Computations on Strings: Intermediate Models
Equivalence of Models: $\mathcal{S}$-Programs and $\mathcal{S}_n$-Programs

This lecture includes details about pseudo-instructions and macros for various $\mathcal{S}_n$-programs, as well as claims about the equivalence of various computational models. This supplemental document give additional details about these.

The notions of ***state***, ***instantaneous description***, and ***computation*** of an $\mathcal{S}$-program can be modified to obtain corresponding notions for $\mathcal{S}_n$-programs as follows.

Suppose, once again, that n is a positive integer and $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.

Definition: An ***$\mathcal{S}_n$-program*** is a sequence of statements as given in the lecture notes —

- "$V \leftarrow \sigma V$" for a variable V and for $\sigma \in \Sigma$,
- "$V \leftarrow V^-$" for a variable V, and
- "IF V ENDS σ GOTO L" for a variable V, $\sigma \in \Sigma$, and for a label L.

Definition: The ***length*** of an $\mathcal{S}_n$-program is the number of instructions in this sequence.

Definition: A ***state*** of an $\mathcal{S}_n$-program $\mathcal{P}$ is a finite sequence of equations $V = \omega$, for a variable V and for $\omega \in \Sigma^\star$, such that

- This includes ***at most one*** equation $V = \omega$ for each variable V appearing in $\mathcal{P}$, as well as for the output variable Y.
- If this includes an equation $V = \omega$ for a variable V that *does not* appear in $\mathcal{P}$ then $\omega = \lambda$.
- If the state includes an equation $V = \omega$ then (as noted above) $\omega \in \Sigma^\star$ — and one should think of ω as being "the current value" of the variable V.

Definition: A ***snapshot*** or ***instantaneous description***, for an $\mathcal{S}_n$-program $\mathcal{P}$ with length ℓ, is an ordered pair $\mathcal{S} = (i, \sigma)$ such that

- $1 \leq i \leq \ell + 1$. One should think of i as being the number of the statement in $\mathcal{P}$ that is about to be executed *next* — and where $i = \ell + 1$ if and only if the computation has already ended.
- σ is a ***state*** of $\mathcal{P}$, as defined above.
- For every variable V that appears in $\mathcal{P}$ that *does not* have an equation "$V = \omega$" included in σ, for any $\omega \in \Sigma^\star$, one can think of the equation "$V = \lambda$" as being implicitly included.

Definition: A ***computation*** of an $\mathcal{S}_n$-program $\mathcal{P}$ with length ℓ is a finite sequence $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$ of snapshots, where

- $\mathcal{S}_{i+1}$ is the ***successor*** of $\mathcal{S}_i$, for every integer i such that $1 \leq i \leq k - 1$. That is, if $\mathcal{S}_i = (j, \sigma_1)$ and $\mathcal{S}_{i+1} = (k, \sigma_2)$, then $1 \leq j \leq \ell$ and, if the program was in state σ_1 and instruction j was executed, then program would be in state σ_2, and j is the number of the instruction that would be executed after this execution of instruction h.
- $\mathcal{S}_k$ is ***terminal***. That is, $\mathcal{S}_k = (\ell + 1, \sigma)$ for some state σ.

Since we will considering the "simulation" of an $\mathcal{S}_n$-program by an $\mathcal{S}$-program (and vice-versa) the following relationship between states of $\mathcal{S}_n$-programs and states of $\mathcal{S}$-programs will also be useful.

Definition: Let $\mathcal{P}$ be an $\mathcal{S}_n$-program, let σ be a state of $\mathcal{P}$, let $\widehat{\mathcal{P}}$ be an $\mathcal{S}$-program, and let $\widehat{\sigma}$ be a state of $\widehat{\mathcal{P}}$. Then $\widehat{\sigma}$ ***corresponds to*** σ if $\widehat{\sigma}$ includes the same number of equations as σ and $\widehat{\sigma}$ includes an equation $V = \#(\omega)$ for every equation $V = \omega$ in σ (so that $\widehat{\sigma}$ does not include any equations besides this.

It will also be useful to confirm that a few more functions and predicates are $\mathcal{S}$-computable. Consider, first, the "monus" function: If $k, \ell \in \mathbb{N}$ then

$$k \dot{-} \ell = \begin{cases} k - \ell & \text{if } k \geq \ell, \\ 0 & \text{if } k < \ell. \end{cases}$$

Lemma 1. *The "monus" function is $\mathcal{S}$-computable.*

Proof. Consider the $\mathcal{S}$-program obtained from the program, shown in Figure 1, on page 3, after macro expansion — and consider an execution of this program when $X_1 = n$ and $X_2 = m$, so that the execution of the program should eventually end, with Y having value $n \dot{-} m$.

Note first that if $m = 0$ then the test at line 3 fails the first time it is checked, so that the step at line 4 is reached — ending the execution of the program with Y having the initial value of X_1 (that is, n), as required for this case.

```
1.        Z_1 ← X_2
2.        Y ← X_1
3.        IF Z_1 ≠ 0 GOTO A_1
4.        GOTO A_2
5.  A_1   Z_1 ← Z_1 − 1
6.        Y ← Y − 1
7.        IF Z_1 ≠ 0 GOTO A_1
```

Figure 1: An $\mathcal{S}$-Program for the "monus" Function

```
1,        Z_1 ← X_1 ∸ X_2
2.        IF Z_1 ≠ 0 GOTO A_1
3.        Z_1 ← X_2 ∸ X_1
4.        IF Z_1 ≠ 0 GOTO A_1
5.        Y ← 1
6.        GOTO A_2
7.  A_1   Y ← 0
```

Figure 2: An $\mathcal{S}$-Program for the Predicate "$X = Z$?"

