Lecture #9: Proving Undecidability — Part Two

What To Do Before the Lecture

1. Watch the videos for Lecture #9 —noting that they will probably be understandable if you play them at double speed. If you do not have time for this then look at the "Key Concepts" document that is found, immediately after the videos for this lecture, on the course web site, instead.
2. ***Print*** and read through the rest of this document and — if you have time — try to solve the problems! These should help you to check that you understand how Turing machines process strings, and that you understand how to prove things about them.
3. The supplemental documents are for interest only. They provide additional details about the proof of undecidability included in the second set of lecture slides for this lecture as well as another example of a proof of undecidability using a man-one reduction.

Problems To Be Solved

1. Consider the alphabet
$$\Sigma_D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$
Let $\sigma_0 = 0$, $\sigma_1 = 1$, $\sigma_2 = 2$, $\sigma_3 = 3$, and so on (so that, for each integer i such that $0 \leq i \leq 9$, σ_i is the symbol in Σ_D that we generally use to represent i). Recall that, for every ***positive*** integer n, there is an ***unpadded decimal representation*** of n, consisting of an integer $k \geq 0$ and digits $d_0, d_1, \ldots, d_k \in \{d \in \mathbb{N} \mid 0 \leq d \leq 9\}$ such that $d_k \geq 1$ and
$$n = \sum_{\ell=0}^{k} d_\ell \times 10^\ell.$$
The ***unpadded decimal representation*** of n is the string $\sigma_{d_k}\sigma_{d_{k-1}} \ldots \sigma_{d_1}\sigma_{d_0}$, a string in $\Sigma_D^\star$ with length $k + 1$.

For example, if $n = 312$ then the ***unpadded decimal representation*** of n consists of the integer $k = 2$ and the digits $d_0 = 2$, $d_1 = 1$ and $d_2 = 3$ — since $d_k = d_2 \geq 1$ — and

$$312 = 2 \times 10^0 + 1 \times 10^1 + 3 \times 10^2$$

— and the ***unpadded decimal representation*** of n is the string 312 — a string in $\Sigma_D^\star$ with length $k + 1 = 3$.[1]

In order to have "unpadded decimal representations" of all natural numbers — including zero — we generally define the ***unpadded decimal representation*** of zero to be the string 0 — a string in $\Sigma_D^\star$ with length one.

Let Unpad $\subseteq \Sigma_D^\star$ be the set of unpadded decimal representations of natural numbers. This is certainly a decidable language — indeed, it is even a ***regular*** language — because this is just the union of the set of all non-empty strings in $\Sigma_D^\star$ that do not begin with 0, and the set $\{0\}$.

A (possibly) ***padded decimal representation*** of a natural number n is any string in $\Sigma_D^\star$ that is the concatenation of a string of zero more 0's and the unpadded decimal representation of n. Thus the ***padded decimal representations*** of zero include the strings 0^i for every positive integer i, and the ***padded decimal representations*** of the natural number 312 are the strings of the form 0^j312 for every non-negative integer j.

Let Pad $\subseteq \Sigma_D^\star$ be the set of (possibly) padded decimal representations of natural numbers. This is also certainly a decidable language — indeed, it is even a ***regular*** language — because this is just $\Sigma_D^+ = \Sigma_D^\star \setminus \{\lambda\}$, that is, the set of all ***non-empty*** strings in $\Sigma_D^\star$.

Now let $S \subseteq \mathbb{N}$. This can be used to define ***two*** languages, that are each subsets of $\Sigma_D^\star$:

- $\mathcal{U}_S \subseteq$ Unpad is the set of all ***unpadded decimal representations*** of numbers $n \in S$.
- $\mathcal{P}_S \subseteq$ Pad is the set of all ***padded decimal representations*** of numbers $n \in S$.

For example, if $S_1 = \emptyset \subseteq \mathbb{N}$ then $\mathcal{U}_{S_1} = \mathcal{P}_{S_1} = \emptyset \subseteq \Sigma_D^\star$. On the other hand, if $S_2 = \{312\} \subseteq \mathbb{N}$ then $\mathcal{U}_{S_2} = \{312\} \subseteq$ Unpad and $\mathcal{P}_{S_2} = \{0^j312 \mid j \in \mathbb{N}\} \subseteq$ Pad.

[1] Describing this can be tricky, because we are so used to working with the "unpadded decimal representations" of positive integers that it might not be easy to see the difference between the unpadded decimal representation of the positive integer n — which is a non-empty string in $\Sigma_D^\star$ — and the integer n, itself.

