Undecidable Languages

1 Existence of Undecidable Languages

Proposition (The set of Turing machines is countable)**.** *Fix some input alphabet and tape alphabet. The set of all Turing machines $\mathcal{T} = \{M : M \text{ is a TM}\}$ is countable.*

Proof. We will show $\mathcal{T}$ is countable by showing it is encodable.

First, based on Remark (**??**), we will normalize TMs and assume the set of states Q is always $\{0, 1, 2, \ldots, k-1\} \cup \{q_{\text{acc}}, q_{\text{rej}}\}$ for some $k \in \mathbb{N}$. Then given any Turing machine, there is a way to encode it with a finite length string because each component of the 7-tuple has a finite description. In particular, the mapping $M \mapsto \langle M \rangle$, where $\langle M \rangle \in \Sigma^*$, for some finite alphabet Σ, is an injective map (two distinct Turing machines cannot have the same encoding).[1] □

Theorem (Almost all languages are undecidable)**.** *Fix some alphabet Σ. There are languages $L \subseteq \Sigma^*$ that are **not** semi-decidable (and therefore undecidable as well).*

Proof. To prove the result, we simply observe that the set of all languages is uncountable whereas the set of semi-decidable languages is countable. First, consider the set of all languages. Since a language L is defined to be a subset of Σ^*, the set of all languages is $\wp(\Sigma^*)$. By Corollary (**??**), we know that this set is uncountable. Now consider the set of all semi-decidable languages over Σ, which we'll denote by $\mathcal{S}$. Let $\mathcal{T}$ be the set of all TMs. By Proposition (The set of Turing machines is countable), we know that $\mathcal{T}$ is countable. Furthermore, the mapping $M \mapsto L(M)$ can be viewed as a surjection from $\mathcal{T}$ to $\mathcal{S}$. So $|\mathcal{S}| \leq |\mathcal{T}|$. Since $\mathcal{T}$ is countable, this shows $\mathcal{S}$ is countable and completes the proof. □

[1] One can spell out this argument more, but we choose not to do so here since even with this level of detail, the argument is already convincing.

Remark (Constructive vs non-constructive proofs)**.** The argument above is called *non-constructive* because it does not present an explicit undecidable language. A *constructive* argument would prove the undecidability of an explicit language. We present such an argument below.

2 An Explicit Undecidable Language

Recall Important Note (**??**). Since our goal in this section is to find an explicit undecidable language, it is natural to consider applying Lemma (**??**). And indeed, we can diagonalize against the set of all decidable languages, or equivalently, diagonalize against the set $\mathcal{D}$ of all decidable decision problems $f : \Sigma^* \to \{0, 1\}$. This gives us an explicit decision problem, f_D, that is undecidable. The reason we can apply diagonalization is because $\mathcal{D}$ is an encodable set, as we have seen in the proof of Theorem (Almost all languages are undecidable). Therefore $|\Sigma^*| \geq |\mathcal{D}|$, which means the condition to apply diagonalization is satisfied.

Note (Diagonalization against a set of function-like objects)**.** In the previous chapter, we presented diagonalization with respect to a set of functions. However, Lemma (**??**) can be applied in a slightly more general setting: it can be applied when $\mathcal{F}$ is a set of objects that map elements of X to elements of Y (e.g. $\mathcal{F}$ can be a set of Turing machines which map elements of Σ^* to elements of $\{0, 1, \infty\}$). In other words, elements of $\mathcal{F}$ do not have to be functions as long as they are "function-like". Once diagonalization is applied, one gets an explicit function $f_D : X \to Y$ such that no object in $\mathcal{F}$ matches the input/output behavior of f_D. We will illustrate this in the proof of the next theorem.

Definition (The $\mathrm{SELF-ACCEPTS_{TM}}$ language/problem)**.** We say that a TM M *self-accepts* if $M(\langle M \rangle) = 1$. The *self-accepts problem* is defined as the decision problem corresponding to the language

$$\mathrm{SA_{TM}} = \mathrm{SELF-ACCEPTS_{TM}} = \{\langle M \rangle : M \text{ is a TM which self-accepts}\}.$$

The complement problem corresponds to the language

$$\overline{\mathrm{SA_{TM}}} = \overline{\mathrm{SELF-ACCEPTS_{TM}}} = \{\langle M \rangle : M \text{ is a TM which does not self-accept}\}.$$

Note that if a TM M does not self-accept, then $M(\langle M \rangle) \in \{0, \infty\}$.

Theorem (Turing's 1st Undecidability Theorem)**.** *The language $\overline{\mathrm{SA_{TM}}}$ is undecidable.*

Proof. Our goal is to show that $\overline{\mathrm{SA_{TM}}}$ is undecidable. To accomplish this, we can either diagonalize against the set of all decidable decision problems or we can diagonalize against the set of all TMs. To illustrate Note (Diagonalization against a set of function-like objects), we will choose the second option and observe that the decision problem corresponding to $\overline{\mathrm{SA_{TM}}}$ is (arguably) the most natural diagonal element that the diagonalization spits out.

Recall that a decision problem $f : \Sigma^* \to \{0, 1\}$ is decidable if there is a TM M whose input/output behavior matches f, that is, for all $x \in \Sigma^*$, $M(x) = f(x)$.

Now let $\mathcal{F}$ denote the set of all Turing machines. A Turing machine maps elements of $X = \Sigma^*$ to an element of $Y = \{0, 1, \infty\}$. Since the set of all Turing machines is countably infinite and X is countably infinite, we have $|X| = |\mathcal{F}|$. Therefore we can diagonalize against $\mathcal{F}$ to construct $f_D : X \to Y$ that cannot correspond to any Turing machine. If we choose f_D in a way such that the range is $\{0, 1\}$, then f_D will be an undecidable decision problem.

