Undecidability

Using a countability argument, we can show that there are some (actually many) languages that are not Turing-recognizable.

Corollary 4.18: Not every language is Turing-recognizable.

Proof Idea:

- There are "countably many" Turing-recognizable languages.
- There are "uncountably many" languages.
- Thus "most languages" are not Turing-recognizable.

Non-constructive proof – does not actually exhibit a specific language that is not Turing-recognizable.

Proof of Corollary 4.18: Choose a suitable alphabet Γ so that every TM has an encoding as a finite string in Γ^*.

Let L be the function that takes each TM M to the language $L(M)$ that it recognizes.

Let $\mathsf{lex} : \mathcal{N} \rightarrow \Gamma^*$ be the (bijective) function that enumerates Γ^* in lexicographic order.

Let T be the function that takes each string in Γ^* to the TM it encodes (if it does), otherwise to some fixed TM.

Then $T \circ \mathsf{lex}$ is a surjective function that maps each $i \in \mathcal{N}$ to a TM $T(\mathsf{lex}(i))$. The set of all TMs is therefore countable.

In contrast, the set $\mathcal{P}(\Sigma^*)$ of all languages over Σ is *uncountable*. Thus, the mapping

$$L \circ T \circ \mathsf{lex} : \mathcal{N} \to \mathcal{P}(\Sigma^*)$$

cannot be surjective, so there must exist a language $L \in \mathcal{P}(\Sigma^*)$ that is *not* $L(T(\mathsf{lex}(i)))$ for any $i \in \mathcal{N}$. That is, there must exist a non-Turing-recognizable language.

The Acceptance Problem for TMs

$$\mathsf{A}_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$$

Claim: A_{TM} is Turing-recognizable.

Proof: We can construct a TM U that operates as follows:

On input $\langle M, w\rangle$*:*

- Simulate M on input w.
- If the simulation ever enters an accepting config., *accept*.
- If the simulation ever enters a rejecting config., *reject*.
- (otherwise the simulation never halts)

Universal Turing Machine

The machine U in the previous proof is called a *universal Turing machine* because it can simulate the behavior of any TM, given its description and an input.

Sipser doesn't say much about the construction of U.

- An encoding $\langle M \rangle$ of a TM M can be done by listing the rows of the (finite) transition table for M.
- Machine U can consult this transition table to carry out the simulation of M.

To make things simpler, use three tapes: one tape as a read-only input, one tape to keep track of the simulated configuration of M and another tape as a scratch area.

Theorem 4.11: A_{TM} is undecidable.

Proof: Suppose (for proof by contradiction) A_{TM} is decidable, and that TM H is a decider for it. Construct a new TM D that operates as follows:

On input $\langle M \rangle$, where M is a TM:

- Run H on input $\langle M, \langle M \rangle \rangle$ (what does this mean?)
- If H accepts, reject; if H rejects, accept
  (there is no other possibility if H is a decider).

Now consider what happens if we run D on input $\langle D \rangle$:

- *If D accepts input $\langle D\rangle$*, then H rejects input $\langle D, \langle D\rangle\rangle$, *so D rejects input $\langle D\rangle$*.
- *If D rejects input $\langle D\rangle$*, then H accepts input $\langle D, \langle D\rangle\rangle$, *so D accepts input $\langle D\rangle$*.

Both cases are contradictory, so H cannot exist.

What does "Run H on input $\langle M, \langle M \rangle \rangle$" mean?

- D takes the description $\langle M \rangle$ of M, uses it to construct a string $\langle M, \langle M \rangle \rangle$ in the input format required by H, then simulates H running on that input.
- The string $\langle M, \langle M \rangle \rangle$ is basically a pair that consists of two copies of the description of M: the first copy describes the machine M that H is being asked about and the second describes the input being given to machine M.

Relationship to Cantor's diagonal method:

H	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\dots$	$\langle D \rangle$	$\dots$
M_1	acc					
M_2	$\dots$	rej				
M_3	$\dots$		rej			
$\dots$						
D	$\dots$				X	
$\dots$						

D	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\dots$	$\langle D \rangle$	$\dots$
M_1	rej					
M_2	$\dots$	acc				
M_3	$\dots$		acc			
$\dots$						
D	$\dots$				$\overline{\text{X}}$	
$\dots$						

Theorem 4.22: A language L is decidable if and only if it and its complement $\overline{L}$ are both Turing-recognizable.

