I said in my haste, All men are liars.

— Psalms 116:11 (King James Version)

Some problems are so complex that you have to be highly intelligent and well informed just to be undecided about them.

— Laurence Johnston Peter, *Peter's Almanac* (September 24, 1982)

"Proving or disproving a formula—once you've encrypted the formula into numbers, that is—is just a calculation on that number. So it means that the answer to the question is, no! Some formulas cannot be proved or disproved by any mechanical process! So I guess there's some point in being human after all!"

Alan looked pleased until Lawrence said this last thing, and then his face collapsed. "Now there you go making unwarranted assumptions."

— Neal Stephenson, *Cryptonomicon* (1999)

No matter how P might perform, Q will scoop it:
Q uses P's output to make P look stupid.
Whatever P says, it cannot predict Q:
P is right when it's wrong, and is false when it's true!

— Geoffrey S. Pullum, "Scooping the Loop Sniffer" (2000)

★★★ Rewrite in the language of algorithms instead of the language of Turing machines, using "source code" instead of "encoding" everywhere. Formulation in terms of TMs makes almost everything much more complicated than it needs to be. (The dovetail/product construction in the proof of Lemma 4 may be an exception.)

7 Undecidability

Perhaps the single most important result in Turing's remarkable 1936 paper that introduces Turing machines is his observation that there are problems that cannot be solved by *any* algorithm. Turing's canonical example of an undecidable problem was the ***halting problem***, which asks whether a given Turing machine halts when given a particular input string. Among other consequences, Turing's undecidability result provided an elegant negative solution to Hilbert's *Entscheidungsproblem*, which asked for an algorithm to decide whether a given statement of first-order logic is true—no such algorithm exists.

7.1 Acceptable versus Decidable

Recall that there are three possible outcomes for a Turing machine M running on any particular input string w: acceptance, rejection, and divergence. Every Turing machine M immediately defines four different languages (over the input alphabet Σ of M):

- The *accepting* language $\textsc{Accept}(M) := \{w \in \Sigma^* \mid M \text{ accepts } w\}$
- The *rejecting* language $\textsc{Reject}(M) := \{w \in \Sigma^* \mid M \text{ rejects } w\}$
- The *halting* language $\textsc{Halt}(M) := \textsc{Accept}(M) \cup \textsc{Reject}(M)$
- The *diverging* language $\textsc{Diverge}(M) := \Sigma^* \setminus \textsc{Halt}(M)$

For any language L, the sentence "***M accepts L***" means $\textsc{Accept}(M) = L$, and the sentence "***M decides L***" means $\textsc{Accept}(M) = L$ and $\textsc{Diverge}(M) = \varnothing$.

Now let L be an arbitrary language. We say that L is ***acceptable*** (or *semi-computable*, or *semi-decidable*, or *recognizable*, or *listable*, or *recursively enumerable*) if some Turing machine accepts L, and ***unacceptable*** otherwise. Similarly, L is ***decidable*** (or *computable*, or *recursive*) if some Turing machine decides L, and ***undecidable*** otherwise.

7.2 Lo, I Have Become Death, Stealer of Pie

There is a subtlety in the definitions of "acceptable" and "decidable" that many beginners miss: A language can be decidable even if we can't exhibit a specific Turing machine decides it. As a canonical example, consider the language $\Pi = \{w \mid 1^{|w|} \text{ appears in the binary expansion of } \pi\}$. Despite appearances, this language *is* decidable! There are only two cases to consider:

- Suppose there is an integer N such that the binary expansion of π contains the substring 1^N but does not contain the substring 1^{N+1}. Let M_N be the Turing machine with $N+3$ states $\{0, 1, \ldots, N, \text{accept}, \text{reject}\}$, start state 0, and the following transition function:

$$\delta(q,a) = \begin{cases} \text{accept} & \text{if } a = \square \\ \text{reject} & \text{if } a \neq \square \text{ and } q = n \\ (q+1, a, +1) & \text{otherwise} \end{cases}$$

  This machine correctly decides Π.

- Suppose the binary expansion of π contains arbitrarily long substrings of 1s. Then any Turing machine that accepts all inputs correctly decides Π.

We have no idea which of these machines correctly decides Π, but one of them does, and that's enough!

7.3 Useful Properties

This subsection lists several simple but useful properties of (un)decidable and (un)acceptable languages. Almost all of these properties follow from routine definition-chasing; readers are strongly encouraged to try to prove each lemma themselves before reading ahead.

One might reasonably ask why we don't also define "rejectable" and "haltable" languages. The following lemma, whose proof is an easy exercise (hint, hint), implies that these sets are both identical to the acceptable languages.

Lemma 1. *Let M be an arbitrary Turing machine.*
(a) There is a Turing machine M^R such that $\text{Accept}(M^R) = \text{Reject}(M)$ and $\text{Reject}(M^R) = \text{Accept}(M)$.
(b) There is a Turing machine M^A such that $\text{Accept}(M^A) = \text{Accept}(M)$ and $\text{Reject}(M^A) = \varnothing$.
(c) There is a Turing machine M^H such that $\text{Accept}(M^H) = \text{Halt}(M)$ and $\text{Reject}(M^H) = \varnothing$.

The decidable languages have several fairly obvious useful closure properties.

Lemma 2. *If L and L' are decidable, then $L \cup L'$, $L \cap L'$, $L \setminus L'$, and $L' \setminus L$ are also decidable.*

Proof: Let M and M' be Turing machines that decide L and L', respectively. We can build a Turing machine $M_\cup$ that decides $L \cup L'$ as follows. First, $M_\cup$ copies its input string w onto a second tape. Then $M_\cup$ runs M on input w (on the first tape), and then runs M' on input w (on the second tape). If either M or M' accepts, then $M_\cup$ accepts; if both M and M' reject, then $M_\cup$ rejects.

The other three languages are similar. □

Corollary 3. *The following hold for all languages L and L'.*
(a) If $L \cap L'$ is undecidable and L' is decidable, then L is undecidable.
(b) If $L \cup L'$ is undecidable and L' is decidable, then L is undecidable.
(c) If $L \setminus L'$ is undecidable and L' is decidable, then L is undecidable.
(d) If $L' \setminus L$ is undecidable and L' is decidable, then L is undecidable.

Unfortunately, acceptable languages are not quite as well-behaved as decidable languages, thanks to the subtle distinction between *rejecting* a string and *not accepting* a string.

Lemma 4. *For all acceptable languages L and L', the languages $L \cup L'$ and $L \cap L'$ are also acceptable.*

Proof: Let M and M' be Turing machines that decide L and L', respectively. We can build a Turing machine $M_\cap$ that decides $L \cap L'$ as follows. First, $M_\cap$ copies its input string w onto a second tape. Then $M_\cap$ runs M on input w using the first tape, and then runs M' on input w using the second tape. If both M and M' accept, then $M_\cap$ accepts; if either M or M' reject, then $M_\cap$ rejects; if either M or M' diverge, then $M_\cap$ diverges (automatically).

The construction for $L \cup L'$ is more subtle; instead of running M and M' in series, we must run them in parallel. Like $M_\cap$, the new machine $M_\cup$ starts by copying its input string w onto a second tape. But then $M_\cup$ runs M and M' simultaneously, with each step of $M_\cup$ simulating both one step of M on the first tape and one step of M' on the second. Ignoring the states and transitions needed for initialization, the state set of $M_\cup$ is the product of the state sets of M and M', and the transition function is

$$\delta_\cup(q,a,q',a') = \begin{cases} \mathsf{accept}_\cup & \text{if } q = \mathsf{accept} \text{ or } q' = \mathsf{accept}' \\ \mathsf{reject}_\cup & \text{if } q = \mathsf{reject} \text{ and } q' = \mathsf{reject}' \\ (\delta(q,a), \delta'(q',a')) & \text{otherwise} \end{cases}$$

Thus, $M_\cup$ accepts as soon as either M or M' accepts, and rejects only after both M or M' reject. □

Lemma 5. *An acceptable language L is decidable if and only if $\Sigma^* \setminus L$ is also acceptable.*

Proof: Let M and $\overline{M}$ be Turing machines that accept L and $\Sigma^* \setminus L$, respectively. Following the previous proof, we construct a new Turing machine M^* that copies its input onto a second tape, and then simulates M and M' in parallel on the two tapes. If M accepts, then M^* accepts; if $\overline{M}$ accepts, then M^* rejects. Since every string is accepted by either M or $\overline{M}$, we conclude that M^* decides L.