To explicitly define f_D, we pick an injection from $\mathcal{F}$ to $X = \Sigma^*$. The most obvious one is $M \mapsto \langle M \rangle$. Then let

$$f_D(\langle M \rangle) = \begin{cases} 1 & \text{if } M(\langle M \rangle) \in \{0, \infty\} \ \ \text{(i.e. } M(\langle M \rangle) \text{ does not accept);} \\ 0 & \text{otherwise.} \end{cases}$$

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\cdots$
M_1	$\boxed{0}$	1	∞	
M_2	1	$\boxed{\infty}$	1	$\cdots$
M_3	0	1	$\boxed{1}$	
$\vdots$		$\vdots$		
f_D	1	1	0	$\cdots$

By construction, any TM M differs from f_D on input $\langle M\rangle$, i.e. for all TMs M, $M(\langle M\rangle) \neq f_D(\langle M\rangle)$. Therefore f_D is not decidable by any TM.

Observe that the language corresponding to the decision problem f_D is precisely $\overline{\text{SA}_{\text{TM}}}$. □

Note ($\overline{\text{SA}_{\text{TM}}}$ is not semi-decidable)**.** Note that the proof of the above theorem establishes something stronger: the language $\overline{\text{SA}_{\text{TM}}}$ is not semi-decidable. To see this, recall that if M is a semi-decider for f_D, by definition, this implies that for all $x \in \Sigma^*$:

- if $f_D(x) = 1$, then $M(x) = 1$,
- if $f_D(x) = 0$, then $M(x) \in \{0, \infty\}$.

However, by construction of f_D, we know that for input $x = \langle M\rangle$, if $f_D(\langle M\rangle) = 1$ then $M(\langle M\rangle) \in \{0, \infty\}$, and if $f_D(\langle M\rangle) = 0$, then $M(\langle M\rangle) = 1$. So for all TMs M, M fails to semi-decide f_D on input $x = \langle M\rangle$.

3 More Undecidable Languages

Theorem (SA_{TM} is undecidable)**.** *The language* SA_{TM} *is undecidable.*

Proof. We want to show that SA_{TM} is undecidable. The proof is by contradiction, so assume SA_{TM} is decidable and let M_{SA} be a decider for it. We will use this decider to come up with a decider for $\overline{\text{SA}_{\text{TM}}}$. Since $\overline{\text{SA}_{\text{TM}}}$ is undecidable (Theorem (Turing's 1st Undecidability Theorem)), this argument will allow us to reach a contradiction.

Here is our decider for $\overline{\text{SA}_{\text{TM}}}$:

```
def  $M_{\overline{\text{SA}}}(\langle \text{TM } M\rangle)$ :
1.  Run $M_{\text{SA}}(\langle M\rangle)$.
2.  If it accepts, reject.
3.  If it rejects, accept.
```

Given our assumption that M_{SA} is a correct decider for SA_{TM}, it is clear that $M_{\overline{\text{SA}}}$ is a correct decider for $\overline{\text{SA}_{\text{TM}}}$. In particular, if the input $\langle M\rangle$ is such that M self-accepts, then $M_{\text{SA}}(\langle M\rangle)$ would accept, and our machine would reject on line 2. And if the input $\langle M\rangle$ is such that M does not self-accept, $M_{\text{SA}}(\langle M\rangle)$ would reject, and our machine would accept on line 3. So for all possible inputs, our decider gives the correct answer. □

Note (The complement of an undecidable language is undecidable)**.** The general principle that the above theorem highlights is that since decidable languages are closed under complementation, if L is an undecidable language, then $\overline{L} = \Sigma^* \setminus L$ must also be undecidable.

Definition (ACCEPTS and HALTS for TMs)**.** We define the following languages:

$$\begin{aligned} \text{ACCEPTS}_{\text{TM}} &= \{\langle M, x\rangle : M \text{ is a TM which accepts input } x\}, \\ \text{HALTS}_{\text{TM}} &= \{\langle M, x\rangle : M \text{ is a TM which halts on input } x\}, \end{aligned}$$

Theorem ($\text{ACCEPTS}_{\text{TM}}$ is undecidable)**.** *The language* $\text{ACCEPTS}_{\text{TM}}$ *is undecidable.*

Proof. We want to show that $\text{ACCEPTS}_{\text{TM}}$ is undecidable. The proof is by contradiction, so assume $\text{ACCEPTS}_{\text{TM}}$ is decidable and let M_{ACCEPTS} be a decider for it. We will use this decider to come up with a decider for SA_{TM}. We have already proved Theorem (SA_{TM} is undecidable), so this argument will allow us to reach a contradiction.

Here is our decider for SA_{TM}:

```
def $M_{\text{SA}}(\langle \text{TM } M\rangle)$ :
1. Run $M_{\text{ACCEPTS}}(\langle M, \langle M\rangle\rangle)$.
2. If it accepts, accept.
3. If it rejects, reject.
```

Given our assumption that M_{ACCEPTS} is a correct decider for SA_{TM}, it is clear that M_{SA} is a correct decider for SA_{TM}. In particular, if the input $\langle M\rangle$ is such that M self-accepts, then $M_{\text{ACCEPTS}}(\langle M, \langle M\rangle\rangle)$ would accept, and our machine would accept on line 2. And if the input $\langle M\rangle$ is such that M does not self-accept, $M_{\text{ACCEPTS}}(\langle M, \langle M\rangle\rangle)$ would reject, and our machine would reject on line 3. So for all possible inputs, our decider gives the correct answer. □

Theorem (HALTS_{TM} is undecidable)**.** *The language* HALTS_{TM} *is undecidable.*

Proof. We want to show that HALTS_{TM} is undecidable. The proof is by contradiction, so assume HALTS_{TM} is decidable and let M_{HALTS} be a decider for it. We will use this decider to come up with a decider for $\text{ACCEPTS}_{\text{TM}}$. We have already proved Theorem ($\text{ACCEPTS}_{\text{TM}}$ is undecidable), so this argument will allow us to reach a contradiction.