Proof: ($\rightarrow$) If L is decidable then so is $\overline{L}$ (interchange the accept and reject states of a decider for L).

($\leftarrow$) Suppose L and $\overline{L}$ are both Turing-recognizable. Let M be a TM that recognizes L and M' be a TM that recognizes $\overline{L}$.

Construct TM D that operates as follows: *On input w:*

- Run M and M' "in parallel" on input w.
- If M accepts, accept. If M' accepts, reject.
- Since w is either in L or $\overline{L}$ we will eventually either accept or reject.

"In parallel" means we alternate steps of M and M' so that they both make progress.

Corollary 4.23: $\overline{A}_{TM}$ is not Turing-recognizable.

Proof: If it were, then A_{TM} and its complement $\overline{A}_{TM}$ would both be Turing-recognizable. This would imply that A_{TM} is decidable, but we have shown that it is not.

$\mathrm{T-recognizable}$ (A_{TM})

$\mathrm{co-T-recognizable}$ $(\overline{A}_{TM})$

$\mathrm{T-decidable}$

- A_{TM} is T-recognizable but not T-decidable, hence not co-T-recognizable
- $\overline{A}_{TM}$ is co-T-recognizable but not T-decidable, hence not T-recognizable

The Halting Problem for TMs

$$\mathsf{HALT_{TM}} = \{\langle M, w\rangle : M \text{ is a TM and } M \text{ halts on input } w\}$$

Theorem 5.1: $\mathsf{HALT_{TM}}$ is undecidable.

Proof: We show that if $\mathsf{HALT_{TM}}$ is decidable, then $\mathsf{A_{TM}}$ would also be decidable, a contradiction.

- Suppose $\mathsf{HALT_{TM}}$ is decidable, and let R be a decider for it.

- Define TM S as follows: *On input $\langle M, w \rangle$:*

  1. Run TM R on input $\langle M, w \rangle$.
  2. If R rejects, *reject*.
  3. If R accepts, simulate M on w until it halts.
  4. If M has accepted, *accept*; if M has rejected, *reject*.

  Note that S is a decider (why?).

- Then S decides A_{TM}: given input $\langle M, w \rangle$, S accepts if and only if M halts and accepts on input w. Contradiction!.

The Emptiness Problem for TMs

$$\mathsf{E}_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM with } L(M) = \emptyset\}$$

Theorem 5.2: E_{TM} is undecidable.

Proof: We show that if E_{TM} is decidable, then A_{TM} would also be decidable, a contradiction.

Suppose E_{TM} is decidable. Let R be a TM that decides E_{TM}. Construct a TM S that operates as follows:

On input $\langle M, w \rangle$:

1. Use $\langle M \rangle$ to construct the description $\langle M_w \rangle$ of a TM M_w that behaves as follows:
   On input x:
   - If $x \neq w$, then *reject*.
   - If $x = w$, then run M on input w and accept if M does.
2. Run R on $\langle M_w \rangle$. If R accepts, then *reject*. If R rejects, then *accept*.

Note: Either $L(M_w) = \{w\}$ or $L(M_w) = \phi$; M accepts w if and only if $L(M_w) \neq \phi$.

An Undecidable Problem for CFGs

Define:

$$\text{ALL}_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$$

Thm: ALL_{CFG} is undecidable.

That is, there is no algorithm to decide, given a CFG G, whether G generates all strings in Σ^*.

Computation Histories

The proof of the Theorem uses the method of Computation Histories.

Def. *(Sipser 5.5):* An *accepting computation history* for a TM M and input string w is a sequence of configurations: C_1, C_2, ..., C_l, where

- C_1 is the start configuration of M on w.
- C_l is an accepting configuration of M.
- C_i yields C_{i+1} according to the transition function for M, for $1 \leq i < l$.

We may choose an alphabet Ξ and a notation for describing computation histories of TMs, such that for every TM M and input string $w \in \Sigma^*$, every accepting computation history for M on input w is described by some string in Ξ^*.

