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

We are going to prove the following theorem:

**Theorem 5.1:** If $f(x_1, \ldots, x_m)$ is partially computable in $\mathcal{L}_n$, then there is a Post-Turing program that computes $f$ strictly.

**Proof:** Let $\varphi$ be a program in $\mathcal{L}_n$ 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 $\mathcal{L}_n$ in $\mathcal{T}$

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 $\mathcal{L}_n$ in the language $\mathcal{T}$.

In the following, we will define some macros that will help us to do this task.

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

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 $\mathcal{L}_n$ in $\mathcal{T}$

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

$$[A] \quad \text{RIGHT}$$

$$\text{IF} \ B \ \text{GOTO E}$$

$$\text{GOTO A}$$

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

$$[A] \quad \text{LEFT}$$

$$\text{IF} \ B \ \text{GOTO E}$$

$$\text{GOTO A}$$

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

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

$$[C] \quad \text{LEFT} \quad \text{Example:}$$

<table>
<thead>
<tr>
<th>LEFT</th>
<th>\text{Example:}</th>
</tr>
</thead>
<tbody>
<tr>
<td>iF \ s_i \ \text{GOTO A}_i</td>
<td>s_1 B s_i s_2</td>
</tr>
<tr>
<td>iF \ s_i \ \text{GOTO A}_i</td>
<td>s_1 B s_i s_2</td>
</tr>
<tr>
<td>iF \ s_i \ \text{GOTO A}_i</td>
<td>s_1 B s_i s_2</td>
</tr>
</tbody>
</table>

Example:

$$s_1 B s_2 B s_3 B$$

$$s_2 B s_3 B$$

$$s_3 B$$

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

The macro $\text{PRINT}$ has the expansion

$$[C] \quad \text{LEFT} \quad \text{Example:}$$

<table>
<thead>
<tr>
<th>LEFT</th>
<th>\text{Example:}</th>
</tr>
</thead>
<tbody>
<tr>
<td>iF \ s_i \ \text{GOTO C}</td>
<td>s_1 B s_i s_2</td>
</tr>
<tr>
<td>iF \ s_i \ \text{GOTO C}</td>
<td>s_1 B s_i s_2</td>
</tr>
<tr>
<td>iF \ s_i \ \text{GOTO C}</td>
<td>s_1 B s_i s_2</td>
</tr>
</tbody>
</table>

Example:

$$s_1 B s_2 B s_3 B$$

$$s_2 B s_3 B$$

$$s_3 B$$
Simulation of \( L_n \) in \( \mathcal{T} \)

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

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

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

Simulation of \( L_n \) in \( \mathcal{T} \)

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:

\[
\begin{align*}
& \text{MOVE BLOCK RIGHT} \\
& \text{MOVE BLOCK RIGHT} \\
& \text{MOVE BLOCK RIGHT} \\
& \text{MOVE BLOCK RIGHT}
\end{align*}
\]

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 \( V_j \leftarrow s_j V_j \). In order to place the symbol \( s_j \) to the left of the \( j \)-th variable on the tape, the values of the variables \( V_j, \ldots, V_l \) must all be moved one square to the right to make room.

After inserting \( s_j \), 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 \( V_j \leftarrow s_j V_j \):

\[
\begin{align*}
& \text{RIGHT TO NEXT BLANK [} j \text{]} \\
& \text{MOVE BLOCK RIGHT [} j - j + 1 \text{]} \\
& \text{PRINT } s_j \\
& \text{LEFT TO NEXT BLANK [} j \text{]}
\end{align*}
\]

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 \( V_j \leftarrow V_j^* \):

\[
\begin{align*}
& \text{RIGHT TO NEXT BLANK [} j \text{]} \\
& \text{LEFT} \\
& \text{IF B GOTO C} \quad \text{// \( V_j \) contains null string} \\
& \text{MOVE BLOCK RIGHT [} j \text{]} \\
& \text{RIGHT} \\
& \text{GOTO E} \\
& \text{[C] LEFT TO NEXT BLANK [} j - 1 \text{]}
\end{align*}
\]
Simulation of $L_n$ in $T$

And finally, here is the program for the simulation of $L_n$ in $T$:

```plaintext
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 $T$

[D] RIGHT // $V_j$ could contain null string
LEFT TO NEXT BLANK [j]
```

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:

```plaintext
ERASE A BLOCK [l – 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 ...
```