At a high level, if we wanted to come up with a TM solving $\text{ACCEPTS}_{\text{TM}}$, on input $\langle M, x\rangle$, we could just try running $M(x)$ and return its answer. Unfortunately, this simple strategy does not work because $M(x)$ can possibly loop forever, so we would not end up with a *decider* solving $\text{ACCEPTS}_{\text{TM}}$. That being said, we are assuming we have access to a correct decider for HALTS, namely M_{HALTS}. So then we can first check if $M(x)$ halts or not. If it does not, we know $M(x)$ does not accept, so we can reject. If it does halt, then it is safe to run $M(x)$. We know it will accept or reject, and so we can return its answer.

Here is the description of our decider for $\text{ACCEPTS}_{\text{TM}}$:

```
def $M_{\text{ACCEPTS}}(\langle \text{TM } M, \text{ string } x\rangle)$ :
1. Run $M_{\text{HALTS}}(\langle M, x\rangle)$.
2. If it rejects, reject.
3. Else:
4.   Run $M(x)$.
5.   If it accepts, accept.
6.   If it rejects, reject.
```

We have basically argued that this decider is correct in our high-level description of the machine. But nevertheless, we'll lay out the detailed argument here.

To show that M_{ACCEPTS} is a correct decider, we need to argue that for all possible inputs, it gives the correct answer.

First let's assume the input $\langle M, x\rangle$ is such that $M(x)$ accepts. Then on line 1, M_{HALTS} will accept, meaning we will not reject on line 2. Then in line 4, $M(x)$ will accept. And therefore our machine will accept on line 5, as desired.

Next, let's assume the input $\langle M, x\rangle$ is such that $M(x)$ does not self-accept. There are two possible cases for this: either $M(x)$ loops, or $M(x)$ rejects.

1. In the first case, $M(x)$ loops. Then $M_{\text{HALTS}}(\langle M, x\rangle)$ on line 1 would reject, and therefore our machine would reject on line 2, giving the correct answer.

2. In the second case, $M(x)$ rejects. Then $M_{\text{HALTS}}(\langle M, x\rangle)$ on line 1 would accept, meaning we would not reject on line 2. Afterwards, on line 4, $M(x)$ would reject. So our machine would reject on line 6 and give the correct answer.

We have shown that for all possible inputs, M_{ACCEPTS} gives the correct answer, which completes the proof of correctness. □

Proposition (SA_{TM}, $\text{ACCEPTS}_{\text{TM}}$, HALTS_{TM} are semi-decidable)**.** *The languages* SA_{TM}*,* $\text{ACCEPTS}_{\text{TM}}$*, and* HALTS_{TM} *are all semi-decidable, i.e., they are all in* RE*.*

Proof. Here are Turing machines that semi-decide SA_{TM}, $\text{ACCEPTS}_{\text{TM}}$, and HALTS_{TM} respectively. It pretty much follows directly from the definition of semi-decidability that they are correct, and therefore we omit explicitly spelling out the correctness proofs.

```
def $M_{\text{SA}}(\langle \text{TM } M\rangle)$ :
    Run $M(\langle M\rangle)$.
    If it accepts, accept.
    If it rejects, reject.
```

```
def $M_{\text{ACCEPTS}}(\langle \text{TM } M, \text{ string } x\rangle)$ :
    Run $M(x)$.
    If it accepts, accept.
    If it rejects, reject.
```

```
def $M_{\text{HALTS}}(\langle \text{TM } M, \text{ string } x\rangle)$ :
    Run $M(x)$.
    If it accepts or rejects, accept.
```

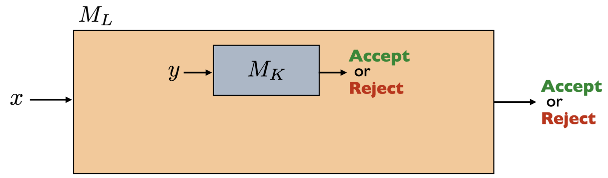
□

Theorem ($\mathsf{R} \neq \mathsf{RE}$)**.** $\mathsf{R} \neq \mathsf{RE}$.

Proof. The languages SA_{TM}, $\text{ACCEPTS}_{\text{TM}}$, and HALTS_{TM} are all in RE but not in R. □

4 Undecidability Proofs by Reductions

Important (Undecidability proofs by reduction)**.** In the last section, we have used the same technique in all the proofs. It will be convenient to abstract away this technique and give it a name. Fix some alphabet Σ. Let L and K be two languages. We say that L *reduces* to K, written $L \leq K$, if we are able to do the following: assume K is decidable (for the sake of argument), and then show that L is decidable by using the decider for K as a black-box subroutine (i.e. a helper function). Here the languages L and K may or may not be decidable to begin with. But observe that if $L \leq K$ and K is decidable, then L is also decidable. Or in other words, if $L \leq K$ and $K \in \mathsf{R}$, then $L \in \mathsf{R}$. Equivalently, taking the contrapositive, if $L \leq K$ and L is undecidable, then K is also undecidable. So when $L \leq K$, we think of K as being at least as hard as L with respect to decidability (or that L is no harder than K), which justifies using the less-than-or-equal-to sign.



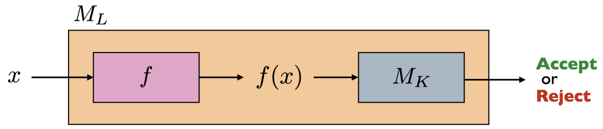
Even though in the diagram above we have drawn one instance of M_K being called within M_L, M_L is allowed to call M_K multiple times with different inputs.

Remark (Turing reductions)**.** In the literature, the above idea is formalized using the notion of a *Turing reduction* (with the corresponding symbol $\leq_T$). In order to define it formally, we need to define Turing machines that have access to an *oracle*. See Section (Bonus: Oracle Turing Machines) for details.

