Modelling Computation of Functions III
Additional Macros, and Proofs of Claims

This document includes various macros for $\mathcal{S}$-programs that were mentioned in the lecture notes, but not give in detail. It also gives more information about the proof that every μ-recursive function is a computable function (as this is defined in this lecture). Proofs of claims are repetitive — and the completion of a proof is often left as an exercise in the use of techniques that have already been introduced.

Macros

Macro: $V \leftarrow c$

Let V be a variable and let c be a non-negative integer.

- If $c = 0$ then one can define a macro,

  $$\mathcal{P}_{V \leftarrow c},$$

  to be the same as the macro "$\mathcal{P}_{V \leftarrow 0}$" that is given as the first example of a macro in the lectures. That is, this should represent the instructions

  $$\begin{array}{ll} [L] & V \leftarrow V - 1 \\ & \text{IF } V \neq 0 \text{ GOTO } L \end{array}$$

  where L is a label that does not appear anywhere else in the larger program where this macro is to be used.

- On the other hand, if c is a *positive* integer, then one can define a macro, "$\mathcal{P}_{V \leftarrow c}$" to be a set of instructions that begins with the above ones (once again, where the label L is not used anywhere else in the larger program) and that continues with c copies of the statement

  $$V \leftarrow V + 1.$$

Thus, the macro "$\mathcal{P}_{V \leftarrow 2}$" is the sequence of statements

$$\begin{array}{ll} [L] & V \leftarrow V - 1 \\ & \text{IF } V \neq 0 \text{ GOTO } L \\ & V \leftarrow V + 1 \\ & V \leftarrow V + 1 \end{array}$$

for L as above.

- One can prove, by induction on c, that an execution of this subprogram always halts — and that, on its termination, the variable V's value has to been set to c, with the values of no other variables having been changed.
- In each case, the macro corresponds to a "new statement in the programming language" (that is, "pseudo-instruction")

$$V \leftarrow c.$$

Macro: GOTO L

Let Z be a local variable that is not used anywhere else (in the program being developed). Then a macro $\mathcal{P}_{\text{GOTO } L}$, for a label L, is as follows.

$$\begin{array}{l} Z \leftarrow 1 \\ \text{IF } Z \neq 0 \text{ GOTO } L \end{array}$$

- Applying "macro expansion" one would obtain a subprogram

$$\begin{array}{ll} [\widehat{L}\,] & Z \leftarrow Z - 1 \\ & \text{IF } Z \neq 0 \text{ GOTO } \widehat{L} \\ & Z \leftarrow Z + 1 \\ & \text{IF } Z \neq 0 \text{ GOTO } L \end{array}$$

  where $\widehat{L}$ is a label that is not included in the larger program where this subprogram will be used — including the pseudo-instruction being replaced, so that it is different from the label L.
- This implements an ***unconditional*** branch to the first statement with label L — or unconditionally terminates the execution of the entire program if there is no such statement. Thus one can use the corresponding "pseudo-instruction"

$$\text{GOTO } L$$

  when developing an $\mathcal{S}$-program.

Macro: STOP

As noted above, the execution of an unconditional branch, to a label that is not used anywhere else in the program, terminates the program's execution. Thus if Z is a local variable that is not used anywhere else in the program, and $\widehat{L}$ and $\widetilde{L}$ are distinct labels that are not used anywhere else in the program, then the execution of the subprogram

$$
\begin{array}{ll}
[\widehat{L}\,] & Z \leftarrow Z - 1 \\
 & \text{IF } Z \neq 0 \text{ GOTO } \widehat{L} \\
 & Z \leftarrow Z + 1 \\
 & \text{IF } Z \neq 0 \text{ GOTO } \widetilde{L}
\end{array}
$$

— which is the same as the macro corresponding to "GOTO $\widetilde{L}$" — always causes the program's execution to end. Thus this subprogram can be used as a macro that corresponds to the pseudo-instruction

STOP

Macro: $W \leftarrow X$

Suppose that W and X are distinct variables. In order to write a program $P_{W \leftarrow X}$, which sets the value of W to be the current value of X — without changing the values of any variables (used in the larger program) besides W — let Z be a local variable that is not the same as either W and X and not used anywhere else in any larger program being developed, and let A, B, C and D be distinct labels that are not used anywhere else in this program. A macro $\mathcal{P}_{W \leftarrow X}$, which sets the value of W to be the current value of X, is as follows.

$$
\begin{array}{ll}
 & W \leftarrow 0 \\
 & Z \leftarrow 0 \\
 & \text{IF } X \neq 0 \text{ GOTO } A \\
 & \text{GOTO } B \\
[A] & X \leftarrow X - 1 \\
 & W \leftarrow W + 1 \\
 & Z \leftarrow Z + 1 \\
 & \text{IF } X \neq 0 \text{ GOTO } A \\
[B] & \text{IF } Z \neq 0 \text{ GOTO } C \\
 & \text{GOTO } D \\
[C] & Z \leftarrow Z - 1 \\
 & X \leftarrow X + 1 \\
 & \text{IF } Z \neq 0 \text{ GOTO } C
\end{array}
$$

As described in the lecture notes, the label D should be used to direct execution to statement immediately after this one in the larger program, if one exists.

Correctness of this macro can be established by completing the following ***exercises***.

(a) Confirm that if this program is executed when the initial value of X is 0 then the program halts and, on termination, X, W and Z all have value 0. Note that the values of no other variables have changed (because none are referenced or modified in any of the above statements).

(b) Confirm that if this program is executed when the initial value of X is a positive integer n, then the statement with label A is executed exactly n times — and that, for $1 \leq m \leq n$, the variables X, W and Z have values $n-m+1$, $m-1$ and $m-1$, respectively, immediately ***before*** the m^{th} execution of this statement.

