Now let us look at the execution of a very simple T program. The arrow in the program indicates the next instruction to be executed.

PRINT s\textsubscript{1}  
LEFT  
PRINT s\textsubscript{2}  
LEFT  
B x  
↑

Post-Turing Programs

Now let us look at the execution of a very simple T program. The arrow in the program indicates the next instruction to be executed.

PRINT s\textsubscript{1}  
→  
PRINT s\textsubscript{2}  
LEFT  

B s\textsubscript{1} x  
↑

Post-Turing Programs

Now let us look at the execution of a very simple T program. The arrow in the program indicates the next instruction to be executed.

PRINT s\textsubscript{1}  
LEFT  
PRINT s\textsubscript{2}  
→  
LEFT  

s\textsubscript{1} x  
↑

Post-Turing Programs

Now let us look at the execution of a very simple T program. The arrow in the program indicates the next instruction to be executed.

PRINT s\textsubscript{1}  
LEFT  
PRINT s\textsubscript{2}  
→  
LEFT  
s\textsubscript{2} s\textsubscript{1} x  
↑

Post-Turing Programs

What function did the previous program compute?

**Definition:** Let f(x\textsubscript{1}, ..., x\textsubscript{m}) be an m-ary partial function on the alphabet \( A = \{ s\textsubscript{1}, ..., s\textsubscript{n} \} \).
Then the program \( \phi \) in the Post-Turing language \( T \) is said to compute \( f \) if when started in the tape configuration

\( B x\textsubscript{1} B x\textsubscript{2} B x\textsubscript{3} ... B x\textsubscript{m} \)

↑

it eventually halts if and only if \( f(x\textsubscript{1}, ..., x\textsubscript{m}) \) is defined and if, on halting, the string \( f(x\textsubscript{1}, ..., x\textsubscript{m}) \) can be read from the tape by ignoring all symbols except \( s\textsubscript{1}, ..., s\textsubscript{n} \).
Post-Turing Programs
Notice that this definition allows \( \varphi \) to contain instructions that mention symbols other than \( s_1, \ldots, s_n \).

**We further define the following:**
The program \( \varphi \) will be said to compute \( f \) **strictly** if two additional conditions are met:
1. no instruction in \( \varphi \) mentions any other symbol than \( s_0, s_1, \ldots, s_n \);
2. whenever \( \varphi \) halts, the tape configuration is
   \[ \ldots B B y B B B \uparrow \]
where the string \( y \) contains no blanks.

Do you remember our simple example program?
It turned the tape configuration

\[ B x \uparrow \]

into the configuration

\[ B s_2 s_1 x \uparrow \]

So what does it compute?
It strictly computes the function \( f(x) = s_2 s_1 x \).

Another example (\( A = \{s_1\} \)):

\[ [A] \quad \text{RIGHT} \quad B \quad s_1 \quad s_1 \]

\[ \rightarrow \quad \text{IF B GOTO E} \quad \uparrow \quad \text{PRINT M} \]

\[ [B] \quad \text{RIGHT} \quad \text{IF s_1 GOTO B} \]

\[ [C] \quad \text{RIGHT} \quad \text{IF s_1 GOTO C} \quad \text{PRINT s_1} \]

\[ [D] \quad \text{LEFT} \quad \text{IF s_1 GOTO D} \quad \text{IF B GOTO D} \quad \text{PRINT s_1} \quad \text{IF s_1 GOTO A} \]

Another example (\( A = \{s_1\} \)):

\[ [A] \quad \text{RIGHT} \quad B \quad s_1 \quad s_1 \]

\[ \rightarrow \quad \text{IF B GOTO E} \quad \uparrow \quad \text{PRINT M} \]

\[ [B] \quad \text{RIGHT} \quad \text{IF s_1 GOTO B} \]

\[ [C] \quad \text{RIGHT} \quad \text{IF s_1 GOTO C} \quad \text{PRINT s_1} \]

\[ [D] \quad \text{LEFT} \quad \text{IF s_1 GOTO D} \quad \text{IF B GOTO D} \quad \text{PRINT s_1} \quad \text{IF s_1 GOTO A} \]

Another example (\( A = \{s_1\} \)):

\[ [A] \quad \text{RIGHT} \quad B \quad M \quad s_1 \]

\[ \rightarrow \quad \text{IF B GOTO E} \quad \uparrow \quad \text{PRINT M} \]

\[ [B] \quad \text{RIGHT} \quad \text{IF s_1 GOTO B} \]

\[ [C] \quad \text{RIGHT} \quad \text{IF s_1 GOTO C} \quad \text{PRINT s_1} \]

\[ [D] \quad \text{LEFT} \quad \text{IF s_1 GOTO D} \quad \text{IF B GOTO D} \quad \text{PRINT s_1} \quad \text{IF s_1 GOTO A} \]
Post-Turing Programs

