Computer Science 513

Computation on Strings: Intermediate Models

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #6

Goal for Today

Goal for Today:

- Introduction of two "intermediate" models of computation that can be used for computations on strings
- Proof that — up to differences in encodings — these are equivalent to the $\mathcal{L}$-program model (for computations of integer functions) already defined
- This will simplify the introduction of ***Turing machines*** and the proof that these are (up to encodings) equivalent too.

The Programming Language $\mathcal{S}_n$

Suppose, once again, that

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

for an integer $n \geq 1$.

- Consider a programming language $\mathcal{S}_n$ whose variables and labels have the same names (and roles) of those as $\mathcal{S}$ — but where the values of variables are strings in $\Sigma^\star$ instead of natural numbers.
- The *statements* in this language are either
  - "$V \leftarrow \sigma V$," where V is a variable and $\sigma \in \Sigma$,
  - "$V \leftarrow V^-$," where V is a variable, or
  - "IF V ENDS σ GOTO L," where V is a variable, $\sigma \in \Sigma$, and L is a label.

  The effect of each statement is defined next.

Statement: $V \leftarrow \sigma V$

- If V is a variable whose current value is

$$\omega = \gamma_m \gamma_{m-1} \cdots \gamma_1 \gamma_0 \in \Sigma^\star$$

and $\sigma \in \Sigma$, then the execution of the statement "$V \leftarrow \sigma V$" changes the value of V to

$$\sigma \gamma_m \gamma_{m-1} \cdots \gamma_1 \gamma_0$$

— that is, σ is appended onto the *left* end of the string that is the value of V.

Statement: $V \leftarrow V^-$

- If V is a variable whose current value is the empty string, λ, then the execution of the statement "$V \leftarrow V^-$" does not change the value of V (or any other variable.).
- On the other hand, if the current value of V is

$$\omega = \gamma_m \gamma_{m-1} \cdots \gamma_1 \gamma_0$$

where $m \geq 0$. then execution of the above statement changes the value of V to

$$\gamma_m \gamma_{m-1} \cdots \gamma_2 \gamma_1$$

— that is, this statement removes the *rightmost* symbol from the current value of V.

Statement: IF V ENDS σ GOTO L

- If the current value of V is either the empty string, or a nonempty string whose *rightmost* symbol is *not* $\sigma \in \Sigma$ then the execution of the statement

  IF V ENDS σ GOTO L

  has no effect.
- Otherwise, the execution of this statement causes a jump to the *first* statement in the program with label L — with the execution ending if there is no such statement at all.

Pseudo-instruction: IF $V \neq \lambda$ GOTO L

The pseudo-instruction "IF $V \neq \lambda$ GOTO L" corresponds to the following macro.

IF V ENDS σ_1 GOTO L
IF V ENDS σ_2 GOTO L
IF V ENDS σ_3 GOTO L
$\vdots$
IF V ENDS σ_n GOTO L

Pseudo-instruction: $V \leftarrow \lambda$

The pseudo-instruction "$V \leftarrow \lambda$" corresponds to the following macro. It is assumed, as usual, that the label A used here does not appear anywhere else.

$$\begin{array}{ll} [A] & V \leftarrow V^- \\ & \text{IF } V \neq \lambda \text{ GOTO } A \end{array}$$

Pseudo-instruction: GOTO L

The pseudo-instruction "GOTO L" corresponds to the following. It is assumed here that V is a variable that does not appear anywhere else.

$V \leftarrow \lambda$
$V \leftarrow \sigma_1 V$
IF V ENDS σ_1 GOTO L

Pseudo-instruction: $V_1 \leftarrow V_2$

- A program corresponding to the pseudo-instruction "$V_1 \leftarrow V_2$" begins as follows. It is assumed here that $A_0, A_1, \ldots, A_n$, $B_0, B_1, \ldots, B_n$, and C_0 are labels that are not used elsewhere, that the variable V_3 is not used elsewhere either.

$$
\begin{array}{ll}
 & V_1 \leftarrow \lambda \\
 & V_3 \leftarrow \lambda \\
[A_0] & \text{IF } V_2 \text{ ENDS } \sigma_1 \text{ GOTO } A_1 \\
 & \text{IF } V_2 \text{ ENDS } \sigma_2 \text{ GOTO } A_2 \\
 & \text{IF } V_2 \text{ ENDS } \sigma_3 \text{ GOTO } A_3 \\
 & \qquad\qquad \vdots \\
 & \text{IF } V_2 \text{ ENDS } \sigma_n \text{ GOTO } A_n \\
 & \text{GOTO } B_0
\end{array}
$$

Pseudo-instruction: $V_1 \leftarrow V_2$

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

$$
\begin{array}{ll}
[A_i] & V_2 \leftarrow V_2^- \\
 & V_1 \leftarrow \sigma_i V_1 \\
 & V_3 \leftarrow \sigma_i V_3 \\
 & \text{GOTO } A_0
\end{array}
$$

Note that since control is transferred back to the beginning of the program, this sets both V_1 and V_3 to have the initial value of V_2 — while setting V_2 to be the empty string.