(c) Using the above, confirm that if this program is executed when the initial value of X is a positive integer n, then the statement with label C is also executed exactly n times — and that, for $1 \leq m \leq n$, the variables X, W and Z have values $m-1$, n and $n-m+1$, respectively, immediately ***before*** the m^{th} execution of this statement.

(d) Using the above, confirm that if this program is executed when the initial value of X is a positive integer n, then the program eventually halts. On termination, the variables X, W and Z have values n, n and 0, respectively. The values of no other variables have changed.

Thus this program corresponds to the pseudo-instruction"

$$W \leftarrow X$$

Macro: $W \leftarrow X + Y$

Suppose that W, X and Y are distinct variables. Suppose, as well, that Z is local variable that is not W, X or Y and that is not used anywhere else in the program being developed. Let A, B, C and D be distinct labels that are also not used anywhere else. A macro $\mathcal{P}_{W \leftarrow X+Y}$, which sets the value of W to be the sum of the current values of X and Y — without changing the values of X, Y, or any other variable appearing elsewhere in the larger program that will include this one, is as follows.

$W \leftarrow X$
$Z \leftarrow 0$
IF $Y \neq 0$ GOTO A
GOTO B
$[A]$ $Y \leftarrow Y - 1$
$W \leftarrow W + 1$
$Z \leftarrow Z + 1$
IF $Y \neq 0$ GOTO A
$[B]$ IF $Z \neq 0$ GOTO C
GOTO D
$[C]$ $Z \leftarrow Z - 1$
$Y \leftarrow Y + 1$
IF $Z \neq 0$ GOTO C

Exercise: After studying the analysis of the "assignment statement" program and tracing execution of the addition program, as needed, modify the analysis of the "assignment" macro, $\mathcal{P}_{W \leftarrow X}$, to prove the following: If the variables X and Y have initial values n and m, respectively, for nonnegative integers n and m, and the above "addition" program is executed, then this execution eventually halts.

On termination of this execution the variables W, X, Y and Z have values $n + m$, n, m, and 0, respectively. The values of no other variables have been changed by the execution of this subprogram.

Pseudo-Instructions, and Macros, for $\mathcal{S}$-Computable Functions

Set k be a positive integer and let $f : \mathbb{N}^k \to \mathbb{N}$ be an $\mathcal{S}$-computable partial or total function — so that there exists an $\mathcal{S}$-program, $\mathcal{Q}$, which computes f (as this is defined in the lecture notes). Let $W, V_1, V_2, \ldots, V_k$ be distinct local variables.

As described below, it is possible to use $\mathcal{Q}$ to produce a macro $\mathcal{P}_{W \leftarrow f(V_1, V_2, \ldots, V_k)}$, and a corresponding pseudo-instruction

$$W \leftarrow f(V_1, V_2, \ldots, V_k)$$

The executions of these should satisfy the following properties.

- Let $x_1, x_2, \ldots, x_k \in \mathbb{N}$ such that $f(x_1, x_2, \ldots, x_k) \in \mathbb{N}$ — that is, $f(x_1, x_2, \ldots, x_k)$ is defined. If the subprogram $\mathcal{P}_{W \leftarrow f(V_1, V_2, \ldots, V_k)}$ is executed when V_i has initial value x_i, for $1 \leq i \leq k$, then the execution of this subprogram eventually ends. When if does, W has value $f(x_1, x_2, \ldots, x_k)$ and V_i has value x_i, once again, for every integer i such that $1 \leq i \leq k$.

- Let $x_1, x_2, \ldots, x_k \in \mathbb{N}$ such that $f(x_1, x_2, \ldots, x_k) = \uparrow$ — that is, $f(x_1, x_2, \ldots, x_k)$ is undefined. If the subprogram $\mathcal{P}_{W \leftarrow f(V_1, V_2, \ldots, V_k)}$ is executed when V_i has initial value x_i, for $1 \leq i \leq k$, then this execution of the subprogram does not ever end.

In order to see that this case, let $\ell \geq 0$ and let

$$T_1, T_2, \ldots, T_\ell$$

be all the variables, besides $X_1, X_2, \ldots, X_k$ and Y, which appear in the $\mathcal{S}$-program $\mathcal{Q}$. If any of these is an input variable X_j, where $j \geq k+1$, then this variable can be replaced with a local variable Z_h that is currently unused in $\mathcal{Q}$ without changing the execution of this program on any inputs $x_1, x_2, \ldots, x_k \in \mathbb{N}$. Thus the resulting program still computes the function f — and we may now assume, without loss of generality, that $T_1, T_2, \ldots, T_\ell$ are all *local variables*.

Now let

$$U_1, U_2, \ldots, U_k$$

be local variables that are not equal to any of W, $V_1, V_2, \ldots, V_k$, or $T_1, T_2, \ldots, T_\ell$. A subprogram, $\mathcal{P}_{W \leftarrow f(V_1, V_2, \ldots, V_k)}$ — which uses local variables $T_1, T_2, \ldots, T_\ell$ and $U_1, U_2, \ldots, U_k$ — can have the following form.

- The subprogram can begin with the following block of pseudo-instructions:

$$\begin{array}{l} U_1 \leftarrow V_1 \\ U_2 \leftarrow V_2 \\ \quad \vdots \\ U_k \leftarrow V_k \\ T_1 \leftarrow 0 \\ T_2 \leftarrow 0 \\ \quad \vdots \\ T_\ell \leftarrow 0 \\ W \leftarrow 0 \end{array}$$

  This ensures that W and $T_1, T_2, \ldots, T_\ell$ have value 0, as Y and $T_1, T_2, \ldots, T_\ell$ do when an execution of the program $\mathcal{Q}$ would begin. It also ensures that U_i has the initial value of V_i, for $1 \leq i \leq k$.

