# First Hard Problems
# Proofs — and Claims for Computations with Turing Machines

This document includes details of proofs for some of the claims found in Lecture #10 — which sometimes concerned claims specific to computations with $\mathcal{S}$-programs. In cases like these one can often state, and prove, similar claims about computations with Turing machines — because the claims concern general properties of ***computation*** that hold for a variety of models, but with different precise statements depending on the model of computation being used. In order to illustrate this, some versions of the claims, for Turing machines, are also stated and proved here.

## A "Universality Theorem" for Computations with Turing Machines

The ***Universality Theorem***, described in the lecture notes, is a statement about computations that depends on encodings, and enumerations of $\mathcal{S}$-programs. If one is to obtain a similar result for computations with Turing machines then both encodings and enumerations of Turing machines are needed too.[1]

### Enumerating Turing Machines

As in the supplemental material for the previous lecture, let us restrict attention to Turing machines

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{halt}})$$

where

$$Q = \{q_0, q_1, q_2, \ldots, q_t, q_{\text{halt}}\}$$

[1] There are *many* ways to encode, and enumerate Turing machines — so that you might find somewhat different versions of a "Universality Theorem for Turing Machines" or, at least, significantly different proofs, in the literature — because these are based on a different encoding scheme for Turing machines than the one used in this course.

for some non-negative integer $t$,

$$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_h\}$$

for some positive integer $h$, and where

$$\Gamma = \{\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_m\}$$

for some positive integer $m$ such that $m \geq h$ — where $\sigma_0$ is "$\sqcup$" and where $\sigma_{h+1}, \sigma_{h+2}, \ldots, \sigma_m$ are non-blank symbols in $M$'s tape alphabet that are not in $M$'s input alphabet (if $m > h$). As discussed in the previous supplemental material, these Turing machines have encodings as strings of symbols over an alphabet

$$\Sigma_{\mathsf{TM}} = \{\texttt{(}, \texttt{)}, \texttt{,}, \texttt{q}, \texttt{s}, \texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}, \texttt{H}, \texttt{L}, \texttt{R}, \texttt{S}, \texttt{\#}\}.$$

Furthermore, it can be shown that the language $\mathsf{TM} \subseteq \Sigma_{\mathsf{TM}}^{\star}$, of encodings of Turing machines, is decidable — and that, for every string $\omega \in \mathsf{TM}$, there is a single Turing machine $M$, with the form described above, that is encoded by $\omega$.

It will be useful to define a total function from $\Sigma_{\mathsf{TM}}^{\star}$ to Turing machines — so that we should also introduce a Turing machine, $M_\omega$, corresponding to each string $\omega \in \Sigma_{\mathsf{TM}}^{\star}$ such that $\omega \notin \mathsf{TM}$, as well. Let us set $M_\omega$, where $\omega \notin \mathsf{TM}$, to be a *fixed* Turing machine that erases its input, so that it computes the function $f_{\mathsf{zero}} : \mathbb{N}^k \to \mathbb{N}$ such that $f_{\mathsf{zero}}(x_1, x_2, \ldots, x_k) = 0$ for all $x_1, x_2, \ldots, x_k \in \mathbb{N}$.

Recall, next, that if $\zeta \in \Sigma_{\mathsf{TM}}^{\star}$ then $\#(\zeta)$ is the natural number encoding $\zeta$, using the encoding scheme for strings (in $\zeta^\star$) introduced in Lecture #5. For $i \in \mathbb{N}$, let us define $M_i$ to be the Turing machine $M_\omega$, as above, for the string $\omega \in \Sigma_{\mathsf{TM}}^{\star}$ such that $\#(\omega) = i$ — so that $\Sigma_{\mathsf{TM}}$ is used as the alphabet, "$\Sigma$", described in the notes for Lecture #5. Thus

$$M_0, M_1, M_2, M_3$$

is an enumeration of all Turing machines.

## "Weak" Computation of Functions by Turing Machines

Unfortunately, this does not lead directly to an enumeration of all Turing-computable functions, because *not all Turing machines compute functions*, as this was defined in the lecture notes for Lecture #7 — because Turing machines do not always halt with the contents of the tape, and the location of the tape head, as described in the definition of a Turing machine that "computes a function".

In order to associate a partial or total function $f_i^{(k)} : \mathbb{N}^k \to \mathbb{N}$ with ever Turing machine $M_i$ (for all $i \in \mathbb{N}$), let us use the notion of "weak computation" of functions that was given, for

Post-Turing machines — so that every Turing machine can now be thought of a computing a partial or total function (in this weaker sense).