Another example ($A = \{s_1\}$):

[A] RIGHT B M s_1
  IF B GOTO E ↑
  PRINT M
  [B] RIGHT
  IF s_1 GOTO B
  [C] RIGHT
  IF s_1 GOTO C
  PRINT s_1
  [D] LEFT
  IF s_1 GOTO D
  IF B GOTO D
  PRINT s_1
  IF s_1 GOTO A

Post-Turing Programs

Another example ($A = \{s_1\}$):

[A] RIGHT B M s_1
  IF B GOTO E ↑
  PRINT M
  → [B] RIGHT
  IF s_1 GOTO B
  [C] RIGHT
  IF s_1 GOTO C
  PRINT s_1
  [D] LEFT
  IF s_1 GOTO D
  IF B GOTO D
  PRINT s_1
  IF s_1 GOTO A

Post-Turing Programs

Another example ($A = \{s_1\}$):

[A] RIGHT B M s_1 B
  IF B GOTO E ↑
  PRINT M
  [B] RIGHT
  IF s_1 GOTO B
  [C] RIGHT
  IF s_1 GOTO C
  PRINT s_1
  [D] LEFT
  IF s_1 GOTO D
  IF B GOTO D
  PRINT s_1
  IF s_1 GOTO A

Post-Turing Programs

Another example ($A = \{s_1\}$):

[A] RIGHT B M s_1 B B
  IF B GOTO E ↑
  PRINT M
  [B] RIGHT
  IF s_1 GOTO B
  [C] RIGHT
  IF s_1 GOTO C
  PRINT s_1
  [D] LEFT
  IF s_1 GOTO D
  IF B GOTO D
  PRINT s_1
  IF s_1 GOTO A
Post-Turing Programs

Another example (A = \{s_1\}):  

\[
\begin{align*}
&A: \text{RIGHT} \quad B \ M \ s_1 \ B \ s_1 \\
&\quad \text{IF } B \text{ GOTO E} \quad \uparrow \\
&\quad \text{PRINT M} \\
&B: \text{RIGHT} \\
&\quad \text{IF } s_1 \text{ GOTO B} \\
&C: \text{RIGHT} \\
&\quad \text{IF } s_1 \text{ GOTO C} \\
&\quad \text{PRINT } s_1 \\
&D: \text{LEFT} \\
&\quad \text{IF } s_1 \text{ GOTO D} \\
&\quad \text{IF } B \text{ GOTO D} \\
&\quad \text{PRINT } s_1 \\
&\quad \text{IF } s_1 \text{ GOTO A}
\end{align*}
\]
Post-Turing Programs

Another example (A = {s1}):  

\[
\begin{align*}
[A] & : \text{RIGHT} \quad B \quad \text{M} \quad s_1 \quad B \quad s_1 \\
& \quad \text{IF B GOTO E} \quad \uparrow \quad \text{PRINT M} \\
[B] & : \text{RIGHT} \quad \text{IF } s_1 \quad \text{GOTO B} \\
[C] & : \text{RIGHT} \quad \text{IF } s_1 \quad \text{GOTO C} \quad \text{PRINT } s_1 \\
[D] & : \text{LEFT} \quad \text{IF } s_1 \quad \text{GOTO D} \quad \text{IF B GOTO D} \quad \text{PRINT } s_1 \quad \text{IF } s_1 \quad \text{GOTO A} \\
\end{align*}
\]

Post-Turing Programs

Another example (A = {s1}):  

\[
\begin{align*}
[A] & : \text{RIGHT} \quad B \quad s_1 \quad B \quad s_1 \\
& \quad \text{IF B GOTO E} \quad \uparrow \quad \text{PRINT M} \\
[B] & : \text{RIGHT} \quad \text{IF } s_1 \quad \text{GOTO B} \\
[C] & : \text{RIGHT} \quad \text{IF } s_1 \quad \text{GOTO C} \quad \text{PRINT } s_1 \\
[D] & : \text{LEFT} \quad \text{IF } s_1 \quad \text{GOTO D} \quad \text{IF B GOTO D} \quad \text{PRINT } s_1 \quad \text{IF } s_1 \quad \text{GOTO A} \\
\end{align*}
\]
Post-Turing Programs