Pseudo-instruction: $V_1 \leftarrow V_2$

Exercise: Let k be a non-negative integer and suppose that V_2 has value

$$\alpha_1\alpha_2\ldots\alpha_k$$

(for $\alpha_1, \alpha_2, \ldots \alpha_k \in \Sigma$) before this subprogram is executed (so that V_2 has value λ if $k = 0$). Prove that the statement with label A_0 is reached and executed exactly $k + 1$ times during this execution of the macro — and that if $1 \leq i \leq k + 1$ then

- V_2 has value $\alpha_1\alpha_2\ldots\alpha_{k-i+1}$, and
- V_1 and V_3 each have value $\alpha_{k-i+2}\alpha_{k-i+3}\ldots\alpha_k$ (that is, the string λ if $i = 1$, and the string α_k if $i = 2$)

when statement with label A is executed for the i^{th} time.

It follows from this that V_2 has value λ and V_1 and V_3 each have value $\alpha_1\alpha_2\ldots\alpha_k$ when the statement with label A is executed for the $k + 1^{\text{st}}$ time. The unconditional branch to a statement with label B_0 will then be reached and executed.

Pseudo-instruction: $V_1 \leftarrow V_2$

- The program continues with similar blocks that serve to restore the value of V_2:

$$
\begin{array}{ll}
[B_0] & \text{IF } V_3 \text{ ENDS } \sigma_1 \text{ GOTO } B_1 \\
 & \text{IF } V_3 \text{ ENDS } \sigma_2 \text{ GOTO } B_2 \\
 & \text{IF } V_3 \text{ ENDS } \sigma_3 \text{ GOTO } B_3 \\
 & \qquad \vdots \\
 & \text{IF } V_3 \text{ ENDS } \sigma_n \text{ GOTO } B_n \\
 & \text{GOTO } C_0
\end{array}
$$

Pseudo-instruction: $V_1 \leftarrow V_2$

- Finally, the program ends with the following block of code for each integer i such that $1 \leq i \leq n$:

$$\begin{array}{ll} [B_i] & V_3 \leftarrow V_3^- \\ & V_2 \leftarrow \sigma_i V_2 \\ & \text{GOTO } B_0 \end{array}$$

Exercise: Confirm that this macro is correct — that is, its execution always halts and, when the execution halts, variable V_1 has the value that V_2 did before the execution began, while the values of all other variables are the same as before.

Equivalence of Computational Models

Once again, let $n = |\Sigma|$ and suppose that $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.

- In the previous lecture a mapping from symbols in Σ to $\mathbb{N}$ was defined by setting $\#(\sigma_i)$ to be i for every integer i such that $1 \leq i \leq n$.
- This was extended, to obtain a mapping from strings in $\Sigma^\star$ to $\mathbb{N}$, by setting $\#(\lambda)$ to be 0 and — for

$$\omega = \alpha_{m-1}\alpha_{m-2}\ldots\alpha_1\alpha_0$$

(where $\alpha_0, \alpha_1, \ldots, \alpha_{m-1} \in \Sigma$) — setting $\#(\omega)$ to be

$$\begin{aligned}\#(\omega) &= \#(\alpha_{m-1}\alpha_{m02}\ldots\alpha_0) \\ &= \#(\alpha_{m-1}) \cdot n^{m-1} + \alpha_{m-2} \cdot n^{m-2} + \cdots + \#(\alpha_0).\end{aligned}$$

- As argued in the previous lecture, every number $\ell \in \mathbb{N}$ represents *exactly one* string $\omega \in \Sigma^\star$ (so that $\#(\omega) = \ell$).

Equivalence of Computational Models

Consider a partial or total function $f : \mathbb{N}^k \to \mathbb{N}$ for a positive integer k.

- ***Definition:*** The function f is ***$\mathcal{S}_n$-computable*** if there exists an $\mathcal{S}_n$-program, $\mathcal{P}$, which satisfies the following properties.
  - For all 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 $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$, such that $\#(\omega_i) = x_i$ for $1 \leq i \leq k$, then this execution of $\mathcal{P}$ eventually ends, and the string $\zeta \in \Sigma^\star$ such that $\#(\zeta) = f(x_1, x_2, \ldots, x_k)$ is returned as output.
  - For all 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 $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ such that $\#(\omega_i) = x_i$ for $1 \leq i \leq k$, then this execution of $\mathcal{P}$ does not ever end, at all.

Equivalence of Computational Models

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 a positive integer. If f is $\mathcal{S}_n$-computable then f is $\mathcal{S}$-computable.

How This is Proved:

- Each instruction of an $\mathcal{S}_n$-program can be simulated using pseudo-instructions and macros for $\mathcal{S}$-programs that were introduced in the previous lecture, along with those that have been given above.

Equivalence of Computational Models

- For example, 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.

- This can be used to define an $\mathcal{S}$-program, $\widehat{\mathcal{P}}$, that simulates the execution of any given $\mathcal{S}_n$-program $\mathcal{P}$.
- Notions of ***state***, ***instantaneous description***, and ***computation*** (defined for $\mathcal{S}$-programs) can be modified — in a straightforward way — to obtain similar notions for $\mathcal{S}_n$-programs — which can be a applied to complete a proof. See the supplemental document for details.