The other direction follows immediately from Lemma 1. □

7.4 Code is Data; Data is Code

Perhaps the single most important observation in developing these undecidability results—and one of the most important observations in computer science more broadly—is that Turing machines can be encoded as strings. At one level, this observation is completely trivial: Any written description of a Turing machine is a string, and modern code *is* just a sequence of bytes, stored in a file like any other data. But this apparently trivial observation is actually incredibly powerful.

Most natural encodings of Turing machines have three important properties.

- **Unique:** Different Turing machines are encoded as different strings.
- **Modifiable:** We can algorithmically modify any Turing machine M, given the encoding of M as input. For example, there are algorithms to swap the accept and reject states of any Turing machine, or to add new states and transitions representing pre- and post-processing phases, or to build a new machine that calls M as a subroutine, or to build a new machine that runs several copies of M in parallel.
- **Executable:** There is a fixed ***universal*** Turing machine U that can simulate the behavior of an arbitrary Turing machine M, given the encodings of M and w as input. For example, if we decided to encode Turing machines as Python programs, then U would be a Python interpreter.

The precise details of the encoding are unimportant, but for the sake of concreteness, let me describe a natural encoding of Turing machines as strings over the six-character alphabet $\{\mathtt{0}, \mathtt{1}, \mathtt{\{}, \bullet, \mathtt{\}}\}$. Let $M = (\Gamma, \square, \Sigma, Q, \mathit{start}, \mathit{accept}, \mathit{reject}, \delta)$ be an arbitrary Turing machine, with a single half-infinite tape and a single read-write head. (I will consistently indicate the states and tape symbols of M in *slanted green* to distinguish them from the upright red symbols in the encoding alphabet.)

- We encode each symbol $a \in \Gamma$ as a unique string $\langle a \rangle$ of $\lceil \lg(|\Gamma|) \rceil$ bits. For example, if $\Gamma = \{\mathit{0}, \mathit{1}, \$, x, \square\}$, we might use the following encoding:

$$\langle \mathit{0} \rangle = \mathtt{001}, \qquad \langle \mathit{1} \rangle = \mathtt{010}, \qquad \langle \$ \rangle = \mathtt{011}, \qquad \langle x \rangle = \mathtt{100}, \qquad \langle \square \rangle = \mathtt{000}.$$

- Similarly, we encode each state $q \in Q$ as a distinct string $\langle q \rangle$ of $\lceil \lg |Q| \rceil$ bits. Without loss of generality, we encode the start state with all $\mathtt{1}$s and the reject state with all $\mathtt{0}$s. For example, if $Q = \{\mathit{start}, \mathit{seek1}, \mathit{seek0}, \mathit{reset}, \mathit{verify}, \mathit{accept}, \mathit{reject}\}$, we might use the following encoding:

$$\begin{array}{llll} \langle \mathit{start} \rangle = \mathtt{111} & \langle \mathit{seek1} \rangle = \mathtt{010} & \langle \mathit{seek0} \rangle = \mathtt{011} & \langle \mathit{reset} \rangle = \mathtt{100} \\ \langle \mathit{verify} \rangle = \mathtt{101} & \langle \mathit{accept} \rangle = \mathtt{110} & \langle \mathit{reject} \rangle = \mathtt{000} & \end{array}$$

- Finally, we encode the machine M itself as the string $\langle M \rangle = \mathtt{\{}\langle \mathit{reject} \rangle \bullet \langle \square \rangle\mathtt{\}}\langle \delta \rangle$, where $\langle \delta \rangle$ is the concatenation of substrings $\mathtt{\{}\langle p \rangle \bullet \langle a \rangle \bullet \langle q \rangle \bullet \langle b \rangle \bullet \langle \Delta \rangle\mathtt{\}}$ encoding each transition $\delta(p, a) = (q, b, \Delta)$ such that $q \neq$ reject. We encode the actions $\Delta = \pm 1$ by defining $\langle -1 \rangle := \mathtt{0}$ and $\langle +1 \rangle := \mathtt{1}$. Conveniently, every transition string has exactly the same length. For example, with the symbol and state encodings described above, the transition $\delta(\mathit{reset}, \$) = (\mathit{start}, \$, +1)$ would be encoded as the string

$$\mathtt{\{100 \bullet 011 \bullet 001 \bullet 011 \bullet 1\}}.$$

Our first example Turing machine for recognizing $\{\mathit{0}^n \mathit{1}^n \mathit{0}^n \mid n \geq 0\}$ would be represented by the following string (broken into multiple lines for readability):

```
{000•000}{{001•001•010•011•1}{001•100•101•011•1}{010•001•010•001•1}
          {010•100•010•100•1}{010•010•011•100•1}{011•010•011•010•1}
          {011•100•011•100•1}{011•001•100•100•1}{100•001•100•001•0}
          {100•010•100•010•0}{100•100•100•100•0}{100•011•001•011•1}
          {101•100•101•011•1}{101•000•110•000•0}}
```

Building a universal Turing machine U that uses this encoding is more a matter of careful bookkeeping than real insight. We can encode any *configuration* of M on U's work tape by encoding each cell of M's tape as a string $\{\langle q\rangle \bullet \langle a\rangle\}$ indicating that (1) the cell contains symbol a; (2) if $q \neq \mathit{reject}$, then M's head is located at this cell, and M is in state q; and (3) if $q = \mathit{reject}$, then M's head is located somewhere else. We also surround the entire tape encoding with brackets $\{$ and $\}$. For example, the initial configuration $(\mathit{start}, \underset{\blacktriangle}{0}0110, 0)$ for our example Turing machine would be encoded as follows.

$$\langle \mathit{start}, \underset{\blacktriangle}{0}0110, 0\rangle = \{\underbrace{\{111\bullet001\}}_{\textbf{start}~0}\underbrace{\{000\bullet001\}}_{\mathit{reject}~0}\underbrace{\{000\bullet010\}}_{\mathit{reject}~1}\underbrace{\{000\bullet010\}}_{\mathit{reject}~1}\underbrace{\{000\bullet001\}}_{\mathit{reject}~0}\}$$

Similarly, the intermediate configuration $(\mathit{reset}, \$0x\underset{\blacktriangle}{1}x, 3)$ would be encoded as follows:

$$\langle \mathit{reset}, \$\$x\underset{\blacktriangle}{1}x, 3\rangle = \{\underbrace{\{000\bullet011\}}_{\mathit{reject}~\$}\underbrace{\{000\bullet011\}}_{\mathit{reject}~0}\underbrace{\{000\bullet100\}}_{\mathit{reject}~x}\underbrace{\{010\bullet010\}}_{\textbf{reset}~1}\underbrace{\{000\bullet100\}}_{\mathit{reject}~x}\}$$

To simulate one step of M's execution, we (1) find the location of the head (or reject if the head has vanished), (2) look up the transition for the state-symbol pair at the head, and (3) update the current cell and one of its neighbors to reflect the transition. The remaining grungy details are left as an exercise.

7.5 Self-Haters Gonna Self-Hate

A Turing machine encoding $\langle M\rangle$ is just a string, and any string (over the correct alphabet) can be used as the input to a Turing machine. Thus, a suitable encoding of *any* Turing machine can be used as the input to *any* Turing machine. In particular:

The encoding $\langle M\rangle$ of Turing machine M
can be used as input to *the same* Turing machine M.