For example:

$$\# \ C_1 \ \# \ C_2 \ \# \ \ldots \ \# \ C_l \ \#,$$

where, *e.g.* the configurations C_i are represented using binary notation for the states and tape symbols.

Proof Idea: We show that, if there were an algorithm to decide $\mathsf{ALL}_{\mathsf{CFG}}$, then we could use it to decide A_{TM}. Suppose D were a decider for $\mathsf{ALL}_{\mathsf{CFG}}$. The idea of the algorithm is as follows:

On input $\langle M, w \rangle$:

- Construct a CFG $G_{M,w}$ that generates all strings that are *not* accepting computation histories for M on input w.
- Run D on $G_{M,w}$. If D accepts, then *reject*. If D rejects, then *accept*.

Note: D accepts $G_{M,w}$ iff there is no accepting computation history for M on input w iff M does not accept w.

The Key Idea

The key idea in the proof is that it is possible to construct $G_{M,w}$ that generates all strings that are *not* accepting computation histories.

Note that $\#\ C_1\ \#\ C_2\ \#\ \ldots\ \#\ C_l\ \#$ describes an accepting computation history iff the following all hold:

1. C_1 describes the start configuration for M on input w.
2. C_l describes an accepting configuration for M.
3. Each C_i yields C_{i+1}, for $1 \leq i < l$.

We can construct a PDA (hence also an equivalent CFG) that accepts whenever one of these conditions is violated.

Construction of the PDA

Start by nondeterministically choosing which of the three conditions to check is violated. Then:

1. Check whether C_1 is the start configuration. If not, *accept*.
2. Check whether C_l ends with the accept state. If not, *accept*.
3. Scan through input, guessing when C_i is reached. Compare with the next configuration C_{i+1} to see if the yields relation is satisfied (how?). If not, *accept*.

(otherwise, do not accept.)

Technical Points

There are some technical points associated with case (3):

- Comparing C_i with C_{i+1} involves pushing C_i onto the stack and popping it off while reading C_{i+1}.
  - *Problem:* C_i comes off the stack in reverse order.
  - *Solution:* Use an encoding for computation histories that reverses the representation of successive configurations.

- Checking whether C_i yields C_{i+1} mostly involves just checking that they are identical, except for the old state/new state and the symbols just before and just after the head position.
  - While reading C_{i+1}, the PDA can remember the last state and the last two tape symbols read.

    Note: the PDA is constructed for a *specific* M and so can be designed to have enough memory in its finite control for this.
  - The remembered information can be checked against what is popped from the stack. The transition rules for M are incorporated into the PDA's finite control.

The Equivalence Problem for CFGs

$$\mathsf{EQ}_{\mathsf{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs with } L(G) = L(H)\}$$

Ex. *(Sipser 5.1):* Show $\mathsf{EQ}_{\mathsf{CFG}}$ is undecidable.

Proof Idea: If $\mathsf{EQ}_{\mathsf{CFG}}$ were decidable, then we could use it to construct a decider for $\mathsf{ALL}_{\mathsf{CFG}}$ (how?).

Reducibility

The previous proofs are implicitly using a general technique, called *reduction*.

- If we want to solve problem A, and we already have a solution to problem B, then we might try to convert instances of problem A into "equivalent" instances of problem B.
- In that case, we say that problem A is *reducible* to problem B, and that the conversion algorithm is called a *reduction* from A to B.
- If there is a reduction from A to B, then in some sense B is "at least as hard as" A, since a solution to B yields a solution to A.

Mapping Reducibility

Recall that we regard a *problem* as a language L and an *instance* of the problem is a string w whose membership in L we would like to determine.

Definition 5.20: Language A is *mapping reducible* to language B ($A \leq_{\mathrm{m}} B$) if there is a *computable* function

$$f : \Sigma^* \to \Sigma^*$$

such that

$$\text{For all } w \in \Sigma^*, \qquad w \in A \qquad \text{iff} \qquad f(w) \in B.$$

In this case, f is called a *reduction*.

Mapping reducibility is also often called *many-one reducibility*.