- The subprogram continues with a copy of $\mathcal{Q}$ — with Y replaced by W and with X_i replaced by V_i for $1 \leq i \leq k$. Thus, if V_i has initial value $x_i \in \mathbb{N}$, for $1 \leq i \leq k$, and $f(x_1, x_2, \ldots, x_k)$ is defined, then the corresponding execution of this part of subprogram would eventually end (since an execution of $\mathcal{Q}$ on inputs $x_1, x_2, \ldots, x_k$ would), and W has value $f(x_1, x_2, \ldots, x_k)$ at this point. On the other hand, if $f(x_1, x_2, \ldots, x_k)$ is undefined then the corresponding execution of this part of the subprogram would never end (since an execution of $\mathcal{Q}$ with inputs $x_1, x_2, \ldots, x_k$ would not end, either).

- Finally, the subprogram ends with the following block of pseudo-instructions:

$$\begin{aligned} V_1 &\leftarrow U_1 \\ V_2 &\leftarrow U_2 \\ &\vdots \\ V_k &\leftarrow U_k \end{aligned}$$

  Since $\mathcal{Q}$ did not include any of the local variables $U_1, U_2, \ldots, U_k$ the middle part of this subprogram does not include any of these variables either — and the values of these variables have not been changed by the execution of the middle part of this subprogram. Thus the execution of this final part of the subprogram restores $V_1, V_2, \ldots, V_k$ to their initial values (without changing the value of W).

Recall that an application of the "macro expansion" process ensures — when the above subprogram is used as a macro — that $U_1, U_2, \ldots, U_k$ and $T_1, T_2, \ldots, T_\ell$ are each replaced (when necessary) with distinct local variables that do not appear anywhere else in the larger program. Furthermore — since executions of the first and third blocks of instructions, above, always halt, an execution of this subprogram when $V_i = x_i \in \mathbb{N}$, for $1 \leq i \leq k$, halts if and only if $f(x_1, x_2, \ldots, x_k)$ is defined. Thus this subprogram can be used as a macro, corresponding to a pseudo-instruction

$$W \leftarrow f(V_1, V_2, \ldots, V_k)$$

as claimed.

Proving That Every Primitive Recursive Function is $\mathcal{S}$-Computable

Proofs of claims in the lecture notes, concerning primitive recursive functions, are sketched below.

Claim 4. *Each of the* **initial functions** *is $\mathcal{S}$-computable.*

Proof. Let $f : \mathbb{N}^k \to \mathbb{N}$ be an initial function. Then one of the following cases must apply.

(a) f is the *successor function*. That is, $k = 1$ and f is the function $s : \mathbb{N} \to \mathbb{N}$ such that $s(x) = x + 1$ for all $x \in \mathbb{N}$.

(b) f is the *nullify function*. That is, $k = 1$ and f is the function $n : \mathbb{N} \to \mathbb{N}$ such that $n(x) = 0$ for all $x \in \mathbb{N}$.

(c) f is a *projection function*. That is, there exists an integer i such that $1 \leq i \leq k$ and f is the function $u_i^k : \mathbb{N}^k \to \mathbb{N}$ such that $u_i^k(x_1, x_2, \ldots, x_k) = x_i$ for all $x_1, x_2, \ldots, x_k \in \mathbb{N}$.

Each of these cases is considered separately, below.

(a) Suppose, first, that f is the *successor function* — so that $k = 1$ and f is the function $s : \mathbb{N} \to \mathbb{N}$ such that $s(x) = x + 1$ for all $x \in \mathbb{N}$.

Since Y and X_1 are distinct variables, the pseudo-instruction "$Y \leftarrow X_1$" can be used to set the value of the output variable, Y to be the value of the first input variable, X_1. The $\mathcal{S}$-program obtained by starting with the following program,

$$Y \leftarrow X_1$$
$$Y \leftarrow Y + 1$$

and applying macro expansion, computes the successor function, as needed to establish the claim in this case.

(b) Suppose, next, that f is the *nullify function* — so that $k = 1$ and f is the function $n : \mathbb{N} \to \mathbb{N}$ such that $n(x) = 0$ for all $x \in \mathbb{N}$.

The $\mathcal{S}$-program obtained by starting with the following program,

$$Y \leftarrow 0$$

and applying macro expansion, computes the nullify function, as needed to establish the claim in this case as well.

(c) Finally, suppose that f is a *projection function* — so that there exists an integer i such that $1 \leq i \leq k$ and f is the function $u_i^k : \mathbb{N}^k \to \mathbb{N}$ such that $u_i^k(x_1, x_2, \ldots, x_k) = x_i$ for all $x_1, x_2, \ldots, x_k \in \mathbb{N}$.

Once again, Y and X_i are distinct variables, so that the pseudo-instruction "$Y \leftarrow X_i$" can be used to set the value of the output variable, Y, to be the value of the i^{th} input variable, X_i. The $\mathcal{S}$-program obtained by starting with the following program,

$$Y \leftarrow X_i$$

and applying macro expansion, computes the above projection function, as needed to establish the claim in this case too.

