Computability and Noncomputability

Up to now, we have been concerned with how efficiently various problems can be computed. Now we will address the issue of which problems can be computed at all, even when we have no concern about efficiency. This work was initiated by Alan Turing in 1936 when he introduced (what are now called) Turing Machines as a model of computation.

From now on, for convenience, we will assume $\Sigma$ is a fixed finite input alphabet. All our Turing machines will have $\Sigma$ as their input alphabet, and whenever we encode stuff using brackets ($\langle, \rangle$) it will be over the alphabet $\Sigma$. We will also assume that $\Sigma$ contains the symbols 0 and 1. Here are our basic definitions. Note that Sipser, in his book, uses “recognizable” instead of “semi-decidable”. (The reader should also note that in some of the literature, the term “recursive” is used instead of “decidable”, and the term “recursively enumerable” is used instead of “semi-decidable”. The reasons for this are historical, and we will not go into them here.)

**Definition 1.** We say $f : \Sigma^* \rightarrow \Sigma^*$ is computable if there is a Turing machine $M$ that computes $f$.

We say $L \subseteq \Sigma^*$ is decidable (or, equivalently, computable) if there is a Turing machine $M$ such that $L = L(M)$ and $M$ halts on every input.

We say $L \subseteq \Sigma^*$ is semi-decidable (or, equivalently, partially decidable or semi-computable or partially computable or recognizable) if there is a Turing machine $M$ such that $L = L(M)$.

**Notation:** We use $D$ to stand for the class of decidable languages, and $SD$ for the class of semi-decidable languages.

The intuition is that a language is decidable or computable if there is an algorithm that, on each input $x$, eventually halts saying whether or not $x$ is in the language. To say that a language $L$ is semi-decidable means that there is an algorithm that accepts precisely the strings in $L$; however, for a string $x \notin L$, the algorithm will either reject or not halt. In order for this intuition to be valid, we have to accept the **Church-Turing Thesis**.

**Church-Turing Thesis:**

Anything that intuitively deserves to be called an “algorithm” can be simulated by a Turing Machine. So any function that can be computed (or any language decided or semi-decided) by some kind of “algorithm” is actually computable (or decidable, or semi-decidable) in our sense.

This Thesis is very widely accepted, because any other kind of algorithm that we are able to think of can be easily simulated by a Turing Machine (although not necessarily efficiently). For example, it is easy to see how to simulate other variations of Turing machines by a
Turing Machine: examples of such variations are many tapes, one-way infinite tapes, and multi-dimensional tapes. It is also easy to see how to simulate RAMs (Random Access Machines). It is also easy to see how to simulate “probabilistic” machines or “quantum” machines (although not efficiently), but we will not define these machines here. The thesis has been challenged by some who (unlike those working in AI) believe that there is something about the Human brain (perhaps based on quantum mechanics) that fundamentally cannot be simulated by a Turing Machine.

It is trivial to see that every function in $\text{FP}$ is computable and that every language in $\text{P}$ is decidable. The following is slightly less obvious.

**Lemma 1.** $\text{NP} \subseteq \text{D}$.  

**Proof:** Let $c, d \in \mathbb{N}$ and let $R$ be a 2-place, (polynomial-time) computable predicate such that for all $x \in \Sigma^*$,  
\[ x \in L \iff \text{for some } y \in \Sigma^*, |y| \leq c|x|^d \text{ and } R(x, y). \]

Let $M$ be a Turing Machine that determines membership in $R$.

Given an input $x \in \Sigma^*$, the following algorithm decides whether or not $x \in L$: for each string $y \in \Sigma^*$ of length less than or equal to $c|x|^d$, run $M$ on $(x, y)$; if at least one of these runs of $M$ accepts, then we halt and accept $x$; otherwise we reject $x$.

Therefore $L$ is decidable. In fact, we have shown that $L$ can be decided by a Turing Machine that runs in time exponential in the length of the input, that is, in time $\leq 2^{2^{c|x|^d}}$ for some $d$.

\[ \square \]

It is clear that if $L$ is a decidable language, then $L$ is also semidecidable. In fact, we have the following chain of containments:

$\text{P} \subseteq \text{NP} \not\subseteq \text{D} \not\subseteq \text{SD} \not\subseteq \{L \mid L \subseteq \Sigma^*\}$

**Analogy:** $\text{P}$ is to $\text{NP}$ as $\text{D}$ is to $\text{SD}$.

Notice that the definitions of $\text{P}$ and $\text{D}$ are almost the same. The difference is that for $\text{D}$ we drop the requirement that the accepting Turing machine runs in polynomial time. Similarly we can characterize $\text{SD}$ in the same style as the definition of $\text{NP}$ (see Definition 1, page 1 in the notes “NP and NP-Completeness”). In fact, we need only delete the words “in polynomial time”, and remove the bound on the length of the certificate $y$.