Equivalence of Computational Models

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.

How This is Proved:

- Each instruction of an $\mathcal{S}$-program can be simulated using pseudo-instructions and macros for $\mathcal{S}_n$-programs. After this is shown, the proof of Claim #1 proceeds like the proof of Claim #1 — with the roles of $\mathcal{S}$-programs and $\mathcal{S}_n$-programs reversed. See the supplemental document for details.

Post-Turing Programs

Let us define a ***Post-Turing program*** to be a 3-tuple

$$\mathcal{P} = (\Sigma, \Gamma, \mathcal{C})$$

where

- $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ is a finite (and nonempty) ***input alphabet*** that does not include the "blank" symbol $\sqcup$;
- Γ is the ***tape alphabet*** — so that

  $$\Gamma = \{\sqcup, \sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_{n+1}, \ldots, \sigma_{n+m}\}$$

  for some nonnegative integer m — and $\Sigma \cup \{\sqcup\}$ is a subset of Γ;
- $\mathcal{C}$ is a sequence of instructions in a programming language $\mathcal{T}$ that will be defined shortly.

Post-Turing Programs: The Tape

- As with some kinds of simple Turing machines, information is stored on a one-dimensional two-way infinite *tape*. Each cell of the tape stores a symbol in Γ and all but a finite number of these cells store $\sqcup$.
- As with most Turing machines, the tape has a single *tape head* that points to one of the cells of the tape — and the only tape symbol that is "visible" is the symbol currently stored in the cell that the tape head points to.

Post-Turing Programs: Statements in $\mathcal{T}$

The programming language $\mathcal{T}$ includes four kinds of statements:

(a) PRINT σ, where $\sigma \in \Gamma$

(b) IF σ GOTO L, where $\sigma \in \Gamma$ and L is a label

(c) RIGHT

(d) LEFT

Note: We will assume that this language includes the same set of ***labels*** for statements as the programming languages $\mathcal{S}$ and $\mathcal{S}_n$.

Post-Turing Programs: Statements in $\mathcal{T}$

(a) If $\sigma \in \Gamma$ then execution of the statement

PRINT σ

causes the contents of the tape cell, pointed to by the tape head, to be replaced by σ. The tape head does not move.

Post-Turing Programs: Statements in $\mathcal{T}$

(b) If $\sigma \in \Gamma$ and the cell currently pointed to by the tape head stores a copy of σ then execution of the statement

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

causes (program) control to jump to the first statement in the program $\mathcal{C}$ with label L — or it causes the execution to end if there is no such statement. The tape head does not move.

On the other hand, this statement has no effect if the contents of the tape cell, pointed to by the tape head, is *not* a copy of σ.

Post-Turing Programs: Statements in $\mathcal{T}$

(c) Execution of the statement

RIGHT

causes the tape head to move one cell to the right, without changing the contents of the cell that is currently visible.

(d) Execution of the statement

LEFT

causes the tape head to move one cell to the left, without changing the contents of the cell that is currently visible.

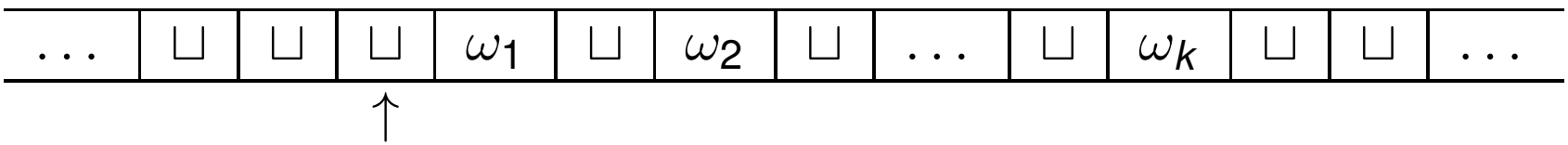
Post-Turing Programs: Initial Configuration of Tape

We will consider Post-Turing programs that compute (partial or total) functions

$$f : (\Sigma^\star)^k \to \Sigma^\star.$$

Given inputs $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ (in order to compute $f(\omega_1, \omega_2, \ldots, \omega_k)$,

- the tape should store the strings $\omega_1, \omega_2, \ldots, \omega_k$, in order, with a single blank between each string, and surrounded by infinitely many blanks on either side;
- the tape head to point to the blank immediately to the ***left*** of the leftmost cell of ω_1.



Post-Turing Programs: Initial Configuration of Tape

- If one or more of $\omega_1, \omega_2, \ldots, \omega_k$ is the empty string, λ, then the "non-blank portion of the tape" may contain two (or more) $\sqcup$'s in a row. Furthermore, if $\omega_1 = \lambda$ and ω_2 is nonempty, then the tape head will initially point to the *second* blank to the left of the first non-blank symbol on the tape.... and so on.

Post-Turing Programs: Strong Computation

Definition: A Post-Turing program ***strongly computes*** a (partial) function

$$f : (\Sigma^\star)^k \to \Sigma^\star$$

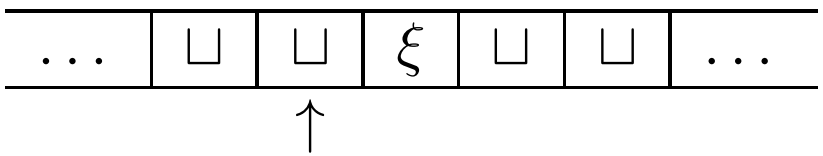
if the following conditions are satisfied.