Since the desired result has been proved for every case, this establishes the claim. □

Claim 5. *If k and n are positive integers, $f : \mathbb{N}^k \to \mathbb{N}$ is an $\mathcal{S}$-computable partial or total function, and $g_i : \mathbb{N}^n \to \mathbb{N}$ is an $\mathcal{S}$-computable partial or total function, for every integer i such that $1 \leq i \leq k$, then the function $h : \mathbb{N}^n \to \mathbb{N}$, obtained from f and $g_1, g_2, \ldots, g_k$ by composition, is an $\mathcal{S}$-computable partial or total function as well.*

Proof. Since the partial or total function $g_i : \mathbb{N}^n \to \mathbb{N}$ is $\mathcal{S}$-computable, there exists an $\mathcal{S}$-program, $\mathcal{P}_i$, that computes g_i, for every integer i such that $1 \leq i \leq k$. Furthermore, if variables $W, V_1, V_2, \ldots, V_n$ are distinct, then $\mathcal{P}_i$ can be used to produce a subprogram, "$\mathcal{P}_{W \leftarrow g_i(V_1, V_2, \ldots, V_n)}$," — with a corresponding pseudo-instruction

$$W \leftarrow g_i(V_1, V_2, \ldots, V_n)$$

— which satisfies the following properties:

- For all $x_1, x_2, \ldots, x_n \in \mathbb{N}$ such that $g_i(x_1, x_2, \ldots, x_n)$ is defined, if variable V_i has value x_i for $1 \leq i \leq n$, then an execution of the pseudo-instruction

  $$W \leftarrow g_i(V_1, V_2, \ldots, V_n)$$

  eventually ends, with the value of W set to $g_i(x_1, x_2, \ldots, x_n)$ and with the values of all other variables (used in the program containing this pseudo-instruction) unchanged.

- For all $x_1, x_2, \ldots, x_n \in \mathbb{N}$ such that $g_i(x_1, x_2, \ldots, x_n)$ is undefined, an execution of the pseudo-instruction

  $$W \leftarrow g_i(V_1, V_2, \ldots, V_n)$$

  does not ever end, at all.

Similarly, since the partial or total function $f : \mathbb{N}^k \to \mathbb{N}$ is $\mathcal{S}$-computable, there exists an $\mathcal{S}$-program, $\mathcal{Q}$, that computes f. Furthermore, if variables $W, V_1, V_2, \ldots, V_k$ are distinct, then $\mathcal{Q}$ can be used to produce a subprogram, "$\mathcal{P}_{W \leftarrow f(V_1, V_2, \ldots, V_k)}$" — with a corresponding pseudo-instruction

$$W \leftarrow f(V_1, V_2, \ldots, V_k)$$

— which satisfies the following properties.

- For all $y_1, y_2, \ldots, y_k \in \mathbb{N}$ such that $f(y_1, y_2, \ldots, y_k)$ is defined, if variable V_i has value y_i for $1 \leq i \leq k$, then an execution of the pseudo-instruction

  $$W \leftarrow f(V_1, V_2, \ldots, V_k)$$

  eventually ends, with the value of W set to $f(y_1, y_2, \ldots, y_k)$ and with the values of all other variables (used in the program containing this pseudo-instruction) unchanged.

- For all $y_1, y_2, \ldots, y_k \in \mathbb{N}$ such that $f(y_1, y_2, \ldots, y_k)$ is undefined, an execution of the pseudo-instruction

  $$W \leftarrow f(V_1, V_2, \ldots, V_k)$$

  does not ever end, at all.

Now consider the following program.

$$\begin{array}{ll} 1 & Z_1 \leftarrow g_1(X_1, X_2, \ldots, X_n) \\ 2 & Z_2 \leftarrow g_2(X_1, X_2, \ldots, X_n) \\ & \vdots \\ k & Z_k \leftarrow g_k(X_1, X_2, \ldots, X_n) \\ k+1 & Y \leftarrow f(Z_1, Z_2, \ldots, Z_k) \end{array}$$

- If $h : \mathbb{N}^n \to \mathbb{N}$ is the function obtained from f and $g_1, g_2, \ldots, g_k$, and $x_1, x_2, \ldots, x_n \in \mathbb{N}$ such that at least one of $g_1(x_1, x_2, \ldots, x_n), g_2(x_1, x_2, \ldots, x_n), \ldots, g_k(x_1, x_2, \ldots, x_n)$ is undefined, then $h(x_1, x_2, \ldots, x_n)$ is undefined as well — and an execution of an $\mathcal{S}$-program computing h should not halt when the program is executed with inputs $x_1, x_2, \ldots, x_n$. The above $\mathcal{S}$-program does not halt when executed with these inputs either — for if i is the smallest integer such that $1 \leq i \leq k$ and $g_i(x_1, x_2, \ldots, x_n)$ is undefined, then (when executed with these inputs) the i^{th} statement in the above $\mathcal{S}$-program is reached and executed, but the execution of this statement does not halt.

- Let h be as above, and let $x_1, x_2, \ldots, x_n \in \mathbb{N}$ such that $g_i(x_1, x_2, \ldots, x_n)$ is defined — so that $g_i(x_1, x_2, \ldots, x_n) = y_i$ for a number $y_i \in \mathbb{N}$, for every integer i such that $1 \leq i \leq k$. If $f(y_1, y_2, \ldots, y_k)$ is undefined then $h(x_1, x_2, \ldots, x_n)$ is undefined as well, so that an $\mathcal{S}$-program computing h should not halt when it is executed with inputs $x_1, x_2, \ldots, x_n$. The above $\mathcal{S}$-program does not halt when executed with these inputs either — for the first k statements are reached and executed, with Z_i set to have value y_i for $1 \leq i \leq k$, but the execution of the $k + 1^{\text{st}}$ statement is an attempt to compute $f(y_1, y_2, \ldots, y_k)$, so that the execution of this statement does not halt.