**Theorem 1.** $L \in \text{SD}$ iff there is a two-place predicate $R \subseteq \Sigma^* \times \Sigma^*$ such that $R$ is computable, and such that for all $x \in \Sigma^*$  
\[ x \in L \iff \text{there exists } y \text{ such that } R(x, y) \]

The proof is left as an exercise.
We will prove shortly that there are languages that are not semi-decidable, and that there are semi-decidable languages that are not decidable. Of course, it is an open question whether \( P \) is properly contained in \( NP \).

We will prove later that there are decidable languages that are not in \( NP \); in fact, there are decidable languages that cannot be decided by any Turing Machine that runs in exponential time. An interesting example is from Mathematical Logic. Consider the predicate calculus language that has a 2-place function symbol +, and let \( PR \) be the set of sentences that are true in the structure with domain \( \mathbb{N} \) where + is interpreted as addition; this language is called "Presburger Arithmetic".

An example of a formula in \( PR \) is
\[ \exists x \forall y \forall z (x = y + z \rightarrow (y \neq z \land (x = y \lor x = z))). \]

We can prove that \( PR \) is decidable. But we can also prove that there is a constant \( c > 0 \) such that every Turing Machine that decides \( PR \) requires time at least \( 2^{2^{cn}} \) on sufficiently large inputs of size \( n \). Since for every language \( L \in NP \) there is a machine that decides \( L \) in time \( 2^{n^d} \), we see that \( PR \notin NP \).

Define \( A_{TM} = \{ \langle M, w \rangle | M \text{ is a Turing Machine and } M \text{ accepts } w \} \).

We will show later that \( A_{TM} \) is not decidable. For now, we show that it is semi-decidable.

**Lemma 2.** \( A_{TM} \in SD \)

**Proof Outline:** The following algorithm accepts precisely the strings in \( A_{TM} \):

Given a string \( x \), check if \( x \) is of the right form \( \langle M, w \rangle \), and if not, reject.

So say \( x = \langle M, w \rangle \). Then simulate machine \( M \) running on input \( w \); if the simulation halts, then accept if \( M \) accepts and reject if \( M \) rejects. Clearly this algorithm accepts precisely \( A_{TM} \).

Note that this algorithm does not halt on every input, since if \( M \) doesn’t halt on \( w \), then the algorithm doesn’t halt on \( \langle M, w \rangle \). □

Note that the above algorithm is essentially an interpreter; that is a program which takes as input both a program \( P \) and an input \( w \) to that program, and simulates \( P \) on input \( w \). In this case the program \( P \) is given by a Turing machine \( M \). A Turing machine interpreter is often called a universal Turing machine. Turing described a universal Turing machine in some detail in his original 1936 paper, an ideal which paved the way for later interpreters operating on real computers.

The following are some basic properties of the classes of decidable and semi-decidable languages. Note that the complement of every decidable language is decidable, but (as we will see a bit later) it is not the case that the complement of every semi-decidable language is semi-decidable. (By \( \overline{L} \), we mean the complement \( \Sigma^* - L \).)

**Notation:** Let \( L \subseteq \Sigma^* \). Let \( C_L : \Sigma^* \rightarrow \Sigma^* \) be defined by:
\( C_L(x) \) has value 1 if \( x \in L \), and value 0 otherwise. Then \( C_L \) is the characteristic function of \( C_L \).
Lemma 3. For any language $L \subseteq \Sigma^*$, $L$ is decidable $\iff C_L$ is computable.

Proof: An easy exercise.  

Lemma 4. Let $L_1, L_2 \subseteq \Sigma^*$ be decidable languages. Then $L_1 \cup L_2$ and $L_1 \cap L_2$ are all decidable languages.

Proof: An easy exercise.

Lemma 5. Let $L_1, L_2 \subseteq \Sigma^*$ be semi-decidable languages. Then $L_1 \cap L_2$ and $L_1 \cup L_2$ are also semi-decidable.

Proof: Say that $L_1 = \mathcal{L}(M_1)$ and $L_2 = \mathcal{L}(M_2)$. To show that $L_1 \cap L_2$ is semi-decidable, we want to construct a machine $M_3$ such that $\mathcal{L}(M_3) = L_1 \cap L_2$. We design $M_3$ as follows. On input $x$, $M_3$ runs $M_1$ on $x$; if $M_1(x)$ halts and rejects, then $M_3$ rejects; if $M_1(x)$ halts and accepts, then $M_3$ runs $M_2$ on $x$, halting if and when $M_2$ halts, and accepting if and only if $M_2$ does.