Note next that (for $i \in \mathbb{N}$) if

$$M_i = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{halt}})$$

then one can produce another Turing machine

$$\widehat{M_i} = (\widehat{Q}, \Sigma, \widehat{\Gamma}, \widehat{\delta}, \widehat{q_0}, \widehat{q_{\mathsf{halt}}})$$

such that $\widehat{M_i}$ computes (that is, "strongly computes") the same partial or total function as the partial or total function, $f_i^{(k)} : \mathbb{N}^k \to \mathbb{N}$, that $M_i$ "weakly computes":

- $\widehat{\Gamma} = \Gamma \cup \{\$\}$, where \$ is a new symbol (not already in $\Gamma$).
- $\widehat{M_i}$ simulates an execution of $M_i$, using the new symbol, "\$", to mark the left and right ends of the non-blank part of the tape (that is, a finite part of the tape that includes all cells storing non-blank symbols, as well as the cell that $M_i$'s tape head currently points to).
- If it is noted, during the simulation, that $M_i$ would halt, then $\widehat{M_i}$ enters a ***cleanup*** phase: The non-blank part of the tape is compressed, by repeatedly detecting the rightmost symbol, stored on this part of the tape, that does not belong to $\Sigma$ (if there is one) — and eliminating this, by shifting the symbols to the *right* of this one, on the non-blank part of the tape, over one position to the left.

  This eventually ends with the non-blank part of the tape storing

  $$\$\,\zeta\,\$$$

  for the string $\zeta \in \Sigma^\star$ that is to be returned as output. At this point the tape head can be swept to the right, to find and erase the right copy of "\$", and then sweep back to the left, to find and erase the left copy of "\$" — leaving the tape head on the copy of "$\sqcup$" that has replaced this symbol.

  Note that, at this point, $\widehat{M_i}$'s tape contents, and location of the tape head, are as required for a Turing machine that is computing a (partial or total) function.

Of course, $\widehat{M_i}$ is also a Turing machine with the form considered in this document — that is, it is $M_j$ for some other natural number $j$. This can be used to argue that the set of "Turing-computable" partial or total functions, is not changed when we consider "weak computation" instead of "strong computation".

## Enumerating Turing-Computable Functions

Suppose, now, that $f_i^{(k)}$ is the (partial or total) function from $\mathbb{N}^k$ to $N$ that is "weakly computed" by $M_i$, and "computed" (that is, "strongly computed") by $\widehat{M}_i$, for $i \in \mathbb{N}$. Then

$$f_0^{(k)}, f_1^{(k)}, f_2^{(k)}, \ldots$$

is an enumeration of all Turing-computable partial or total functions from $\mathbb{N}^k$ to $N$.

For $x_1, x_2, \ldots, x_k, y \in \mathbb{N}$, let $\widehat{\Phi}^{(k)} : \mathbb{N}^{k+1} \to \mathbb{N}$ be the partial or total function such that

$$\widehat{\Phi}^{(k)}(x_1, x_2, \ldots, x_k, y) = f_y^{(k)}(x_1, x_2, \ldots, x_k)$$

for all $x_1, x_2, \ldots, x_k, y \in \mathbb{N}$ — so that $\widehat{\Phi}^{(k)}$ is undefined at the inputs $x_1, x_2, \ldots, x_k, y \in \mathbb{N}$ if and only if the partial or total function $f_y^{(k)}$ is undefined at inputs $x_1, x_2, \ldots, x_k$. Then the function $\widehat{\Phi}^{(k)} : \mathbb{N}^{k+1} \to \mathbb{N}$, which has now been defined, is similar to the function $\Phi^{(k)} : \mathbb{N}^{k+1} \to \mathbb{N}$ introduced in the lecture notes — with the chief difference being that the first has been defined using an enumeration of $\mathcal{S}$-programs, while the second has been defined using an enumeration of Turing machines.

## A "Universality Theorem" for Turing Machines

**Theorem** (Universality Theorem for Turing Machines)**.** *For every positive integer $k$, the above partial or total function $\widehat{\Phi}^{(k)} : \mathbb{N}^{k+1} \to \mathbb{N}$ is Turing-computable.*

***Note:*** A proof of this is more complicated than you might expect! Requirements for the representations of inputs and outputs, for Turing machines that compute functions (as given in definitions concerning this) require quite a bit of pre-computation and post-computation to be introduced.

