An example of a problem that is Turing-recognizable but not Turing-decidable. Let $D = \{p \mid p$ is a polynomial with an integer root$\}$. Hilbert's 10^{th} problem is asking whether D is decidable.

Let us simplify the problem to the one variable case:

$$D_1 = \{p \mid p \text{ is a polynomial in variable } X \text{ with an integer root}\}.$$

We can easily write down a Turing machine M_1 that recognizes D_1:

M_1 = On input p, where p is a polynomial in X

1. Evaluate p with X set successively to the values $0, 1, -1, 2, -2, \dots$ If at any value the polynomial evaluates to 0, accept.

If p does indeed have an integer root, M_1 will eventually find it and accept p. If p does not have an integer root, then M_1 will run forever.

Principle behind M_1: $\mathbb{Z} \sim \mathbb{N}$, i.e. $\mathbb{Z}$ is countably infinite, so we can write $\mathbb{Z}$ as a sequence (enumerate it) $\mathbb{Z} = \{s_1, s_2, \dots\} = \{s_i\}_{i=1,2,\dots}$ $= \{0, 1, -1, 2, -2, \dots\}$

Now, consider polynomials of n variables $p(x_1, \dots, x_n)$. We want to find $(x_1, \dots, x_n) \in \mathbb{Z}^n$ such that $p(x_1, \dots, x_n) = 0$, so in general Hilbert's 10^{th} problem is asking us to build a decider for

$$D_n = \{p(x_1, \dots, x_n) \mid \exists (x_1, \dots, x_n) \in \mathbb{Z}^n \text{ such that } p(x_1, \dots, x_n) = 0\}.$$

We can easily build a Turing machine M_n that recognizes D_n using the principle behind M_1: $\mathbb{Z}^n$ is countably infinite because it is the Cartesian product of a countably infinite set with itself n times. Since $\mathbb{Z}^n$ is countably infinite, we can enumerate it, namely write it as a sequence $\mathbb{Z}^n = \{c_1, c_2, \dots\}$, where $c_i = (x_1^{(i)}, \dots, x_n^{(i)})$.

Then M_n = On input p, where p is a polynomial in $x_1, \dots, x_n$

1. Evaluate p with $(x_1, \dots, x_n)$ set successively to the values $c_1, c_2, \dots$. If at any value $c_i = (x_1^{(i)}, \dots, x_n^{(i)})$, $p(x_1^{(i)}, \dots, x_n^{(i)}) = 0$, accept p.

If p has an integer root $(x_1^{(i)}, \ldots, x_n^{(i)}) \in \mathbb{Z}^n$, then the Turing machine accepts; otherwise, it goes on forever (it loops) just like M_1. It turns out M_1 can be converted into a decider because if $p(x)$ of one variable has a root, then that root must fall between certain bounds, so the checking of possible values can be made to terminate when those bounds are reached. By contrast, no such bounds exist when the polynomial is of two variables or more $\Rightarrow$ M_n for $n \geq 2$ CANNOT be converted into a decider. This is what Matijasević proved.

Decidable languages

Task Explore whether certain languages are decidable that come from our study of formal languages and grammars.

① The acceptance problem for deterministic finite state acceptors (DFA's)

Test whether a given deterministic finite state acceptor (DFA) B accepts a given string w.

We can rewrite the acceptance problem as a language:

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

Theorem L_{DFA} is a Turing decidable language.

Proof We construct a Turing machine M that decides L_{DFA} as follows: 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 of B, accept $\langle B, w \rangle$. If it ends in a non-accepting state of B, reject $\langle B, w \rangle$.

We need to provide more details on the input $\langle B, w \rangle$. B is a finite state acceptor, which we defined as a 5-tuple (S, A, i, t, F) w/ S the set of states, A the alphabet, i the initial state, t the transition mapping $t: S \times A \to S$, and F the set of finishing states. The string w is over the alphabet A, so the pair $\langle B, w \rangle$ as input for our Turing machine is in fact $(S, A, i, t, F; w)$. The Turing machine M starts in the configuration $\varepsilon i w$ (remind yourself what a configuration is). If $w = uv$, where $u \in A$ is the first character in the word w and if $t(i,u) = s$, then the next configuration of the Turing machine M is usv, i.e. the new state corresponds to the state s in which B enters from the initial state i upon receiving input character u and the tape head has moved right past u ready to examine the second character of w. Once the string w has been completely processed, then the configuration of the Turing machine is $w s_\omega \varepsilon$. If the final state s_ω where we ended up is an accepting state, i.e. $s_\omega \in F$, then we accept $\langle B, w \rangle$; otherwise, we reject $\langle B, w \rangle$. (q.e.d.)