It can be shown, for the case that $m > 0$, that the step at line 5 (with label A_1) is executed exactly m times. Furthermore, if $1 \leq i \leq m$ then the variables Z_1 and Y have values $m \dot{-} (i-1)$ and $n-(i-1)$, respectively, immediately before the i^{th} execution of the step at line 5. Following the m^{th} execution of this step, and the execution of the steps at line 6, the variables Z_1 and Y have values $m \dot{-} m = 0$ and $n \dot{-} m$ respectively — so that the test at line 7 fails and the execution of the program halts, with $Y = n \dot{-} m$, as required to establish the correctness of this program and to prove the claim. □

Next consider the predicate "$X = Z$?", that is, the function $p_{X=Z?} : \mathbb{N}^2 \to \{0, 1\}$ such that, for all $n, m \in \mathbb{N}$,

$$p_{X=Z?}(n, m) = \begin{cases} 1 & \text{if } n = m, \\ 0 & \text{if } n \neq m. \end{cases}$$

Lemma 2. *The predicate "$X = Z$?" is $\mathcal{S}$-computable.*

Proof. Consider the $\mathcal{S}$-program obtained from the program shown in Figure 2 — and, in particular, consider an execution of this program on inputs n and m (for $n, m \in \mathbb{N}$).

Since the "monus" function is $\mathcal{S}$-program, by Lemma 1, there is a macro corresponding to the

pseudo-instruction

$$W \leftarrow X \dot{-} Z$$

such that, if X and Z have variables k and ℓ, respectively, when this macro is executed, then the execution of this macro eventually ends, with the value of W set to be

$$k \dot{-} \ell = \begin{cases} k - \ell & \text{if } k \geq \ell, \\ 0 & \text{if } k < \ell, \end{cases}$$

and with the values of all other variables (used elsewhere in the program) unchanged. Thus, after the execution of the step at line 1, the variables X_1, X_2 and Z_1 have values n, m, and $n \dot{-} m$ respectively — and, after the execution of the step at line 3 (if it is reached), the variables X_1, X_2 and Z_1 have values n, m, and $m \dot{-} n$ respectively.

Either $n < m$, $n > m$, or $n = m$. These cases are considered separately, below.

- If $n < m$ then the value 0 should be returned. During this execution of this program the step at line 1 is reached and executed, causing the value of Z_1 to be set to be $n \dot{-} m = 0$. The test at line 2 therefore fails, so that the step at line 3 is reached and executed. The value of Z_1 has now been set to be $m \dot{-} n = m - n$ — a positive integer. Thus the test at line 4 is passed and the statement at line 7 (with label A_1) is reached executed — setting the value of Y to be 0 when the execution of the program ends. The execution of the program therefore ends with the correct output returned for this case.
- If $n > m$ then the value 0 should be returned. During the execution of this program the step at line 1 is reached and executed, causing the value of Z_1 to be set to be $n \dot{-} m - n - m$ — a positive integer. The test at line 2 is therefore passed, so that the step at line 7 (with label A_1) is reached and executed — setting the value of Y to be 0 when the execution of the program ends. The execution of the program therefore ends with the correct output returned for this case, as well.
- Finally, if $n = m$ then the value 1 should be returned. During the execution of this program the step at line 1 is reached and executed, causing the value of Z_1 to be set to be $n \dot{-} m = n - m = 0$. The test at line 2 therefore fails, so that the step at line 3 is reached executed. The value of Z_1 has now been set to be $m \dot{-} n = m - n = 0$. The test at line 4 is therefore failed, so that the step at line 5 is reached, and the value of Y is set to be 1. The unconditional branch at line 6 is reached and executed and — since there is no statement with label A_2 — the execution of the algorithm ends. At this point Y has the desired output value, once again.

Since the execution of the program ends with the required output returned in every case, this program computes the predicate "$X = Z$?" — as needed to show that this predicated is $\mathcal{S}$-computable. □

Claim 1. *Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial or total function, for a positive integer k, and let n be positive integer. If f is $\mathcal{S}_n$-computable then f is $\mathcal{S}$-computable.*

Proof. To begin, note that every instruction of an $\mathcal{S}_n$-program an be simulated using pseudo-instructions and macros for $\mathcal{S}$-programs that were introduced in the previous lecture or in the lecture notes for this lecture, before the statement of this claim:

- If $\sigma \in \Sigma$ and V is a variable, then the instruction

  $$V \leftarrow \sigma V$$

  can be simulated using the $\mathcal{S}$-program

  $$W \leftarrow \text{concat}(\sigma, V)$$
  $$V \leftarrow W$$

  where W is a variable that is not used anywhere else. That is, the above instruction can be simulated using the $\mathcal{S}$-program obtained, from the above, after macro expansion.

- If V is a variable then the instruction

  $$V \leftarrow V^-$$

  can be simulated using the $\mathcal{S}$-program obtained from

  $$W \leftarrow \text{rtrunc}(v)$$
  $$V \leftarrow W$$

  using macro expansion, where W is a variable that is not used anywhere else.