(a) $\Gamma = \Sigma \cup \{\sqcup\}$ — so that the statements in $\mathcal{C}$ only refer to the symbols in $\Sigma \cup \{\sqcup\}$;

(b) If $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ and

$$f(\omega_1, \omega_2, \ldots, \omega_k) = \xi \in \Sigma^\star$$

then execution of the program (with these inputs) eventually halts. When it does so, the tape stores ξ, surrounded with infinitely many blanks on either side, and with the tape head pointing to the blank immediately to the left of ξ:



Post-Turing Programs: Strong Computation

(c) Finally, if $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ and $f(\omega_1, \omega_2, \ldots, \omega_k)$ is undefined, then the execution of the program with these inputs never halts, at all.

Note: This might also be called "strict computation" of the function f.

Post-Turing Programs: Weak Computation

Definition: A Post-Turing program ***weakly computes*** a (partial) function

$$f : (\Sigma^\star)^k \to \Sigma^\star$$

if the following conditions are satisfied, instead.

(a) If $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ and

$$f(\omega_1, \omega_2, \ldots, \omega_k) = \xi \in \Sigma^\star$$

then the execution of the program (with these inputs) eventually halts and, on termination, the sequence of symbols on the tape ***that belong to*** $\boldsymbol{\Sigma}$ form the string ξ when read from left to right.

There may be other symbols in $\Gamma \setminus \Sigma$ on the tape, the symbols in ξ do not have be in one contiguous block on the tape, and there is no requirement concerning the final location of the tape head.

Post-Turing Programs: Weak Computation

(b) On the other hand, if $\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$ and $f(\omega_1, \omega_2, \ldots, \omega_k)$ is undefined, then the execution of the program with these inputs never halts, at all.

Note: This might also be called "computation" of the function f.

Post-Turing Programs: Computation of Integer Functions

- Post-Turing programs can now be viewed as computing functions $f : \mathbb{N}^k \to \mathbb{N}$ in the same way as programs written in the language $\mathcal{S}_n$: In order to compute the value of such a function on inputs

$$\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathbb{N}$$

the Post-Turing program should be run on input strings

$$\omega_1, \omega_2, \ldots, \omega_k \in \Sigma^\star$$

such that $\alpha_i = \#(\omega_i)$ for $1 \leq i \leq k$.

Post-Turing Programs: Computation of Integer Functions

- If computation terminates, with a string $\xi \in \Sigma^\star$ being computed then (one should require that)

$$f(\alpha_1, \alpha_2, \ldots, \alpha_k) = \#(\xi).$$

Otherwise (one should require that) $f(\alpha_1, \alpha_2, \ldots, \alpha_k)$ is undefined.

Simulating $\mathcal{S}_n$-Programs using Post-Turing Programs

Suppose again that $n \geq 1$, $m \geq 0$, that

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

and

$$\Gamma = \{\sqcup, \sigma_1, \sigma_2, \ldots, \sigma_n, \sigma_{n+1}, \ldots, \sigma_{n+m}\}.$$

The following ***pseudo-instructions*** and ***macros***, for Post-Turing programs, will eventually be useful to prove that if $f : \mathbb{N}^k \to \mathbb{N}$ is an $\mathcal{S}$-computable (partial or total) function, then there exists a Post-Turing program that strongly computes this function.

Pseudo-Instruction: GOTO *L*

If *L* is a label then the pseudo-instruction "GOTO *L*" corresponds to the macro

IF $\sqcup$ GOTO *L*
IF σ_1 GOTO *L*
IF σ_2 GOTO *L*
IF σ_3 GOTO *L*
$\vdots$
IF σ_{n+m} GOTO *L*

Pseudo-Instruction: RIGHT TO NEXT BLANK

The pseudo-instruction “RIGHT TO NEXT BLANK” corresponds to the following macro — assuming that the label *A* is not used elsewhere, and *E* is first used to label the instruction after this pseudo-instruction (if such a statement exists).

[*A*] RIGHT
IF ⊔ GOTO *E*
GOTO *A*

Note: This is defined to guarantee that the tape head *always* moves at least one symbol to the right — in order to reach the “next” ⊔ in that direction.

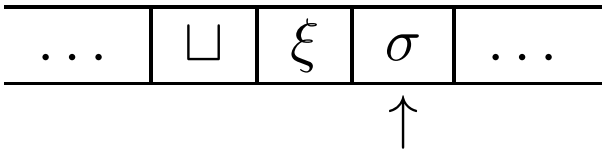
Pseudo-Instruction: LEFT TO NEXT BLANK

Similarly, the pseudo-instruction “LEFT TO NEXT BLANK” corresponds to the following macro (under the same conditions).