To show that $L_1 \cup L_2$ is semi-decidable, we want to construct a machine $M_4$ such that $\mathcal{L}(M_4) = L_1 \cup L_2$. This is harder than the previous construction, since we cannot first run $M_1$ and then run $M_2$, since if $M_1$ doesn’t halt we will never get a chance to run $M_2$ (which is necessary if we want to accept the union of $L_1$ and $L_2$). Instead, we will run $M_1$ and $M_2$ together, in effect interleaving their computations. This idea of running different computations at the same time is commonly called “dovetailing”. (Apparently this use of the word “dovetail” comes from card shuffling, and its use there comes from a certain kind of interleaved joint in cabinet making, and its use there comes from the fact that a part of the joint resembles a dove’s tail.)

On input $x$, we define $M_4$ to work as follows. $M_4$ runs $M_1$ on $x$ and $M_2$ on $x$ for 1 step each, then $M_4$ runs $M_1$ on $x$ and $M_2$ on $x$ for 2 steps each, then $M_4$ runs $M_1$ on $x$ and $M_2$ on $x$ for 3 steps each, etc. (Note that when we say to “run $M_i$ on $x$ for $k$ steps”, we mean that we run $M_i$ on $x$ for $k$ steps or until it halts, whichever comes first.) This continues until (which may never happen) either $M_1(x)$ or $M_2(x)$ is discovered to accept, at which point $M_4$ halts and accepts. Clearly $\mathcal{L}(M_4) = L_1 \cup L_2$.

Theorem 2. Let $L \subseteq \Sigma^*$, then $L$ is decidable $\iff$ both $L$ and $\overline{L}$ are semi-decidable.

Proof:
Proof of $\Rightarrow$: Say that $L$ is decidable. By a previous Lemma, $\overline{L}$ is also decidable. Since every decidable language is also semi-decidable, both $L$ and $\overline{L}$ are semi-decidable.

Proof of $\Leftarrow$: Say that $L = \mathcal{L}(M_1)$ and $\overline{L} = \mathcal{L}(M_2)$. We want to construct a machine $M$ such that on any input $x$, $M$ halts, deciding whether or not $x \in L$. 

4
We define $M$ to work on input $x$ as follows. We will use dovetailing. $M$ runs $M_1$ on $x$ and $M_2$ on $x$ for 1 step each, then $M$ runs $M_1$ on $x$ and $M_2$ on $x$ for 2 steps each, then $M$ runs $M_1$ on $x$ and $M_2$ on $x$ for 3 steps each, etc. Since $M_2$ accepts the complement of $M_1$, eventually $M$ will discover that one of these two machines accepts $x$. If $M$ finds that $M_1$ accepts $x$, then $M$ halts and accepts; if $M$ finds that $M_2$ accepts $x$, then $M$ halts and rejects.

So $\mathcal{L}(M) = L$ and $M$ halts on every input, so $L$ is decidable. □

We will now give an alternative characterization of the notion of semi-decidable. It turns out that a language $L$ is semi-decidable if and only if it can be “enumerated” by a Turing Machine. To make this precise we have to define the notion of an Enumerating Turing machine. An Enumerating Turing Machine $M_e$ will have no inputs, but rather will always start with a blank tape. The machine will have, in addition, an “output tape”; this tape is initially blank, and it contains a head that can only write, and that at each step either stays where it is or moves one square to the right; this head can only write symbols from $\Sigma \cup \{\$\}$. If and when the output head has written a string $\alpha\$$ where $\alpha \in \Sigma^*$, then the output tape is erased and we say that $M_e$ has printed (or enumerated, or outputted) the string $\alpha$. Note that $M_e$ may eventually halt, or may run forever; $M_e$ may output no strings, a finite sequence of strings, or an infinite sequence of strings; some strings may be enumerated by $M_e$ more than once. We define $E(M_e)$ to be the set of strings $\alpha$ such that $M_e$, at some point, outputs $\alpha$.

Theorem 3. Let $L \subseteq \Sigma^*$. Then
$L$ is semi-decidable $\iff$ $L = E(M_e)$ for some Enumerating Turing Machine $M_e$.

Proof:
Proof of $\Leftarrow$: Let $M_e$ be an Enumerating Turing Machine such that $L = E(M_e)$. We want to construct a Turing Machine $M$ such that $\mathcal{L}(M) = L$.