- Let $\sigma \in \Sigma$ — so that $\sigma = \sigma_i$ for an integer i such that $1 \leq i \leq n$. If V is a variable and L is a label then the instruction

  $$\text{IF } V \text{ ENDS } \sigma \text{ GOTO } L$$

  can be simulated using the $\mathcal{S}$-program obtained from

  $$W_1 \leftarrow \text{rtend}(V)$$
  $$W_2 \leftarrow i$$
  $$W_3 \leftarrow p_{W_1 = W_2?}$$
  $$\text{IF } W_3 \neq 0 \text{ GOTO } L$$

  The third statement uses a macro for the predicate $p_{X=Z?}$ — which is an $\mathcal{S}$-computable predicate, by Lemma 2 – so, after its execution, $W_3 = 1$ if $W_1 = W_2$, and $W_3 = 0$ otherwise. Thus the test on the final line is passed if and only if $V \neq \lambda$ and the rightmost symbol of V is σ_i, as required.

With that noted, suppose that $f : \mathbb{N}^k \to \mathbb{N}$ is an $\mathcal{S}_n$-computable partial or total function, for positive integers k and n. Then there exists an $\mathcal{S}_n$ program $\mathcal{P}$ which computes f. That is, this program satisfies the following properties.

- For all natural numbers $x_1, x_2, \ldots, x_k$ such that $f(x_1, x_2, \ldots, x_k)$ is defined, if $\mathcal{P}$ is executed with inputs $\omega_1, \omega_2, \ldots, \omega_k$, where $\omega_i \in \Sigma^\star$ such that $\#(\omega_i) = x_i$ for $1 \leq i \leq k$, then this execution of $\mathcal{P}$ eventually ends, and Y has value μ, for a string $\mu \in \Sigma^\star$ such that $\#(\mu) = f(x_1, x_2, \ldots, x_k)$, when this execution ends.
- For all natural numbers $x_1, x_2, \ldots, x_k$ such that $f(x_1, x_2, \ldots, x_k)$ is undefined, if $\mathcal{P}$ is executed with inputs $\omega_1, \omega_2, \ldots, \omega_k$, where $\omega_i \in \Sigma^\star$ such that $\#(\omega_i) = x_i$ for $1 \leq i \leq k$, then this execution of $\mathcal{P}$ does not end.

Suppose that the program $\mathcal{P}$ has length ℓ — so that it is a sequence

$$i_1, i_2, \ldots, i_\ell$$

of instructions in the programming language $\mathcal{S}_n$ with length ℓ. For $1 \leq j \leq k$, since i_j is either "$V \leftarrow \sigma V$" for $\sigma \in \Sigma$ and a variable V, "$V \leftarrow V^-$" for a variable V, or "IF V ENDS σ GOTO L" for $\sigma \in \Sigma$, a variable V and a label L, there exists an $\mathcal{S}$-program $\mathcal{Q}_j$ that simulates an execution of instruction j. In particular, the following property is satisfied:

- Suppose that σ is a state in $\mathcal{P}$ and $\widehat{\sigma}$ is a state in $\mathcal{Q}_j$ (or any larger $\mathcal{S}$-program that includes $\mathcal{Q}_j$ as a subprogram) such that $\widehat{\sigma}$ corresponds to σ.
- Suppose that if instruction i_j is executed when $\mathcal{P}$ is in state in σ. Let τ be the state that $\mathcal{P}$ is in, immediately after the execution of instruction i_j ends.
- Then, if the $\mathcal{S}$-program is executed, beginning in state $\widehat{\sigma}$, then this execution also ends — and, on termination of the program, the program is in the state $\widehat{\tau}$ that corresponds to τ.

Consider an $\mathcal{S}_n$-program $\widehat{\mathcal{P}}$ that is the concatenation

$$\mathcal{Q}_1 \mathcal{Q}_2 \ldots \mathcal{Q}_\ell$$

of the $\mathcal{S}$-programs that simulate the instructions in $\mathcal{P}$. For $1 \leq j \leq \ell$, let len_j be the length of $\mathcal{Q}_j$, so that $\widehat{\mathcal{P}}$ has length $\sum_{j=1}^{\ell} \text{len}_j$.

The following subclaim is now easily established by induction on h.

> **Subclaim:** Let $x_1, x_2, \ldots, x_k \in \mathbb{N}$ and let $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ such that $\#(\omega_i) = x_i$ for every integer i such that $1 \leq i \leq k$. Consider executions of $\mathcal{P}$ with inputs $\omega_1, \omega_2, \ldots, \omega_k$ and of $\widehat{\mathcal{P}}$ with inputs $x_1, x_2, \ldots, x_k$.

Let $h \in \mathbb{N}$ such that at least h statements are executed during the above execution of $\mathcal{P}$, and let $\widehat{h}$ be the total number of steps used, in the above execution of $\widehat{\mathcal{P}}$, to simulate the first steps in the execution of $\mathcal{P}$.

