Post-Turing Programs

**Example 3:** We want to compute a function \( f(x_1, x_2, x_3) \), where \( A = \{s_1, s_2\} \), \( x_1 = 0 \), \( x_2 = s_2s_1s_2 \), and \( x_3 = 0 \). Then the initial tape configuration is as follows:

\[
\begin{array}{cccccc}
B & B & s_2 & s_1 & s_2 & B \\
\uparrow 
\end{array}
\]

Notice that it is impossible to distinguish this initial tape configuration from that with two inputs \( x_1 = 0 \) and \( x_2 = s_2s_1s_2 \). Therefore, the number of arguments must be provided externally.

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.

1. PRINT \( s_1 \) LEFT
2. PRINT \( s_2 \) LEFT
3. \( s_1 \) \( x \) \( \uparrow \)

program terminated
Post-Turing Programs
What function did the previous program compute?

**Definition:** Let \( f(x_1, \ldots, x_m) \) be an \( m \)-ary partial function on the alphabet \( A = \{s_1, \ldots, s_n\} \). Then the program \( \mathcal{P} \) in the Post-Turing language \( \mathcal{T} \) is said to **compute** \( f \) if when started in the tape configuration
\[
B \ x_1 \ B \ x_2 \ B \ x_3 \ldots B \ x_m
\]
it eventually halts if and only if \( f(x_1, \ldots, x_m) \) is defined and if, on halting, the string \( f(x_1, \ldots, x_m) \) can be read from the tape by ignoring all symbols except \( s_1, \ldots, s_n \).

Post-Turing Programs
Notice that this definition allows \( \mathcal{P} \) to contain instructions that mention symbols other than \( s_1, \ldots, s_n \).

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

Post-Turing Programs
Do you remember our simple example program?
It turned the tape configuration
\[
B \ x
\]
into the configuration
\[
B \ s_2 \ s_1 \ x
\]
So what does it compute?
It strictly computes the function \( f(x) = s_2 s_1 x \).

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

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A] RIGHT</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF B GOTO E</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>PRINT M</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>[B] RIGHT</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF s_1 GOTO B</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>[C] RIGHT</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF s_1 GOTO C</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>PRINT s_1</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>[D] LEFT</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF s_1 GOTO D</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF B GOTO D</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>PRINT s_1</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF s_1 GOTO A</td>
<td>B s_1 s_1</td>
</tr>
</tbody>
</table>

Post-Turing Programs
Another example (\( A = \{s_1\} \)):

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A] RIGHT</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF B GOTO E</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>PRINT M</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>[B] RIGHT</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF s_1 GOTO B</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>[C] RIGHT</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF s_1 GOTO C</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>PRINT s_1</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>[D] LEFT</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF s_1 GOTO D</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF B GOTO D</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>PRINT s_1</td>
<td>B s_1 s_1</td>
</tr>
<tr>
<td>IF s_1 GOTO A</td>
<td>B s_1 s_1</td>
</tr>
</tbody>
</table>
Another example (A = \{s_1\}):  

\begin{align*}
\text{[A]} & \quad \text{RIGHT} \quad B \quad M \quad s_1 \\
& \quad \text{IF B GOTO E} \\
& \quad \text{PRINT M} \\
\rightarrow & \quad \text{[B]} \quad \text{RIGHT} \\
& \quad \text{IF s_1 GOTO B} \\
& \quad \text{[C]} \quad \text{RIGHT} \\
& \quad \text{IF s_1 GOTO C} \\
& \quad \text{PRINT s_1} \\
& \quad \text{[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\}):  

[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\}):  

[A] RIGHT B M s_1 B 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 s_1  
IF B GOTO E  
PRINT M  
[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 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 ($A = \{s_1\}$):

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A]</td>
<td>RIGHT</td>
<td>B M s₁ B s₁</td>
</tr>
<tr>
<td></td>
<td>IF B GOTO E</td>
<td>PRINT M</td>
</tr>
<tr>
<td>[B]</td>
<td>RIGHT</td>
<td>IF s₁ GOTO B</td>
</tr>
<tr>
<td>[C]</td>
<td>RIGHT</td>
<td>IF s₁ GOTO C</td>
</tr>
<tr>
<td></td>
<td>PRINT s₁</td>
<td></td>
</tr>
<tr>
<td>[D]</td>
<td>LEFT</td>
<td>IF s₁ GOTO D</td>
</tr>
<tr>
<td></td>
<td>IF B GOTO D</td>
<td>PRINT s₁</td>
</tr>
<tr>
<td></td>
<td>IF s₁ GOTO A</td>
<td></td>
</tr>
</tbody>
</table>

### Post-Turing Programs

Another example ($A = \{s₁\}$):

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A]</td>
<td>RIGHT</td>
<td>B M s₁ B s₁</td>
</tr>
<tr>
<td></td>
<td>IF B GOTO E</td>
<td>PRINT M</td>
</tr>
<tr>
<td>[B]</td>
<td>RIGHT</td>
<td>IF s₁ GOTO B</td>
</tr>
<tr>
<td>[C]</td>
<td>RIGHT</td>
<td>IF s₁ GOTO C</td>
</tr>
<tr>
<td></td>
<td>PRINT s₁</td>
<td></td>
</tr>
<tr>
<td>[D]</td>
<td>LEFT</td>
<td>IF s₁ GOTO D</td>
</tr>
<tr>
<td></td>
<td>IF B GOTO D</td>
<td>PRINT s₁</td>
</tr>
<tr>
<td></td>
<td>IF s₁ GOTO A</td>
<td></td>
</tr>
</tbody>
</table>
Post-Turing Programs