The proofs of Theorem (SA_{TM} is undecidable), Theorem ($\text{ACCEPTS}_{\text{TM}}$ is undecidable), and Theorem (HALTS_{TM} is undecidable) correspond to

$$\begin{aligned} \overline{\text{SA}_{\text{TM}}} &\leq \text{SA}_{\text{TM}}, \\ \text{SA}_{\text{TM}} &\leq \text{ACCEPTS}_{\text{TM}}, \\ \text{ACCEPTS}_{\text{TM}} &\leq \text{HALTS}_{\text{TM}}. \end{aligned}$$

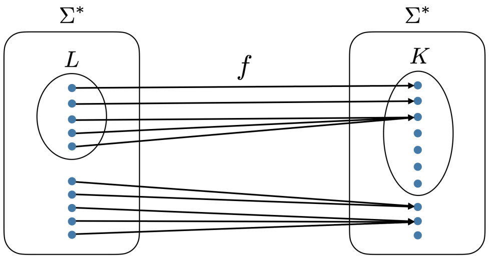
Note (Mapping reductions)**.** Many reductions $L \leq K$ have the following very special structure. The TM M_L is such that on input x, it first transforms x into a new string $y = f(x)$ by applying some computable transformation f. Then $f(x)$ is fed into M_K. The output of M_L is the same as the output of M_K. In other words $M_L(x) = M_K(f(x))$.



These special kinds of reductions are called *mapping reductions* (or *many-one reductions*), with the corresponding notation $L \leq_m K$. Note that the reduction we have seen from SA_{TM} to $\text{ACCEPTS}_{\text{TM}}$ is a mapping reduction. However, the reduction from $\overline{\text{SA}_{\text{TM}}}$ to SA_{TM} is *not* a mapping reduction because the output of M_K is flipped, or in other words, we have $M_L(x) = \textbf{not } M_K(f(x))$.

Almost all the reductions in this section will be mapping reductions.

Note that to specify a mapping reduction from L to K, all one needs to do is specify a TM M_f that computes $f : \Sigma^* \to \Sigma^*$ with the property that for any $x \in \Sigma^*$, $x \in L$ if and only if $f(x) \in K$.



Definition (More languages related to encodings of TMs)**.** A TM M is *satisfiable* if there is some input string that M accepts. In other words, M is satisfiable if $L(M) \neq \varnothing$. We now define the following languages:

$$\begin{aligned}\mathrm{SAT_{TM}} &= \{\langle M\rangle : M \text{ is a satisfiable TM}\},\\ \mathrm{NEQ_{TM}} &= \{\langle M_1, M_2\rangle : M_1 \text{ and } M_2 \text{ are TMs with } L(M_1) \neq L(M_2)\},\\ \mathrm{HALTS-EMPTY_{TM}} &= \{\langle M\rangle : M \text{ is a TM such that } M(\epsilon) \text{ halts}\},\\ \mathrm{FINITE_{TM}} &= \{\langle M\rangle : M \text{ is a TM such that } L(M) \text{ is finite}\}.\end{aligned}$$

Theorem ($\mathrm{SAT_{TM}}$ is undecidable)**.** *The language* $\mathrm{SAT_{TM}}$ *is undecidable.*

Proof. We want to show that $\mathrm{SAT_{TM}}$ is undecidable. We will do so by reducing a known undecidable language to $\mathrm{SAT_{TM}}$. In particular, we will show that $\mathrm{ACCEPTS_{TM}} \leq \mathrm{SAT_{TM}}$. To carry out this reduction, assume $\mathrm{SAT_{TM}}$ is decidable and let M_{SAT} be a decider for it. Using this decider, we will construct a decider for $\mathrm{ACCEPTS_{TM}}$ to establish the reduction.

We construct a TM that decides $\mathrm{ACCEPTS_{TM}}$ as follows.

```
def $M_{\mathrm{ACCEPTS}}(\langle \text{TM } M,\ \text{string } x\rangle)$:
1. Construct the following string, which we call $\langle M'\rangle$.
2. "def $M'(y)$:
3.    Run $M(x)$.
4.    If it accepts, accept.
5.    If it rejects, reject."
6. Run $M_{\mathrm{SAT}}(\langle M'\rangle)$.
7. If it accepts, accept.
8. If it rejects, reject.
```

We now argue that this machine indeed decides $\mathrm{ACCEPTS_{TM}}$. To do this, we'll show that no matter what input is given to our machine, it always gives the correct answer.

First let's assume we get an input $\langle M, x\rangle$ such that $\langle M, x\rangle \in \mathrm{ACCEPTS_{TM}}$, i.e. $x \in L(M)$. Then observe that $L(M') = \Sigma^*$, because for any input y, $M'(y)$ will accept. When we run $M_{\mathrm{SAT}}(\langle M'\rangle)$ on line 6, it accepts, and so our machine accepts and gives the correct answer.

Now assume that we get an input $\langle M, x\rangle$ such that $\langle M, x\rangle \notin \mathrm{ACCEPTS_{TM}}$, i.e. $x \notin L(M)$. Then either $M(x)$ rejects, or loops forever. If it rejects, then $M'(y)$ rejects for any y. If it loops forever, then $M'(y)$ gets stuck on line 3 for any y. In both cases, $L(M') = \varnothing$.

When we run $M_{\text{SAT}}(\langle M' \rangle)$ on line 6, it rejects, and so our machine rejects and gives the correct answer.

Our machine always gives the correct answer, so we are done. □

Remark (Creating an encoding of a machine vs running it)**.** In the proof above, we have defined the decider M_{ACCEPTS} in which we create the encoding of the machine M', denoted $\langle M' \rangle$. Note that creating $\langle M' \rangle$ (which is simply creating a string) is very different from actually running the machine M'. In particular, even if $M(x)$ loops forever, $M_{\text{ACCEPTS}}(\langle M, x \rangle)$ does **not** loop forever because

(i) M_{ACCEPTS} does not run M', and

(ii) M_{SAT} is assumed to be a decider, which means it always halts.