- Once again, let h be as above, and let $x_1, x_2, \ldots, x_n \in \mathbb{N}$ such that $g_i(x_1, x_2, \ldots, x_n)$ is defined — so that $g_i(x_1, x_2, \ldots, x_n) = y_i$ for a number $y_i \in \mathbb{N}$, for every integer i such that $1 \leq i \leq k$. If $f(y_1, y_2, \ldots, y_k)$ is defined then $h(x_1, x_2, \ldots, x_n)$ is defined as well and is equal to $f(y_1, y_2, \ldots, y_k)$, so that an $\mathcal{S}$-program computing h should halt, returning $f(y_1, y_2, \ldots, y_k)$, when it is executed with inputs $x_1, x_2, \ldots, x_n$. Since each of the first k statements is reached and executed, and the execution halts — with Z_i set to have value y_i for $1 \leq i \leq k$ — one can see that the execution of this program on inputs $x_1, x_2, \ldots, x_n$ halts as well, with Z set to be the $h(x_1, x_2, \ldots, x_n)$ on termination, as required.

Thus the program obtained from this one, after macro expansion, is an $\mathcal{S}$-program computing the function h — and h is $\mathcal{S}$-computable, as claimed. □

Claim 6.

(a) Let k be a fixed nonnegative integer and let $g : \mathbb{N}^2 \to \mathbb{N}$ be a partial or total function. If g is $\mathcal{S}$-computable then the function partial or total function $h : \mathbb{N} \to \mathbb{N}$, defined from k and f using the first form of primitive recursion, is an $\mathcal{S}$-computable function as well.

(b) Let n be an integer such that $n \geq 1$ and let $f : \mathbb{N}^n \to \mathbb{N}$ and $g : \mathbb{N}^{n+2} \to \mathbb{N}$ be partial or total functions. If f and g are $\mathcal{S}$-computable then the partial or total function $h : \mathbb{N}^{n+1} \to \mathbb{N}$, defined from f and g using the second form of primitive recursive, is an $\mathcal{S}$-computable function as well.

Proof. To begin, consider part (b) of the claim. Let n be a positive integer and let $f : \mathbb{N}^n \to \mathbb{N}$ and $g : \mathbb{N}^{n+2} \to \mathbb{N}$ be partial or total functions that are both $\mathcal{S}$-computable.

- Since the partial or total function $f : \mathbb{N}^n \to \mathbb{N}$ is $\mathcal{S}$-computable, there exists an $\mathcal{S}$-program, $\mathcal{P}$, that computes f. Furthermore, if variables $W, V_1, V_2, \ldots, V_n$ are distinct, then $\mathcal{P}$ can be used to produce a subprogram, "$\mathcal{P}_{W \leftarrow f(V_1, V_2, \ldots, V_n)}$" — with a corresponding pseudo-instruction
$$W \leftarrow f(V_1, V_2, \ldots, V_n)$$
— which satisfies the following properties.
  - For all $y_1, y_2, \ldots, y_n \in \mathbb{N}$ such that $f(y_1, y_2, \ldots, y_n)$ is defined, if variable V_i has value y_i for $1 \leq i \leq n$, then an execution of the pseudo-instruction
$$W \leftarrow f(V_1, V_2, \ldots, V_n)$$
eventually ends, with the value of W set to $f(y_1, y_2, \ldots, y_n)$ and with the values of all other variables (used in the program containing this pseudo-instruction) unchanged.
  - For all $y_1, y_2, \ldots, y_n \in \mathbb{N}$ such that $f(y_1, y_2, \ldots, y_n)$ is undefined, an execution of the pseudo-instruction
$$W \leftarrow f(V_1, V_2, \ldots, V_n)$$
does not ever end, at all.
- Since the partial or total function $g : \mathbb{N}^{n+2} \to \mathbb{N}$ is $\mathcal{S}$-computable, there exists an $\mathcal{S}$-program, $\mathcal{Q}$, that computes g. Furthermore, if variables $W, V_1, V_2, \ldots, V_{n+2}$ are distinct, then $\mathcal{Q}$ can be used to produce a subprogram, "$\mathcal{P}_{W \leftarrow g(V_1, V_2, \ldots, V_{n+2})}$" — with a corresponding pseudo-instruction
$$W \leftarrow g(V_1, V_2, \ldots, V_{n+2})$$
— which satisfies the following properties.

– For all $y_1, y_2, \ldots, y_{n+2} \in \mathbb{N}$ such that $g(y_1, y_2, \ldots, y_{n+2})$ is defined, if variable V_i has value y_i for $1 \leq i \leq n+2$, then an execution of the pseudo-instruction

$$W \leftarrow g(V_1, V_2, \ldots, V_{n+2})$$

eventually ends, with the value of W set to $g(y_1, y_2, \ldots, y_{n+2})$ and with the values of all other variables (used in the program containing this pseudo-instruction) unchanged.

– For all $y_1, y_2, \ldots, y_{n+2} \in \mathbb{N}$ such that $g(y_1, y_2, \ldots, y_{n+2})$ is undefined, an execution of the pseudo-instruction

$$W \leftarrow g(V_1, V_2, \ldots, V_{n+2})$$

does not ever end, at all.

Now consider the following program.

1.		$Y \leftarrow f(X_1, X_2, \ldots, X_n)$
2.		$Z_1 \leftarrow 0$
3.	$[A]$	IF $X_{n+1} \neq 0$ GOTO B
4.		GOTO C
5.	$[B]$	$Z_2 \leftarrow g(Z_1, Y, X_1, X_2, \ldots, X_n)$
6.		$Y \leftarrow Z_2$
7.		$Z_1 \leftarrow Z_1 + 1$
8.		$X_{n+1} \leftarrow X_{n+1} - 1$
9.		GOTO A

Consider an execution of this program on inputs $x_1, x_2, \ldots, x_{n+1} \in \mathbb{N}$.