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

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

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

[A] RIGHT $B \ s_1 \ M \ B \ s_1$
IF $B$ GOTO E $\uparrow$
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 $B \ s_1 \ M \ B \ s_1$
IF $B$ GOTO E $\uparrow$
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
Another example \((A = \{s_1\})\):

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

\begin{align*}
\text{[A]} & \rightarrow \text{RIGHT} \quad B \ s_1 \ s_1 \ B \ s_1 \ s_1 \\
& \quad \text{IF B GOTO E} \\
& \quad \text{PRINT M} \\
\text{[B]} & \rightarrow \text{RIGHT} \\
& \quad \text{IF s}_1 \ \text{GOTO B} \\
\text{[C]} & \rightarrow \text{RIGHT} \\
& \quad \text{IF s}_1 \ \text{GOTO C} \\
& \quad \text{PRINT s}_1 \\
\text{[D]} & \rightarrow \text{LEFT} \\
& \quad \text{IF s}_1 \ \text{GOTO D} \\
& \quad \text{IF B GOTO D} \\
& \quad \text{PRINT s}_1 \\
& \quad \text{IF s}_1 \ \text{GOTO A}
\end{align*}

Simulation of $L_n$ in $T$

We are going to prove the following theorem:

**Theorem 5.1:** If $f(x_1, \ldots, x_m)$ 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:

- $x_1, \ldots, x_m, z_1, \ldots, z_k, y$.

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):

$V_1, \ldots, V_l$.

We will now construct a Post-Turing program $Q$ that simulates $\varphi$ 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:

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

where $x_1, x_2, \ldots, x_m, z_1, z_2, \ldots, z_k, y$ are the current values of the variables $x_1, x_2, \ldots, x_m, z_1, z_2, \ldots, z_k, y$ (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:

\[ B \, x_1 \, B \, x_2 \, B \ldots B \, x_m \, . \]

↑

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.

The macro **GOTO** \( L \) has the expansion

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

\[
\text{IF } s_1 \text{ GOTO } L
\]

\[
\vdots
\]

\[
\text{IF } s_n \text{ GOTO } L
\]

The macro **RIGHT TO NEXT BLANK** has the expansion

\[
\begin{align*}
[A] & \quad \text{RIGHT} \\
& \quad \text{IF } B \text{ GOTO E} \\
& \quad \text{GOTO A}
\end{align*}
\]

The macro **LEFT TO NEXT BLANK** has the expansion

\[
\begin{align*}
[A] & \quad \text{LEFT} \\
& \quad \text{IF } B \text{ GOTO E} \\
& \quad \text{GOTO A}
\end{align*}
\]

The macro **MOVE BLOCK RIGHT** has the expansion

\[
\begin{align*}
[C] & \quad \text{LEFT} \\
& \quad \text{IF } s_i \text{ GOTO } A_{i1} \\
& \quad \text{IF } s_i \text{ GOTO } A_{i2} \\
& \quad \vdots \\
& \quad \text{IF } s_i \text{ GOTO } A_{in} \\
& \quad \text{RIGHT} \quad i = 1, \ldots, n \\
& \quad \text{PRINT } s_i \\
& \quad \text{LEFT} \\
& \quad \text{GOTO C} \\
& \quad \text{RIGHT} \\
& \quad \text{PRINT B} \\
& \quad \text{LEFT} \\
& \quad s_{i1} \\
& \quad s_{i2} \\
& \quad s_{in} \\
& \quad \vdots
\end{align*}
\]

The macro **ERASE A BLOCK** has the expansion

\[
\begin{align*}
[A] & \quad \text{RIGHT} \\
& \quad \text{IF } B \text{ GOTO E} \\
& \quad \text{PRINT } B \\
& \quad \text{GOTO A}
\end{align*}
\]

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

The macro **MOVE BLOCK RIGHT** has the expansion

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, **MOVE BLOCK RIGHT** [4] is short for

**MOVE BLOCK RIGHT**

**MOVE BLOCK RIGHT**

**MOVE BLOCK RIGHT**

**MOVE BLOCK RIGHT**

**MOVE BLOCK RIGHT**
Simulation of $L_n$ in $\mathcal{T}$

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 $V_j$, ..., $V_1$ 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 $V_1$ in order to be ready for the next simulated instruction.

Simulation of $L_n$ in $\mathcal{T}$

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 $\mathcal{T}$

Now we want to show how to simulate $V_j \leftarrow V_j^*$.

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 $\mathcal{T}$

Here is the program for the simulation of $V_j \leftarrow V_j^*$:

- 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 $\mathcal{T}$

And finally, here is the program for the simulation of $IF V_j ENDS s_i, GOTO L$:

- 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 $\mathcal{T}$
- [D] RIGHT // $V_j$ could contain null string
- LEFT TO NEXT BLANK [j]

Simulation of $L_n$ in $\mathcal{T}$

Now we are able to translate any program in the language $L_n$ into a corresponding program in $\mathcal{T}$.

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 $\mathcal{T}$ 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

... $B$ $B$ $B$ $y$ $B$ $B$ $B$ ...

↑
Simulation of $\mathcal{T}$ in $\mathcal{L}$

The next thing we want to prove is the following:

**Theorem 6.1:** If there is a Post-Turing program that computes the partial function $f(x_1, \ldots, x_m)$, then $f$ is partially computable.

Since our definition of partial computability is based on the language $\mathcal{L}$, this theorem states the following:

If the $m$-ary partial function $f$ on $A^*$ is computed by a program of $\mathcal{T}$, then there is a program of $\mathcal{L}$ that computes $f$ (using base $n$ values of strings).