*Proof.* If one reorders the inputs, to define a partial or total function $\widetilde{\Phi}^{(k)} : \mathbb{N}^{k+1} \to \mathbb{N}$ such that

$$\widetilde{\Phi}^{(k)}(y, x_1, x_2, \ldots, x_k) = \widehat{\Phi}^{(k)}(x_1, x_2, \ldots, x_k, y)$$

for all $x_1, x_2, \ldots, x_k, y \in \mathbb{N}$, then the resulting partial or total function $\widetilde{\Phi}^{(k)}$ is the function computed by the ***universal Turing machine*** described in the supplemental document for this previous lecture — provided that this is modified to include the following.

- The computation should begin with each of the inputs $x_1, x_2, \ldots, x_k, y$ given as strings $\mu_1, \mu_2, \ldots, \mu_k, \nu \in \Sigma_{\mathsf{TM}}^\star$ such that $\#(\mu_i) = x_i$ for $1 \leq i \leq k$, and such that $\#(\nu) = y$ — using the encoding scheme from strings described in Lecture #5 where, once again, $\Sigma_{\mathsf{TM}}$ is the alphabet used to define this encoding.

Now, if the Turing machine, $M$, that is encoded by $y$ (and whose computation is to be simulated) has input alphabet $\Sigma$ and tape alphabet $\Gamma$ then the $i^{\text{th}}$ input for $M$ should be a string $\widehat{\mu}_i \in \Sigma^\star$ such that $\#(\widehat{\mu}_i) = x_i$ — where, here, the alphabet used to define the encoding is $M$'s input alphabet, $\Sigma$, instead.

However, the initial representation of the universal's tape should initially include a representation of $\widehat{\mu}_i$ as a string in $\Gamma^\star$, instead — where $\Gamma$ is $M$'s tape alphabet.

Thus a significantly extended "initialization" phase is needed before the universal Turing machine's simulation of steps can begin.

- If is detected that the simulated Turing machine is about to halt then the post-processing stage described, above, (when obtaining $\widehat{M_i}$ from $M_i$) must also be performed, so that a Turing machine that "strongly computes" a function is being described.
- Furthermore, the representation of the output integer as a string in $\Gamma^\star$ must be used to produce a different sting $\Sigma^\star_{\text{TM}}$ — that encodes the same output integer using the encoding scheme from Lecture #5 (with $\Sigma_{\text{TM}}$ as the alphabet used to define the encoding) — and this string should be stored on the universal Turing machine's tape, with the tape head located on a "$\sqcup$" immediately to the left of this string, when the execution ends.

The resulting, modified, universal Turing machine (strongly) computes the above function $\widetilde{\Phi}^{(k)}$ — as this is defined in the lecture notes — so that this function is certainly Turing-computable. Since the function $\widehat{\Phi}^{(k)}$ is obtained from the function $\widetilde{\Phi}^{(k)}$ by reordering the inputs (in a straightforward way), the function $\widehat{\Phi}^{(k)}$ must certainly be Turing-computable too. □

## Parameter Theorem

**Theorem** (Parameter Theorem)**.** *For all positive integers $n$ and $m$ there exists a primitive recursive function*

$$S^n_m : \mathbb{N}^{n+1} \to \mathbb{N}$$

*such that, for all $x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_n, y \in \mathbb{N}$,*

$$\Phi^{(m+n)}(x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_n, y) = \Phi^{(m)}(x_1, x_2, \ldots, x_m, S^n_m(u_1, u_2, \ldots, u_n, y)).$$

*Proof.* By induction on $n$.

*Basis:* Suppose that $n = 1$. Then $S^n_m = S^1_m$ should be a function $S^m_1 : \mathbb{N}^2 \to \mathbb{N}$ such that, for all $u_1, y \in \mathbb{N}$, $S^1_m(u_1, y)$ is the encoding of a program starting with $u_1$ copies of the statement

$$X_{m+1} \leftarrow X_{m+1} + 1$$

and continuing with the program encoded by $y$.

Recall how programs are encoded: If $\#(\mathcal{P}) = y$, for some program $\mathcal{P}$, then

$$y + 1 = \prod_{i=1}^{\mathsf{Lt}(y+1)} p_i^{(y+1)_i}$$

where $\mathsf{Lt}(y+1)$ is the number of instructions in the program $\mathcal{P}$, after trailing copies of the unlabelled dummy statement $Y \leftarrow Y$ are deleted — and where $(y+1)_i$ is the encoding of the $i^{\text{th}}$ instruction in $\mathcal{P}$ for $1 \leq i \leq \mathsf{Lt}(y+1)$.