Suppose, first, that $x_{n+1} = 0$.

- If $f(x_1, x_2, \ldots, x_n)$ is defined then $h(x_1, x_2, \ldots, x_n, 0)$ is defined as well, and

$$h(x_1, x_2, \ldots, x_n, 0) = f(x_1, x_2, \ldots, x_n, 0).$$

Now, the variable Y is set to have the desired output value when the step at line 1 is executed (and, the execution of this step eventually ends). Z_1 is set to be 0 when the step at line 2 is executed (and the execution of this step ends too), Since $X_{n+1} = 0$ the step at line 4 is reached and executed, after the step at line 3 and, since C is not the label of any instruction in the program, this execution of the program ends — with $Y = h(x_1, x_2, \ldots, x_n, 0)$, as required.

- On the other hand, if $f(x_1, x_2, \ldots, x_n)$ is undefined then the execution of this program fails to end (as needed, here), since the execution of the step at line 1 does not end.

Suppose, next, that $x_{n+1} > 0$. In particular, let $x_{n+1} = k$ for a positive integer k. Three cases might arise:

- *Case:* $f(x_1, x_2, \ldots, x_n, 0)$ is undefined.

  In this case $f(x_1, x_2, \ldots, x_n, i)$ is also undefined for every positive integer i as well. In particular, $f(x_1, x_2, \ldots, x_n, k)$ is undefined, so that an $\mathcal{S}$-program computing the function h should fail to halt, when executed on inputs $x_1, x_2, \ldots, x_n, x_{n+1}$ (with $x_{n+1} = k$).

  Now consider the execution of the above program on inputs $x_1, x_2, \ldots, x_n, k$. The execution begins with an attempt to compute $f(x_1, x_2, \ldots, x_n)$ at line 1. Since this value is undefined the execution of this step never ends — so that this execution of this program never ends, as required for this case.

- *Case:* $f(x_1, x_2, \ldots, x_n, 0)$ is defined — so that $h(x_1, x_2, \ldots, x_n, 0)$ is defined as well — and there exists an integer i such that $1 \leq i \leq k$, $h(x_1, x_2, \ldots, x_n, j)$ is defined for every integer j such that $0 \leq j \leq i - 1$, and $h(x_1, x_2, \ldots, x_n, i)$ is undefined.

  In this case $f(x_1, x_2, \ldots, x_n, \ell)$ is undefined, as well, for every integer ℓ such that $\ell \geq i$, so that $f(x_1, x_2, \ldots, x_n, k)$ is undefined. Once again, an $\mathcal{S}$-program computing the function h should fail to halt, when executed on inputs $x_1, x_2, \ldots, x_{n+1}$ (with $x_{n+1} = k$).

  Now consider the execution of the above program on inputs $x_1, x_2, \ldots, x_n, k$. The execution begins with an attempt to compute $f(x_1, x_2, \ldots, x_n)$ at line 1. Since this value is defined the execution of this step ends, and Y has value $f(x_1, x_2, \ldots, x_n) = h(x_1, x_2, \ldots, x_n, 0)$ at this point. The variable Z_1 receives value 0 when the step at line 2 is executed. Since X_{n+1} has positive value k when the step at line 3 is reached, the execution of this step causes a branch to the step at line 5.

  Since $h(x_1, x_2, \ldots, x_n, j)$ is defined for every integer j such that $0 \leq j \leq i - 1$, the following can be established by induction on j: If $j \in \mathbb{N}$ such that $0 \leq j \leq i - 1$ then the statement (at line 3) with label A is reached at least $j + 1$ times — and the following conditions are satisfied when this statement is reached for the $j + 1^{\text{st}}$ time:

  (i) Z_1 has value j both immediately before and immediately after this execution of the statement with label A.

  (ii) X_{n+1} has value $k - j$ — a positive integer — both immediately before and immediately after this execution of the statement with label A.

  (iii) Y has value $h(x_1, x_2, \ldots, x_n, j)$ both immediately before and immediately after this execution of the statement with label A.

  Since the value, $k - j$, of X_{n+1} is a positive integer, the test at line 3 (included in this execution of the statement) is passed — so that the execution continues with the step (at line 5) with label B. When this step is executed, an attempt is made to assign Z_2 the

value

$$g(j, h(x_1, x_2, \ldots, x_n, j), x_1, x_2, \ldots, x_n) = h(x_1, x_2, \ldots, x_n, j+1). \quad (1)$$

Now, if $j \leq i-2$, then $j+1 \leq i-1$ and the above value is defined — so that this execution of the step with label B halts, with Z_2 receiving this value. The steps at lines 6, 7 and 8 are then executed, so that Y receives value $h(x_1, x_2, \ldots, x_n, j+1)$, Z_1 receives value $j+1$, and X_{n+1} receives value $(k-j)-1 = k-(j+1)$ — so that the above properties are satisfied with j replaced by $j+1$ when the step at line 9 is reached and executed, and the step with label A is recached once again.

On the other hand, if $j = i-1$ the the value at line (1) is not defined and this execution of the label B never ends — so that this execution of the program never ends, as required for this case as well.

- *Case:* $f(x_1, x_2, \ldots, x_n, 0)$ is defined — so that $h(x_1, x_2, \ldots, x_n, 0)$ is defined as well — and $h(x_1, x_2, \ldots, x_n, i)$ is defined for every integer i such that $1 \leq i \leq k$. In particular, $h(x_1, x_2, \ldots, x_n, k)$ is defined, and an $\mathcal{S}$-program should halt, with Y equal to $h(x_1, x_2, \ldots, x_n, k)$, when executed on inputs $x_1, x_2, \ldots, x_{n+1}$ (with $x_{n+1} = k$).