Def. *(Sipser 5.17):* A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if there is a Turing machine M, such that when started on any input w, machine M eventually halts with just $f(w)$ on its tape.

Theorem 5.22: If $A \leq_{\mathrm{m}} B$ and B is decidable, then A is decidable.

Proof: Suppose $A \leq_{\mathrm{m}} B$ via reduction f and B is decided by M. We construct a decider N for A that operates as follows: On input w:

- Compute $f(w)$. [**Important:** f must be computable!]
- Run M on input $f(w)$ and accept/reject as M does.

If $w \in A$ then $f(w) \in B$ so M accepts $f(w)$ so N accepts w.
If $w \notin A$ then $f(w) \notin B$ so M rejects $f(w)$ so N rejects w.

Corollary 5.23: If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Theorem 5.28: If $A \leq_{\mathsf{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Proof: Similar to Theorem 5.22.

Corollary 5.23: If $A \leq_{\mathsf{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

The Equivalence Problem for TMs

$\mathsf{EQ}_{\mathsf{TM}} = \{\langle M_1, M_2\rangle : M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\}$

Theorem 5.4: $\mathsf{EQ}_{\mathsf{TM}}$ is undecidable.

Proof: We can show $\mathsf{E}_{\mathsf{TM}} \leq_{\mathsf{m}} \mathsf{EQ}_{\mathsf{TM}}$.

Let f take $\langle M\rangle$ to $\langle M, E\rangle$, where E is a TM that always rejects. Then $\langle M\rangle \in \mathsf{E}_{\mathsf{TM}}$ iff $\langle M, E\rangle \in \mathsf{EQ}_{\mathsf{TM}}$.

Theorem 5.30: $\mathsf{EQ_{TM}}$ is neither Turing-recognizable nor co-Turing-recognizable.

Proof: To show $\mathsf{EQ_{TM}}$ not Turing-recognizable, we show $\mathsf{A_{TM}} \leq_{\mathsf{m}} \overline{\mathsf{EQ}}_{\mathsf{TM}}$. Then $\overline{\mathsf{A}}_{\mathsf{TM}} \leq_{\mathsf{m}} \mathsf{EQ_{TM}}$ *(see Lemma)*. Since $\overline{\mathsf{A}}_{\mathsf{TM}}$ is not Turing-recognizable, neither is $\mathsf{EQ_{TM}}$.

To show $\mathsf{EQ_{TM}}$ not co-Turing-recognizable, we show $\mathsf{A_{TM}} \leq_{\mathsf{m}} \mathsf{EQ_{TM}}$. Since $\mathsf{A_{TM}}$ is not co-recognizable, neither is $\mathsf{EQ_{TM}}$.

Lemma: $A \leq_{\mathsf{m}} B$ iff $\overline{A} \leq_{\mathsf{m}} \overline{B}$.

Proof: A reduction f from A to B is also a reduction from $\overline{A} \leq_{\mathsf{m}} \overline{B}$.

The reduction from $\mathsf{A_{TM}}$ to $\overline{\mathsf{EQ}}_{\mathsf{TM}}$ is the function computed by TM F:

On input $\langle M, w \rangle$:

- Construct TM M_1 that behaves as follows:
  On any input:
  - Run M on input w.
  - If M accepts, accept, otherwise loop.
- Output $\langle M_1, E \rangle$, where E is a TM that always rejects.

If $\langle M, w \rangle \in \mathsf{A_{TM}}$, then M accepts w, hence $\langle M_1, E \rangle \in \overline{\mathsf{EQ}}_{\mathsf{TM}}$.

If $\langle M, w \rangle \notin \mathsf{A_{TM}}$, then M does not accept w, so M_1 loops on any input, hence $\langle M_1, E \rangle \notin \overline{\mathsf{EQ}}_{\mathsf{TM}}$.

To show $\mathsf{EQ}_{\mathsf{TM}}$ not co-Turing-recognizable, we show $\mathsf{A}_{\mathsf{TM}} \leq_{\mathsf{m}} \mathsf{EQ}_{\mathsf{TM}}$. Since A_{TM} is not co-recognizable, neither is $\mathsf{EQ}_{\mathsf{TM}}$.