Turing used this observation about self-reference to derive his first undecidable language as follows. Let's say that a Turing machine M is ***self-rejecting*** if it rejects its own encoding $\langle M\rangle$. Let SelfReject be the set of all encodings of self-rejecting Turing machines:

$$\textsc{SelfReject} := \big\{\langle M\rangle \bigm| M \text{ rejects } \langle M\rangle\big\}$$

Theorem 6. *SelfReject is undecidable.*

Proof: Suppose to the contrary that there is a Turing machine SR that decides SelfReject. Then by definition, $\textsc{Accept}(SR) = \textsc{SelfReject}$ and $\textsc{Diverge}(SR) = \varnothing$. More explicitly, for ***any*** Turing machine M,

- SR accepts $\langle M\rangle \iff M$ rejects $\langle M\rangle$, and
- SR rejects $\langle M\rangle \iff M$ does not reject $\langle M\rangle$.

In particular, these equivalences must hold when M is the machine SR. Thus,

- SR accepts $\langle SR\rangle \iff SR$ rejects $\langle SR\rangle$, and
- SR rejects $\langle SR\rangle \iff SR$ does not reject $\langle SR\rangle$.

In short, SR accepts $\langle SR\rangle$ if and only if SR rejects $\langle SR\rangle$, which is impossible! The only logical conclusion is that the Turing machine SR does not exist. □

7.6 Aside: Uncountable Barbers

Turing's proof by contradiction is an avatar of the famous ***diagonalization argument*** that uncountable sets exist, published by Georg Cantor in 1891. Indeed, SELFREJECT is sometimes called "the diagonal language". Recall that a function $f : A \to B$ is a ***surjection***[1] if $f(A) = \{f(a) \mid a \in A\} = B$.

Cantor's Theorem. *Let $f : X \to 2^X$ be an **arbitrary** function from an **arbitrary** set X to its power set. This function f is not a surjection.*

Proof: Fix an arbitrary function $f : X \to 2^X$. Call an element $x \in X$ ***happy*** if $x \in f(x)$ and ***sad*** if $x \notin f(x)$. Let Y be the set of all sad elements of X; that is, for *every* element $x \in X$, we have

$$x \in Y \iff x \notin f(x).$$

For the sake of argument, suppose f is a surjection. Then (by definition of surjection) there must be an element $y \in X$ such that $f(y) = Y$. Then for *every* element $x \in X$, we have

$$x \in f(y) \iff x \notin f(x).$$

In particular, the previous equivalence must hold when $x = y$:

$$y \in f(y) \iff y \notin f(y).$$

We have a contradiction! We conclude that f is not a surjection after all. □

Now let $X = \Sigma^*$, and define the function $f : X \to 2^X$ as follows:

$$f(w) := \begin{cases} \text{ACCEPT}(M) & \text{if } w = \langle M \rangle \text{ for some Turing machine } M \\ \varnothing & \text{if } w \text{ is not the encoding of a Turing machine} \end{cases}$$

Cantor's theorem immediately implies that not all languages are acceptable.

Alternatively, let X be the set of all Turing machines that halt on all inputs. For any Turing machine $M \in X$, let $f(M)$ be the set of all Turing machines $N \in X$ such that M accepts the encoding $\langle N \rangle$. Then a Turing machine M is *sad* if it rejects its own encoding $\langle M \rangle$; thus, Y is essentially the set SELFREJECT. Cantor's argument now immediately implies that no Turing machine decides the language SELFREJECT.

The core of Cantor's diagonalization argument also appears in the "barber paradox" popularized by Bertrand Russell in the 1910s. In a certain small town, every resident has a haircut on Haircut Day. Some residents cut their own hair; others have their hair cut by another resident of the same town. To obtain an official barber's license, a resident must cut the hair of all residents who don't cut their own hair, and no one else. Given these assumptions, we can immediately conclude that there are no licensed barbers. After all, who would cut the barber's hair?

To map Russell's barber paradox back to Cantor's theorem, let X be the set of residents, and let $f(x)$ be the set of residents who have their hair cut by x; then a resident is *sad* if they do not cut their own hair. To prove that SELFREJECT is undecidable, replace "resident" with "a Turing machine that halts on all inputs", and replace "A cuts B's hair" with "A accepts $\langle B \rangle$".

[1] more commonly, flouting all reasonable standards of grammatical English, "an onto function"

7.7 Just Don't Know What to Do with Myself

Similar diagonal arguments imply that three other languages are also undecidable:

$$\begin{aligned}\textsc{SelfAccept} &:= \big\{\langle M\rangle \;\big|\; M \text{ accepts } \langle M\rangle\big\}\\ \textsc{SelfHalt} &:= \big\{\langle M\rangle \;\big|\; M \text{ halts on } \langle M\rangle\big\}\\ \textsc{SelfDiverge} &:= \big\{\langle M\rangle \;\big|\; M \text{ diverges on } \langle M\rangle\big\}\end{aligned}$$

The proofs for these three languages are not quite as direct as the proof for SelfReject; each fictional deciding machine requires a small modification to create the contradiction.

Theorem 7. *SelfAccept is undecidable.*

Proof: For the sake of argument, suppose there is a Turing machine SA such that $\textsc{Accept}(SA) = \textsc{SelfAccept}$ and $\textsc{Diverge}(M) = \varnothing$. Let SA^R be the Turing machine obtained from SA by swapping its accept and reject states (as in the proof of Lemma 1). Then $\textsc{Reject}(SA^R) = \textsc{SelfAccept}$ and $\textsc{Diverge}(SA^R) = \varnothing$. It follows that SA^R rejects $\langle SA^R\rangle$ if and only if SA^R accepts $\langle SA^R\rangle$, which is impossible. □

Theorem 8. *SelfHalt is undecidable.*

Proof: Suppose to the contrary that there is a Turing machine SH such that $\textsc{Accept}(SH) = \textsc{SelfHalt}$ and $\textsc{Diverge}(SH) = \varnothing$. Let SH^X be the Turing machine obtained from SH by redirecting every transition to accept to a new hanging state hang, and then redirecting every transition to reject to accept. Then $\textsc{Accept}(SH^X) = \Sigma^* \setminus \textsc{SelfHalt}$ and $\textsc{Reject}(SH^X) = \varnothing$. It follows that SH^X accepts $\langle SH^X\rangle$ if and only if SH^X does not halt on $\langle SH^X\rangle$, and we have a contradiction. □

Theorem 9. *SelfDiverge is unacceptable and therefore undecidable.*

Proof: Suppose to the contrary that there is a Turing machine SD such that $\textsc{Accept}(M) = \textsc{SelfDiverge}$. Let SD^A be the Turing machine obtained from M by redirecting every transition to reject to a new hanging state hang such that $\delta(\text{hang}, a) = (\text{hang}, a, +1)$ for every symbol a. Then $\textsc{Accept}(SD^A) = \textsc{SelfDiverge}$ and $\textsc{Reject}(SD^A) = \varnothing$. It follows that SD^A accepts $\langle SD^A\rangle$ if and only if SD^A does not halt on $\langle SD^A\rangle$, which is impossible. □

*7.8 Nevertheless, Acceptable

Our undecidability argument for SelfDiverge actually implies the stronger result that SelfDiverge is unacceptable; we never assumed that the hypothetical accepting machine SD halts on all inputs. However, we can use or modify our universal Turing machine U to *accept* the other three self-referential languages.

Theorem 10. *SelfAccept is acceptable.*

Proof: We describe a Turing machine SA that accepts the language SelfAccept. Given any string w as input, SA first verifies that w is the encoding of a Turing machine. If w is not the encoding of a Turing machine, then SA diverges. Otherwise, $w = \langle M\rangle$ for some Turing machine M; in this case, SA writes the string $ww = \langle M\rangle\langle M\rangle$ onto its tape and passes control to the universal Turing machine U. U then simulates M (the machine encoded by the first half of its input) on the string $\langle M\rangle$ (the second half of its input).[2] In particular, U accepts $\langle M, M\rangle$ if and only if M accepts $\langle M\rangle$. We conclude that SR accepts $\langle M\rangle$ if and only if M accepts $\langle M\rangle$. □

Theorem 11. *SelfReject is acceptable.*

Proof: Let U^R be the Turing machine obtained from our universal machine U by swapping the accept and reject states. We describe a Turing machine SR that accepts the language SelfReject as follows. SR first verifies that its input string w is the encoding of a Turing machine and diverges if not. Otherwise, SR writes the string $ww = \langle M, M\rangle$ onto its tape and passes control to the reversed universal Turing machine U^R. Then U^R accepts $\langle M, M\rangle$ if and only if M rejects $\langle M\rangle$. We conclude that SR accepts $\langle M\rangle$ if and only if M rejects $\langle M\rangle$. □

Finally, because SelfHalt is the union of two acceptable languages, SelfHalt is also acceptable.