Recall, as well, that the encoding of an instruction $I$ has the form

$$\langle a, \langle b, c \rangle \rangle$$

where

- $a$ gives information about $I$'s label, and is $0$ if this instruction *has* no label;
- $b$ indicates the instruction type, and is $1$ if this an increment statement; and
- $c+1$ is encoding of the variable mentioned in the statement — so that $c = 2(m+1)-1 = 2m+1$ if this variable is $X_{m+1}$.

Finally, recall that

$$\langle x, y \rangle = 2^x \cdot (2y+1) \dot{-} 1$$

for all $x, y \in \mathbb{N}$.

It follows that the instruction $X_{m+1} \to X_{m+1} + 1$ has the encoding

$$\begin{aligned}
\langle 0, \langle 1, 2m+1 \rangle \rangle &= 2^0 \cdot (2 \cdot \langle 1, 2m+1 \rangle + 1) \dot{-} 1 \\
&= 2 \cdot \langle 1, 2m+1 \rangle \\
&= 2 \cdot (2^1 \cdot (2 \cdot (2m+1) + 1) \dot{-} 1) \\
&= 2 \cdot (2 \cdot (4m+3) \dot{-} 1) \\
&= 2 \cdot (8m+5) \\
&= 16m + 10.
\end{aligned}$$

The encoding of a program $\mathcal{Q}$ that consists of $u_1$ copies of the statement $X_{m+1} \leftarrow X_{m+1} + 1$ is $\ell \dot{-} 1$ for

$$\ell = \prod_{i=1}^{u_1} p_i^{16m+10}.$$

It remains to notice that, for $1 \leq i \leq \mathsf{Lt}(y+1)$, the $i^{\text{th}}$ instruction of $\mathcal{P}$ becomes the $i + u_1^{\text{th}}$ instruction in the program obtained by including $u_1$ copies of the instruction $X_{m+1} \leftarrow X_{m+1}+1$ at the beginning.

This can be used to see that if

$$S_m^1(u_1, y) = \left( \prod_{i=1}^{u_1} p_i^{16m+10} \times \prod_{i=1}^{\mathsf{Lt}(y+1)} p_{u_1+i}^{(y+1)_i} \right) \dot{-} 1$$

then

$$\Phi^{(m+1)}(x_1, x_2, \ldots, x_n, u_1, y) = \Phi^{(m)}(x_1, x_2, \ldots, x_n, S_m^1(u_1, y))$$

Furthermore, the above function is primitive recursive, establishing the result claimed in the basis.

*Inductive Step:* Let $k$ be an integer such that $k \geq 1$. It is necessary, and sufficient, to use the following:

> Inductive Hypothesis: For every positive integer $m$ there exists a primitive recursive function $S_m^k : \mathbb{N}^{k+1} \to \mathbb{N}$ such that, for all $x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_k, y \in \mathbb{N}$,
>
> $$\Phi^{(m+k)}(x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_k, y) = \Phi^{(m)}(x_1, x_2, \ldots, x_m, S_m^k(u_1, u_2, \ldots, u_k, y))$$

to prove the following

> Inductive Claim: For every positive integer $m$ there exists a primitive recursive function $S_m^{k+1} : \mathbb{N}^{k+2} \to \mathbb{N}$ such that, for all $x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_{k+1}, y \in \mathbb{N}$,
>
> $$\Phi^{(m+k)}(x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_k, u_{k+1}, y) = \Phi^{(m)}(x_1, x_2, \ldots, x_m, S_m^{k+1}(u_1, u_2, \ldots, u_k, u_{k+1}, y)).$$

With that noted let $S_m^{k+1} : \mathbb{N}^{k+2} \to \mathbb{N}$ be the function such that, for all $u_1, u_2, \ldots, u_k, u_{k+1}, y \in \mathbb{N}$,

$$S_m^{k+1}(u_1, u_2, \ldots, u_k, u_{k+1}, y) = S_m^k(u_1, u_2, \ldots, u_k, S_{m+k}^1(u_{k+1}, y))$$

where $S_{m+k}^1 : \mathbb{N}^2 \to \mathbb{N}$ is the function whose existence and properties have been proved in the *basis*, and where $S_m^k : \mathbb{N}^{k+1} \to \mathbb{N}$ is the function whose existence and properties are consequences of the *inductive hypothesis*.