Finally, suppose that the execution of $\widehat{P}$ has instantaneous description (m, σ) after the first h steps have been executed — so that m is an integer such that $1 \leq m \leq \ell + 1$, and σ is a state of $\mathcal{P}$. Then $\widehat{\mathcal{P}}$ has instantaneous description

$$\left(\left(\sum_{j=1}^{m-1} \mathsf{len}_j \right) + 1, \widehat{\sigma} \right)$$

after the first h steps of $\mathcal{P}$ have been simulated (that is, after $\widehat{h}$ steps have been carried out), where $\widehat{\sigma}$ is a state of $\widehat{\mathcal{P}}$ that corresponds to σ.

Now let $x_1, x_2, \ldots, x_k \in \mathbb{N}$ and let $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ such that $\#(\omega_j) = x_j$ for $1 \leq j \leq k$.

- If $f(x_1, x_2, \ldots, x_k)$ is defined then the execution of $\mathcal{P}$ on inputs $\omega_1, \omega_2, \ldots, \omega_k$ terminates, and the value of Y is a string $\mu \in \Sigma^\star$ such that $\#(\mu) = f(x_1, x_2, \ldots, x_k)$ when the execution ends.

  Let t be the number of steps included in this execution of $\mathcal{P}$ and let $\widehat{t}$ be the number of steps used, in the execution of $\widehat{\mathcal{P}}$ on inputs $x_1, x_2, \ldots, x_k$, to simulate the t steps of the execution of $\mathcal{P}$. Then, since the execution of $\widehat{P}$ has a snapshot $(\ell + 1, \sigma)$ after these steps, where σ is a state that includes the equation $Y = \#(f(x_1, x_2, \ldots, x_k)$. It follows by the subclaim that, after $\widehat{t}$ steps of the execution of $\widehat{\mathcal{P}}$ have been carried out, $\widehat{\mathcal{P}}$ has an instantaneous description

  $$\left(\left(\sum_{j=1}^{\ell} \mathsf{len}_j \right) + 1, \widehat{\sigma} \right)$$

  — which is terminal, since $\widehat{\mathcal{P}}$ has length $\sum_{j=1}^{\ell} \mathsf{len}_j$. Furthermore, since $\widehat{\sigma}$ is consistent with σ, and $\#(\mu) = f(x_1, x_2, \ldots, x_k)$, $\widehat{\sigma}$ must include an equation

  $$Y = f(x_1, x_2, \ldots, x_k).$$

  That is, the desired value, $f(x_1, x_2, \ldots, x_k)$ is returned when the execution of $\widehat{\mathcal{P}}$ on inputs $x_1, x_2, \ldots, x_k$ ends.

- On the other hand, if $f(x_1, x_2, \ldots, x_k)$ is undefined, then the execution of $\mathcal{P}$ on inputs $\omega_1, \omega_2, \ldots, \omega_k$ never ends — so that it includes the execution of at least t steps for all $t \in \mathbb{N}$. Since at least one step of $\widehat{\mathcal{P}}$ is required to simulate any step in an execution of $\mathcal{P}$, it follows, by the above subclaim, that the execution of $\widehat{\mathcal{P}}$ on inputs $x_1, x_2, \ldots, x_k$ must include at least t steps as well, for all $t \in \mathbb{N}$. Thus the execution of $\widehat{\mathcal{P}}$ on inputs $x_1, x_2, \ldots, x_k$ never ends — as required for this case.

Since all inputs $x_1, x_2, \ldots, x_k \in \mathbb{N}$ have been considered, it follows that the $\mathcal{S}$-program $\widehat{\mathcal{P}}$ computes the function f, so that f is $\mathcal{S}$-computable.

Since f was an arbitrarily chosen $\mathcal{S}_n$-computable function, this establishes the claim. □

Now, in order to establish a converse for Claim 1, it is necessary to show that every instruction in an $\mathcal{S}$-program can be simulated using a pseudo-instruction and macro in an $\mathcal{S}_n$-program (for an arbitrary positive integer n). It will be useful to recall that if n is a positive integer then the variables in an $\mathcal{S}_n$-program have strings in $\Sigma^\star$ as values, for the alphabet

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

with size n. Each string $\omega \in \Sigma^\star$ corresponds to a natural number, $\#(\omega) \in \mathbb{N}$:

- $\#(\lambda) = 0$.
- If $\omega = \alpha_m \alpha_{m-1} \ldots \alpha_2 \alpha_1$ is a string in $\Sigma^\star$ with positive length m, then

  $$\#(\omega) = \#(\alpha_m \alpha_{m-1} \ldots \alpha_2 \alpha_1) = \sum_{i=1}^{m} \#(\alpha_i) \cdot n^{i-1}$$

  where $(\sigma_i) = i$ for every integer i such that $1 \leq i \leq n$.

Recall that a function $f : \mathbb{N}^k \to \mathbb{N}$ is computed by an $\mathcal{S}_n$-program $\mathcal{P}$ if the following properties are satisfied for all natural numbers $x_1, x_2, \ldots, x_k \in \mathbb{N}$:

- If $f(x_1, x_2, \ldots, x_k)$ is defined and $\mathcal{P}$ is executed with inputs $\omega_1, \omega_2, \ldots, \omega_k$, where $\#(\omega_i) = x_i$ for $1 \leq i \leq k$, then this execution of the program eventually ends and, when it does, the value of the output variable, Y, is a string $\mu \in \Sigma^\star$ such that $\#(\mu) = f(x_1, x_2, \ldots, x_k)$.
- If $f(x_1, x_2, \ldots, x_k)$ is undefined and $\mathcal{P}$ is executed with inputs $\omega_1, \omega_2, \ldots, \omega_k$, where $\#(\omega_i) = x_i$ for $1 \leq i \leq k$, then this execution of the program never ends.

