1. Macros for Use in $S_n$

2. Two Important Examples

3. The Languages $S$ and $S_n$

4. Post-Turing Programs

5. Simulation of $S_n$ in $S$

6. Simulation of $\mathcal{T}$ in $S$
We introduce for each $n > 0$ a programming language $S_n$ designed for string calculations on an alphabet with $n$ symbols.
The instructions of $S_n$ are:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V \leftarrow \sigma V$</td>
<td>place symbol $\sigma$ at the left of $V$</td>
</tr>
<tr>
<td>$V \leftarrow V^-$</td>
<td>delete the final symbol of the string that is the value of $V$; if the value is 0 leave it unchanged</td>
</tr>
<tr>
<td>$V \leftarrow V$</td>
<td>do nothing instruction</td>
</tr>
<tr>
<td>IF $V$ ENDS $\sigma$ GOTO $L$</td>
<td>if the value of $V$ ends in $\sigma$ then execute the first instruction with label $L$; otherwise proceed with next instruction</td>
</tr>
</tbody>
</table>
Example

Suppose that the alphabet $A$ consists of the symbols $s_1, s_2, s_3$ and $x = s_3s_2s_2s_1$ is a string of length 4 on the alphabet $V$. The effect of the above instructions applied to $x$ is shown below:

<table>
<thead>
<tr>
<th>Instr.</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leftarrow s_2x$</td>
<td>$s_2s_3s_2s_2s_1$</td>
</tr>
<tr>
<td>$x \leftarrow \bar{x}$</td>
<td>$s_3s_2s_2$</td>
</tr>
<tr>
<td>$x \leftarrow x$</td>
<td>$s_3s_2s_2s_1$</td>
</tr>
<tr>
<td>IF $x$ ENDS $s_2$ GOTO L</td>
<td>no effect</td>
</tr>
<tr>
<td>IF $x$ ENDS $s_1$ GOTO L</td>
<td>jump to $L$</td>
</tr>
</tbody>
</table>
Also the instructions of $S_n$ refer to strings, we can also think of them as referring to numbers that the strings represent.

Example

The numerical effect of $X \leftarrow s_i X$ in the $n$-symbol alphabet $\{s_1, \ldots, s_n\}$ is to replace numerical value $x$ by $i \cdot n|x| + x$. 
The macro

$$\text{IF } V \neq 0 \text{ GOTO } L$$

has the expression

$$\text{IF } V \text{ ENDS } s_1 \text{ GOTO } L$$
$$\text{IF } V \text{ ENDS } s_2 \text{ GOTO } L$$

...  
$$\text{IF } V \text{ ENDS } s_n \text{ GOTO } L$$
The macro $V \leftarrow 0$ has the expansion

\[
[A] \quad V \leftarrow V^{-} \\
\text{IF } V \neq 0 \text{ GOTO } A
\]
The macro

GOTO L

has the expansion

Z ← 0
Z ← s_1 Z
IF Z ENDS s_1 GOTO L
The block of instructions

\[
\begin{align*}
\text{IF } V \text{ ENDS } s_1 & \text{ GOTO } B_1 \\
\text{IF } V \text{ ENDS } s_2 & \text{ GOTO } B_2 \\
& \vdots \\
\text{IF } V \text{ ENDS } s_n & \text{ GOTO } B_n
\end{align*}
\]

is abbreviated as

\[
\text{IF } V \text{ ENDS } s_i \text{ GOTO } B_i (1 \leq i \leq n)
\]
The macro $V' \leftarrow V$ has the expansion

$$
\begin{align*}
Z &\leftarrow 0 \\
V' &\leftarrow 0 \\
[A] &\quad \text{IF } V \text{ ENDS } s_i \text{ GOTO } B_i (1 \leq i \leq n) \\
&\quad \text{GOTO } C \\
[B_i] &\quad V \leftarrow V^- \text{(This group of 4 repeated for } 1 \leq i \leq n) \\
&\quad V' \leftarrow s_i V' \\
&\quad Z \leftarrow s_i Z \\
&\quad \text{GOTO } A \text{(end group)} \\
[C] &\quad \text{IF } Z \text{ ENDS } s_i \text{ GOTO } D_i (1 \leq i \leq n) \\
[D_i] &\quad Z \leftarrow Z^- \\
&\quad V \leftarrow s_i V \\
&\quad \text{GOTO } C
\end{align*}
$$
The function $x + 1$ is computable in $S_n$, as shown by the following flowchart.
Example

Start with the string \( s = s_2s_1s_1s_3s_1 \). The numerical values is 208. Strings produced by the algorithm are:

<table>
<thead>
<tr>
<th>( X )</th>
<th>( Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_2s_1s_3 )</td>
<td>( s_2 )</td>
</tr>
<tr>
<td>( s_2s_1s_1 )</td>
<td>( s_3s_2 )</td>
</tr>
<tr>
<td>( s_2s_1 )</td>
<td>( s_1s_3s_2 )</td>
</tr>
<tr>
<td>( s_2 )</td>
<td>( s_1s_1s_3s_2 )</td>
</tr>
<tr>
<td>( 0 )</td>
<td>( s_2s_1s_1s_3s_2 )</td>
</tr>
</tbody>
</table>

The value of \( Y \) is \( 2 \cdot 3^4 + 1 \cdot 3^3 + 1 \cdot 3^2 + 3 \cdot 3^1 + 2 = 209 \).
Example

Start with the string $s = s_2s_1s_3s_3$. The numerical values is 75. Strings produced by the algorithm are:

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_2s_1s_3$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_2s_1$</td>
<td>$s_1s_1$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_2s_1s_1$</td>
</tr>
<tr>
<td>$0$</td>
<td>$s_2s_2s_1s_1$</td>
</tr>
</tbody>
</table>

The value of $Y$ is $2 \cdot 3^3 + 2 \cdot 3^2 + 1 \cdot 3^1 + 1 = 76$. 
The previous flowchart corresponds to the program

\[ [B] \quad \text{IF } X \text{ ENDS } s_1 \text{ GOTO } A_i (1 \leq i \leq n) \]
\[ Y \leftarrow s_1 Y \]
\[ \text{GOTO } E \]

\[ [A_i] \quad X \leftarrow X^- \text{ (This group of 3 repeated for } 1 \leq i \leq n) \]
\[ Y \leftarrow s_{i+1} Y \]
\[ \text{GOTO } C \]

\[ [A_n] \quad X \leftarrow X^- \]
\[ Y \leftarrow s_1 Y \]
\[ \text{GOTO } B \]

\[ [C] \quad \text{IF } X \text{ ENDS } s_i \text{ GOTO } D_i (1 \leq i \leq n) \]
\[ \text{GOTO } E \]

\[ [D_i] \quad X \leftarrow X^- \text{ (This group of 3 repeated for } 1 \leq i \leq n) \]
\[ Y \leftarrow s_i Y \]
\[ \text{GOTO } C \]
The function $x \div 1$ is computed by the following flowchart:
Example

Start with the string $s = s_2s_1s_1s_3s_2$ having the numerical value 209. The successive values of $X$ and $Y$ are:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_2s_1s_1s_3$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_2s_1s_1$</td>
<td>$s_3s_1$</td>
</tr>
<tr>
<td>$s_2s_1$</td>
<td>$s_1s_3s_1$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_1s_1s_3s_1$</td>
</tr>
<tr>
<td>0</td>
<td>$s_2s_1s_1s_3s_1$</td>
</tr>
</tbody>
</table>

The numerical equivalent is 208.
Example

Let \( s = s_3s_2s_1s_1 \) having the numerical equivalent
\[ 3 \cdot 3^3 + 2 \cdot 3^2 + 1 \cdot 3^1 + 1 = 103. \]
The successive values of \( X \) and \( Y \) are:

<table>
<thead>
<tr>
<th>( X )</th>
<th>( Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_3s_2s_1 )</td>
<td>0</td>
</tr>
<tr>
<td>( s_3s_2s_1 )</td>
<td>( s_3 ) (carry is propagated)</td>
</tr>
<tr>
<td>( s_3s_2 )</td>
<td>( s_3s_3 )</td>
</tr>
<tr>
<td>( s_3 )</td>
<td>( s_1s_3s_3 ) (carry is absorbed)</td>
</tr>
<tr>
<td>0</td>
<td>( s_3s_1s_3s_3 )</td>
</tr>
</tbody>
</table>

The numerical equivalent of \( s_3s_1s_3s_3 \) is 102.
The previous flowchart corresponds to the program

\[ [B] \quad \text{IF } X \text{ ENDS } s_i \text{ GOTO } A \text{ (This is repeated for } 1 \leq i \leq n) \\
\quad \text{GOTO } E \]

\[ [A_i] \quad X \leftarrow X^- \text{ (This group of 3 repeated for } 1 \leq i \leq n) \\
\quad Y \leftarrow s_{i-1}Y \\
\quad \text{GOTO } C \]

\[ [A_1] \quad X \leftarrow X^- \\
\quad \text{IF } X \neq 0 \text{ GOTO } C_2 \\
\quad \text{GOTO } E \]

\[ [C_2] \quad Y \leftarrow s_nY \\
\quad \text{GOTO } B \]

\[ [C] \quad \text{IF } X \text{ ENDS } s_i \text{ GOTO } D_i \text{ (This group of 2 repeated for } 1 \leq i \leq n) \\
\quad \text{GOTO } E \]

\[ [D_i] \quad X \leftarrow X^- \text{ (This group of 3 repeated for } 1 \leq i \leq n) \\
\quad Y \leftarrow s_iY \\
\quad \text{GOTO } C \]
In either $S$ or $S_n$ computations are really dealing with numbers and strings on an $n$ letter alphabets are objects being used to represent numbers in the base $n$.