Theorem (NEQ_{TM} is undecidable)**.** *The language* NEQ_{TM} *is undecidable.*

Proof. We want to show that NEQ_{TM} is undecidable and we will do so by reducing SAT_{TM} (which we know is undecidable) to NEQ_{TM}. To carry out this reduction, assume NEQ_{TM} is decidable and let M_{NEQ} be a decider for it. Using this decider, we will construct a decider for SAT_{TM} to establish the reduction. The construction is as follows.

```
def $M_{\text{SAT}}(\langle \text{TM } M \rangle)$:
1. Construct string $\langle M' \rangle$ where $M'$ is a TM that rejects every input.
2. Run $M_{\text{NEQ}}(\langle M, M' \rangle)$.
3. If it accepts, accept.
4. If it rejects, reject.
```

It is not difficult to see that this machine indeed decides SAT_{TM}. Notice that $L(M') = \varnothing$. So when we run $M_{\text{NEQ}}(\langle M, M' \rangle)$ on line 2, we are deciding whether $L(M) = L(M')$, which means we are deciding whether $L(M) = \varnothing$.

In more detail, if M is such that $\langle M \rangle \in \text{SAT}_{\text{TM}}$, then $L(M) \neq \varnothing$. Then on line 2, M_{NEQ} accepts, and therefore M_{SAT} accepts, giving the correct output. If on the other hand M is such that $\langle M \rangle \notin \text{SAT}_{\text{TM}}$, then $L(M) = \varnothing$. In this case M_{NEQ} on line 2 rejects, and therefore M_{SAT} rejects as well, which is again the correct output. □

Exercise (Practice with undecidability proofs)**.** Show that the following languages are undecidable.

1. $\text{HALTS} - \text{EMPTY}_{\text{TM}} = \{\langle M \rangle : \ M \text{ is a TM and } M(\epsilon) \text{ halts}\}$
2. $\text{FINITE}_{\text{TM}} = \{\langle M \rangle : M \text{ is a TM that accepts finitely many strings}\}$

Solution. Part 1: We want to show $\text{HALTS} - \text{EMPTY}_{\text{TM}}$ is undecidable. We will reduce HALTS_{TM} to $\text{HALTS} - \text{EMPTY}_{\text{TM}}$. So assume $\text{HALTS} - \text{EMPTY}_{\text{TM}}$ is decidable and let $M_{\text{HALTS}-\text{EMPTY}}$ be a decider for it. We construct a decider for HALTS_{TM} as follows.

```
def $M_{\text{HALTS}}(\langle \text{TM } M, \text{ string } x \rangle)$:
Construct the following string, which we call $\langle M' \rangle$.
"def $M'(y)$:
  Run $M(x)$.
  Ignore the output and accept."
Run $M_{\text{HALTS}-\text{EMPTY}}(\langle M' \rangle)$.
If it accepts, accept.
If it rejects, reject.
```

To see that this is a correct decider for HALTS_{TM}, first consider any input $\langle M, x\rangle$ such that $\langle M, x\rangle \in \text{HALTS}_{\text{TM}}$, i.e., $M(x)$ halts. By the construction of M', this implies that $M'(y)$ halts (and accepts) for any string y. So $M_{\text{HALTS}-\text{EMPTY}}(\langle M'\rangle)$ accepts, and our decider above accepts as well. So in this case, the decider gives the correct answer.

Now consider any input $\langle M, x\rangle$ such that $\langle M, x\rangle \notin \text{HALTS}_{\text{TM}}$, i.e., $M(x)$ loops. Then for any input y, $M'(y)$ would get stuck on line 3, and would never halt. This means $M_{\text{HALTS}-\text{EMPTY}}(\langle M'\rangle)$ rejects, and our decider rejects as well, as desired.

For any input, our decider gives the correct answer, and the proof is complete.

Part 2: Our goal is to show that $\text{FINITE}_{\text{TM}}$ is undecidable. For this, we will reduce HALTS_{TM} to $\text{FINITE}_{\text{TM}}$. We assume $\text{FINITE}_{\text{TM}}$ is decidable, so let M_{FINITE} be a decider for it. Here is the description of a decider for HALTS_{TM}.

```
def $M_{\text{HALTS}}(\langle \text{TM } M, \text{ string } x\rangle)$:
1. Construct the following string, which we call $\langle M'\rangle$.
2. "def $M'(y)$:
3.     Run $M(x)$.
4.     Ignore the output and accept."
5. Run $M_{\text{FINITE}}(\langle M'\rangle)$.
6. If it accepts, reject.
7. If it rejects, accept.
```

To see that this is a correct decider for HALTS_{TM}, first consider any input $\langle M, x\rangle$ such that $\langle M, x\rangle \in \text{HALTS}_{\text{TM}}$, i.e., $M(x)$ halts. By the construction of M', this implies that $M'(y)$ accepts for any string y. So $L(M') = \Sigma^*$ (an infinite set), and therefore $M_{\text{FINITE}}(\langle M'\rangle)$ rejects. In this case, our decider for HALTS_{TM} accepts and gives the correct answer.

Now consider any input $\langle M, x\rangle$ such that $\langle M, x\rangle \notin \text{HALTS}_{\text{TM}}$, i.e., $M(x)$ loops. Then for any input y, $M'(y)$ would get stuck on line 3, and would never halt. So $L(M') = \varnothing$ (a finite set), and therefore $M_{\text{FINITE}}(\langle M'\rangle)$ accepts. In this case, our decider for HALTS_{TM} rejects and gives the correct answer.

For any input, our decider gives the correct answer, and the proof is complete. ■

Theorem ($\text{SAT}_{\text{TM}} \leq \text{HALTS}_{\text{TM}}$)**.** $\text{SAT}_{\text{TM}} \leq \text{HALTS}_{\text{TM}}$.