7.9 The Halting Problem via Reduction

Now consider the following related languages:[3]

$$\begin{aligned}
\textsc{Accept} &:= \big\{\langle M, w\rangle \bigm| M \text{ accepts } w\big\}\\
\textsc{Reject} &:= \big\{\langle M, w\rangle \bigm| M \text{ rejects } w\big\}\\
\textsc{Halt} &:= \big\{\langle M, w\rangle \bigm| M \text{ halts on } w\big\}\\
\textsc{Diverge} &:= \big\{\langle M, w\rangle \bigm| M \text{ diverges on } w\big\}
\end{aligned}$$

Deciding the language Halt is usually called the ***halting problem***: Given a program M and an input w to that program, does the program halt? This problem may seem trivial; why not just run the program and see? More formally, why not just pass the input string $\langle M, x\rangle$ to our universal Turing machine U? That strategy works perfectly if we just want to *accept* Halt, but we actually want to *decide* Halt; if M is not going to halt on w, we still want an answer in a finite amount of time. Sadly, we can't always get what we want.

Theorem 12. *Halt is undecidable.*

Proof: Suppose to the contrary that there is a Turing machine H that decides Halt. Then we can use H to build another Turing machine SH that decides the language SelfHalt. Given any string w, the machine SH first verifies that $w = \langle M\rangle$ for some Turing machine M (rejecting if not), then writes the string $ww = \langle M, M\rangle$ onto the tape, and finally passes control to H. But SelfHalt is undecidable, so no such machine SH exists. We conclude that H does not exist either. □

Nearly identical arguments imply that the languages Accept, Reject, and Diverge are undecidable.

[2]To simplify the presentation, I am implicitly assuming here that $\langle M\rangle = \langle\langle M\rangle\rangle$. Without this assumption, we need a Turing machine that transforms an arbitrary string $w \in \Sigma_M^*$ into its encoding $\langle w\rangle \in \Sigma_U^*$; building such a Turing machine is straightforward.

[3]Many sources including Sipser and Wikipedia uses the shorter name A_{TM} instead of Accept, but uses $HALT_{TM}$ instead of Halt. I have no idea why Sipser thought four-letter names are okay, but six-letter names are not. The subscript TM is just a reminder that these are languages of *Turing machine* encodings, as opposed to encodings of DFAs or some other machine model.

Here we have our first example of an undecidability proof by ***reduction***. Specifically, we ***reduced*** the language SELFHALT to the language HALT. More generally, to reduce one language X to another language Y, we assume (for the sake of argument) that there is a program P_Y that decides Y, and we write another program that decides X, using P_Y as a black-box subroutine. If later we discover that Y is decidable, we can immediately conclude that X is decidable. Equivalently, if we later discover that X is undecidable, we can immediately conclude that Y is undecidable.

To prove that a language L is undecidable,
reduce a known undecidable language to L.

Perhaps the most confusing aspect of reduction arguments is that the *languages* we want to prove undecidable nearly (but not quite) always involve encodings of Turing machines, while at the same time, the *programs* that we build to prove them undecidable are also Turing machines. Our proof that HALT is undecidable involved three different machines:

- The hypothetical Turing machine H that decides HALT.
- The new Turing machine SH that decides SELFHALT, using H as a subroutine.
- The Turing machine M whose encoding is the input to H.

It is *incredibly* easy to get confused about which machines are playing each in the proof. Therefore, it is absolutely vital that we give each machine in a reduction proof a unique and mnemonic name, and then ***always*** refer to each machine ***by name***. Never write, say, or even *think* "the Turing machine" or "the state" or "the tape" or "the input" or (gods forbid) "it". You also may find it useful to think of the working ***programs*** we are trying to construct (H and SH in this proof) as being written in a different language than the arbitrary ***source code*** that we want those programs to analyze ($\langle M\rangle$ in this proof).

7.10 One Million Years Dungeon!

As a more complex set of examples, consider the following languages:

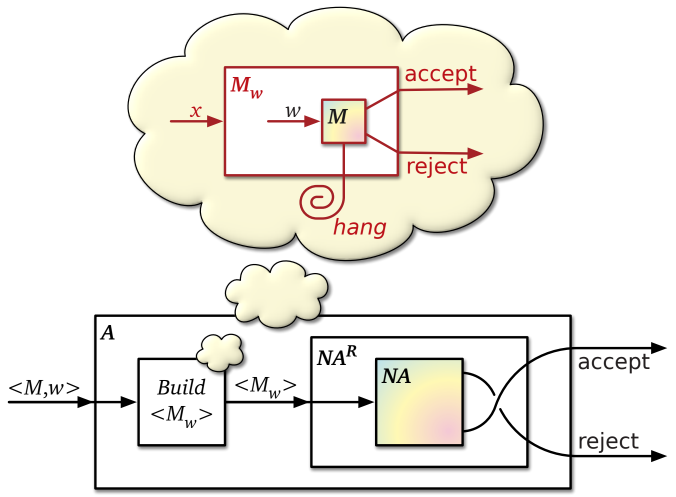
$$\begin{aligned}
\text{NEVERACCEPT} &:= \{\langle M\rangle \mid \text{ACCEPT}(M) = \varnothing\}\\
\text{NEVERREJECT} &:= \{\langle M\rangle \mid \text{REJECT}(M) = \varnothing\}\\
\text{NEVERHALT} &:= \{\langle M\rangle \mid \text{HALT}(M) = \varnothing\}\\
\text{NEVERDIVERGE} &:= \{\langle M\rangle \mid \text{DIVERGE}(M) = \varnothing\}
\end{aligned}$$

Theorem 13. *NEVERACCEPT is undecidable.*

Proof: Suppose to the contrary that there is a Turing machine NA that decides NEVERACCEPT. Then by swapping the accept and reject states, we obtain a Turing machine NA^R that decides the complementary language $\Sigma^* \setminus$ NEVERACCEPT.

To reach a contradiction, we construct a Turing machine A that decides ACCEPT as follows. Given the encoding $\langle M, w\rangle$ of an arbitrary machine M and an arbitrary string w as input, A writes the encoding $\langle M_w\rangle$ of a new Turing machine M_w that ignores its input, writes w onto the tape, and then passes control to M. Finally, A passes the new encoding $\langle M_w\rangle$ as input to NA^R. The following cartoon tries to illustrate the overall construction.

Before going any further, it may be helpful to list the various Turing machines that appear in this construction.



A reduction from from Accept to NeverAccept, which proves NeverAccept undecidable.

- The hypothetical Turing machine NA that decides NeverAccept.
- The Turing machine NA^R that decides $\Sigma^* \setminus$ NeverAccept, which we constructed by modifying NA.
- The Turing machine A that we are building, which decides Accept using NA^R as a black-box subroutine.
- The Turing machine M, whose encoding is part of the input to A.
- The Turing machine M_w whose encoding A constructs from $\langle M, w\rangle$ and then passes to NA^R as input.

Now let M be an arbitrary Turing machine and w be an arbitrary string, and suppose we run our new Turing machine A on the encoding $\langle M, w\rangle$. To complete the proof, we need to consider two cases: Either M accepts w or M does not accept w.