  Now consider the execution of the above program on inputs $x_1, x_2, \ldots, x_n, k$. Once again, the execution begins with an attempt to compute $f(x_1, x_2, \ldots, x_n)$ at line 1. Since this value is defined the execution of this step ends, and Y has value $f(x_1, x_2, \ldots, x_n) = h(x_1, x_2, \ldots, x_n, 0)$ at this point. The variable Z_1 receives value 0 when the step at line 2 is executed. Since X_{n+1} has positive value k when the step at line 3 is reached, the execution of this step causes a branch to the step at line 5.

  Since $h(x_1, x_2, \ldots, x_n, j)$ is defined for every integer j such that $0 \leq j \leq k$, the following can be established by induction on j: If $j \in \mathbb{N}$ such that $0 \leq j \leq k$ then the statement (at line 3) with label A is reached at least $j+1$ times — and conditions (i), (ii) and (iii), as given above for the previous case, are all satisfied when this statement is reached for the $j+1^{\text{st}}$ time. In particular — when $j = k$ — the variable X_{n+1} has value $k-k = 0$. Thus the test at this line fails, so the execution continues with the step immediately below this (at line 4). Since there is no statement in the program with label C the execution of this step causes the execution of the program to halt. In particular, the execution halts, with the output variable Y having value $h(x_1, x_2, \ldots, x_n, k)$ — as required for this case.

Thus the program obtained from the above one, after macro expansion, is an $\mathcal{S}$-program computing the function h — as required to establish part (b) of the claim.

The claim of part (a) is almost the same — but slightly simpler. Instead of the program given above one should consider a program whose first step uses a macro to carry out the following

$$1. \quad Y \leftarrow k$$

— and whose remaining eight steps are the same as above. The proof that this program computes h is almost the same — but slightly simpler, because an execution of the first step,

"$Y \leftarrow k$" always halts (and a case corresponding to the first of the three cases, given in the proof of part (b), cannot arise). □

Claim 7. *Every primitive recursive function is $\mathcal{S}$-computable.*

Proof. Let $f : \mathbb{N}^n \to \mathbb{N}$ be a primitive recursive function (for some positive integer n). Then there is a positive integer k and a sequence

$$f_1, f_2, \ldots, f_k$$

of functions satisfying the following properties:

- For every integer i such that $1 \leq i \leq k$, one of the following properties holds.
  - (a) f_i is an initial function.
  - (b) There exists a positive integer ℓ, and functions $h : \mathbb{N}^\ell \to \mathbb{N}$ and $g_1, g_2, \ldots, g_\ell : \mathbb{N}^n \to \mathbb{N}$, such that each of the functions $h, g_1, g_2, \ldots, g_\ell$ is one of the functions $f_1, f_2, \ldots, f_{i-1}$, and such that f_i is obtained from h and $g_1, g_2, \ldots, g_\ell$ using composition.
  - (c) $n = 1$, and there exists a number $k \in \mathbb{N}$, and a function $g : \mathbb{N}^2 \to \mathbb{N}$, such that g is one of the functions $f_1, f_2, \ldots, f_{i-1}$ and such that f_i is obtained from k and g using the first form of primitive recursion.
  - (d) $n \geq 2$ and there exist functions $h : \mathbb{N}^{n-1} \to \mathbb{N}$ and $g : \mathbb{N}^{n+1} \to \mathbb{N}$ such that h and g are each one of the functions $f_1, f_2, \ldots, f_{i-1}$ and such that f_i is obtained from h and g using the second form of primitive recursion.
- $f = f_k$.

It can now be proved, by induction on i, using the strong form of mathematical induction, that the function f_i is primitive recursive for every integer i such that $1 \leq i \leq k$: For the basis (when the case $i = 1$ is considered) the desired result is a consequence of Claim 4. For the inductive step, one should apply either Claim 4, Claim 5, or either part (a) or part (b) of Claim 6 — depending on which of the above cases (a), (b), (c) or (d) applies to f_i.

Since $f = f_k$, and f is an arbitrarily chosen primitive recursive function, this establishes the claim. □

Proving That Every μ-Recursive Function is $\mathcal{S}$-Computable

Claim 8. *Let n be a positive integer and let $p : \mathbb{N}^{n+1} \to \{0, 1\}$ be a partial or total predicate. If p is $\mathcal{S}$-computable then the (partial or total) function $\mu p : \mathbb{N}^n \to \mathbb{N}$, obtained from p by unbounded minimization, is $\mathcal{S}$-computable as well.*

Proof. Let n be a positive integer and let $p : \mathbb{N}^{n+1} \to \{0, 1\}$ be an $\mathcal{S}$-computable partial or total predicate.

Since p is $\mathcal{S}$-computable, there exists an $\mathcal{S}$-program, $\mathcal{P}$, that computes p. Furthermore, if variables $W, V_1, V_2, \ldots, V_{n+1}$ are distinct, then $\mathcal{P}$ can be used to produce a subprogram, "$\mathcal{P}_{W \leftarrow p(X_1, X_2, \ldots, X_{n+1})}$" — with a corresponding pseudo-instruction

$$W \leftarrow p(V_1, V_2, \ldots, V_{n+1})$$

— which satisfies the following properties.

- For all $y_1, y_2, \ldots, y_{n+1} \in \mathbb{N}$ such that $p(y_1, y_2, \ldots, y_{n+1})$ is defined, if variable V_i has value y_i for $1 \leq i \leq n + 1$, then an execution of the pseudo-instruction

  $$W \leftarrow p(V_1, V_2, \ldots, V_{n+1})$$

  eventually ends, with the value of W set to $p(y_1, y_2, \ldots, y_{n+1})$ and with the values of all other variables (used in the program containing this pseudo-instruction) unchanged.