Proof. We want to show that deciding SAT_{TM} reduces to deciding HALTS_{TM}. For this, we assume HALTS_{TM} is decidable. Let M_{HALTS} be a decider for HALTS_{TM}. Using it, we need to construct a decider M_{SAT} for SAT_{TM}.

The main idea in the construction of M_{SAT} is as follows. Given as input $\langle M\rangle$, we want to define the description of another TM M' so that M is satisfiable (i.e. there exists x such that $M(x)$ accepts) if and only if M' halts (regardless of what the input is). The idea behind the definition of M' is to do an exhaustive search, going through every possible input string y, in order to find one y that M accepts. If we can define such an M', then indeed, M is satisfiable if and only if M' halts. That being said, we have to be careful about how M' is defined. In particular, we have to be careful that M' does not get stuck in a loop even though there is some y that M accepts. With this in mind, we construct M_{SAT} as follows.

```
def $M_{\text{SAT}}(\langle \text{TM } M \rangle)$:
Construct the following string, which we call $\langle M' \rangle$.
"def $M'(x)$:
  For $t = 1, 2, 3, \ldots$ :
    For each string $y$ with $|y| \leq t$:
      Simulate $M(y)$ for at most $t$ steps.
      If it accepts, accept."
Run $M_{\text{HALTS}}(\langle M', \epsilon \rangle)$.
If it accepts, accept.
If it rejects, reject.
```

We now argue that this machine indeed decides SAT_{TM}. First consider an input $\langle M \rangle$ such that $\langle M \rangle \in \text{SAT}_{\text{TM}}$. This means that there is some word y such that $M(y)$ accepts. Note that M', by construction, does an exhaustive search, so if such a y exists, then M' will eventually find it, and accept. So $M'(x)$ halts for any x. When we run $M_{\text{HALTS}}(\langle M', \epsilon \rangle)$, it accepts, and our machine accepts, giving the correct answer.

Now consider an input $\langle M \rangle$ such that $\langle M \rangle \notin \text{SAT}_{\text{TM}}$. This means $L(M) = \varnothing$. Observe that the only way M' halts is if $M(y)$ accepts for some string y. Since $L(M) = \varnothing$, $M'(x)$ loops forever for any string x. This means that when we run $M_{\text{HALTS}}(\langle M', \epsilon \rangle)$, it rejects, and so our decider above rejects, as desired. □

Exercise (SAT_{TM} is in RE)**.** Describe a TM that semi-decides SAT_{TM}. Proof of correctness is not required.

Solution.

```
def $M_{\text{SAT}}(\langle \text{TM } M \rangle)$:
For $t = 1, 2, 3, \ldots$ :
  For each string $y$ with $|y| \leq t$:
    Simulate $M(y)$ for at most $t$ steps.
    If it accepts, accept.
```

■

Exercise (Practice with reduction definition)**.** Let $L, K \subseteq \{0, 1\}^*$ be languages. Prove or disprove the following claims.

1. If $L \leq K$ then $K \leq L$.
2. If $L \leq K$ and K is regular, then L is regular.

Solution. Part 1: The claim is false. Let L be any decidable language. For example, we can take $L = \varnothing$. The decider for L is a machine that rejects no matter what the input is. Let $K = \text{HALTS}_{\text{TM}}$. Then to establish $L \leq K$, we need to argue that given a decider for HALTS_{TM}, we can decide $\varnothing$. Since $\varnothing$ is decidable, this is true (and we don't even need to make use of a decider for HALTS_{TM}). On the other hand, it is **not** true that $\text{HALTS}_{\text{TM}} \leq \varnothing$. For the sake of contradiction, if it was true, then this would mean that using a decider for $\varnothing$, we can decide HALTS_{TM}. And this would imply that HALTS_{TM} is decidable, a contradiction.

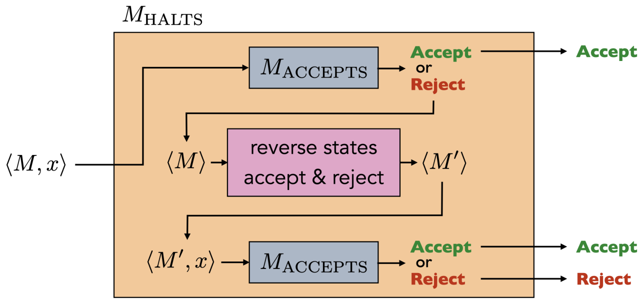
Part 2: The claim is false. Consider $L = \{0^n 1^n : n \in \mathbb{N}\}$ and $K = \varnothing$. We have $L \leq K$ because L is a decidable language (we don't even need to make use of the decider for K). Furthermore, K is regular, but L is not. ■

Exercise (Calling the helper function more than once)**.** In all the reductions $L \leq K$ presented in this chapter, the TM M_L calls the TM M_K exactly once. (And in fact, almost all the reductions in this chapter are mapping reductions). Present a reduction from HALTS_{TM} to $\text{ACCEPTS}_{\text{TM}}$ in which M_{HALTS} calls M_{ACCEPTS} twice, and both calls/outputs are used in a meaningful way.

Solution. We are assuming $\text{ACCEPTS}_{\text{TM}}$ is decidable. Let M_{ACCEPTS} be a decider for it. Based on this assumption, we will show that HALTS_{TM} is decidable. Here is the description of a decider solving HALTS_{TM}:

```
def $M_{\text{HALTS}}(\langle \text{TM } M, \text{ string } x\rangle)$ :
1. Run $M_{\text{ACCEPTS}}(\langle M, x\rangle)$.
2. If it accepts, accept.
3. Construct string $\langle M'\rangle$ by flipping the accept and reject states of $\langle M\rangle$.
4. Run $M_{\text{ACCEPTS}}(\langle M', x\rangle)$.
5. If it accepts, accept.
6. If it rejects, reject.
```

Or equivalently, as a diagram:



Let's now prove that this is indeed a correct decider for HALTS_{TM}. To do this, we'll show that no matter what input is, the machine always gives the correct answer.