Lemma 3. *Let $s : \mathbb{N} \to \mathbb{N}$ such that $s(x) = x + 1$ for all $x \in \mathbb{N}$. Then s is S_n-computable for every positive integer n.*

Proof. In order to show that the function s is $\mathcal{S}_n$-computable, for a positive integer n, it is necessary to describe, and prove the correctness of, an $\mathcal{S}_n$-program that returns the following value as output, when executed on a given input $\omega \in \Sigma^\star$:

- If $\omega = \lambda$ then, since $\#(\lambda) = 0$, the string σ_1 should be returned, since $\#(\sigma_1) = 1$.

- If $\omega = \alpha_k \alpha_{k-1} \ldots \alpha_2 \alpha_1$ is a string in $\Sigma^\star$ with positive length k, and $\alpha_i = \sigma_n$ for every integer i such that $1 \leq i \leq k$, then

$$\#(\omega) = \sum_{i=1}^{k} \#(\alpha_i) \cdot n^{i-1} = \sum_{i=1}^{k} n \cdot n^{i-1} = \sum_{i=1}^{k} n^i.$$

  Consider the string $\mu = \beta_{k+1} \beta_k \ldots \beta_2 \beta_1 \in \Sigma^\star$, with length $k+1$, where $\beta_i = \alpha_1$ for every integer i such that $1 \leq i \leq k+1$. Since

$$\#(\mu) = \sum_{i=1}^{k+1} \#(\beta_i) \cdot n^{i-1} = \sum_{i=1}^{k+1} 1 \cdot n^{i-1} = \left(\sum_{i=1}^{k} n^i \right) + 1 = \#(\omega) + 1,$$

  the string μ should be returned in this case.

- In the only remaining case $\omega = \alpha_k \alpha_{k-1} \ldots \alpha_2 \alpha_1$ is a string in $\Sigma^\star$ with positive length k, and there exists at least one integer i such that $1 \leq i \leq k$ and $\alpha_i \neq \sigma_n$. Consider the *smallest* such integer i — so that $\alpha_j = \sigma_n$ for an integer j such that $1 \leq j \leq i-1$, and such that $\alpha_i = \sigma_\ell$ for an integer ℓ such that $1 \leq \ell \leq n-1$. Then

$$\begin{aligned}
\#(\omega) &= \sum_{j=1}^{k} \#(\alpha_j) \cdot n^{j-1} \\
&= \sum_{j=i+1}^{k} \#(\alpha_j) \cdot n^{j-1} + \#(\alpha_i) \cdot n^{i-1} + \sum_{j=1}^{i-1} \#(\alpha_j) \cdot n^{j-1} \\
&= \sum_{j=i+1}^{k} \#(\alpha_j) \cdot n^{j-1} + \ell \cdot n^{i-1} + \sum_{j=1}^{i-1} n \cdot n^{j-1} \\
&= \sum_{j=i+1}^{k} \#(\alpha_j) \cdot n^{j-1} + \ell \cdot n^{i-1} + \sum_{j=1}^{i-1} n^j \\
&= \sum_{j=i+1}^{k} \#(\alpha_j) \cdot n^{j-1} + (\ell+1) \cdot n^{i-1} + \sum_{j=1}^{i-2} n^j.
\end{aligned}$$

  Consider the string $\nu = \alpha_k \alpha_{k-1} \ldots \alpha_{i+1} \beta_i \beta_{i-1} \ldots \beta_2 \beta_1 \in \Sigma^\star$, with length k, such that $\beta_i = \alpha_{\ell+1}$ and such that $\beta_j = \alpha_1$ for every integer j such that $1 \leq j \leq i-1$:

$$\begin{aligned}
\#(\nu) &= \sum_{j=i+1}^{k} \#(\alpha_j) \cdot n^{j-1} + \sum_{j=1}^{i} \#(\beta_j) \cdot n^{j-1} \\
&= \sum_{j=i+1}^{k} \#(\alpha_j) \cdot n^{j-1} + (\ell+1) \cdot n^{i-1} + \sum_{j=1}^{i-1} 1 \cdot n^{j-1}
\end{aligned}$$

$$= \sum_{j=i+1}^{k} \#(\alpha_j) \cdot n^{j-1} + (\ell + 1) \cdot n^{i-1} + \sum_{j=0}^{i-2} n^j$$
$$= \#(\omega) + 1.$$

Thus the string ν should be returned, as output, in this final case.