Then (once again) this function is primitive recursive, and

$$\begin{aligned}
&\Phi^{(m+k+1)}(x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_k, u_{k+1}, y) \\
&\quad = \Phi^{(m+k)}(x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_k, S_{m+k}^1(u_{k+1}, y)) \\
&\qquad\qquad \text{(by the results established in the basis)} \\
&= \Phi^{(m)}(x_1, x_2, \ldots, x_m, S_m^k(u_1, u_2, \ldots, u_k, S_{m+k}^1(u_{k+1}, y))) \quad \text{(by the inductive hypothesis)} \\
&= \Phi^{(m)}(x_1, x_2, \ldots, x_m, S_m^{k+1}(u_1, u_2, \ldots, u_k, u_{k+1}, y)) \qquad \text{(by the definition of } S_m^{k+1})
\end{aligned}$$

as needed to complete the inductive step, and the proof of the claim. □

## A Parameter Theorem for Turing Machine Computations

**Theorem** (Parameter Theorem for Turing Machines)**.** *For positive integers $n$ and $m$ there exists a primitive recursive function*

$$\widehat{S}^{\,n}_{\,m} : \mathbb{N}^{n+1} \to \mathbb{N}$$

*such that, for all $x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_n, y \in \mathbb{N}$,*

$$\widehat{\Phi}^{(n+m)}(x_1, x_2, \ldots, x_m, u_1, u_2, \ldots, u_n, y) = \widehat{\Phi}^{(m)}(x_1, x_2, \ldots, x_m, \widehat{S}^{\,n}_{\,m}(u_1, u_2, \ldots, u_n, y)).$$

*Sketch of Proof.* This result can also be proved by induction on $n$.

*Basis:* Suppose that $n = 1$. Then $\widehat{S}^n_m = \widehat{S}^1_m$ can be a function $\widehat{S}^1_m : \mathbb{N}^2 \to \mathbb{N}$ whose output on inputs $u_1, y \in \mathbb{N}$ is as follows.

- If $y$ is not a valid encoding of a Turing machine (so that it can be thought of as computing the constant function $0$), then it suffices to set $\widehat{S}^1_m(u_1, y)$ to be $y$ — since the value returned should also be a string that can be thought of as encoding a Turing machine that computes the constant function $0$.
- Suppose, instead, that $y$ is a valid encoding of a Turing machine

  $$M_y = (Q_y, \Sigma_y, \delta_y, q_{0,y}, q_{H,y})$$

  Then $S^1_m(u_1, y)$ can be the encoding of a Turing machine that carries out the following computation.

  (a) Move to the right past the first $m$ inputs, and a blank to the right of these (by scanning and moving right past $m + 1$ blanks, including the one the tape head initially rests on), without changing any tape symbols that it has seen, in the process).

  (b) Write the string in $\Sigma^*_y$ encoded by the input integer $u_1$, and another blank, on the cells to the right of this — resting on the blank that it has just written.

  (c) Scan back to the left past the string that has just been written, along with the first $m$ inputs (by scanning and moving left past $m + 2$ blanks) without changing any tape symbols that has seen in the process — ending back at the cell where the tape head was when the computation began.

  (d) Simulate the computation of the Turing machine encoded by $y$.

Then

$$\widehat{\Phi}^{(m+1)}(x_1, x_2, \ldots, x_m, u_1, y) = \widehat{\Phi}^{(m)}(x_1, x_2, \ldots, x_m, \widehat{S}^1_m(u_1, y))$$

for all $x_1, x_2, \ldots, x_n, u_1, y \in \mathbb{N}$, as required.

With effort it can be shown that the function $\widehat{S}^1_m$ is primitive recursive, as claimed.

The details of the *inductive step* are identical to those for the original version of the Parameter Theorem — with $\Phi^{(m+k)}$, $\Phi^{(m+k+1)}$, $S^k_m$ and $S^{k+1}_m$ replaced by $\widehat{\Phi}^{(m+k)}$, $\widehat{\Phi}^{(m+k+1)}$, $\widehat{S}^k_m$ and $\widehat{S}^{k+1}_m$, respectively. □

# Proofs of Results Concerning Computations on Sets

## Relationships Between Computable and Computably Enumerable Sets

**Claim 1.** *If a set $S \subseteq \mathbb{N}$ is* **computable** *then $S$ is also* **computably enumerable.**

*Proof.* Suppose that $S$ is computable. Then its characteristic function is a computable predicate — and there exists an $\mathcal{S}$-program (obtained from the following after macro expansion) that computes the same partial function as the following.

$$[A_1] \quad \text{IF } \neg(X_1 \in S) \text{ GOTO } A_1$$

This program computes a partial function $g : \mathbb{N} \to \mathbb{N}$ such that, for all $x \in \mathbb{N}$,

$$g(x) = \begin{cases} 0 & \text{if } x \in S \\ \uparrow & \text{if } x \notin S \end{cases}$$

Applying the definition of a "computably enumerable set," using this as the computable partial function in the definition, establishes that $S$ is computable enumerable — as needed to establish the claim. □

The proof of the next result uses a technique called ***dovetailing*** (sometimes also called "interleaving") which you might have seen already in CPSC 351 or 313.