- First, suppose M accepts w.
  - Then for all strings x, the machine M_w accepts x.
  - So Accept$(M_w) = \Sigma^*$, by the definition of Accept(M_w).
  - So $\langle M_w\rangle \notin$ NeverAccept, by definition of NeverAccept.
  - So NA rejects $\langle M_w\rangle$, because NA decides NeverAccept.
  - So NA^R accepts $\langle M_w\rangle$, buy construction of NA^R.
  - We conclude that A accepts $\langle M, w\rangle$, by construction of A.
- On the other hand, suppose M does not accept w, either rejecting or diverging instead.
  - Then for all strings x, the machine M_w does not accept x.
  - So Accept$(M_w) = \varnothing$, by the definition of Accept(M_w).
  - So $\langle M_w\rangle \in$ NeverAccept, by definition of NeverAccept.
  - So NA accepts $\langle M_w\rangle$, because NA decides NeverAccept.
  - So NA^R rejects $\langle M_w\rangle$, buy construction of NA^R.
  - We conclude that A rejects $\langle M, w\rangle$, by construction of A.

In short, A decides the language ACCEPT, which is impossible. We conclude that NA does not exist. □

Again, similar arguments imply that the languages NEVERREJECT, NEVERHALT, and NEVERDIVERGE are undecidable. In each case, the core of the argument is describing how to transform the incoming machine-and-input encoding $\langle M, w\rangle$ into the encoding of an appropriate new Turing machine $\langle M_w\rangle$.

Now that we know that NEVERACCEPT and its relatives are undecidable, we can use them as the basis of further reduction proofs. Here is a typical example:

Theorem 14. *The language* DIVERGESAME $:= \{\langle M_1\rangle\langle M_2\rangle \mid \text{DIVERGE}(M_1) = \text{DIVERGE}(M_2)\}$ *is undecidable.*

Proof: Suppose for the sake of argument that there is a Turing machine DS that decides DIVERGESAME. Then we can build a Turing machine ND that decides NEVERDIVERGE as follows. Fix a Turing machine Y that accepts Σ^* (for example, by defining $\delta(\text{start}, a) = (\text{accept}, \cdot, \cdot)$ for all $a \in \Gamma$). Given an arbitrary Turing machine encoding $\langle M\rangle$ as input, ND writes the string $\langle M\rangle\langle Y\rangle$ onto the tape and then passes control to DS. There are two cases to consider:

- If DS accepts $\langle M\rangle\langle Y\rangle$, then $\text{DIVERGE}(M) = \text{DIVERGE}(Y) = \varnothing$, so $\langle M\rangle \in$ NEVERDIVERGE.
- If DS rejects $\langle M\rangle\langle Y\rangle$, then $\text{DIVERGE}(M) \neq \text{DIVERGE}(Y) = \varnothing$, so $\langle M\rangle \notin$ NEVERDIVERGE.

In short, ND accepts $\langle M\rangle$ if and only if $\langle M\rangle \in$ NEVERDIVERGE, which is impossible. We conclude that DS does not exist. □

7.11 Rice's Theorem

In 1953, Henry Rice proved the following extremely powerful theorem, which essentially states that ***every*** interesting question about the language accepted by a Turing machine is undecidable.

★★★

The following formulation is closer to the proof and may be (slightly) easier to use:

Rice's Theorem. *For any set $\mathcal{L}$ of languages, if $\varnothing \notin \mathcal{L}$ and there is a Turing machine M such that $\text{ACCEPT}(M) \in \mathcal{L}$, then the language $\text{ACCEPTIN}(\mathcal{L}) := \{\langle M\rangle \mid \text{ACCEPT}(M) \in \mathcal{L}\}$ is undecidable.*

The only downside of this formulation is that when $\varnothing \in \mathcal{L}$, we need to consider either the complementary property $\overline{\mathcal{L}} = 2^{\Sigma^*} \setminus \mathcal{L}$ or the complementary language $\{\langle M\rangle \mid \text{ACCEPT}(M) \notin \mathcal{L}\}$.

Rice's Theorem. *Let $\mathcal{L}$ be any set of languages that satisfies the following conditions:*
- *There is a Turing machine Y such that $\text{ACCEPT}(Y) \in \mathcal{L}$.*
- *There is a Turing machine N such that $\text{ACCEPT}(N) \notin \mathcal{L}$.*

The language $\text{ACCEPTIN}(\mathcal{L}) := \{\langle M\rangle \mid \text{ACCEPT}(M) \in \mathcal{L}\}$ is undecidable.

Proof: Without loss of generality, suppose $\varnothing \notin \mathcal{L}$. (A symmetric argument establishes the theorem in the opposite case $\varnothing \in \mathcal{L}$.) Fix an arbitrary Turing machine Y such that $\text{ACCEPT}(Y) \in \mathcal{L}$.

Suppose to the contrary that there is a Turing machine $A_{\mathcal{L}}$ that decides $\text{ACCEPTIN}(\mathcal{L})$. To derive a contradiction, we describe a Turing machine H that decides the halting language HALT, using $A_{\mathcal{L}}$ as a black-box subroutine. Given the encoding $\langle M, w\rangle$ of an arbitrary Turing machine M and an arbitrary string w as input, H writes the encoding $\langle WTF\rangle$ of a new Turing machine WTF that executes the following algorithm:

```
WTF(x):
    run M on input w (and discard the result)
    run Y on input x
```

H then passes the new encoding $\langle WTF \rangle$ to $A_{\mathcal{L}}$.

Now let M be an arbitrary Turing machine and w be an arbitrary string, and suppose we run our new Turing machine H on the encoding $\langle M, w \rangle$. There are two cases to consider.

- Suppose M halts on input w.
  - Then for all strings x, the machine WTF accepts x if and only if Y accepts x.
  - So $\text{Accept}(WTF) = \text{Accept}(Y)$, by definition of $\text{Accept}(\,\cdot\,)$.
  - So $\text{Accept}(WTF) \in \mathcal{L}$, by definition of Y.
  - So $A_{\mathcal{L}}$ accepts $\langle WTF \rangle$, because $A_{\mathcal{L}}$ decides $\text{AcceptIn}(\mathcal{L})$.
  - So H accepts $\langle M, w \rangle$, by definition of H.
- Suppose M does not halt on input w.
  - Then for all strings x, the machine WTF does not halt on input x, and therefore does not accept x.
  - So $\text{Accept}(WTF) = \varnothing$, by definition of $\text{Accept}(WTF)$.
  - So $\text{Accept}(WTF) \notin \mathcal{L}$, by our assumption that $\varnothing \notin \mathcal{L}$.
  - So $A_{\mathcal{L}}$ rejects $\langle WTF \rangle$, because $A_{\mathcal{L}}$ decides $\text{AcceptIn}(\mathcal{L})$.
  - So H rejects $\langle M, w \rangle$, by definition of H.

In short, H decides the language Halt, which is impossible. We conclude that $A_{\mathcal{L}}$ does not exist. □

The set $\mathcal{L}$ in the statement of Rice's Theorem is often called a ***property*** of languages, rather than a *set*, to avoid the inevitable confusion about sets of sets of finite sequences of characters. We can also think of $\mathcal{L}$ as a ***decision problem*** about languages, where the languages are represented by Turing machines that accept or decide them. Rice's theorem states that the ***only*** properties of languages that are decidable are the trivial properties "Does this Turing machine accept an acceptable language?" (Answer: Yes, by definition.) and "Does this Turing machine accept Discover?" (Answer: No, because Discover is a credit card, not a language.)

Rice's Theorem makes it incredibly easy to prove that language properties are undecidable; we only need to exhibit one acceptable language that has the property and another acceptable language that does not. In fact, ***every*** proof using Rice's theorem can use at least one of the following Turing machines:

- M_{Accept} accepts every string, by defining $\delta(\text{start}, a) = \text{accept}$ for every tape symbol a.
- M_{Reject} rejects every string, by defining $\delta(\text{start}, a) = \text{reject}$ for every tape symbol a.
- M_{Diverge} diverges on every string, by defining $\delta(\text{start}, a) = (\text{start}, a, +1)$ for every tape symbol a.