- For all $y_1, y_2, \ldots, y_{n+1} \in \mathbb{N}$ such that $p(y_1, y_2, \ldots, y_{n+1})$ is undefined, an execution of the pseudo-instruction

  $$W \leftarrow p(V_1, V_2, \ldots, V_{n+1})$$

  does not ever end, at all.

Now consider the following program.

1.		$Y \leftarrow 0$
2.	$[A]$	$Z_1 \leftarrow p(X_1, X_2, \ldots, X_n, Y)$
3.		IF $Z_1 \neq 0$ GOTO B
4.		$Y \leftarrow Y + 1$
5.		GOTO A

Let $x_1, x_2, \ldots, x_n \in \mathbb{N}$, and consider an execution of the above program with inputs $x_1, x_2, \ldots, x_n$. One of the following cases must arise.

- *Case:* There exists a number $i \in \mathbb{N}$ such that $p(x_1, x_2, \ldots, x_n, j) = 0$ for every integer j such that $0 \leq j \leq i - 1$, and such that $p(x_1, x_2, \ldots, x_n, i) = 1$.

  In this case $\mu p(x_1, x_2, \ldots, x_n) = i$ — so that an $\mathcal{S}$-program computing the function μp should halt, when executed on inputs $x_1, x_2, \ldots, x_n$, with Y having value i when this execution of the program ends.

  Consider an execution of the above program with inputs $x_1, x_2, \ldots, x_n$. The following properties can be proved to hold, for every integer h such that $1 \leq h \leq i + 1$, by induction on h:

(a) The step at line 2 (with label A) is executed at least h times.

(b) On the h^{th} execution of this step, X_ℓ has value x_ℓ, for every integer ℓ such that $1 \leq \ell \leq n$, and Y has value $h-1$, so the the execution of this step is an attempt to set the value of Z_1 to be

$$p(x_1, x_2, \ldots, x_n, h-1).$$

Now consider the execution of this program after the step at line 2 has been executed for the $i+1^{\text{st}}$ time. It follows by properties (a) and (b), above, that the execution of this step is an attempt to set the value of Z_1 to be

$$p(x_1, x_2, \ldots, x_n, i) = 1.$$

This attempt succeeds (so that the execution of this step ends) and the step at line 3 is reached. Since $Z_1 = 1$ at this point the test at line 3 passes and, since there is no statement with label B, the execution of the program ends. Since the value of Y has not been changed since the $i+1^{\text{st}}$ execution of the step at line 2 it follows, by property (b), that the value of this variable is $(i+1)-1 = i$. This i is returned as output, as needed in this case.

- *Case:* There exists a number $i \in \mathbb{N}$ such that $p(x_1, x_2, \ldots, x_n, j) = 0$ for every integer j such that $0 \leq j \leq i-1$, and such that $p(x_1, x_2, \ldots, x_n, i)$ is undefined.

  In this case $\mu p(x_1, x_2, \ldots, x_n)$ is undefined, since there is no integer j such that

  $$p(x_1, x_2, \ldots, x_n, \ell) = 0$$

  for every integer ℓ such that $0 \leq \ell \leq j-1$ and such that $p(x_1, x_2, \ldots, x_n, j) = 1$ — so that an $\mathcal{S}$-program computing the function μp should not halt, when executed on inputs $x_1, x_2, \ldots, x_n$.

  Consider an execution of the above program with inputs $x_1, x_2, \ldots, x_n$. Once again, properties (a) and (b), as given in the previous case, can be proved to hold for every integer h such that $1 \leq h \leq \ell + 1$.

  Now consider the execution of this program after the step at line 2 has been executed for the $i+1^{\text{st}}$ time. It follows by properties (a) and (b), above, that the execution of this step is an attempt to set the value of Z_1 to be

  $$p(x_1, x_2, \ldots, x_n, i)$$

  — which is undefined. Thus the execution of this step (or, more precisely, the subprogram it represents) never ends. It follows that the execution of the entire program on inputs $x_1, x_2, \ldots, x_n$ never ends, either, as needed for this case.

- *Case:* $p(x_1, x_2, \ldots, x_n, i) = 0$ for every number $i \in \mathbb{N}$.

  Once again $\mu p(x_1, x_2, \ldots, x_n)$ is undefined, since there is no integer j such that

  $$p(x_1, x_2, \ldots, x_n, \ell) = 0$$

  for every integer ℓ such that $0 \leq \ell \leq j - 1$ and such that $p(x_1, x_2, \ldots, x_n, j) = 1$ — so that an $\mathcal{S}$-program computing the function μp should not halt, when executed on inputs $x_1, x_2, \ldots, x_n$.

  Consider an execution of the above program with inputs $x_1, x_2, \ldots, x_n$. In this case properties (a) and (b), as given for the first case can be proved to hold for every integer i such that $i \geq 1$ — by induction on i.

  Thus (by property (a)) there is an i^{th} execution of the step at line 2 for every positive integer i. This execution of the program never ends, as needed for this case too.

Thus the $\mathcal{S}$-program obtained from the above one, by macro expansion, is an $\mathcal{S}$-program computing the function μp — so that the function μp is $\mathcal{S}$-computable. Since p was an arbitrarily chosen $\mathcal{S}$-computable partial or total predicate, this establishes the claim. □

Claim 9. *Every μ-recursive partial or total function is $\mathcal{S}$-computable.*

Exercise: Using Claim 8 as needed, modify the proof of Claim 7 to prove Claim 9.