**Claim 2.** *Let $S \subseteq \mathbb{N}$. If both $S$ and its complement $\overline{S}$ are computably enumerable, then the set $S$ is computable*

*Proof.* Suppose that $S \subseteq \mathbb{N}$ such that both $S$ and $\overline{S}$ are computably enumerable.

Since $S$ is computably enumerable, there exists a computable partial or total function

$$g_{\text{yes}} : \mathbb{N} \to \mathbb{N}$$

such that, for all $x \in \mathbb{N}$, $g_{\text{yes}}(x)$ is defined if and only if $x \in S$. Let $\mathcal{P}_{\text{yes}}$ be an $\mathcal{S}$-program that computes $g_{\text{yes}}$ — some such program must exist. Let $z_{\text{yes}} = \#(\mathcal{P}_{\text{yes}}) \in \mathbb{N}$.

Note that, for all $x \in \mathbb{N}$, $\mathcal{P}_{\text{yes}}$ halts when executed on input $x$ if and only if $x \in S$.

Similarly, since $\overline{S}$ is computably enumerable, there exists a computable partial or total function

$$g_{\text{no}} : \mathbb{N} \to \mathbb{N}$$

such that, for all $x \in \mathbb{N}$, $g_{\text{no}}(x)$ is defined if and only if $x \notin S$. Let $\mathcal{P}_{\text{no}}$ be an $\mathcal{S}$-program that computes $g_{\text{no}}$ — some such program must exist. Let $z_{\text{no}} = \#(\mathcal{P}_{\text{no}}) \in \mathbb{N}$.

Note that, for all $x \in \mathbb{N}$, $\mathcal{P}_{\text{no}}$ halts when executed on input $x$ if and only if $x \notin S$.

Recall the total predicate $\mathsf{Halt}_k : \mathbb{N}^{k+2} \to \{0,1\}$ (for a positive integer $k$) such that, for all $x_1, x_2, \ldots, x_k, y, z \in \mathbb{N}$, $\mathsf{Halt}_k(x_1, x_2, \ldots, x_k, y, z) = 1$ if and only if the program $\mathcal{P}$, such that $\#(\mathcal{P}) = y$, halts after executing at most $z$ instructions when it is executed on the inputs $x_1, x_2, \ldots, x_k$. This predicate is primitive recursive, so it is certainly computable — and can be used as a macro — particularly, when $k = 1$.

Dovetailing can now be used: There exists an $\mathcal{S}$-program that computes the same function as the following one.

$$
\begin{array}{ll}
 & Z_1 \leftarrow 0 \\
 & Z_4 \leftarrow z_{\mathsf{yes}} \\
 & Z_5 \leftarrow z_{\mathsf{no}} \\
[A_1] & Z_2 \leftarrow \mathsf{Halt}_1(X_1, Z_4, Z_1) \\
 & \mathsf{IF}\ Z_2 \neq 0\ \mathsf{GOTO}\ A_2 \\
 & Z_3 \leftarrow \mathsf{Halt}_1(X_1, Z_5, Z_1) \\
 & \mathsf{IF}\ Z_3 \neq 0\ \mathsf{GOTO}\ A_3 \\
 & Z_1 \leftarrow Z_1 + 1 \\
 & \mathsf{GOTO}\ A_1 \\
[A_2] & Y \leftarrow Y + 1
\end{array}
$$

Consider the execution of this program on an input $x \in \mathbb{N}$ — considering the cases $x \in S$ and $x \notin S$ separately.

*Case:* $x \in S$.

- Since $x \in S$ and $z_{\mathsf{no}} = \#(\mathcal{P}_{\mathsf{no}})$ the variable $Z_3$ always receives value $0$ when the sixth statement is executed — so there is never an attempt to go to a statement with label $A_3$.
- On the other hand, since $z_{\mathsf{yes}} = \#(\mathcal{P}_{\mathsf{yes}})$, $Z_2$ *does* eventually receive the value $1$. In particular, this happens the $k+1^{\mathsf{st}}$ time that the fourth statement is executed, where $k$ is the number of instructions executed by $\mathcal{P}_{\mathsf{yes}}$ on the input $x$.
- Since the label $A_2$ is followed after that, execution halts with the output value $1$ returned.