Corollary 15. *Each of the following languages is undecidable.*
(a) $\{\langle M \rangle \mid M \textit{ accepts given an empty initial tape}\}$
(b) $\{\langle M \rangle \mid M \textit{ accepts the string } \texttt{UIUC}\}$
(c) $\{\langle M \rangle \mid M \textit{ accepts exactly three strings}\}$

(d) $\{\langle M\rangle \mid M$ *accepts all palindromes*$\}$
(e) $\{\langle M\rangle \mid \textsc{Accept}(M)$ *is regular*$\}$
(f) $\{\langle M\rangle \mid \textsc{Accept}(M)$ *is not regular*$\}$
(g) $\{\langle M\rangle \mid \textsc{Accept}(M)$ *is undecidable*$\}$
(h) $\{\langle M\rangle \mid \textsc{Accept}(M) = \textsc{Accept}(N)\}$*, for some arbitrary fixed Turing machine* N*.*

Proof: In all cases, undecidability follows from Rice's theorem.

(a) Let $\mathcal{L}$ be the set of all languages that contain the empty string. Then $\textsc{AcceptIn}(\mathcal{L}) = \{\langle M\rangle \mid M$ accepts given an empty initial tape$\}$.

- Given an empty initial tape, $M_{\textsc{Accept}}$ accepts, so $\textsc{Accept}(M_{\textsc{Accept}}) \in \mathcal{L}$.
- Given an empty initial tape, $M_{\textsc{Diverge}}$ does not accept, so $\textsc{Accept}(M_{\textsc{Diverge}}) \notin \mathcal{L}$.

Therefore, Rice's Theorem implies that $\textsc{AcceptIn}(\mathcal{L})$ is undecidable.

(b) Let $\mathcal{L}$ be the set of all languages that contain the string `UIUC`.

- $M_{\textsc{Accept}}$ accepts `UIUC`, so $\textsc{Accept}(M_{\textsc{Accept}}) \in \mathcal{L}$.
- $M_{\textsc{Diverge}}$ does not accept `UIUC`, so $\textsc{Accept}(M_{\textsc{Diverge}}) \notin \mathcal{L}$.

Therefore, $\textsc{AcceptIn}(\mathcal{L}) = \{\langle M\rangle \mid M$ accepts the string `UIUC`$\}$ is undecidable by Rice's Theorem.

(c) There is a Turing machine that accepts the language $\{$`larry`, `curly`, `moe`$\}$. On the other hand, $M_{\textsc{Reject}}$ does not accept exactly three strings.

(d) $M_{\textsc{Accept}}$ accepts all palindromes, and $M_{\textsc{Reject}}$ does not accept all palindromes.

(e) $M_{\textsc{Reject}}$ accepts the regular language $\varnothing$, and there is a Turing machine $M_{0^n1^n}$ that accepts the non-regular language $\{0^n1^n \mid n \ge 0\}$.

(f) $M_{\textsc{Reject}}$ accepts the regular language $\varnothing$, and there is a Turing machine $M_{0^n1^n}$ that accepts the non-regular language $\{0^n1^n \mid n \ge 0\}$.[4]

(g) $M_{\textsc{Reject}}$ accepts the decidable language $\varnothing$, and there is a Turing machine that *accepts* the undecidable language $\textsc{SelfReject}$.

(h) The Turing machine N accepts $\textsc{Accept}(N)$ by definition. For the negative Turing machine $M_{\textsc{Accept}}$ accepts Σ^* and the Turing machine $M_{\textsc{Reject}}$ accepts $\varnothing$, so at least one of those two machines does not accept $\textsc{Accept}(N)$. □

We can also use Rice's theorem as a component in more complex undecidability proofs, where the target language consists of more than just a single Turing machine encoding.

Theorem 16. *The language* $L := \left\{\langle M, w\rangle \;\middle|\; M \text{ accepts } w^k \text{ for every integer } k \ge 0\right\}$ *is undecidable.*

Proof: Fix an arbitrary string w, and let $\mathcal{L}$ be the set of all languages that contain w^k for all k. Then $\textsc{Accept}(M_{\textsc{Accept}}) = \Sigma^* \in \mathcal{L}$ and $\textsc{Accept}(M_{\textsc{Reject}}) = \varnothing \notin \mathcal{L}$. Thus, *even if the string w is fixed in advance*, no Turing machine can decide L. □

Nearly identical reduction arguments imply the following variants of Rice's theorem. (The names of these theorems are not standard.)

[4] Yes, parts (e) and (f) have exactly the same proof.

Rice's Rejection Theorem. *Let $\mathcal{L}$ be any set of languages that satisfies the following conditions:*

- *There is a Turing machine Y such that $\textsc{Reject}(Y) \in \mathcal{L}$*
- *There is a Turing machine N such that $\textsc{Reject}(N) \notin \mathcal{L}$.*

The language $\textsc{RejectIn}(\mathcal{L}) := \big\{\langle M\rangle \;\big|\; \textsc{Reject}(M) \in \mathcal{L}\big\}$ is undecidable.

Rice's Halting Theorem. *Let $\mathcal{L}$ be any set of languages that satisfies the following conditions:*

- *There is a Turing machine Y such that $\textsc{Halt}(Y) \in \mathcal{L}$*
- *There is a Turing machine N such that $\textsc{Halt}(N) \notin \mathcal{L}$.*

The language $\textsc{HaltIn}(\mathcal{L}) := \big\{\langle M\rangle \;\big|\; \textsc{Halt}(M) \in \mathcal{L}\big\}$ is undecidable.

Rice's Divergence Theorem. *Let $\mathcal{L}$ be any set of languages that satisfies the following conditions:*

- *There is a Turing machine Y such that $\textsc{Diverge}(Y) \in \mathcal{L}$*
- *There is a Turing machine N such that $\textsc{Diverge}(N) \notin \mathcal{L}$.*

The language $\textsc{DivergeIn}(\mathcal{L}) := \big\{\langle M\rangle \;\big|\; \textsc{Diverge}(M) \in \mathcal{L}\big\}$ is undecidable.

Rice's Decision Theorem. *Let $\mathcal{L}$ be any set of languages that satisfies the following conditions:*

- *There is a Turing machine Y that* ***decides*** *an language in $\mathcal{L}$.*
- *There is a Turing machine N that* ***decides*** *an language not in $\mathcal{L}$.*

The language $\textsc{DecideIn}(\mathcal{L}) := \big\{\langle M\rangle \;\big|\; M$ ***decides*** *a language in $\mathcal{L}\big\}$ is undecidable.*

As easy as it is to use Rice's theorem and its variants, they cannot be used for all undecidability proofs; these theorems only apply to properties of *languages*. For example, the language $\textsc{ThisIsSparta} := \{\langle M\rangle \mid M$ accepts the string SPARTA after exactly 300 steps$\}$ is decidable, even though there are Turing machines that accept the string SPARTA after exactly 300 steps and there are other Turing machines that do not.

More subtly, Rice's theorem cannot be applied to self-referential languages like $\textsc{RevAccept} := \{\langle M\rangle \mid M \text{ accepts } \langle M\rangle^R\}$, because membership depends on details of the encoded machine and not just the language that the encoded machine accepts. To be clear: $\textsc{RevAccept}$ ***is*** undecidable; you just can't use Rice's theorem to prove that fact.

⋆7.12 The Rice-McNaughton-Myhill-Shapiro Theorem

The following subtle generalization of Rice's theorem precisely characterizes which properties of acceptable languages are *acceptable*. This result was partially proved by Henry Rice in 1953, in the same paper that proved Rice's Theorem; Robert McNaughton, John Myhill, and Norman Shapiro completed the proof a few years later, each independently from the other two.[5]

The Rice-McNaughton-Myhill-Shapiro Theorem. *Let $\mathcal{L}$ be an arbitrary set of acceptable languages. The language $\textsc{AcceptIn}(\mathcal{L}) := \{\langle M\rangle \mid \textsc{Accept}(M) \in \mathcal{L}\}$ is* ***acceptable*** *if and only if $\mathcal{L}$ satisfies the following conditions:*

(a) $\mathcal{L}$ is ***monotone****: For any language $L \in \mathcal{L}$, every superset of L is also in $\mathcal{L}$.*

(b) $\mathcal{L}$ is ***compact****: Every language in $\mathcal{L}$ has a finite subset that is also in $\mathcal{L}$.*

[5]McNaughton never published his proof (although he did announce the result); consequently, this theorem is sometimes called "The Rice-Myhill-Shapiro Theorem". Even more confusingly, Myhill published his proof twice, once in a paper with John Shepherdson and again in a later paper with Jacob Dekker. So maybe it should be called the Rice–Dekker-Myhill–(McNaughton–)Myhill-Shepherdson–Shapiro Theorem.