First let's assume we get any input $\langle M, x\rangle$ such that $\langle M, x\rangle \in \text{HALTS}_{\text{TM}}$. In this case our machine is supposed to accept. Since $M(x)$ halts, we know that $M(x)$ either ends up in the accepting state, or it ends up in the rejecting state. If it ends up in the accepting state, then $M_{\text{ACCEPTS}}(\langle M, x\rangle)$ accepts (on line 1 of our machine's description), and so our program accepts and gives the correct answer on line 2. If on the other hand, $M(x)$ ends up in the rejecting state, then $M'(x)$ ends up in the accepting state. Therefore $M_{\text{ACCEPTS}}(\langle M', x\rangle)$ accepts (on line 4 of our machine's description), and so our program accepts and gives the correct answer on line 5.

Now let's assume we get any input $\langle M, x\rangle$ such that $\langle M, x\rangle \notin \text{HALTS}_{\text{TM}}$. In this case our machine is supposed to reject. Since $M(x)$ does not halt, it never reaches the accepting or the rejecting state. By the construction of M', this also implies that $M'(x)$ never reaches the accepting or the rejecting state. Therefore first $M_{\text{ACCEPTS}}(\langle M, x\rangle)$ (on line 1 of our machine's description) will reject. And then $M_{\text{ACCEPTS}}(\langle M', x\rangle)$ (on line 4 of our machine's description) will reject. Thus our program will reject as well, and give the correct answer on line 6. ■

5 Non-Semi-Decidable Languages

We have already proved that $\overline{\text{SA}_{\text{TM}}}$ is not semi-decidable directly by diagonalization. We can show that other languages are not semi-decidable by making use of Theorem (**??**).

Theorem (Non-semi-decidability through semi-decidable undecidable languages)**.** *If a language L is semi-decidable but undecidable, then $\overline{L}$ is not semi-decidable. In other words, if $L \in \mathsf{RE} \setminus \mathsf{R}$, then $\overline{L} \notin \mathsf{RE}$.*

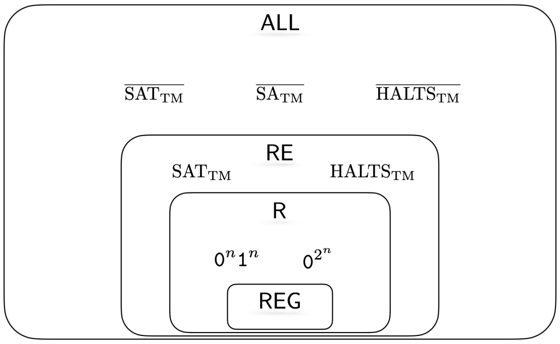
Proof. We know L is semi-decidable. If (for the sake of contradiction) $\overline{L}$ is semi-decidable as well, then by Theorem (**??**), L would be decidable, which contradicts our assumption that it is undecidable. □

Corollary (Non-semi-decidable languages)**.** *The languages $\overline{\mathrm{SA}_{\mathrm{TM}}}$, $\overline{\mathrm{ACCEPTS}_{\mathrm{TM}}}$, and $\overline{\mathrm{HALTS}_{\mathrm{TM}}}$ are not semi-decidable, i.e. they are not in* RE.

Proof. The languages $\mathrm{SA}_{\mathrm{TM}}$, $\mathrm{ACCEPTS}_{\mathrm{TM}}$, and $\mathrm{HALTS}_{\mathrm{TM}}$ are all semi-decidable (i.e. are in RE), but are not decidable (i.e. are not in R). So the statement of the corollary follows from the previous theorem. □

Note (Comparing REG, R, and RE)**.** The relationship among the complexity classes REG, R, and RE and can be summarized as follows.

- REG $\subsetneq$ R since $\{0^n1^n : n \in \mathbb{N}\}$ is not regular but is decidable.
- There are languages like $\mathrm{SA}_{\mathrm{TM}}$, $\mathrm{ACCEPTS}_{\mathrm{TM}}$, $\mathrm{HALTS}_{\mathrm{TM}}$, $\mathrm{SAT}_{\mathrm{TM}}$ that are undecidable (not in R), but are semi-decidable (in RE).
- The complements of the languages above are not in RE.


ALL
$\overline{\mathrm{SAT}_{\mathrm{TM}}}$
$\overline{\mathrm{SA}_{\mathrm{TM}}}$
$\overline{\mathrm{HALTS}_{\mathrm{TM}}}$
RE
$\mathrm{SAT}_{\mathrm{TM}}$
$\mathrm{HALTS}_{\mathrm{TM}}$
R
0^n1^n
0^{2^n}
REG

6 Bonus: Oracle Turing Machines

Definition (Oracle Turing machines)**.** Given some decision problem $g : \Sigma^* \to \{0,1\}$, we define a *g-oracle Turing machine* (or simply g-OTM, or just OTM if g is understood from the context) as a Turing machine with an additional *oracle instruction* which can be used in the TM's high-level description. The additional instruction has the following form. For any string x and any variable name y of our choosing, we are allowed to use

$$y = g(x).$$

After this instruction, the variable y holds the value $g(x)$ and can be used in the rest of the Turing machine's description. In this context, the function g is called an *oracle*.

We'll use a superscript of g (e.g. M^g) to indicate that a machine is a g-OTM.

Similar to TMs, we write $M^g(x) = 1$ if the oracle TM M^g, on input x, accepts. We write $M^g(x) = 0$ if it rejects. And we write $M^g(x) = \infty$ if it loops forever.

If $f : \Sigma^* \to \{0,1\}$ is a decision problem such that for all $x \in \Sigma^*$, $M^g(x) = f(x)$, then we say M^g *describes* f (and the language corresponding to f).