We define $M$ to work on input $x$ as follows.
$M$ simulates $M_e$ (where $M_e$ starts with an initially blank tape), checking to see if $M_e$ ever outputs $x$. If and when $M_e$ outputs $x$, $M$ halts and accepts; if $M_e$ halts without ever having outputted $x$, $M$ halts and rejects. (Note that if $M_e$ runs forever without outputting $x$, then $M$ will run forever.) Clearly $\mathcal{L}(M) = L$.

Proof of $\Rightarrow$: Let $M$ be a Turing Machine such that $L = \mathcal{L}(M)$. We want to construct an Enumerating Turing Machine $M_e$ such that $E(M_e) = L$. We will use dovetailing, simulating (in parallel) $M$ running on all possible inputs.

We define $M_e$ to work (starting with the blank tape) as follows.
$M_e$ will simulate $M$ running on all strings of length at most 1 for 1 step; then $M_e$ will simulate $M$ running on all strings of length at most 2 for 2 steps; then $M_e$ will simulate $M$ running on all strings of length at most 3 for 3 steps; etc; whenever one of the simulations of $M$ on a string $x$ is discovered to halt and accept, then $M_e$, before continuing with its simulations, outputs the string $x$. Clearly $E(M_e) = L$. □
Proving Undecidability

We wish to show that some languages are not decidable, and in particular that $A_{TM}$ is not decidable. Toward this end, we will define a language DIAG in such a way as to make it as easy as possible to show DIAG is not decidable; we will then use the fact that DIAG is not decidable to show that $A_{TM}$ is not decidable.

The language DIAG will be defined using the technique of “diagonalization”. The idea is that we want to define DIAG so as to make sure it is different from $L(M)$ for every Turing Machine $M$ (this will actually imply that DIAG is not even semi-decidable); we will do this by making sure that DIAG differs from $L(M)$ on the string $⟨M⟩$.

We define

$$\text{DIAG} = \{⟨M⟩ | M \text{ is a Turing machine and } ⟨M⟩ \notin L(M)\}.$$ 

Lemma 6. $\text{DIAG} \notin \text{D}$

Proof: We will actually show that DIAG is not even semi-decidable. Assume, to the contrary, that there is a Turing Machine $M$ such that $L(M) = \text{DIAG}$. Consider the string $⟨M⟩$. By definition of DIAG, $⟨M⟩ \in \text{DIAG} ⇔ ⟨M⟩ \notin L(M)$. But since $L(M) = \text{DIAG}$, $⟨M⟩ \notin L(M) ⇔ ⟨M⟩ \notin \text{DIAG}$. So $⟨M⟩ \in \text{DIAG} ⇔ ⟨M⟩ \notin \text{DIAG}$, which is impossible. □

To see why the above proof is called a “diagonal argument”, consider an enumeration $M_1, M_2, ...$ of all possible Turing machines (obtained by enumerating the strings that describe them). Consider an infinite matrix, in which the columns and rows are each labeled successively by $M_1, M_2, ...$. The entry in column $M_i$ and row $M_j$ is either Y or N depending on whether $⟨M_i⟩ \in L(M_j)$. The language DIAG is obtained from the diagonal of this matrix by interchanging Y and N.

Theorem 4. $A_{TM} \notin \text{D}$

Proof: Assume, to the contrary, that $M_0$ is a Turing Machine such that $L(M_0) = A_{TM}$ and such that $M_0$ halts on every input. We can now construct a Turing Machine $M_1$ that halts on every input and accepts DIAG.

Define $M_1$ to work as follows on input $x$.

If $x$ is not of the form $⟨M⟩$ where $M$ is a Turing machine, then $M_1$ rejects.

So assume $x = ⟨M⟩$.

Now $M_1$ runs $M_0$ on $⟨M, ⟨M⟩⟩$;

if $M_0$ accepts, then $M_1$ rejects;

and if $M_0$ rejects, then $M_1$ accepts.

Clearly $L(M_1) = \text{DIAG}$ and $M_1$ halts on every input, implying that DIAG is decidable, which is a contradiction. □

Remark: Looking ahead to Definition 2 below, the above proof essentially shows that $\text{DIAG} \leq_m A_{TM}$. 