② <u>The acceptance problem for nondeterministic finite state acceptors (NFA's)</u>

Test whether a given nondeterministic finite state acceptor B accepts a given string w.

Rewrite this acceptance problem as a language:

$L_{NFA} = \{ \langle B, w \rangle \mid B$ is a NFA that accepts input string $w \}$.

<u>Theorem</u> L_{NFA} is a Turing-decidable language.

<u>Proof</u> This result is in fact a corollary to the previous theorem. As we showed in our unit on formal languages and grammars, given any NFA B, $\exists$ a deterministic finite state acceptor (DFA) B' that corresponds to it (with potentially many more states). Therefore, to any pair $\langle B, w \rangle \in L_{NFA}$, there corresponds a pair $\langle B', w \rangle \in L_{DFA}$. Since L_{DFA} is a Turing-decidable language, L_{NFA} is Turing-decidable as well. (q.e.d.)

③ <u>The acceptance problem for regular expressions</u>

Test whether a regular expression R generates a string w.

We rewrite this acceptance problem as the language

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

__Theorem__ L_{REX} is a Turing-decidable language.

__Proof__ Recall that a language L is regular $\Leftrightarrow$ L is accepted by a deterministic or nondeterministic finite state acceptor $\Leftrightarrow$ L is given by a regular expression. There exists an algorithm to construct a nondeterministic finite state acceptor from any given regular expression $\Rightarrow \forall \langle R, w \rangle \in L_{REX}$, $\exists \langle B, w \rangle \in L_{NFA}$ that corresponds to it. Since L_{NFA} is Turing-decidable, L_{REX} is Turing decidable.

(q.e.d.)

④ __Emptiness testing for the language of an automaton__

Given a DFA B, figure out whether the language recognized by B, $L(B)$ is empty or not, i.e. whether $L(B) \neq \emptyset$ or $L(B) = \emptyset$.

Rewrite the emptiness testing problem as a language:

$$E_{DFA} = \{ \langle B \rangle \mid B \text{ is a DFA and } L(B) = \emptyset \}.$$

__Theorem__ E_{DFA} is a Turing-decidable language.

__Proof__ A DFA B accepts a certain string w if we are in an accepting state when the last character of w has been processed. We design a Turing machine M to test this condition as follows:

M = On input $\langle B \rangle$, where B is a DFA:

1. Mark the initial state of B.
2. Repeat until no new states of B get marked:
3. Mark any state that has a transition coming into it from any state that is already marked.
4. If no accept state is marked, then accept; otherwise, reject.

We have thus marked all states of B where we can end up given an input string. If no such state is an accepting state, then B will not accept any string, i.e. $L(B) = \emptyset$ as needed!

(q.e.d.)

⑤ <u>Checking whether two given DFA's accept the same language</u>

Given B_1, B_2 DFA's, test whether $L(B_1) = L(B_2)$.

We rewrite this problem as the language

$$EQ_{DFA} = \{\langle B_1, B_2 \rangle \mid B_1 \text{ and } B_2 \text{ are DFA's and } L(B_1) = L(B_2)\}.$$

<u>Theorem</u> EQ_{DFA} is a Turing-decidable language.

<u>Proof</u> Given two sets Γ and Σ, $\Gamma \neq \Sigma$ if $\exists\, x \in \Gamma$ such that $x \notin \Sigma$ (i.e. $\Gamma \setminus \Sigma \neq \emptyset$) or $\exists\, x \in \Sigma$ such that $x \notin \Gamma$ (i.e. $\Sigma \setminus \Gamma \neq \emptyset$).

Recall from our unit on set theory that $\Gamma \setminus \Sigma = \Gamma \cap \overline{\Sigma}$, Γ intersect