**Theorem**

*A function $f$ is partially computable if and only if it is partially computable in $S_1$.***
Proof.

Note that the languages $S$ and $S_1$ are the same. Indeed, the effect of the $S_1$ instructions

$$V \leftarrow s_1 V \text{ and } V \leftarrow V^-$$

is identical to the effect of the $S$ instructions

$$V \leftarrow V + 1 \text{ and } V \leftarrow V - 1.$$ 

The condition $V$ ENDS $s_1$ in $S_1$ is equivalent to $V \neq 0$ in $S$.

Thus, the results involving $S_n$ can be specialized to $n = 1$ to give results about $S$. 


Theorem

If a function is partially computable, then it also partially computable in $S_n$ for each $n$.

Proof.

Suppose $f$ is computed by $P$ in $S$. $P$ is translated into a program in $S_n$ by replacing instructions in $P$ by a macro in $S_n$:

- $V \leftarrow V + 1$ is replaced by the macro $V \leftarrow V + 1$ in $S_n$;
- $V \leftarrow V - 1$ is replaced by the macro $V \leftarrow V - 1$ in $S_n$;
- IF $V \neq 0$ GOTO $L$ by the macro IF $V \neq 0$ GOTO $L$ in $S_n$. 

$\blacksquare$
$\mathcal{T}$ is another programming language for string manipulation named the Post-Turing language.

- there is a unique variable and its content is placed on a tape;
- the tape is divided into cells; each cell is able to contain a symbol of the alphabet $A = \{s_1, \ldots, s_n\}$;
- there is a special symbol $s_0$ (also denoted by $B$ and referred to as blank);
- only one symbol is observed at any given time.
- All **but a finite number of cells** contain \( B \). The content of the tape is shown by exhibiting a finite portion of the tape containing the non-blank symbols.

- At any given moment only one tape symbol is being scanned by a **head**. This is indicated by an arrow.

- The head can move one square to the left or to the right of the square that is currently scanned.
This is indicated by writing

\[ a_2 \ B \ a_3 \ a_1 \]
There are four types of instructions in the Post-Turing Language:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINT$_\sigma$</td>
<td>replace the symbol on the square being scanned by $\sigma$</td>
</tr>
<tr>
<td>IF $\sigma$ GOTO L</td>
<td>goto the first instruction labeled $L$ if the symbol currently scanned is $\sigma$; otherwise continue to the next instruction.</td>
</tr>
<tr>
<td>RIGHT</td>
<td>scan the square to the right of the current square.</td>
</tr>
<tr>
<td>LEFT</td>
<td>scan the square to the left of the current square.</td>
</tr>
</tbody>
</table>
To compute a partial function $f(x_1, \ldots, x_m)$ of $m$ variables we start with the initial tape configuration

$$B \ x_1 \ B \ x_2 \ \cdots \ x_m$$

The inputs are separated by single blanks, and the symbol initially scanned is the blank immediately at left of $x_1$. 
Example

If $n = 1$, the alphabet is $\{s_1\}$. We want to compute a function $f(x_1, x_2)$ and the initial values are $x_1 = s_1s_1$, $x_2 = s_1$. Then, the initial configuration is:

$$B \ s_1 \ s_1 \ B \ s_1$$
Example

\[ n = 2, \ x_1 = s_1 s_2, \ x_2 = s_2 s_1. \]\n
The initial configuration is

\[ B \ s_1 \ s_2 \ B \ s_2 \ s_1 \]

↑
Example

Suppose $n = 2$, $x_1 = 0$, $x_2 = s_1 s_1$, $x_3 = s_2$. The tape configuration is

```
B B s_1 s_1 B s_2
```

↑
Example

For $n = 2$, $x_1 = s_1 s_2$, $x_2 = s_2 s_1$, $x_3 = 0$ the tape configuration is initially

$$B \ s_1 \ s_2 \ B \ s_2 \ s_1 \ B$$

The number of arguments placed on tape must be provided externally.
An example of a Post-Turing program that begins with the input $x$ and outputs $s_2s_1x$ is

```
PRINTs_1
LEFT
PRINTs_2
LEFT
```