Our goal is to prove that $\mathcal{U}_S \preceq_{\mathsf{M}} \mathcal{P}_S$ for *every* subset $S \subseteq \mathbb{N}$.

What We Need To Provide — and the Properties It Must Satisfy:

Let $\omega \in \Sigma_D^\star$. ***What Can We Set $f(\omega)$ To Be, When $\omega \notin$ Unpad? Why?***

What Can We Set $f(\omega)$ To Be, When $\omega \in$ Unpad Why?

The Function f:

A First Claim about f and Its Proof:

A Second Claim About f and Its Proof:

A Third Claim About f and Its Proof:

2. The following decision problems have now been introduced:

   Acceptance Problem

   Instance: A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ and an input string $\omega \in \Sigma^\star$

   Question: Does M accept ω?

   Halting Problem

   Instance: A Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ and an input string $\omega \in \Sigma^\star$

   Question: Does M halt, when executed on input ω?

   These have languages of Yes-instances A_{TM} and *Halt*$_{\text{TM}}$, respectively. Suppose that we wish to show that the "Acceptance Problem' is reducible to the "Halting Problem Problem", that is,

   $$A_{\text{TM}} \preceq_{\text{M}} \textit{Halt}_{\text{TM}}.$$

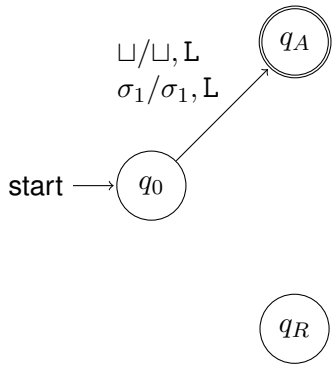
   What do we need to provide — and what properties must it satisfy?

Let $f_1 : \Sigma^\star_{\mathsf{TM}} \to \Sigma^\star_{\mathsf{TM}}$ such that $f_1(\mu) = \mu$ for every string $\mu \in \Sigma^\star_{\mathsf{TM}}$.

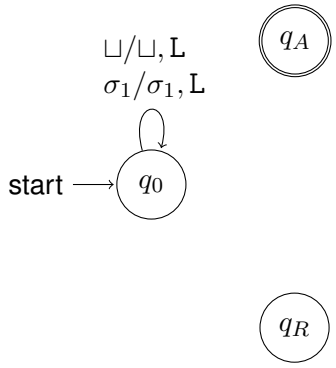
Is f_1 a Many-One Reduction from $\mathbf{A_{TM}}$ to $\mathbf{HALT_{TM}}$?

Why — or Why Not?

Next let us consider a pair of Turing machines with input alphabet $\Sigma = \{\sigma_1\}$ and tape alphabet $\Gamma = \{\sigma_1, \sqcup\}$. The first of these Turing machines, M_Y, is as follows:



The second of these Turing machines, M_N, is as follows:



In both of these pictures the accept state is shown as "q_A" instead of q_{accept}, and the reject state is shown as "q_R" instead of "q_{reject}", to simplify the picture.

Now note that if $x_{\text{Yes}} \in \Sigma^\star_{\text{TM}}$ is the encoding of the above Turing machine M_Y, and the empty string λ, then $x_{\text{Yes}} \in \text{HALT}_{\text{TM}}$. On the other hand, if $x_{\text{No}} \in \Sigma^\star_{\text{TM}}$ is the encoding of the Turing machine M_N, and the empty string λ, then $x_{\text{no}} \notin \text{HALT}_{\text{TM}}$.

Let $f_2 : \Sigma^\star_{\mathsf{TM}} \to \Sigma^\star_{\mathsf{TM}}$ such that, for every string $\mu \in \Sigma^\star_{\mathsf{TM}}$,

$$f_2(\mu) = \begin{cases} x_{\mathsf{Yes}} & \text{if } \mu \in A_{\mathsf{TM}}, \\ x_{\mathsf{No}} & \text{if } \mu \notin A_{\mathsf{TM}}. \end{cases}$$

Is f_2 a Many-One Reduction from $\mathbf{A_{TM}}$ to $\mathbf{HALT_{TM}}$?

Why — or Why Not?

One More Attempt — A Mapping That Can Be Used:

Proof That This Mapping Would Work

Adding Detail: Considering Encodings

Confirming that the Usual Situation Arises — and Handling Other Input Strings

What Is The Function Being Used?

Two Claims, and Their Proofs

Proving That f is Computable

One More Claim To State and Prove

A (Somewhat) "High-Level Algorithm" To Compute This Function

Additional Information To Provide

Finishing Things