Another example ($A = \{s_1\}$):

[A] RIGHT B s₁ M B s₁  
IF B GOTO E  
PRINT M  
\[\uparrow\]

[B] RIGHT  
IF s₁ GOTO B  
[C] RIGHT  
IF s₁ GOTO C  
PRINT s₁  
[D] LEFT  
IF s₁ GOTO D  
IF B GOTO D  
PRINT s₁  
IF s₁ GOTO A

Post-Turing Programs

Another example ($A = \{s₁\}$):

[A] RIGHT B s₁ M B s₁  
IF B GOTO E  
PRINT M  
\[\uparrow\]

[B] RIGHT  
IF s₁ GOTO B  
[C] RIGHT  
IF s₁ GOTO C  
PRINT s₁  
[D] LEFT  
IF s₁ GOTO D  
IF B GOTO D  
PRINT s₁  
IF s₁ GOTO A

Post-Turing Programs

Another example ($A = \{s₁\}$):

[A] RIGHT B s₁ M B s₁  
IF B GOTO E  
PRINT M  
\[\uparrow\]

[B] RIGHT  
IF s₁ GOTO B  
[C] RIGHT  
IF s₁ GOTO C  
PRINT s₁  
[D] LEFT  
IF s₁ GOTO D  
IF B GOTO D  
PRINT s₁  
IF s₁ GOTO A

Post-Turing Programs

Another example ($A = \{s₁\}$):

[A] RIGHT B s₁ M B s₁  
IF B GOTO E  
PRINT M  
\[\uparrow\]

[B] RIGHT  
IF s₁ GOTO B  
[C] RIGHT  
IF s₁ GOTO C  
PRINT s₁  
[D] LEFT  
IF s₁ GOTO D  
IF B GOTO D  
PRINT s₁  
IF s₁ GOTO A
Post-Turing Programs

Another example (A = \{s_1\}):

[A] RIGHT B s_1 M B s_1 B
IF B GOTO E
PRINT s_1
[B] RIGHT
IF s_1 GOTO B
[C] RIGHT
IF s_1 GOTO C
PRINT s_1
→ [D] LEFT
IF s_1 GOTO D
IF B GOTO D
PRINT s_1
IF s_1 GOTO A

Post-Turing Programs

Another example (A = \{s_1\}):

[A] RIGHT B s_1 M B s_1 B
IF B GOTO E
PRINT s_1
[B] RIGHT
IF s_1 GOTO B
[C] RIGHT
IF s_1 GOTO C
PRINT s_1
→ [D] LEFT
IF s_1 GOTO D
IF B GOTO D
PRINT s_1
IF s_1 GOTO A
Post-Turing Programs

Another example (A = \{s_1\}):

\[
\begin{align*}
[A] & \quad \text{RIGHT} \quad B\ s_1\ M\ B\ s_1\ s_1 \\
& \quad \text{IF B GOTO E} \\
& \quad \text{PRINT M} \\
[B] & \quad \text{RIGHT} \\
& \quad \text{IF s_1 GOTO B} \\
[C] & \quad \text{RIGHT} \\
& \quad \text{IF s_1 GOTO C} \\
& \quad \text{PRINT s_1} \\
[D] & \quad \text{LEFT} \\
& \quad \text{IF s_1 GOTO D} \\
& \quad \text{IF B GOTO D} \\
& \quad \text{PRINT s_1} \\
& \quad \text{IF s_1 GOTO A} \\
\end{align*}
\]

Post-Turing Programs

Another example (A = \{s_1\}):

\[
\begin{align*}
[A] & \quad \text{RIGHT} \quad B\ s_1\ M\ B\ s_1\ s_1 \\
& \quad \text{IF B GOTO E} \\
& \quad \text{PRINT M} \\
[B] & \quad \text{RIGHT} \\
& \quad \text{IF s_1 GOTO B} \\
[C] & \quad \text{RIGHT} \\
& \quad \text{IF s_1 GOTO C} \\
& \quad \text{PRINT s_1} \\
[D] & \quad \text{LEFT} \\
& \quad \text{IF s_1 GOTO D} \\
& \quad \text{IF B GOTO D} \\
& \quad \text{PRINT s_1} \\
& \quad \text{IF s_1 GOTO A} \\
\end{align*}
\]
Post-Turing Programs
Another example (A = \{s_1\}):  
\[
\begin{align*}
[A] & \quad \text{RIGHT} \quad B \ s_1 \ s_1 \ s_1 \\
& \quad \quad \text{IF } B \text{ GOTO E} \\
& \quad \quad \text{PRINT } M \\
[B] & \quad \text{RIGHT} \quad \text{IF } s_1 \text{ GOTO B} \\
[C] & \quad \text{RIGHT} \quad \text{IF } s_1 \text{ GOTO C} \\
&D & \quad \text{LEFT} \quad \text{IF } s_1 \text{ GOTO D} \\
& \quad \quad \text{IF } B \text{ GOTO D} \\
& \quad \quad \text{PRINT } s_1 \\
& \quad \quad \text{IF } s_1 \text{ GOTO A}
\end{align*}
\]