(c) $\mathcal{L}$ *is* ***finitely acceptable****: The language* $\bigl\{\langle L\rangle \mid L \in \mathcal{L} \text{ and } L \text{ is finite}\bigr\}$ *is acceptable.*[6]

I won't give a complete proof of this theorem (in part because it requires techniques I haven't introduced), but the following lemma is arguably the most interesting component:

Lemma 17. *Let* $\mathcal{L}$ *be a set of acceptable languages. If* $\mathcal{L}$ *is not monotone, then* $\textsc{AcceptIn}(\mathcal{L})$ *is unacceptable.*

Proof: Suppose to the contrary that there is a Turing machine $AI_{\mathcal{L}}$ that accepts $\textsc{AcceptIn}(\mathcal{L})$. Using this Turing machine as a black box, we describe a Turing machine SD that accepts the unacceptable language $\textsc{SelfDiverge}$. Fix two Turing machines Y and N such that

$$\begin{aligned} &\textsc{Accept}(Y) \in \mathcal{L}, \\ &\textsc{Accept}(N) \notin \mathcal{L}, \\ \text{and}\quad &\textsc{Accept}(Y) \subseteq \textsc{Accept}(N). \end{aligned}$$

Let w be the input to SD. After verifying that $w = \langle M\rangle$ for some Turing machine M (and rejecting otherwise), SD writes the encoding $\langle WTF\rangle$ or a new Turing machine WTF that implements the following algorithm:

```
WTF(x):
  write x to second tape
  write ⟨M⟩ to third tape
  in parallel:
      run Y on the first tape
      run N on the second tape
      run M on the third tape
  if Y accepts x
      accept
  if N accepts x and M halts on ⟨M⟩
      accept
```

Finally, SD passes the new encoding $\langle WTF\rangle$ to $AI_{\mathcal{L}}$. There are two cases to consider:

- If M halts on $\langle M\rangle$, then $\textsc{Accept}(WTF) = \textsc{Accept}(N) \notin \mathcal{L}$, and therefore $AI_{\mathcal{L}}$ does not accept $\langle WTF\rangle$.
- If M does not halt on $\langle M\rangle$, then $\textsc{Accept}(WTF) = \textsc{Accept}(Y) \in \mathcal{L}$, and therefore $AI_{\mathcal{L}}$ accepts $\langle WTF\rangle$.

In short, SD accepts $\textsc{SelfDiverge}$, which is impossible. We conclude that SD does not exist. $\square$

Corollary 18. *Each of the following languages is unacceptable.*
(a) $\{\langle M\rangle \mid \textsc{Accept}(M) \text{ is finite}\}$
(b) $\{\langle M\rangle \mid \textsc{Accept}(M) \text{ is infinite}\}$
(c) $\{\langle M\rangle \mid \textsc{Accept}(M) \text{ is regular}\}$
(d) $\{\langle M\rangle \mid \textsc{Accept}(M) \text{ is not regular}\}$
(e) $\{\langle M\rangle \mid \textsc{Accept}(M) \text{ is decidable}\}$

[6]Here the encoding $\langle L\rangle$ of a finite language $L \subseteq \Sigma^*$ is exactly the string that you would write down to explicitly describe L. Formally, $\langle L\rangle$ is the unique string over the alphabet $\Sigma \cup \{\texttt{\{}, \bullet, \texttt{\}}, \varepsilon\}$ that contains the strings in L in lexicographic order, separated by commas $\bullet$ and surrounded by braces $\texttt{\{\}}$, with ε representing the empty string. For example, $\bigl\langle\{\varepsilon, \texttt{0}, \texttt{01}, \texttt{0110}, \texttt{01101001}\}\bigr\rangle = \texttt{\{}\varepsilon\bullet\texttt{0}\bullet\texttt{01}\bullet\texttt{0110}\bullet\texttt{01101001\}}$.

(f) $\{\langle M\rangle \mid \textsc{Accept}(M) \text{ is undecidable}\}$
(g) $\{\langle M\rangle \mid M \text{ accepts at least one string in } \textsc{SelfDiverge}\}$
(h) $\{\langle M\rangle \mid \textsc{Accept}(M) = \textsc{Accept}(N)\}$*, for some arbitrary fixed Turing machine* N*.*

Proof: (a) The set of finite languages is not monotone: $\varnothing$ is finite; Σ^* is not finite; both $\varnothing$ and Σ^* are acceptable (in fact decidable); and $\varnothing \subset \Sigma^*$.

(b) The set of infinite acceptable languages is not compact: No finite subset of the infinite acceptable language Σ^* is infinite!

(c) The set of regular languages is not monotone: Consider the languages $\varnothing$ and $\{0^n1^n \mid n \geq 0\}$.

(d) The set of non-regular acceptable languages is not monotone: Consider the languages $\{0^n1^n \mid n \geq 0\}$ and Σ^*.

(e) The set of decidable languages is not monotone: Consider the languages $\varnothing$ and $\textsc{SelfReject}$.

(f) The set of undecidable acceptable languages is not monotone: Consider the languages $\textsc{SelfReject}$ and Σ^*.

(g) The set $\mathcal{L} = \{L \mid L \cap \textsc{SelfDiverge} \neq \varnothing\}$ is not finitely acceptable. For any string w, deciding whether $\{w\} \in \mathcal{L}$ is equivalent to deciding whether $w \in \textsc{SelfDiverge}$, which is impossible.

(h) If $\textsc{Accept}(N) \neq \Sigma^*$, then the set $\{\textsc{Accept}(N)\}$ is not monotone. On the other hand, if $\textsc{Accept}(N) = \Sigma^*$, then the set $\{\textsc{Accept}(N)\}$ is not compact: No finite subset of Σ^* is equal to Σ^*!

□

7.13 Turing Machine Behavior: It's Complicated

Rice's theorems imply that every interesting question about the language that a Turing machine accepts—or more generally, the function that a program computes—is undecidable. A more subtle question is whether we can recognize Turing machines that exhibit certain *internal behavior*. Some behaviors we can recognize; others we can't.

Theorem 19. *The language* $\textsc{NeverLeft} := \{\langle M, w\rangle \mid \text{Given } w \text{ as input, } M \text{ never moves left}\}$ *is decidable.*

Proof: Given the encoding $\langle M, w\rangle$, we simulate M with input w using our universal Turing machine U, but with the following termination conditions. If M ever moves its head to the left, then we reject. If M halts without moving its head to the left, then we accept. Finally, if M reads more than $|Q|$ blanks, where Q is the state set of M, then we accept. If the first two cases do not apply, M only moves to the right; moreover, after reading the entire input string, M only reads blanks. Thus, after reading $|Q|$ blanks, it must repeat some state, and therefore loop forever without moving to the left. The three cases are exhaustive. □

Theorem 20. *The language* $\textsc{LeftThree} := \{\langle M, w\rangle \mid \text{Given } w \text{ as input, } M \text{ eventually moves left three times in a row}\}$ *is undecidable.*

Proof: Given $\langle M\rangle$, we build a new Turing machine M' that accepts the same language as M and moves left three times in a row if and only if it accepts, as follows. For each non-accepting state p

of M, the new machine M' has three states p_1, p_2, p_3, with the following transitions:

$$\begin{aligned}
\delta'(p_1, a) &= (q_2, b, \Delta), && \text{where } (q, b, \Delta) = \delta(p, a) \text{ and } q \neq \text{accept}\\
\delta'(p_2, a) &= (p_3, a, +1)\\
\delta'(p_3, a) &= (p_1, a, -1)
\end{aligned}$$

In other words, after each non-accepting transition, M' moves once to the right and then once to the left. For each transition to accept, M' has a sequence of seven transitions: three steps to the right, then three steps to the left, and then finally accept$'$, all without modifying the tape. (The three steps to the right ensure that M' does not fall off the left end of the tape.)