Example. Let $h : \Sigma^* \to \{0, 1\}$ denote the decision problem corresponding to HALTS_{TM}. Here is a description of an h-oracle Turing machine.

```
def $M^h(\langle \text{TM } M,\ \text{string } x\rangle)$ :
y = $h(\langle M, x\rangle)$
If $y = 1$:
  Run $M(x)$.
  If it accepts, accept.
  If it rejects, reject.
Else:  reject.
```

Note that M^h describes the language $\text{ACCEPTS}_{\text{TM}}$.

Remark (Low-level definition of oracle TMs)**.** One can give a precise low-level definition for an oracle TM, however, we will not need or use that level of detail, so we choose to omit it.

Definition (Turing reduction)**.** Let L and K be two languages, and let k be the decision problem corresponding to K. We say that L Turing-reduces to K, written $L \leq_T K$, if there is an oracle Turing machine M^k describing L.

Note (Turing reductions and decidability)**.** Observe that if $L \leq_T K$ and K is decidable, then L is decidable. This is because if $L \leq_T K$, then by definition, there is an oracle Turing machine M^k that describes L. If K is decidable, there is a TM M_K deciding K. Then in M^k, if we replace all calls to the oracle k with M_K, we end up with a TM deciding L.

It follows (taking the contrapositive of the observation) that if $L \leq_T K$ and L is undecidable, then K is undecidable.

Important (TMs as a description model)**.** As we have already seen, the set of all languages, $\wp(\Sigma^*)$, is not encodable. Most languages do not have a finite description. In addition to thinking about the TM model as a computational model, it is useful to also think of it as a *finite description* model: Every TM finitely describes/defines the language that it decides (or semi-decides).

The TM model is not a comprehensive finite description model. There are many finitely describable languages (e.g., $\overline{\text{SA}_{\text{TM}}}$, $\text{ACCEPTS}_{\text{TM}}$, HALTS_{TM}) that a TM cannot finitely describe. The oracle TM model extends the reach of the TM model as a finite description model.

If we inspect the proof of Theorem (Turing's 1st Undecidability Theorem), we can see that we never really made use of the fact that TMs represent a computational model. The crucial part of the argument was that the set of TMs is encodable/countable. Or in other words, we only needed that the TM model is a finite description model for languages/decision problems. This means that for other finite description models, like the oracle TM model, we can carry out the same argument.

An important lesson here is that the main limitation Theorem (Turing's 1st Undecidability Theorem) highlights about TMs is not that they form a computing model, but that they form a finite description model.

Note (Languages involving encodings of oracle TMs)**.** Fix an oracle g. Then we can define the languages $\text{ACCEPTS}_{\text{OTM}}$, $\text{HALTS}_{\text{OTM}}$, SA_{OTM}, and so on, analogous to how they are defined for TMs. For example,

$$\text{HALTS}_{\text{OTM}} = \{\langle M^g, x\rangle : M^g \text{ is a } g\text{-OTM which halts on input } x\}.$$

Theorem (Limits of oracle Turing machines)**.** *Fix an oracle g. Then there is no oracle Turing machine that describes $\overline{\text{SA}_{\text{OTM}}}$.*

Proof. The proof is exactly the same as the proof of Theorem (Turing's 1st Undecidability Theorem). Note that the set of all g-oracle Turing machines is encodable/countable. Therefore we can diagonalize against the set of all g-oracle TMs. The natural diagonal element that the diagonalization spits out is $\overline{\text{SA}_{\text{OTM}}}$. □

7 Check Your Understanding

Problem.

1. State 4 languages that are undecidable.
2. State an undecidable language whose description does not involve Turing machines.
3. Describe 3 languages that are not semi-decidable.
4. Describe how one can show that a language is undecidable using the notion of a reduction.
5. True or false: For languages K and L, if $K \leq L$, then L is undecidable.
6. True or false: For languages K and L, if $K \leq L$ and $L \leq K$, then L and K are both decidable.
7. True or false: Fix some alphabet Σ. The set of regular languages over Σ is countable.
8. True or false: Fix an alphabet Σ. The set of undecidable languages is countable.
9. True or false: Fix an alphabet Σ. The set of decidable languages is infinite.
10. True or false: If languages K and L are both undecidable, then their union is also undecidable.
11. True or false: If a language L is undecidable, then L is infinite.
12. True or false: $\Sigma^* \leq \varnothing$.
13. True or false: $\text{HALTS}_{\text{TM}} \leq \Sigma^*$.
14. True or false: Every decidable language reduces to HALTS_{TM}.
15. True or false: For all unary languages L (i.e. languages over the alphabet $\Sigma = \{\mathtt{1}\}$), L is decidable.
16. In a previous chapter, we saw that the language SAT_{DFA} is decidable. In particular, we saw that given the encoding of a DFA D, we can determine if D accepts a string by checking if in the state diagram of D, there is a directed path from the initial state to one of the accepting states.

    In this chapter, we saw that the language SAT_{TM} is undecidable. Show that the strategy above that we used for DFAs does not apply to TMs by drawing the state diagram of a TM M with $L(M) = \varnothing$, but there *is* a path from the starting state to the accepting state.

8 High-Order Bits

Important. Here are the important things to keep in mind from this chapter.

1. The existence of undecidable languages follows by a counting argument: The set of all languages is uncountable whereas the set of decidable languages is countable. This shows that almost all languages are undecidable, however, it does not give us an explicit undecidable language.

2. We can use diagonalization to come up with an explicit language that is undecidable.

3. Reductions are very important. We have seen in an earlier chapter that reductions can be used to expand the landscape of decidable languages. In this chapter, you see that it can be used to expand the landscape of undecidable languages.

4. This chapter has many examples of undecidability proofs. It is easy to fall in the trap of trying to memorize the template of such proofs rather than really understanding the logic behind how and why these proofs work. If you struggle to understand these proofs, it is usually a sign that there are gaps in your knowledge from previous chapters. Come talk to us so we can help you identify those gaps.