6
Corollary 1. \( \overline{A_{TM}} \notin \text{SD} \).

**Proof:** We know that \( A_{TM} \) is semi-decidable. If \( \overline{A_{TM}} \) were also semi-decidable, then by a previous Theorem, \( A_{TM} \) would be decidable, and we have just shown that it isn’t. \( \square \)

Corollary 2. Let \( L_1 = \{ \langle M, w \rangle \mid M \text{ is a Turing machine and } w \notin \mathcal{L}(M) \} \).
Then \( L_1 \) is not semi-decidable.

**Proof:** \( L_1 \) is close to being the complement of \( A_{TM} \), except that \( L_1 \) only contains strings that are of the form \( \langle M, w \rangle \). One way of seeing that \( L_1 \) is not semi-decidable is as follows. Letting \( L_0 = \{ x \mid x \text{ is not of the form } \langle M, w \rangle \} \), we see that \( \overline{A_{TM}} = L_1 \cup L_0 \). Clearly \( L_0 \) is decidable, and hence semi-decidable. If \( L_1 \) were also semi-decidable, then by a previous Lemma we would have \( \overline{A_{TM}} \) semi-decidable. Hence, \( L_1 \) is not semi-decidable. \( \square \)

An important natural language is the “blank tape halting problem”. Define \( HB = \{ \langle M \rangle \mid M \text{ is a Turing machine and } M \text{ halts on the blank input tape} \} \).

**Theorem 5.** \( HB \) is semi-decidable, but not decidable.

**Proof:** To show that \( HB \) is semi-decidable, we observe that \( HB = \mathcal{L}(M_0) \) where \( M_0 \) is as follows.
On input \( x \), \( M_0 \) checks if \( x \) is of the form \( \langle M \rangle \), and if not, rejects. So say \( x = \langle M \rangle \). \( M_0 \) then runs \( M \) on the blank tape; if and when \( M \) halts, \( M_0 \) halts and accepts.

We now want to show that \( HB \) is not decidable. Assume, to the contrary, that \( HB \) is decidable. Say that \( HB = \mathcal{L}(M_1) \), where \( M_1 \) is a Turing Machine that halts on every input.
We will construct a Turing Machine \( M_2 \) such that \( A_{TM} = \mathcal{L}(M_2) \) and \( M_2 \) halts on every input.

Define \( M_2 \) to work as follows on input \( x \).
If \( x \) is not of the form \( \langle M, w \rangle \), then \( M_2 \) halts and rejects. So assume that \( x = \langle M, w \rangle \). \( M_2 \) constructs a new machine \( M' \) that when started on a blank tape, writes \( w \) on the tape and runs \( M \) on \( w \); if \( M \) halts and accepts \( w \), then \( M' \) will halt and accept; if \( M \) halts and rejects \( w \), then \( M' \) will go into an infinite loop.
Clearly \( M \) accepts \( w \) if and only if \( M' \) halts on the blank tape.

\( M_2 \) then runs \( M_1 \) on \( \langle M' \rangle \).
Clearly \( M_2 \) halts on every input, and \( \mathcal{L}(M_2) = A_{TM} \). This means that \( A_{TM} \) is decidable, a contradiction. \( \square \)

**Reducibilities**

In the previous proof, we used the fact that \( A_{TM} \) is not decidable to show that \( HB \) is not decidable. This proof can be simplified by using the idea of reducibilities. We will introduce the notion of language \( L_1 \) being reducible, or transformable, to \( L_2 \), written \( L_1 \leq m L_2 \), and
show that $A_{T M} \leq_m HB$. The definition of $\leq_m$ will be the same as the definition of $\leq_p$, except that the transforming function $f$ is only required to be computable, instead of polynomial-time computable. (The “$m$” in $\leq_m$ exists for historical reasons, and stands for “many-one” reducible.)

**Definition 2.** Let $L_1, L_2 \subseteq \Sigma^*$. We say that $L_1 \leq_m L_2$ if there is a computable function $f : \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$,

$x \in L_1 \iff f(x) \in L_2$.

**Theorem 6.** Let $L_1, L_2 \subseteq \Sigma^*$ such that $L_1 \leq_m L_2$. Then

1) $\overline{L_1} \leq_m \overline{L_2}$

2) If $L_2 \in D$ then $L_1 \in D$.

(And hence, if $L_1 \notin D$ then $L_2 \notin D$).

3) If $L_2 \in SD$ then $L_1 \in SD$.

(And hence, if $L_1 \notin SD$ then $L_2 \notin SD$.)

**Proof:**

1) Say that $L_1 \leq_m L_2$ via the computable function $f$. Then we also have $\overline{L_1} \leq_m \overline{L_2}$ via $f$, since $x \in L_1 \iff f(x) \in L_2$ implies that $x \in \overline{L_1} \iff f(x) \in \overline{L_2}$.