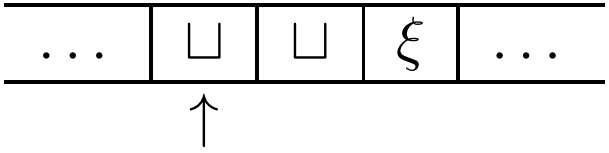
$$
\begin{array}{ll}
[A] & \text{LEFT} \\
 & \text{IF } \sqcup \text{ GOTO } E \\
 & \text{GOTO } A
\end{array}
$$

Pseudo-Instruction: MOVE BLOCK RIGHT

Consider a pseudo-instruction "MOVE BLOCK RIGHT" that is intended to do the following: If $\sigma \in \Gamma$, $\xi \in (\Gamma \setminus \{\sqcup\})^\star$ (so that ξ is a string that does not include any $\sqcup$'s) and the region near the tape head is as follows *before* the execution of this instruction,



then this region of the tape should be as follows *after* the execution of this pseudo-instruction (with σ erased, ξ moved over, another $\sqcup$ added to the left of it, and nothing else changed):



Pseudo-Instruction: MOVE BLOCK RIGHT

A macro corresponding to this pseudo-instruction begins as follows.

$$
\begin{array}{ll}
[C] & \text{LEFT} \\
 & \text{IF } \sqcup \text{ GOTO } A_0 \\
 & \text{IF } \sigma_1 \text{ GOTO } A_1 \\
 & \text{IF } \sigma_2 \text{ GOTO } A_2 \\
 & \text{IF } \sigma_3 \text{ GOTO } A_3 \\
 & \qquad \vdots \\
 & \text{IF } \sigma_{n+m} \text{ GOTO } A_{n+m}
\end{array}
$$

This effectively reads and remembers the rightmost cell of ξ that has not been copied yet, or confirms that all of ξ has been copied.

Pseudo-Instruction: MOVE BLOCK RIGHT

The macro continues with a block as follows for $1 \leq i \leq n + m$ — in order to shift the non-blank symbol that is now visible over to the right, reposition the tape head, and continue:

$$
\begin{array}{ll}
[A_i] & \text{RIGHT} \\
 & \text{PRINT } \sigma_i \\
 & \text{LEFT} \\
 & \text{GOTO } C
\end{array}
$$

Pseudo-Instruction: MOVE BLOCK RIGHT

Finally, this macro ends with the following — which writes the new blank that should be to the left of ξ and repositions the tape head:

$$\begin{array}{ll} [A_0] & \text{RIGHT} \\ & \text{PRINT } \sqcup \\ & \text{LEFT} \end{array}$$

Exercise: Confirm that this macro is consistent with this pseudo-instruction — that is, it has the desired effects.

Pseudo-Instruction: ERASE A BLOCK

Consider a pseudo-instruction "ERASE A BLOCK," which should cause the tape head to move to the right, erasing (that is, overwriting with ␣'s) everything between (but not including) the cell of the tape where it begins and the next blank to the right.

This corresponds to the following macro — assuming that E is a label that is only once and that labels the instruction after this (or is not used at all, if this is the last pseudo-instruction):

$[A]$ RIGHT
IF ␣ GOTO E
PRINT ␣
GOTO A

Naming Convention

From now on, if $\ell \in \mathbb{N}$ then writing $[\ell]$ after the name of a pseudo-instruction indicates that this pseudo-instruction should be repeated ℓ times.

- For example,

  RIGHT TO NEXT BLANK [4]

  is short for

  RIGHT TO NEXT BLANK
  RIGHT TO NEXT BLANK
  RIGHT TO NEXT BLANK
  RIGHT TO NEXT BLANK

Simulating $\mathcal{S}_n$ in $\mathcal{T}$

Claim #3: If $f : (\Sigma^\star)^k \to \Sigma^\star$ is a (partial or total) function that is $\mathcal{S}_n$-computable, for a positive integer n, then there exists a Post-Turing program that strongly computes f.

Proof:

- Suppose $f : (\Sigma^\star)^k \to \Sigma^\star$ is a (partial or total) function that is $\mathcal{S}_n$-computable — so that f is computed using some program $\mathcal{P}$ in the language $\mathcal{S}_n$.

Simulating $\mathcal{S}_n$ in $\mathcal{T}$

- We may assume that the only *input variables* mentioned in $\mathcal{P}$ are

  $$X_1, X_2, \ldots, X_k$$

  since all others have initial value 0, so that they can be replaced by local variables.
- We may also assume that no label is used to label more than one statement in $\mathcal{P}$, since uses of a label after the first can simply be eliminated without changing what the program does.