Simulation of $L_n$ in $\mathcal{T}$
We are going to prove the following theorem:

**Theorem 5.1:** If $f(x_1, ..., x_m)$ is partially computable in $L_n$, then there is a Post-Turing program that computes $f$ strictly.

**Proof:** Let $\varphi$ be a program in $L_n$ which computes $f$. We assume that $\varphi$ uses the following variables: $X_1, ..., X_m, Z_1, ..., Z_k, Y$.

So all in all there are $l$ variables, where $l = m + k + 1$. Therefore, we can rename the variables as follows (while keeping their order):

$V_1, ..., V_l$.

An advantage of this system is that the initial tape configuration is already in the correct form:

$B \ x_1 \ B \ x_2 \ B \ ... \ B \ x_m \ B \ z_1 \ B \ z_2 \ ... \ B \ z_k \ B \ y$,  

where $x_1, x_2, ..., x_m, z_1, z_2, ..., z_k, y$ are the current values of the variables $X_1, X_2, ..., X_m, Z_1, Z_2, ..., Z_k, Y$ (using the original variable names).
Simulation of $L_n$ in $T$

The macro $\text{GOTO L}$ has the expansion

IF $s_0$ GOTO L // Remember: $s_0 = B$
IF $s_1$ GOTO L :
IF $s_n$ GOTO L

Simulation of $L_n$ in $T$

The macro $\text{RIGHT TO NEXT BLANK}$ has the expansion

[A] RIGHT
  IF B GOTO E
  GOTO A

The macro $\text{LEFT TO NEXT BLANK}$ has the expansion

[A] LEFT
  IF B GOTO E
  GOTO A

Simulation of $L_n$ in $T$

The macro $\text{MOVE BLOCK RIGHT}$ has the expansion

[C] LEFT
  IF $s_0$ GOTO $A_0$
  IF $s_1$ GOTO $A_1$
  :
  IF $s_n$ GOTO $A_n$

[A] RIGHT
  i = 1, 2, ..., n
  PRINT $s_i$
  LEFT
  GOTO C
[Ai] RIGHT
  PRINT B
  LEFT

Example:

$\begin{array}{c}
\text{↑} \\
\text{↑} \\
\text{↑} \\
\text{↑} \\
\text{↑} \\
\text{↑} \\
\end{array}$

Simulation of $L_n$ in $T$

The macro $\text{ERASE A BLOCK}$ has the expansion

[A] RIGHT
  IF B GOTO E
  PRINT B
  GOTO A

This program causes the head to move to the right, erasing everything between its initial position and the first blank to its right.

Simulation of $L_n$ in $T$

We will now construct a Post-Turing program $Q$ that simulates $P$ step by step.

Of course, the information available to $Q$ must be put onto the tape.

Therefore, we have to use a system for storing the values of all variables at certain positions on the tape:

$B x_1 B x_2 B ... B x_m B z_1 B z_2 B ... B z_k B y,$

where $x_1, x_2, ..., x_m, z_1, z_2, ..., z_k, y$ are the current values of the variables $X_1, X_2, ..., X_m, Z_1, Z_2, ..., Z_k, Y$ (using the original variable names).

Simulation of $L_n$ in $T$

An advantage of this system is that the initial tape configuration is already in the correct form:

$B x_1 B x_2 B ... B x_m B$.

What needs to be done now is to show how to program the effect of each instruction type of $L_n$ in the language $T$.