The reduction is computed as follows:
On input $\langle M, w \rangle$:

- Construct TM M_1 that behaves as follows:
  On any input:
  - Run M on input w.
  - If M accepts, accept, otherwise loop.
- Output $\langle M_1, A \rangle$, where A is a TM that always accepts.

If $\langle M, w \rangle \in \mathsf{A}_{\mathsf{TM}}$, then M accepts w, hence $\langle M_1, A \rangle \in \mathsf{EQ}_{\mathsf{TM}}$.

If $\langle M, w \rangle \notin \mathsf{A}_{\mathsf{TM}}$, then M does not accept w, so M_1 loops on any input, hence $\langle M_1, A \rangle \notin \overline{\mathsf{EQ}}_{\mathsf{TM}}$.

Summary

Almost every problem you can think of about the language accepted by a TM is undecidable.

The following asks you to show such a result explicitly:

Exercise 5.28 (Rice's Theorem): Let P be any nontrivial property of the language of a TM. Prove that the problem of determining if a given TM's language has property P is undecidable.

Why is the word "nontrivial" included in the statement?

We have to clarify what we mean by a "property of the language of a TM".

Fix an alphabet Σ and a particular encoding of TMs as strings in Σ^*.

Definition: A *property of Turing machine descriptions* is a language P whose elements are Turing machine descriptions. (Say that P is *true* for a TM M if $\langle M\rangle \in P$, otherwise P is *false* for M.)

Note that

$$P = \{\langle M\rangle : M \text{ is a TM with fewer than 100 states}\}$$

is a property of TM descriptions that is readily decidable. So we have to consider properties P that only depend on $L(M)$, not specific details of M.

Definition: Say that property P is a *property of the language of a Turing Machine* if whenever M_1 and M_2 are TMs with $L(M_1) = L(M_2)$, then $\langle M_1 \rangle \in P$ if and only if $\langle M_2 \rangle \in P$.

There is still an issue: a language A that contains *all* TM descriptions and a language N that contains *no* TM descriptions are trivially decidable.

Definition: Property P is *nontrivial* if it neither contains *all* TM descriptions nor *no* TM descriptions.

Proof of Rice's Theorem:

Suppose P is a nontrival property of the language of a Turing Machine. Then one of P or $\overline{P}$ contains the description of a TM that always rejects.

Without loss of generality, suppose it is $\overline{P}$; otherwise we can proceed using the nontrivial property $\overline{P}$ instead of P.

Since P is nontrivial, it contains the description $\langle T \rangle$ of some TM T.

We show that P is undecidable by giving a reduction of A_{TM} to P.

The reduction is computed as follows: On input $\langle M, w\rangle$:

- Construct the TM M_w that behaves as follows:
  On input x:
  - Simulate M on w.
  - If M halts and rejects, then reject.
  - If M accepts, simulate T on x. If T accepts, accept, otherwise loop.
- Output $\langle M_w\rangle$.

If $\langle M, w\rangle \in \mathsf{A}_{\mathsf{TM}}$, then M accepts w, hence M_w simulates T and $L(M_w) = L(T)$. But since $\langle T\rangle \in P$ also $\langle M_w\rangle \in P$.

If $\langle M, w\rangle \notin \mathsf{A}_{\mathsf{TM}}$, then M_w either rejects or loops, so $L(M_w) = \emptyset$. But then $\langle M_w\rangle \in \overline{P}$.

Corollaries of Rice's Theorem

The following languages are undecidable:

- All descriptions of TMs that always accept ($L(M) = \Sigma^*$).
- All descriptions of TMs that never accept ($L(M) = \emptyset$).
- All descriptions of TMs that accept at least one string ($L(M) \neq \emptyset$).
- All descriptions of TMs that always halt.
  If decidable, then we could decide the "always accept" language: Given $\langle M \rangle$, construct $\langle M' \rangle$ that accepts if M accepts and loops otherwise. Then check if M' always halts.

- All descriptions of TMs that always loop.

  If decidable, then we could decide the "accept at least one string" language, using the same construction.

Practical Consequence: No compiler (which must always terminate) can decide *any* nontrivial "behavioral property" of a program (such as termination).