*Case:* $x \notin S$.

- Since $x \notin S$ and $z_{\mathsf{yes}} = \#(\mathcal{P}_{\mathsf{yes}})$ the variable $Z_2$ always receives value $0$ when the fourth statement is executed — so there is never an attempt to go to a statement with label $A_2$.
- On the other hand, since $z_{\mathsf{no}} = \#(\mathcal{P}_{\mathsf{no}})$, $Z_3$ *does* eventually receive the value $1$. In particular, this happens the $k+1^{\mathsf{st}}$ time that the sixth statement is executed, where $k$ is the number of instructions executed by $\mathcal{P}_{\mathsf{no}}$ on the input $x$.
- Since the label $A_3$ is followed after that, execution halts with the output value $0$ returned.

It follows that this program computes the total predicate $p_S : \mathbb{N} \to \{0, 1\}$ such that, for all $x \in \mathbb{N}$,

$$p_S(x) = \begin{cases} 1 & \text{if } x \in S, \\ 0 & \text{if } x \notin S. \end{cases}$$

Since $p_S$ is the characteristic function of $S$ it now follows that $S$ is recursive, as claimed. □

## Closure Properties for Computably Enumerable Sets

**Claim 4.** *If $S, T \subseteq \mathbb{N}$ are both computably enumerable sets, then so is $S \cap T$.*

*Proof.* Since $S$ is computably enumerable there exists a computable partial or total function $g_S : \mathbb{N} \to \mathbb{N}$ such that, for all $x \in \mathbb{N}$, $g_S(x)$ is defined if and only if $x \in S$. Similarly, since $T$ is computably enumerable there exists a computable partial or total function $g_T : \mathbb{N} \to \mathbb{N}$ such that, for all $x \in \mathbb{N}$, $g_T(x)$ is defined if and only $x \in T$.

Consider the $\mathcal{S}$-program corresponding to the following one (which uses macros to compute $g_S$ and $g_T$) as well as the partial or total function that it computes.

$$Z_1 \leftarrow g_S(X_1)$$
$$Z_2 \leftarrow g_T(X_1)$$

This computes the partial or total function $g : \mathbb{N} \to \mathbb{N}$ such that, for all $x \in \mathbb{N}$,

$$g(x) = \begin{cases} 0 & \text{if } x \in S \cap T \\ \uparrow & \text{otherwise.} \end{cases}$$

It follows by an application of the definition of a "computably enumerable set," using the above partial or total function $g$, that $S \cap T$ is computably enumerable. □

## Computably Enumerable Sets: Alternative Characterizations

**Claim 8.** *Let $S \subseteq \mathbb{N}$ be a computably enumerable set. Then there exists a primitive recursive predicate $R : \mathbb{N}^2 \to \{0, 1\}$ such that, for all $x \in \mathbb{N}$, $x \in S$ if and only if there exists a number $t \in \mathbb{N}$ such that $R(x, t) = 1$.*

*Proof.* Let $S$ be a nonempty computably enumerable set. Then, by the Enumeration Theorem, $S = W_n$ for some natural number $n$. Let $R$ be the predicate such that, for all $x, t \in \mathbb{N}$,

$$R(x, t) = \mathsf{Halt}_1(x, n, t).$$

Note that $n$ is ***constant***, here, depending only on $S$. Since $x \in S$ if and only if the program $\mathcal{P}$ encoded by $n$ halts when executed on input $x$, this is a primitive recursive predicate that satisfies the property given in the claim. □

**Claim 9.** *Let $S \subseteq \mathbb{N}$ be a **nonempty** computably enumerable set. Then there exists a primitive recursive function $f : \mathbb{N} \to \mathbb{N}$ such that*

$$S = \{f(n) \mid n \in \mathbb{N}\},$$

*that is, $S$ is the **range** of the primitive recursive function $f$.*

*Proof.* Let $S$ be a nonempty computably enumerable set — and consider the predicate recursive predicate $R : \mathbb{N}^2 \to \{0, 1\}$ whose existence is established by Claim 8. Since $S$ is nonempty there exists an element $x_0$ of $S$.

Consider the function $f : \mathbb{N} \to \mathbb{N}$ such that, for all $y \in \mathbb{N}$,

$$f(y) = \begin{cases} \ell(y) & \text{if } R(\ell(y), r(y)) = 1, \\ x_0 & \text{otherwise.} \end{cases}$$

This is a primitive recursive function of $y$, since $R$ is a primitive predicate and $x_0$ is a constant.

Suppose that $z \in S$.