Now consider an $\mathcal{S}_n$-program obtained, after macro expansion, from a program that begins as follows.

$$\begin{array}{ll} [C_1] & \text{IF } X_1 \text{ ENDS } \sigma_1 \text{ GOTO } A_1 \\ & \text{IF } X_1 \text{ ENDS } \sigma_2 \text{ GOTO } A_2 \\ & \qquad \vdots \\ & \text{IF } X_1 \text{ ENDS } \sigma_n \text{ GOTO } A_n \\ & Y \leftarrow \sigma_1 Y \\ & \text{GOTO } D_1 \end{array}$$

The program continues with a block, as follows, for every integer i such that $1 \leq i \leq n-1$:

$$\begin{array}{ll} [A_i] & X_1 \leftarrow X_1^- \\ & Y \leftarrow \sigma_{i+1} Y \\ & \text{GOTO } C_2 \end{array}$$

The program continues with a different block (which implements the "carry" operation that is needed here):

$$\begin{array}{ll} [A_n] & X_1 \leftarrow X_1^- \\ & Y \leftarrow \sigma_1 Y \\ & \text{GOTO } C_1 \end{array}$$

The next block of the program is used to complete the generation of the output, after an input symbol that is different from σ_n has been found.

$$\begin{array}{ll} [C_2] & \text{IF } X_1 \text{ ENDS } \sigma_1 \text{ GOTO } B_1 \\ & \text{IF } X_1 \text{ ENDS } \sigma_2 \text{ GOTO } B_2 \\ & \qquad \vdots \\ & \text{IF } X_1 \text{ ENDS } \sigma_n \text{ GOTO } B_n \\ & \text{GOTO } D_1 \end{array}$$

The program continues (and ends) with the following blocks, for every integer i such that $1 \leq i \leq n$:

$$
\begin{array}{ll}
[B_i] & X_1 \leftarrow X_1^- \\
& Y \leftarrow \sigma_i Y \\
& \text{GOTO } C_2
\end{array}
$$

Consider, now, the execution of this program on an input string $\omega \in \Sigma^\star$ for each of the cases mentioned above.

- If $\omega = \lambda$ then the tests in the first n instructions all fail, so that the value of Y is changed from λ to σ_1 when the $n + 1^{\text{st}}$ instruction is reached and executed. Since there is no instruction with label D_1 the unconditional branch after this causes the program's execution to end — with σ_1 returned, as required.

- If $\omega = \alpha_k \alpha_{k-1} \ldots \alpha_2 \alpha_1$ is a string in $\Sigma^\star$ with positive length k, and $\alpha_1 = \sigma_n$ for every integer i such that $1 \leq i \leq k$, then one can prove by induction on i that if $1 \leq i \leq k$, and the program is executed on input ω, then the n^{th} instruction in the program (including the tests checking whether X_1 ends with σ_n) is reached, with the test passed, at least i times. Furthermore, when this test is reached for the i^{th} time, X_1 is a string with length $k - i + 1$ including this number of copies of σ_n, while Y is a string with length $i = 1$, including this number of copies of σ_1.

  Following the k^{th} execution of this instruction — and passing of the test — control transfers, again, to the statement with label A_n. When this instruction is executed the value of X_1 is changed from σ_n to λ. When the instruction after it is executed, the value of Y is changed from a string with length $k - 1$, including this number of copies of σ_1, to a string with length k, including this number of copies of σ_1, once again. The unconditional branch after this instruction is reached, transferring control back to the first statement in the program.

  At this point, since $X_1 = \lambda$, the tests in the first n instructions all fail. The $n + 1^{\text{st}}$ instruction is executed, after which Y is a string with length $k + 1$, all of whose symbols are copies of σ_1 — so that the value of Y is as desired for this case. Since there is no instruction with label with label D_1 the unconditional branch causes execution to end — with the required output set as the value for Y.

- Finally, if $\omega = \alpha_k \alpha_{k-1} \ldots \alpha_2 \alpha_1$ is a string in $\Sigma^\star$ with positive length k, such that there exists at least one integer i such that $1 \leq i \leq k$ and $\alpha_i \neq \sigma_n$. Once again, consider the smallest such integer i — so that $\alpha_j = \sigma_n$ for every integer j such that $1 \leq j \leq i - 1$, and so that $\alpha_i = \sigma_\ell$ for some integer ℓ such that $1 \leq \ell \leq n - 1$.

  – Suppose, first, that $i = 1$. In this case $\alpha_1 = \sigma_\ell$ (where $1 \leq \ell \leq n - 1$, as noted above), and the n^{th} statement is never reached at all — because the ℓ^{th} statement is reached, the test at this line is passed, and control is passed to the statement with label A_ℓ. Following the execution of this statement, and the one after it, X_1

has value $\alpha_k\alpha_{k-1}\ldots\alpha_3\alpha_2$ and Y has value $\sigma_{\ell+1}$. An unconditional branch is then executed, passing control to the statement with label C_2.

– Suppose, next, that $i \geq 2$. In this case one can prove, by induction on j, that if the program is executed with input ω, then the n^{th} instruction in the program (including the test whether X_1 ends with σ_n) is reached, with the test passed, at least j times, for every integer j such that $1 \leq j \leq i-1$. Furthermore, when this test is reached and passed for the j^{th} time, the variable X_1 has value $\alpha_k\alpha_{k-1}\ldots,\alpha_j$, and the variable Y is a string with length $j-1$, including $j-1$ copes of the symbol σ_1.

  Now, after this test has been reached and passed for the $i-1^{\text{th}}$ time, control is passed to the statement with label A_n. When this statement is executed, the value of X_1 is changed from $\alpha_k\alpha_{k-1}\ldots,\alpha_i\alpha_{i-1}$ to $\alpha_k\alpha_{k-1}\ldots\alpha_{i+1}\alpha_i$ and, when the statement after this is executed, the value of Y is changed from a string with length $i-2$, including $i-2$ copies of σ_1, to a string of length $i-1$, including $i-1$ copies of this symbol. An unconditional branch to a statement with label C_1 is then reached and executed, so that control is passed to the first statement in this program.