In the following, we will define some macros that will help us to do this task.
Simulation of $L_n$ in $T$ 
The macro $\text{GOTO L}$ has the expansion 

\[
\text{IF } s_0 \text{ GOTO L} \quad \text{// Remember: } s_0 = B \\
\text{IF } s_1 \text{ GOTO L} \\
\vdots \\
\text{IF } s_n \text{ GOTO L}
\]

Simulation of $L_n$ in $T$ 
The macro $\text{RIGHT TO NEXT BLANK}$ has the expansion

\[
[A] \quad \text{RIGHT} \\
\quad \text{IF B GOTO E} \\
\quad \text{GOTO A}
\]

The macro $\text{LEFT TO NEXT BLANK}$ has the expansion

\[
[A] \quad \text{LEFT} \\
\quad \text{IF B GOTO E} \\
\quad \text{GOTO A}
\]

Simulation of $L_n$ in $T$ 
The macro $\text{MOVE BLOCK RIGHT}$ has the expansion

\[
[C] \quad \text{LEFT} \\
\text{IF } s_0 \text{ GOTO A_0} \\
\text{IF } s_1 \text{ GOTO A_1} \\
\vdots \\
\text{IF } s_n \text{ GOTO A_n} \\
[A] \quad \text{RIGHT} \quad i = 1, 2, \ldots, n \\
\text{PRINT } s_i \\
\text{LEFT} \\
\text{GOTO C} \\
[A_1] \quad \text{RIGHT} \\
\text{PRINT B} \\
\text{LEFT}
\]

Simulation of $L_n$ in $T$ 
The macro $\text{ERASE A BLOCK}$ has the expansion

\[
[A] \quad \text{RIGHT} \\
\quad \text{IF B GOTO E} \\
\quad \text{PRINT B} \\
\quad \text{GOTO A}
\]

This program causes the head to move to the right, erasing everything between its initial position and the first blank to its right.

Simulation of $L_n$ in $T$ 
We introduce another convention: 
A number in square brackets after the name of a macro indicates how many times the macro expansion is to be inserted into the program. 

For example, 

\[
\text{MOVE BLOCK RIGHT [4]}
\]

is short for 

\[
\text{MOVE BLOCK RIGHT} \\
\text{MOVE BLOCK RIGHT} \\
\text{MOVE BLOCK RIGHT} \\
\text{MOVE BLOCK RIGHT}
\]

Simulation of $L_n$ in $T$ 
Now we can start simulating the three instruction types in the language $L_n$ by Post-Turing programs.

We begin the instruction type $V_j \leftarrow s_i V_j$.

In order to place the symbol $s_i$ to the left of the $j$-th variable on the tape, the values of the variables $V_{j'}$, ..., $V_i$ must all be moved one square to the right to make room.

After inserting $s_i$, the tapehead must go back to the blank at the left of the value of $V_i$ in order to be ready for the next simulated instruction.
Simulation of \( L_n \) in \( T \)

Here is the program for the simulation of \( V_j \leftarrow s_i V_j \):

- RIGHT TO NEXT BLANK [I]
- MOVE BLOCK RIGHT [I – j + 1]
- RIGHT
- PRINT \( s_i \)
- LEFT TO NEXT BLANK [j]

Simulation of \( L_n \) in \( T \)

Now we want to show how to simulate \( V_j \leftarrow V_j^{i} \).

The problem here is that if \( V_j \) contains the null string, it must be left unchanged.

Thus, we move to the blank immediately to the right of the value of \( V_j \).

Then we move one step to the left, and if we find another blank there, \( V_j \) must contain the null string (indicated by two successive blanks).

Simulation of \( L_n \) in \( T \)

Here is the program for the simulation of \( V_j \leftarrow V_j^{i} \):

- RIGHT TO NEXT BLANK [j]
- LEFT
- IF B GOTO C // \( V_j \) contains null string
- MOVE BLOCK RIGHT [j]
- RIGHT
- GOTO E

[C] LEFT TO NEXT BLANK [j - 1]

Simulation of \( L_n \) in \( T \)

And finally, here is the program for the simulation of \( IF V_j \text{ ENDS } s_i \text{ GOTO L} \):

- RIGHT TO NEXT BLANK [j]
- LEFT
- IF \( s_i \) GOTO C // \( V_j \) ends in \( s_i \)
- GOTO D

[C] LEFT TO NEXT BLANK [j]
- GOTO L // Note: transfer all labels from \( L_n \) to \( T \)

[D] RIGHT // \( V_j \) could contain null string
- LEFT TO NEXT BLANK [j]

Simulation of \( L_n \) in \( T \)

Now we are able to translate any program in the language \( L_n \) into a corresponding program in \( T \).

There is only one thing that needs to be fixed:

After the program terminates, we want only the string \( y \) to remain on the tape as the program’s output.

This can be done by appending the following code to our generated \( T \) program:

ERASE A BLOCK [1 – 1]

This will erase the values of the first \( l – 1 \) variables on the tape, so only the last variable will remain and the final tape configuration will be

\[
\ldots B B B y B B B \ldots
\]