2) Say that $L_2 = \mathcal{L}(M_2)$ where $M_2$ is a Turing machine that halts on every input. Let $M$ be a Turing machine that computes $f$. We now define Turing machine $M_1$ as follows.

On input $x$, $M_1$ runs $M$ on $x$ to get $f(x)$, and then runs $M_2$ on $f(x)$, accepting or rejecting as $M_2$ does. Clearly $M_1$ halts on every input, and $L_1 = \mathcal{L}(M_1)$, so $L_1$ is decidable.

3) Say that $L_2 = \mathcal{L}(M_2)$ where $M_2$ is a Turing machine. Let $M$ be a Turing machine that computes $f$. We now define Turing machine $M_1$ as follows.

On input $x$, $M_1$ runs $M$ on $x$ to get $f(x)$, and then runs $M_2$ on $f(x)$, accepting or rejecting as $M_2$ does if and when $M_2$ halts. Clearly $L_1 = \mathcal{L}(M_1)$, so $L_1$ is semi-decidable.

(Proof continued on next page.)
Corollary 3. \( \overline{HB} \) is not semi-decidable.

Proof: We know HB is semi-decidable but not decidable, so \( \overline{HB} \) is not semi-decidable. \( \square \)

Let NE be the language consisting of Turing Machines that accept a nonempty language. That is, \( NE = \{ \langle M \rangle \mid M \text{ is a Turing Machine and } L(M) \neq \emptyset \} \).

Lemma 7. NE is semi-decidable, but not decidable. (Hence, \( \overline{NE} \) is not semi-decidable.)

Proof: We will design a Turing Machine \( M_0 \) such that \( NE = L(M_0) \).
Say that the input to \( M_0 \) is \( x = \langle M \rangle \); \( M_0 \) behaves as follows:
run \( M \) on all inputs of length \( \leq 1 \) for 1 step;
run \( M \) on all inputs of length \( \leq 2 \) for 2 steps;
run \( M \) on all inputs of length \( \leq 3 \) for 3 steps; etc.
if and when it is discovered that \( M \) accepts some input, \( M_0 \) halts and accepts.
Clearly \( NE = L(M_0) \), so NE is semi-decidable.
To show that NE is not decidable we will prove that HB \( \leq_m NE \).
Let \( x \) be an input for HB, and assume that \( x \) is well-formed, that is, \( x = \langle M \rangle \).
Define \( f(x) = \langle M' \rangle \) where \( M' \) works as follows.
On any input, \( M' \) erases its input and runs \( M \) on the blank tape;
if and when \( M \) halts, \( M' \) halts and accepts.
Clearly \( \langle M \rangle \in HB \iff \langle M' \rangle \in NE \), so HB \( \leq_m NE \), so NE is not decidable. \( \square \)

Let \( T \) be the language of “total” machines, that is, of machines that halt on every input. \( T = \{ \langle M \rangle \mid M \text{ is a Turing Machine that halts on every input} \} \).

Lemma 8. Neither \( T \) nor \( \overline{T} \) is semi-decidable.

Proof:
We first show that HB \( \leq_m T \), implying that \( \overline{HB} \leq_m \overline{T} \), implying that \( \overline{T} \) is not semi-decidable.
So let \( x \) be an input for HB, and assume that \( x \) is well-formed, \( x = \langle M \rangle \).
Let \( f(x) = \langle M' \rangle \) where \( M' \) is a machine that always erases its input and runs \( M \) on the blank tape. We have \( M \) halts on the blank tape \( \iff \) \( M' \) halts on every input, so we are done.

We next show that HB \( \leq_m T \), implying that \( \overline{HB} \leq_m T \), implying that \( T \) is not semi-decidable. This reduction is a bit tricky.
Let \( x \) be an input for HB, and assume that \( x \) is well-formed, \( x = \langle M \rangle \).
Let \( f(x) = \langle M' \rangle \) where \( M' \) is as follows.
On input \( \alpha \), \( M' \) simulates \( M \) on the blank tape for \( |\alpha| \) steps;
if \( M \) halts within \( |\alpha| \) steps, then \( M' \) goes into an infinite loop;
if \( M \) doesn’t halt within \( |\alpha| \) steps, then \( M' \) halts (and, say, accepts).
Clearly \( M \) halts on the blank tape \( \iff \) \( M' \) is not a total machine. \( \square \)

Define the language \( INF = \{ \langle M \rangle \mid M \text{ is a Turing Machine and } L(M) \text{ is infinite} \} \).
Lemma 9. Neither $\text{INF}$ nor $\overline{\text{INF}}$ is semi-decidable.