  At this point X_1 ends with the symbol $\alpha_i = \sigma_\ell$, so that the ℓ^{th} statement in the program is reached (as the execution continues) and the test is passed; control is then transferred to the statement with label A_ℓ. As a result of the execution of the next three statements the value of X is changed from $\alpha_k\alpha_{k-1}\ldots\alpha_{i+1}\alpha_i$ to $\alpha_k\alpha_{k-1}\ldots\alpha_{i+2}\alpha_{i+1}$, the value of Y is changed from $\beta_{i-1}\beta_{i-2}\ldots\beta_2\beta_1$ to the string $\beta_i\beta_{i-1}\ldots\beta_2\beta_1$ — where $\beta_i = \sigma_{\ell+1}$ and $\beta_j = \sigma_1$ for $1 \leq j \leq i-1$ — and, after the execution of an unconditional branch, control is passed to the statement with label C_2.

In both cases the current value of X_1 is $\alpha_k\alpha_{k-1}\ldots\alpha_{i+2}\alpha_{i+1}$ and the current value of Y is $\beta_i\beta_{i-1}\ldots\beta_2\beta_1$ — so that the desired output is the concatenation of the current values of X_1 and Y.

It can now be shown, by induction on j, that — for every integer j such that $1 \leq j \leq k-i$, the statement with label C_2 is reached at least j times. Furthermore, when this statement is reached (but not yet executed) for the j^{th} time, the variable X_1 has value $\alpha_k\alpha_{k-1}\ldots\alpha_{i+j}\alpha_{i+j}$ and the variable Y has value

$$\alpha_{i+j-1}\alpha_{i+j-2}\ldots\alpha_{i+2}\alpha_{i+1}\beta_i\beta_{i-1}\ldots\beta_2\beta_1$$

(which is $\beta_i\beta_{i-1}\ldots\beta_2\beta_1$ when $j=1$, and $\alpha_{i+1}\beta_i\beta_{i-1}\ldots\beta_2\beta_1$ when $j=2$). In particular, immediately before the $k-i^{\text{th}}$ execution of this statement the variable X_1 has value α_k and the variable Y has value $\alpha_{k-1}\alpha_{k-2}\ldots\alpha_{i+2}\alpha_{i+1}\beta_i\beta_{i-1}\ldots\beta_2\beta_1$.

The symbol α_k is equal to σ_h for some integer h such that $1 \leq h \leq n$ and the statement at, or after, the statement with label C_2, including the test "X_1 ENDS σ_h" is reached

and passed — so that control is transferred to the statement with label B_h. After the next three steps are executed $X_1 = \lambda$, $Y = \alpha_k\alpha_{k-1}\ldots\alpha_{i+2}\alpha_{i+1}\beta_i\beta_{i-2}\ldots\beta_1\beta_0$ — the value which should be returned as output, for this case — and an unconditional branch, transferring control back to the statement with C_2, for a final time, has been executed. Since $X_1 = \lambda$ at this points the tests in the n statements including and following the statement with label C_2 all fail, so the unconditional branch after these statements is reached and executed. Since there is no statement with label D_1 this causes the execution to end — with the desired output value returned in this final case.

Since the execution of the program ends with Y set to the desired output value, it follows that this program computes the function s — which is therefore $\mathcal{S}_n$-computable, as claimed. □

An $\mathcal{S}_n$ program computing the above function is easily modified to produce a subprogram (that is, macro) corresponding to the pseudo-instruction

$$V \leftarrow V + 1$$

for a variable V — that is, the execution of the subprogram always ends, with the value of the variable V incremented and with the values of no other variables changed.

Lemma 4. *Let mon* : $\mathbb{N} \to \mathbb{N}$ *such that*

$$\mathit{mon}(x) = x \dot{-} 1 = \begin{cases} x - 1 & \text{if } x \geq 1, \\ 0 & \text{if } x = 0. \end{cases}$$

Then the function mon is $\mathcal{S}_n$-computable for every positive integer n.

Proof. In order to show that the function mon is $\mathcal{S}_n$-computable, for a positive integer n, it is necessary to describe, and prove the correctness of, an $\mathcal{S}_n$-program that returns the following value as output, when executed on a given input $\omega \in \Sigma^\star$:

- If either $\omega = \lambda$ (so that $\#(\omega) = 0$) or $\omega = \sigma_1$ (so that $\#(\omega) = 1$ then the string λ should be returned as output.
- If ω has length $k \geq 2$ and every symbol in ω is a copy of σ_1 — so that

$$\#(\omega) = \sum_{j=1}^{k} 1 \cdot n^{j-1} = \sum_{j=0}^{k-1} n^j$$

then a string $\mu \in \Sigma^\star$, with length $k-1$, such that every symbol in μ is a copy of σ_n should be returned as output.