Simulating $\mathcal{S}_n$ in $\mathcal{T}$

- Suppose that $\ell \geq 0$ and that the only *local* variables mentioned in the program are

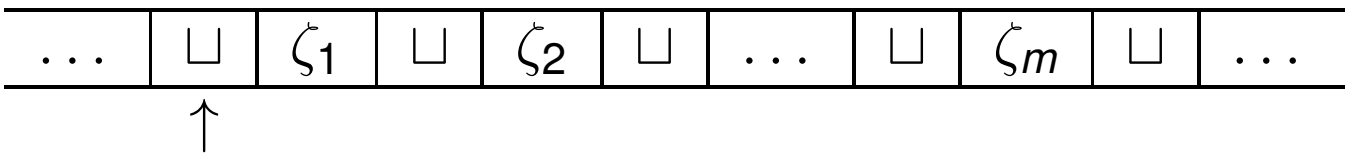
$$Z_1, Z_2, \ldots, Z_\ell.$$

- Then the only other variable that might be mentioned in the program is the output variable Y.
- With that noted let $m = k + \ell + 1$, and, for $1 \leq i \leq m$, let

$$V_i = \begin{cases} X_i & \text{if } 1 \leq i \leq k, \\ Z_{i-k} & \text{if } k+1 \leq i \leq k+\ell, \\ Y & \text{if } i = k + \ell + 1 = m. \end{cases}$$

Simulating $\mathcal{S}_n$ in $\mathcal{T}$

- We will prove the result by writing a Post-Turing program — whose tape alphabet is $\Gamma = \Sigma \cup \{\sqcup\}$ — that uses its tape to store the current values of the above variables and simulates the execution of $\mathcal{P}$, one step at a time.
- In particular, if V_i has value $\zeta_i \in \Sigma^\star$, for $1 \leq i \leq m$ after a step of $\mathcal{P}$ then — after the simulation of this step — the non-blank portion of the Post-Turing program's tape will be as follows.



Simulating $\mathcal{S}_n$ in $\mathcal{T}$: Initialization

No Initialization of the Simulation is Needed:

- Notice that, since the initial values of $Z_1, Z_2, \ldots, Z_\ell$ and Y are all $\sqcup$, no "initialization" is needed: The tape's contents and the location of the tape head are already all what and where they should be, before the first step of the program should be simulated.

Simulating $\mathcal{S}_n$ in $\mathcal{T}$: Structure of $\mathcal{C}$

Structure of the Post-Turing Program's Code:

Suppose that a program $\mathcal{P}$ to be simulated consists of a sequence of instructions

$$\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_t$$

- This will be simulated using a Post-Turing program $\mathcal{C}$, consisting of a sequence

  $$M(\mathcal{S}_1), M(\mathcal{S}_2), \ldots, M(\mathcal{S}_t), \text{CLEANUP}$$

  of subprograms, as described next.

Simulating $\mathcal{S}_n$ in $\mathcal{T}$: Structure of $\mathcal{C}$

For $1 \leq i \leq t$, it will generally be necessary to assign a ***label*** to the first statement in the subprogram $M(\mathcal{S}_i)$ that is used to simulate an execution of the step S_i.

- If statement $\mathcal{S}_i$ has a label in $\mathcal{P}$ then the same label should be used for the first statement in $M(\mathcal{S}_i)$, so that transfers of control to $\mathcal{S}_i$ in $\mathcal{P}$ are being replaced with transfers of control to the beginning of $M(\mathcal{S}_i)$.
- A new label (that is not used elsewhere) should be used for the first statement in $M(\mathcal{S}_i)$ if $\mathcal{S}_i$ does not have a label in $\mathcal{P}$.

New labels (which are not used anywhere else) should be used for any other statements in $M(\mathcal{S}_i)$ that need them.

Let L_C be another label that is not used here. This will be used to label the first statement in the CLEANUP subprogram.

Case: $\mathcal{S}_i$ is "$V_h \leftarrow \sigma_j V_h$"

Simulating Instruction "$V_h \leftarrow \sigma_j V_h$"

- If $\mathcal{S}_i$ is as above then $M(\mathcal{S}_i)$ should begin with statements that move the tape head to the leftmost blank to the *right* of the portion of the tape representing the values of variables.

  The symbols representing the values of the rightmost $m - h + 1$ variables should then be shifted over to the right, to make room for the new symbol to be printed.

  After this is printed, the tape head should be repositioned:

  RIGHT TO NEXT BLANK [m]
  MOVE BLOCK RIGHT [$m - h + 1$]
  RIGHT
  PRINT σ_j
  LEFT TO NEXT BLANK [h]

Case: $\mathcal{S}_i$ is "$V_h \leftarrow V_h^-$"

Simulating Instruction "$V_h \leftarrow V_h^-$"

- A complication, when simulating this instruction, is that the contents of the tape should not be changed if the value of V_h is already the empty string — so we need to check that the symbol to be removed is non-blank before deleting it.

Case: $\mathcal{S}_i$ is "$V_h \leftarrow V_h^-$"

With that noted, $M(\mathcal{S}_i)$ should be as follows when $\mathcal{S}_i$ is the instruction "$V_h \leftarrow V_h^-$":

	RIGHT TO NEXT BLANK [h]
	LEFT
	IF $\sqcup$ GOTO C
	MOVE BLOCK RIGHT [h]
	RIGHT
	GOTO E
[C]	LEFT TO NEXT BLANK [$h-1$]

Once again, E should label the first statement of the next subprogram $\mathcal{S}_{i+1}$ if $i < t$. E should be the label L_C if $i = t$.