Proof: Exercise.

Define the language $\text{ALL} = \{ \langle M \rangle \mid M$ is a Turing Machine and $L(M) = \Sigma^* \}.$

Lemma 10. Neither $\text{ALL}$ nor $\overline{\text{ALL}}$ is semi-decidable.

Proof: We will show $\mathbb{T} \leq_m \text{ALL}.$ This implies $\overline{\mathbb{T}} \leq_m \overline{\text{ALL}}.$ Since both $\mathbb{T}$ and $\overline{\mathbb{T}}$ are not semi-decidable, we have that both $\text{ALL}$ and $\overline{\text{ALL}}$ are not semi-decidable.

So let $x$ be an input for $\mathbb{T},$ and assume $x$ is well-formed, that is, $x = \langle M \rangle.$ Let $f(x) = \langle M' \rangle$ where $M'$ is obtained from $M$ by changing every transfer to $q_{\text{reject}}$ by a transfer to $q_{\text{accept}}.$ Since the only two halting states are $q_{\text{reject}}$ and $q_{\text{accept}},$ it follows that $M$ halts on every input $\iff M'$ accepts every input. $\square$

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

Lemma 11. Neither $\text{EQ}$ nor $\overline{\text{EQ}}$ is semi-decidable.

Proof: As above, it is sufficient to show that $\text{ALL} \leq_m \text{EQ}.$

Let $x$ be an input for $\text{ALL},$ and assume that $x$ is well-formed, that is, $x = \langle M \rangle.$ Let $f(x) = \langle M, M_0 \rangle$ where $M_0$ is some fixed, simple Turing Machine that accepts $\Sigma^*.$ Clearly $x \in \text{ALL} \iff f(x) \in \text{EQ}.$ $\square$

Some Other Undecidable Languages

All the undecidable languages we have discussed so far involve Turing Machines in some way. However, there are many other interesting types of undecidable languages.

It is easy to see that various questions about other types of computing models are also undecidable. For example, let $\text{AOB} =$ the set of $\langle U \rangle$ such that $U$ is a C program that has no input statements, and that causes an Array Out of Bounds reference when executed.

Lemma 12. $\text{AOB}$ is semi-decidable, but not decidable.

Proof: To see that $\text{AOB}$ is semi-decidable, we construct a Turing Machine $M$ such that $L(M) = L.$ $M$ works on input $x$ as follows.

$M$ checks that $x = \langle U \rangle$ where $U$ is a C program with no input statements (and if not, rejects).

Then $M$ simulates $U;$ if $U$ halts without causing an Array Out of Bounds reference, then $M$ halts and rejects; if at some point $U$ causes an Array Out of Bounds reference, then $M$ halts and accepts.
To see that AOB is not decidable, we will show that HB \leq_m AOB.
So let \( x \) be an input for HB and assume that \( x = \langle M \rangle \) where \( M \) is a Turing Machine.
Let \( f(x) = \langle U \rangle \) where \( U \) is a C program that works as follows.
\( U \) simulates \( M \) running on the blank tape, without making any Array Out of Bounds reference; if and when \( M \) is discovered to halt, \( U \) causes an Array Out of Bounds reference (and halts). Clearly \( M \) halts on the blank tape \iff \( U \) causes an Array Out of Bounds reference. 

Another source of undecidable languages is Mathematical Logic. We discussed above the language PR (Presburger Arithmetic) of predicate logic sentences that are true about \( \mathbb{N} \) when we have the function symbol “+” (interpreted as addition). This language is decidable, but requires (at least) double-exponential time.

Say that we now allow the function symbol “\( \times \)” as well, and let TA (for True Arithmetic) be the set of sentences true about \( \mathbb{N} \) when “+” is interpreted as addition and “\( \times \)” as multiplication.

Let us also define VALID as the set of sentences of the predicate calculus that are valid, that is, true in every structure. The following are important facts, but we will not prove them here.

**Theorem 7.** VALID is semi-decidable, but not decidable.
TA is not semi-decidable, and neither is its complement.

The proofs that these languages are not decidable involve showing how to “simulate” Turing Machines in them. The proof that VALID is semi-decidable involves showing the existence of a sound and complete proof system (such as a Frege System, or a Natural Deduction System) for the predicate calculus; we then have VALID = \( L(M) \) where \( M \), given a formula \( F \), searches through all possible “proofs”, accepting if and when a proof for \( F \) is found.