- In the only remaining case ω is a string with length $k \geq 1$ and there is at least one symbol in ω that is not a copy of σ_1. Consider the *rightmost* such symbol, in some position i — $1 \leq i \leq k$ and

  $$\omega = \alpha_k \alpha_{k-1} \dots \alpha_2 \alpha_1$$

  where $\alpha_i = \sigma_h$, for some integer h such that $2 \leq h \leq n$, and where $\alpha_j = \sigma_1$ for every integer j such that $1 \leq j \leq i-1$. In this case the value that should be returned is a string

  $$\mu = \alpha_k \alpha_{k-1} \dots \alpha_{i+2} \alpha_{i+1} \beta_i \beta_{i-2} \dots \beta_1 \beta_0$$

  such that $\beta_i = \sigma_{h-1}$ and $\beta_j = \sigma_n$ for every integer j such that $1 \leq j \leq i-1$.

Exercise: Confirm that if μ is as described in the last two cases then $\#(\mu) = \#(\omega) - 1$, as required.

Now consider an $\mathcal{S}_n$-program obtained, after macro expansion, from a program that begins as follows. This is similar to the program described in the proof of Lemma 3:

$$\begin{array}{ll} [C_1] & \text{IF } X_1 \text{ ENDS } \sigma_1 \text{ GOTO } A_1 \\ & \text{IF } X_1 \text{ ENDS } \sigma_2 \text{ GOTO } A_2 \\ & \qquad \vdots \\ & \text{IF } X_1 \text{ ENDS } \sigma_n \text{ GOTO } A_n \\ Y \leftarrow Y^- & \\ \text{GOTO } D_1 & \end{array}$$

This time, the first block to follow (which implements a "borrow" operation) is different from others in this part of the program:

$$\begin{array}{ll} [A_1] & X_1 \leftarrow X_1^- \\ & Y \leftarrow \sigma_n Y \\ & \text{GOTO } C_1 \end{array}$$

The program continues with a block, as follows, for every integer i such that $2 \leq i \leq n$:

$$\begin{array}{ll} [A_i] & X_1 \leftarrow X_1^- \\ & Y \leftarrow \sigma_{i-1} Y \\ \text{GOTO } C_2 & \end{array}$$

The rest of the program is the same as the rest of the program described in the proof of Lemma 3. That is, the program continues as follows.

$$
\begin{array}{ll}
[C_2] & \text{IF } X_1 \text{ ENDS } \sigma_1 \text{ GOTO } B_1 \\
& \text{IF } X_1 \text{ ENDS } \sigma_2 \text{ GOTO } B_2 \\
& \qquad \vdots \\
& \text{IF } X_1 \text{ ENDS } \sigma_n \text{ GOTO } B_n \\
& \text{GOTO } D_1
\end{array}
$$

The program continues (and ends) with the following blocks, for every integer i such that $1 \leq i \leq n$:

$$
\begin{array}{ll}
[B_i] & X_1 \leftarrow X_1^- \\
& Y \leftarrow \sigma_i Y \\
& \text{GOTO } C_2
\end{array}
$$

Completion of the proof — that is, establishing that this program correctly computes the *mon* function — is left as another ***exercise***, which can be completed by modifying the end of the proof of Lemma 3. □

An $\mathcal{S}_n$ program computing the above function is easily modified to produce a subprogram (that is, macro) corresponding to the pseudo-instruction

$$V \leftarrow V - 1$$

for a variable V — that is, the execution of the subprogram always ends, with the value of the variable V decremented (or unchanged, if its initial value is 0) and with the values of no other variables changed.

Claim 2. *Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial or total function, for a positive integer k, and let n be a positive integer. If f is $\mathcal{S}$-computable then f is $\mathcal{S}_n$-computable.*

Sketch of Proof. To begin, note that every instruction of an $\mathcal{S}$-program can be simulated using pseudo-instructions and macros for $\mathcal{S}_n$-programs, for any positive integer n:

- If V is a variable then it follows by Lemma 3, and the remark following it, that there is a macro and pseudo-instruction, for $\mathcal{S}_n$-programs, that simulates the instruction

$$V \leftarrow V + 1$$

- If V is a variable then it follows by Lemma 4, and the remark following it, hat there is a macro and pseudo-instruction, for $\mathcal{S}_n$-programs, that simulates the instruction

$$V \leftarrow V - 1$$

- If V is a variable and L is a label then the instruction "IF $V \neq 0$ GOTO L" can be simulated by the following macro:

$$\begin{array}{c} \text{IF } V \text{ ENDS } \sigma_1 \text{ GOTO } L \\ \text{IF } V \text{ ENDS } \sigma_2 \text{ GOTO } L \\ \vdots \\ \text{IF } V \text{ ENDS } \sigma_n \text{ GOTO } L \end{array}$$

  If V is a variable then — since its execution does not change the program's state — the instruction "$V \leftarrow V$" can be simulated using the macro

$$W \leftarrow W^-$$

  where W is a variable that is not used anywhere else (so that its value is λ, both before and after the macro is executed).

With that noted, suppose that $f : \mathbb{N}^k \to \mathbb{N}$ is an $\mathcal{S}$-computable partial or total function, for positive integers k and n. Then there exists an $\mathcal{S}$-program $\mathcal{P}$ which computes f. The construction of an $\mathcal{S}_n$-program that also computes f (and proof that it is correct) proceeds essentially as the construction of an $\mathcal{S}$-program from an $\mathcal{S}_n$-program does, in the proof of Clam 1 — with the roles of the programming languages $\mathcal{S}$ and $\mathcal{S}_n$ reversed. □