- It follows by the choice of $R$ that there exists an element $t$ of $\mathbb{N}$ such that $R(z, t) = 1$.
- Then $f(\langle z, t\rangle) = z$, so that $z$ is in the range of $f$.

Suppose, on the other hand, that $z \notin S$.

- Notice that — by the definition of $f$ — either $f(\langle s, t\rangle) = s$ or $f(\langle s, t\rangle) = x_0$ for all $s, t \in \mathbb{N}$.
- Furthermore, $f(\langle s, t\rangle) = s$ ***only*** if either $R(s, t) = 1$ or $s = x_0$ — and every integer $n \in \mathbb{N}$ is equal to $\langle s, t\rangle$ for some pair $s, t \in \mathbb{N}$.
- It now follows by the choice of the predicate $R$ that $f(\langle z, y\rangle) \neq z$ for all $y \in \mathbb{N}$ — unless $z = x_0$: The integer $z$ cannot be in the range of $f$, otherwise.
- However, $x_0 \in S$ and $z \notin S$, so this is impossible too. Thus $z$ is not in the range of $f$ — as needed to establish the claim. □

**Claim 10.** *Let $f : \mathbb{N} \to \mathbb{N}$ be a computable partial or total function and let*

$$S = \{f(x) \mid x \in \mathbb{N} \textit{ and } f(x) \textit{ is defined}\},$$

*so that $S$ is the* **range** *of a Turing-computable partial or total function. Then $S$ is computably enumerable.*

*Proof.* Let $f$ be a Turing-computable partial or total function. Then there exists an $\mathcal{S}$-program $\mathcal{P}$ that computes this function. Let $p = \#(\mathcal{P})$.

The following program makes a more complicated use of ***dovetailing*** to compute some partial or total function $g : \mathbb{N} \to \mathbb{N}$ such that

$$S = \{x \in \mathbb{N} \mid g(x) \text{ is defined}\}$$

as needed to establish that $S$ is computably enumerable.

| | | |
|---|---|---|
| 1. | | $Z_4 \leftarrow p$ |
| 2. | $[A_1]$ | IF $\neg\text{Halt}_1(Z_1, Z_4, Z_2)$ GOTO $B_1$ |
| 3. | | $Z_3 \leftarrow f(Z_1)$ |
| 4. | | IF $Z_3 = X_1$ GOTO $E_1$ |
| 5. | $[B_1]$ | $Z_1 \leftarrow Z_1 + 1$ |
| 6. | | IF $Z_1 \leq Z_2$ GOTO $A_1$ |
| 7. | | $Z_2 \leftarrow Z_2 + 1$ |
| 8. | | $Z_1 \leftarrow 0$ |
| 9. | | GOTO $A_1$ |

***Note:*** The statement at line $3$ is only executed if it has already been confirmed, using the test at line $2$, that $f(Z_1)$ is defined. Thus every execution of the step at line $3$ halts.

Notice, as well, that this program is computing — and comparing to $X_1$ —

- the value computed by $\mathcal{P}$ on input $0$, if $\mathcal{P}$ halts immediately, followed by
- the values computed by $\mathcal{P}$ on inputs $0$ and $1$ if $\mathcal{P}$ halts after at most one step when executed on each input, followed by
- the values computed by $\mathcal{P}$ on inputs $0$, $1$ and $2$ if $\mathcal{P}$ halts after at most two steps when executed on each input...
- ... and so on.

Suppose that $y \in S$.

- Then $y = f(s)$ for some $s \in \mathbb{N}$, and — for some integer $k \geq 0$ — $\mathcal{P}$ will halt on the input $s$, returning $y$, after executing exactly $k$ instructions.
- When executed with input $X_1 = y$, this program will confirm that $f(s) = y$, halting (and returning the output $0$) after the $\max(s, k) + 1^{\text{st}}$ iteration of the process described above.
- Thus $g(y)$ is defined (and $g(y) = 0$), where $g$ is the partial or total function computed by this program.

Suppose, instead, that $y \notin S$.

- Then $y \neq f(s)$ for any integer $s \in \mathbb{N}$ and the test at line $4$ will always fail (when $X_1 = y$) — so that execution will never branch to the nonexistent statement with label $E_1$.
- An examination of the code confirms that this is the only way that the execution of this program can end.
- Thus $g(y)$ is undefined, where $g$ is the partial function computed by this program.

It has now been established that $g$ is a computable partial or total function such that

$$S = \{x \in \mathbb{N} \mid g(x) \text{ is defined}\},$$

establishing that $S$ is computably enumerable, as claimed. □