Lastly, we consider the language DE (for Diophantine Equations) consisting of strings of the form \( \langle Q \rangle \) where \( Q \) is a multivariate polynomial with integer coefficients, such that there are integer values for the variables of \( Q \) which make \( Q \) evaluate to 0. For example \( \langle Q \rangle \in \text{DE} \) if \( Q \) is \( x^3 + 2xy + x^2 - y - 9 \), since \( Q \) evaluates to 0 for \( [x = 2, y = -1] \).

It is easy to see that DE is semi-decidable (exercise). However, the following Theorem is very difficult. In fact, it easily implies that TA is not decidable (exercise).

**Theorem 8.** DE is not decidable.

**A Decidable Language not in NP**

We wish to prove that there is a language \( L \) such that \( L \in \mathbf{D} \), but \( L \notin \mathbf{NP} \) (and hence \( L \notin \mathbf{P} \)). We claimed earlier that the language PR has this property, but this is too hard to prove here. Instead, we will prove this for a much less natural language \( L \). We will obtain \( L \) using diagonalization, in much the same way that we first described a language that was
semi-decidable but not decidable. In fact we will define a decidable language \( L \) such that every Turing machine that accepts \( L \) runs in (at least) double exponential time.

**Theorem 9.** There is a decidable language \( L \) such that \( L \not\in \text{NP} \).

**Proof:**
Define \( L = \{ \langle M \rangle \mid M \text{ is a Turing machine, and } M \text{ does not accept } \langle M \rangle \text{ within } 2^{2^{2^{|M|}}} \text{ steps} \} \).

We will now show that \( L \not\in \text{NP} \). Assume otherwise. This implies that there is a Turing machine \( M_1 \) and constant \( d \) such that \( L = L(M_1) \) and such that \( M_1 \) runs in time \( O(2^{n^d}) \).

This implies that there is a number \( n_0 \) such that for all \( x \in \Sigma^* \) of length \( \geq n_0 \), \( M_1 \) on input \( x \) halts within \( 2^{2^{|x|}} \) steps. We would like it to be the case that \( |\langle M_1 \rangle| \geq n_0 \), but this may not be true. So let \( M_2 \) be such that \( M_2 \) is the same as \( M_1 \), except that some additional, useless states have been added so that \( |\langle M_2 \rangle| \geq n_0 \).

We now have that \( L = L(M_2) \), and \( M_2 \) running on \( \langle M_2 \rangle \) halts within \( 2^{2^{2^{|M_2|}}} \) steps. So \( \langle M_2 \rangle \in L \iff \langle M_2 \rangle \in L(M_2) \iff M_2 \) accepts \( \langle M_2 \rangle \) within \( 2^{2^{|M_2|}} \) steps \( \iff \langle M_2 \rangle \not\in L \).

This is a contradiction. \( \square \)

In fact, the above proof technique can be used to show that for every computable time bound \( t(n) \), there is a decidable language \( L \) such that no Turing machine that accepts \( L \) runs in time \( O(t(n)) \).

Actually, in order to make sense of this, we have to say what it means for a function \( t : \mathbb{N} \to \mathbb{N} \) to be computable, since so far we have only discussed what it means for a function mapping strings to strings to be computable. However, as computer scientists we are comfortable with the idea that strings can represent numbers and that numbers can represent strings. For every integer \( n \), we denote by \( \hat{n} \) the string of bits such that \( \hat{n} \) is the unique binary representation of \( n \) that has no leading 0's; note that this implies that \( \hat{0} \) is the empty string.

We now make the following definition.

**Definition 3.** Let \( t : \mathbb{N} \to \mathbb{N} \). We say \( t \) is computable if there is a computable function \( f : \Sigma^* \to \Sigma^* \) such that for every \( n \in \mathbb{N} \), \( f(\hat{n}) = \hat{t(n)} \).

It is important to understand that this notion of computability of functions on integers does not depend on exactly what relationship we choose between strings and numbers. If we had chosen any other reasonable relationship, we would have defined exactly the same notion.

We can now state our theorem.

**Theorem 10.** Let \( t : \mathbb{N} \to \mathbb{N} \) be a computable function. Then there is a decidable language \( L \) such that for every Turing machine \( M \) that accepts \( L \), \( M \) does not run in time \( O(t(n)) \).

**Proof:**
Let \( L = \{ \langle M \rangle \mid M \text{ is a Turing machine, and } M \text{ does not accept } \langle M \rangle \text{ within } n \cdot t(n) \text{ steps} \}, \)
where \( n = |\langle M \rangle| \).
We leave it as an exercise to prove that \( L \) has the two desired properties. \( \Box \)