Case: $\mathcal{S}_i$ is "IF V_h ENDS σ_j GOTO L"

Simulating Instruction "IF V_h ENDS σ_j GOTO L"

- If $\mathcal{S}_i$ is as above then $M(\mathcal{S}_i)$ should be as follows:

	RIGHT TO NEXT BLANK [h]
	LEFT
	IF σ_j GOTO C
	GOTO D
[C]	LEFT TO NEXT BLANK [h]
	GOTO L
[D]	RIGHT
	LEFT TO NEXT BLANK [h]

Note that if the label L is not used to label any statements in the program then it should be replaced by the label, L_C, used for the first statement in the CLEANUP subprogram.

CLEANUP

***Implementing* CLEANUP:**

- Finally, CLEANUP simply needs to erase any non-blank symbols representing the first $k + \ell = m - 1$ variables. Since "ERASE A BLOCK" leaves the tape head at the blank following the block that has been erased (or simply shifts the tape head over by one, if this block was empty), CLEANUP is simply the following:

$$[L_C] \quad \text{ERASE A BLOCK } [m - 1]$$

Simulating $\mathcal{S}_n$ in $\mathcal{T}$

The correctness of the simulation is reasonably easy to establish by induction on the number of steps that have been simulated so far, after checking that all of the above subprograms do what they should.

This completes the proof of the theorem. □

Simulating $\mathcal{T}$ in $\mathcal{S}$

Recall — as noted above — that Post-Turing programs can also be viewed as programs that compute functions $f : \mathbb{N}^k \to \mathbb{N}$ in a reasonably straightforward way.

Claim #4: If $f : \mathbb{N}^k \to \mathbb{N}$ is a (partial or total) function that is weakly computed by a Post-Turing program, then f is $\mathcal{S}$-computable. as well.

Proof:

- Let

$$\mathcal{P} = (\Sigma, \Gamma, \mathcal{C})$$

be a Post-Turing program that weakly computes the (partial or total) function f.

Simulating $\mathcal{T}$ in $\mathcal{S}$

- Let $n \geq 1$ and $m \geq 0$ such that

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

and

$$\Gamma = \{\sqcup, \sigma_1, \sigma_2, \ldots, \sigma_{m+n}\}.$$

- ***Note:*** In the construction to follow, the encoding of $\sqcup$ will be set to $n + m + 1$ (that is, we will pretend that $\sqcup = \sigma_{n+m+1}$).
- ***Method:*** We will describe an $\mathcal{S}$-program, $\mathcal{Q}$, that simulates an execution of $\mathcal{P}$.

Simulating $\mathcal{T}$ in $\mathcal{S}$: Representing the Tape (and Tape Head Location)

Three (local) variables in $\mathcal{Q}$ — L, H, and R — will be used to represent the contents of the tape before and after the simulation of each step taken by the program $\mathcal{P}$:

- If there are no non-blank symbols to the left of the tape head then $L = 0$, the encoding of the empty string.

  Otherwise L is the encoding of the string in $\Gamma^\star$ that begins with the *leftmost* non-blank symbol on the tape, ends with the symbol in the cell immediately to the left of the tape head, and has all the intermediate symbols in-between.

- H stores the encoding of the symbol that is currently visible (that is, is stored at the location of the tape head).

Simulating $\mathcal{T}$ in $\mathcal{S}$: Representing the Tape (and Tape Head Location)

- If there are no non-blank symbols to the right of the tape head then $R = 0$, the encoding of the empty string.

  Otherwise R is the encoding of the string in $\Sigma^\star$ that begins with the symbol immediately to the *right* of the tape head on the tape, ends with the rightmost non-blank symbol on the tape, and has all the intermediate symbols in between.

Simulating $\mathcal{T}$ in $\mathcal{S}$: Structure of $\mathcal{Q}$

- The program $\mathcal{Q}$ will have three parts:

  BEGINNING
  MIDDLE
  END

- Recall that a variety of functions involving strings were shown to be primitive recursive — and, therefore, computable — during the previous lectures.

  Macros for these functions will be used in order to describe the above parts of $\mathcal{Q}$.

BEGINNING

This part of the program begins by setting R to be the encoding — as a string in $\Gamma^\star$ — of the portion of the tape (to the right of the tape head) encoding the inputs $X_1, X_2, \ldots, X_k$. Since these are given as encodings of strings in $\Sigma^\star$, a macro to compute the primitive recursive function $\text{upchange}_{n,n+m+1}$, described in the previous lecture, will be useful:

$$
\begin{aligned}
&R \leftarrow \text{upchange}_{n,n+m+1}(X_1)\\
&R \leftarrow \text{concat}(R, n+m+1)\\
&R \leftarrow \text{concat}(R, \text{upchange}_{n,m+m+1}(X_2))\\
&R \leftarrow \text{concat}(R, n+m+1)\\
&\qquad\qquad\vdots\\
&R \leftarrow \text{concat}(R, n+m+1)\\
&R \leftarrow \text{concat}(R, \text{upchange}_{n,n+m+1}(X_k))
\end{aligned}
$$

BEGINNING

However, if X_k was the encoding 0 of the empty string then some trailing blanks can be removed from the string encoded by R. The following code attends to this.

$$
\begin{array}{ll}
[A] & Z_1 \leftarrow \mathrm{rtend}(R) \\
 & \text{IF } Z_1 = n + m + 1 \text{ GOTO } B \\
 & \text{GOTO } C \\
[B] & R \leftarrow \mathrm{rtrunc}(R) \\
 & \text{GOTO } A
\end{array}
$$

The initial values of L and H must also be set. This is easy, because L should encode the empty string and H should be the encoding of $\sqcup$:

$$
\begin{array}{ll}
[C] & L \leftarrow 0 \\
 & H \leftarrow n + m + 1
\end{array}
$$

MIDDLE

If the sequence $\mathcal{C}$ of instructions the Post-Turing program P is

$$\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_t$$

then MIDDLE should consist of the sequence of subprograms

$$M(\mathcal{S}_1), M(\mathcal{S}_2), \ldots, M(\mathcal{S}_t)$$

— with labels renamed and added as needed, for correct macro expansion.

