a Turing-recognizable language.

<u>Proof</u> We define a Turing machine U that recognizes L_{TM}:

U = On input $\langle M, w \rangle$, where M is a Turing machine and w is a string

1. Simulate M on string w.
2. If M ever enters its accept state, then accept; if M ever enters

its reject state, then reject. U loops on input $\langle M, w\rangle$ if M loops on w $\Rightarrow$ U is a recognizer but not a decider.
(q.e.d.)

<u>Remark</u> The Turing machine U is an example of the <u>universal Turing machine</u> first proposed by Turing in 1936. It is called universal because it simulates any other Turing machine. This idea of an universal Turing machine led to the development of stored-program computers.

<u>NB</u> Philosophically, the universal Turing machine we just constructed runs into the following big issues:

① U itself is a Turing machine. What happens when U is given an input $\langle U, w\rangle$?

② The encoding of a Turing machine is a string. What happens when we input $\langle M, \langle M\rangle\rangle$ or even worse $\langle U, \langle U\rangle\rangle$?

We are getting very close to Russell's paradox, the set $\Gamma = \{D \mid D \notin D\}$ which showed the axioms of naive set theory were inconsistent and led to more complicated axioms.

In our case, these issues lead to showing the language L_{TM} cannot possibly be Turing-decidable.

<u>Proposition</u> L_{TM} is not Turing-decidable.

<u>Proof</u> Assume L_{TM} is Turing-decidable and obtain a contradiction.

If L_{TM} is Turing decidable, then $\exists$ decider H for L_{TM}.

Given input $\langle M, w\rangle$, H $\rightarrow$ accepts if M accepts w
$\searrow$ rejects if M does not accept w.

We now construct another Turing machine D with H as a subroutine, which behaves like the set Γ defined by Russell

D = on input $\langle M\rangle$, where M is a Turing machine:

1. Run H on input $\langle M, \langle M\rangle\rangle$.
2. Output the opposite of what H outputs. If H accepts, then reject; if H rejects, then accept.

Now, let us run D on its own encoding $\langle D\rangle$:

D on input $\langle D \rangle$ → accepts if D does not accept $\langle D \rangle$
→ rejects if D accepts $\langle D \rangle$.

$\Rightarrow$ D cannot exist, hence H cannot exist. The language L_{TM} has no decider.
(q.e.d.)

<u>Example of a language that is not Turing-recognizable</u>

<u>task</u> Use what we know about L_{TM} to build an example of a language that is not Turing-recognizable.

<u>Def</u> Given an alphabet A that is finite, $A^* = \bigcup_{j=0}^{\infty} A^j$, and then a language $L \subset A^*$, we define the <u>complement</u> $\overline{L}$ of L as $\overline{L} = A^* \setminus L$, i.e. all words over A that are not in L.

<u>Def</u> A language L is called <u>co-Turing-recognizable</u> if its complement $\overline{L}$ is Turing-recognizable.

<u>Theorem</u> A language L is decidable $\Leftrightarrow$ L is Turing-recognizable and co-Turing-recognizable.

<u>Proof</u> "$\Rightarrow$" If L is decidable $\Rightarrow$ L is Turing-recognizable. Note that if L is decidable $\Rightarrow$ $\exists$ a Turing machine M that decides L. Build a Turing machine $\tilde{M}$ that reverses the output of M, i.e. if M accepts a string w then $\tilde{M}$ rejects the same string w. If M rejects w, then $\tilde{M}$ accepts w. $\tilde{M}$ is therefore a decider for $\overline{L}$ $\Rightarrow$ $\overline{L}$ is Turing-decidable $\Rightarrow$ $\overline{L}$ is Turing-recognizable, so L is Turing-recognizable and co-Turing-recognizable.