Finally, M' moves left three times in a row if and only if M accepts w. Thus, if we could decide LeftThree, we could also decide Accept, which is impossible. □

There is no hard and fast rule like Rice's theorem to distinguish decidable behaviors from undecidable behaviors, but I can offer two rules of thumb.

- If it is possible to simulate an arbitrary Turing machine while avoiding the target behavior, then the behavior is not decidable. For example, there is no algorithm to determine whether a given Turing machine reenters its start state, or revisits the left end of the tape, or writes a blank.
- If a Turing machine with the target behavior is limited to a finite number of configurations, or is guaranteed to force an infinite loop after a finite number of transitions, then the behavior is likely to be decidable. For example, there *are* algorithms to determine whether a given Turing machine ever leaves its start state, or reads its entire input string, or writes a non-blank symbol over a blank.

Exercises

1. Let M be an arbitrary Turing machine.
   (a) Describe a Turing machine M^R such that
   $$\text{Accept}(M^R) = \text{Reject}(M) \quad\text{and}\quad \text{Reject}(M^R) = \text{Accept}(M).$$
   (b) Describe a Turing machine M^A such that
   $$\text{Accept}(M^A) = \text{Accept}(M) \quad\text{and}\quad \text{Reject}(M^A) = \varnothing.$$
   (c) Describe a Turing machine M^H such that
   $$\text{Accept}(M^H) = \text{Halt}(M) \quad\text{and}\quad \text{Reject}(M^H) = \varnothing.$$

2. (a) Prove that Accept is undecidable.
   (b) Prove that Reject is undecidable.
   (c) Prove that Diverge is undecidable.

3. (a) Prove that NeverReject is undecidable.

(b) Prove that NeverHalt is undecidable.

(c) Prove that NeverDiverge is undecidable.

4. Prove that each of the following languages is undecidable.

   (a) $\textsc{AlwaysAccept} := \{\langle M\rangle \mid \textsc{Accept}(M) = \Sigma^*\}$

   (b) $\textsc{AlwaysReject} := \{\langle M\rangle \mid \textsc{Reject}(M) = \Sigma^*\}$

   (c) $\textsc{AlwaysHalt} := \{\langle M\rangle \mid \textsc{Halt}(M) = \Sigma^*\}$

   (d) $\textsc{AlwaysDiverge} := \{\langle M\rangle \mid \textsc{Diverge}(M) = \Sigma^*\}$

5. Let $\mathcal{L}$ be a non-empty proper subset of the set of acceptable languages. Prove that the following languages are undecidable:

   (a) $\textsc{RejectIn}(\mathcal{L}) := \big\{\langle M\rangle \bigm| \textsc{Reject}(M) \in \mathcal{L}\big\}$

   (b) $\textsc{HaltIn}(\mathcal{L}) := \big\{\langle M\rangle \bigm| \textsc{Halt}(M) \in \mathcal{L}\big\}$

   (c) $\textsc{DivergeIn}(\mathcal{L}) := \big\{\langle M\rangle \bigm| \textsc{Diverge}(M) \in \mathcal{L}\big\}$

6. For each of the following decision problems, either *sketch* an algorithm or prove that the problem is undecidable. Recall that w^R denotes the reversal of string w. For each problem, the input is the encoding $\langle M\rangle$ of a Turing machine M.

   (a) Does M reject the empty string?

   (b) Does M accept $\langle M\rangle^R$?

   (c) Does M accept $\langle M\rangle\langle M\rangle$?

   (d) Does M accept $\langle M\rangle^k$ for any integer k?

   (e) Does M accept the encoding of any Turing machine?

   (f) Is there a Turing machine that accepts $\langle M\rangle$?

   (g) Is $\langle M\rangle$ a palindrome?

   (h) Does M reject any palindrome?

   (i) Does M accept all palindromes?

   (j) Does M diverge only on palindromes?

   (k) Is there an input string that forces M to move left?

   (l) Is there an input string that forces M to move left three times in a row?

   (m) Does M accept the encoding of any Turing machine N such that $\textsc{Accept}(N) = \textsc{SelfDiverge}$?

7. For each of the following decision problems, either *sketch* an algorithm or prove that the problem is undecidable. Recall that w^R denotes the reversal of string w. For each problem, the input is an encoding $\langle M, w\rangle$ of a Turing machine M and its input string w.

   (a) Does M accept the string ww^R?

(b) Does M accept either w or w^R?

(c) Does M either accept w or reject w^R?

(d) Does M accept the string w^k for some integer k?

(e) Does M accept w in at most $2^{|w|}$ steps?

(f) If we run M on input w, does M ever change a symbol on its tape?

(g) If we run M on input w, does M ever move to the right?

(h) If we run M on input w, does M ever move to the right twice in a row?

(i) If we run M on input w, does M move its head to the right more than $2^{|w|}$ times (not necessarily consecutively)?

(j) If we run M with input w, does M ever change a $\square$ on the tape to any other symbol?

(k) If we run M with input w, does M ever change a $\square$ on the tape to 1?

(l) If we run M with input w, does M ever write a $\square$?

(m) If we run M with input w, does M ever leave its start state?

(n) If we run M with input w, does M ever reenter its start state?

(o) If we run M with input w, does M ever reenter a state that it previously left? That is, are there states $p \neq q$ such that M moves from state p to state q and then later moves back to state p?

8. Let M be a Turing machine, let w be an arbitrary input string, and let s and t be positive integers integer. We say that M accepts w ***in space s*** if M accepts w after accessing at most the first s cells on the tape, and M accepts w ***in time t*** if M accepts w after at most t transitions.

(a) Prove that the following languages are decidable:

i. $\left\{\langle M, w\rangle \;\middle|\; M \text{ accepts } w \text{ in time } |w|^2\right\}$

ii. $\left\{\langle M, w\rangle \;\middle|\; M \text{ accepts } w \text{ in space } |w|^2\right\}$

(b) Prove that the following languages are undecidable:

i. $\left\{\langle M\rangle \;\middle|\; M \text{ accepts at least one string } w \text{ in time } |w|^2\right\}$

ii. $\left\{\langle M\rangle \;\middle|\; M \text{ accepts at least one string } w \text{ in space } |w|^2\right\}$

9. Let L_0 be an arbitrary language. For any integer $i > 0$, define the language

$$L_i := \left\{\langle M\rangle \;\middle|\; M \text{ decides } L_{i-1}\right\}.$$

For which integers $i > 0$ is L_i decidable? Obviously the answer depends on the initial language L_0; give a complete characterization of all possible cases. Prove your answer is correct. *[Hint: This question is a lot easier than it looks!]*

10. Argue that each of the following decision problems about programs in your favorite programming language are undecidable.

(a) Does this program correctly compute Fibonacci numbers?

(b) Can this program fall into an infinite loop?

(c) Will the value of this variable ever change?

(d) Will this program every attempt to deference a null pointer?

(e) Does this program free every block of memory that it dynamically allocates?

(f) Is any statement in this program unreachable?

(g) Do these two programs compute the same function?

*11. Call a Turing machine ***conservative*** if it never writes over its input string. More formally, a Turing machine is conservative if for every transition $\delta(p,a) = (q,b,\Delta)$ where $a \in \Sigma$, we have $b = a$; and for every transition $\delta(p,a) = (q,b,\Delta)$ where $a \notin \Sigma$, we have $b \neq \Sigma$.

(a) Prove that if M is a conservative Turing machine, then $\textsc{Accept}(M)$ is a regular language.

(b) Prove that the language $\{\langle M\rangle \mid M \text{ is conservative and } M \text{ accepts } \varepsilon\}$ is undecidable.

Together, these two results imply that every conservative Turing machine accepts the same language as some DFA, but it is impossible to determine *which* DFA.

★12. (a) Prove that it is undecidable whether a given C++ program is syntactically correct. *[Hint: Use templates!]*

(b) Prove that it is undecidable whether a given ANSI C program is syntactically correct. *[Hint: Use the preprocessor!]*

(c) Prove that it is undecidable whether a given Perl program is syntactically correct. *[Hint: Does that slash character / delimit a regular expression or represent division?]*