Case: $\mathcal{S}_i$ is "PRINT σ_j"

***Simulating the Instruction* "PRINT σ_j"**

In $\mathcal{S}_i$ is as above then it suffices to set $M(\mathcal{S}_i)$ to be the subprogram

$$H \leftarrow j$$

Case: $\mathcal{S}_i$ is "IF σ_j GOTO L"

***Simulating the Instruction* "IF σ_j GOTO L**

If $\mathcal{S}_i$ is as above then $\mathcal{S}_i$ should be a subprogram obtained (after macro expansion) corresponding to the pseudo-instruction

$$\text{IF } H = j \text{ GOTO } L$$

If L does not label any statement in this program then it should be replaced by a label L_E that labels the first statement in END.

Case: $\mathcal{S}_i$ is "RIGHT"

***Simulating the Instruction* "RIGHT"**

If $\mathcal{S}_i$ is above then $M(\mathcal{S}_i)$ should be what is obtained from the following after macro expansion.

	IF $L \neq 0$ GOTO A
	IF $H \neq n + m + 1$ GOTO A
	GOTO B
$[A]$	$L \leftarrow \text{concat}(L, H)$
$[B]$	IF $R \neq 0$ GOTO C
	$H \leftarrow n + m + 1$
	GOTO D
$[C]$	$H \leftarrow \text{ltend}(R)$
$[D]$	$R \leftarrow \text{ltrunc}(R)$

Case: $\mathcal{S}_i$ is "RIGHT'

Special cases that are checked for, and handled here, are as follows:

- L encodes the empty string and H encodes $\sqcup$, in which case the value of L should not be changed, and
- R encodes the empty string, in which case the value of H should be set to the encoding of $\sqcup$ instead of 0.

Case: $\mathcal{S}_i$ is "LEFT"

***Simulating the Instruction* "LEFT"**

If $\mathcal{S}_i$ is as above then $M(\mathcal{S}_i)$ should be what is obtained from the following after macro expansion.

	IF $R \neq 0$ GOTO A
	IF $H \neq n + m + 1$ GOTO A
	GOTO B
$[A]$	$R \leftarrow \text{concat}(H, R)$
$[B]$	IF $L \neq 0$ GOTO C
	$H \leftarrow n + m + 1$
	GOTO D
$[C]$	$H \leftarrow \text{rtend}(L)$
$[D]$	$L \leftarrow \text{rtrunc}(L)$

Case: $\mathcal{S}_i$ is "LEFT"

Special cases that are checked for, and handled here, are as follows:

- R encodes the empty string and H encodes $\sqcup$, in which case the value of R should not be changed, and
- L encodes the empty string, in which case the value of H should be set to the encoding of $\sqcup$ instead of 0.

END

Finally, recall that if $\mathcal{P}$ weakly computes f then, on termination, the encoding of the output is what is obtained by discarding all symbols in the tape that do not belong to Σ and reading what remains in order from left to right.

The value of Y is, therefore, correctly set by the following.

$$
\begin{array}{ll}
[L_E] & Z_1 \leftarrow \mathrm{concat}(L, H) \\
 & Z_2 \leftarrow \mathrm{concat}(Z_1, R) \\
 & Y \leftarrow \mathrm{downchange}_{n+m+1,n}(Z_2)
\end{array}
$$

Simulating $\mathcal{T}$ in $\mathcal{S}$

The correctness of the simulation is reasonably easy to establish by induction on the number of steps that have been simulated so far, after checking that all of the above programs do what they should.

This completes the proof of the claim. □

Conclusions

Theorem: Let $f : \mathbb{N}^k \to \mathbb{N}$ be a (partial or total) function. Then the following are equivalent:

(a) f is $\mathcal{S}$-computable — that is, there is an $\mathcal{S}$-program that computes f.

(b) There is a $\mathcal{S}_n$-program that computes f for every integer $n \geq 1$.

(c) There is a $\mathcal{S}_n$-program that computes f for some integer $n \geq 1$.

(d) There is a Post-Turing program that strongly computes f.

(e) There is a Post-Turing program that weakly-computes f.

Conclusions

Proof:

- $(a) \Rightarrow (b)$ by Claim #2.
- $(b) \Rightarrow (c)$ should be pretty clear!
- $(c) \Rightarrow (d)$ by Claim #3.
- $(d) \Rightarrow (e)$ should also be pretty clear!
- $(e) \Rightarrow (a)$ by Claim #4.