The program starts with

```
B x
```

and ends with

```
B s_2 s_1 x
```
Example

Suppose now that the alphabet is \( \{s_1, s_2, s_3\} \) and let \( x \in \{s_1, s_2, s_3\}^* \). Beginning with

\[
B \ x \\
\uparrow
\]

the program needs to halt with the tape configuration

\[
B \ x \ s_1 \ s_1 \\
\uparrow
\]

The computation proceeds by first moving right until the blank to the right of \( x \) is located. Then, \( s_1 \) is printed twice and then the computation moves to the left until first \( B \) is located.
Example cont’d

Example

[A]  RIGHT
  IF \( s_1 \) GOTO A
  IF \( s_2 \) GOTO A
  IF \( s_3 \) GOTO A
  PRINT \( s_1 \)
  RIGHT
  PRINT \( s_1 \)

[C]  LEFT
  IF \( s_1 \) GOTO C
  IF \( s_2 \) GOTO C
  IF \( s_3 \) GOTO C
Example

The alphabet is \( \{s_1, s_2\} \) and the next program aims to erase all occurrences of \( s_2 \) in the input string (that is, replace \( s_2 \) by \( B \)). For the purpose of reading output values from the tape, additional \( B \)s are ignored.
Example cont’d

Example

[C] RIGHT
  IF B GOTO E
  IF \( s_2 \) GOTO A
  IF \( s_1 \) GOTO C

[A] PRINT B
  IF B GOTO C

The function computed by this program satisfies

\[
\begin{align*}
  f(s_2 s_1 s_2) &= s_1, \\
  f(s_1 s_2 s_1) &= s_1 s_1.
\end{align*}
\]
Example

The previous program achieves the following computation:

\[
\begin{align*}
B & ~s_1 ~s_2 ~s_1 \\
\uparrow & \\
B & ~s_1 ~s_2 ~s_1 \\
\uparrow & \\
B & ~s_1 ~s_2 ~s_1 \\
\uparrow & \\
B & ~s_1 ~B ~s_1 \\
\uparrow & \\
\end{align*}
\]

ending with \(Bs_1 Bs_1 B\) on the tape.
We discuss a number of macros in $\mathcal{T}$. The $\mathcal{T}$ macro $\text{GOTO } L$ has the expansion

$$
\text{IF } s_0 \text{ GOTO } L \\
\text{IF } s_1 \text{ GOTO } L \\
\vdots \\
\text{IF } s_n \text{ GOTO } L
$$
The $\mathcal{T}$ macro **RIGHT TO NEXT BLANK** has the expansion

```
[A]    RIGHT
    IF $B$ GOTO $E$
    GOTO $A$
```

Similarly, **LEFT TO NEXT BLANK** has the expansion

```
[A]    LEFT
    IF $B$ GOTO $E$
    GOTO $A$
```
The macro **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_i]  RIGHT
    PRINT $s_i$
    LEFT
    GOTO $C$

[A_0]  RIGHT
    PRINT $B$
    LEFT
```

this group of 4 instructions is executed for $1 \leq i \leq n$
The macro **ERASE A BLOCK** causes the head to move to the right with everything erased between the square at which it begins and the first blank to the right. Its expansion is

```
[A]   RIGHT
    IF B GOTO E
    PRINT B
    GOTO A
```
Convention: a non-negative number between brackets after the name of a macro indicates that the macro is repeated that number of times.

Example

RIGHT TO NEXT BLANK[3]

is short for

RIGHT TO NEXT BLANK
RIGHT TO NEXT BLANK
RIGHT TO NEXT BLANK
Exercise in class!
The next program uses three symbols: $s_1$ from the input alphabet \{s_1\}, $B$, and a marker symbol $M$. Beginning with the tape $B \ u$

$\uparrow$

where $u$ is a string in \{s_1\}*, the program terminates with a tape $B \ u \ B \ u$

$\uparrow$
A program $P$ in $T$ computes a function $f(x_1, \ldots, x_m)$ on the alphabet $\{s_1, \ldots, s_n\}$ if when started with a tape configuration

$$B \ x_1 \ B \ \cdots \ B \ x_m$$

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 off the tape by ignoring all symbols other than $s_1, \ldots, s_n$.

Note that in the final configuration all markers and blanks are ignored.
A program $P$ computes $f$ strictly if two additional conditions are met:

- no instruction in $P$ mentions other symbol than $s_0 = B, s_1, \ldots, s_n$, and
- whenever $P$ halts, the tape configuration is

\[
\cdots B B y B \cdots
\]

where $y = f(x_1, \ldots, x_m)$.

Thus, when $P$ computes $f$ strictly, the output is available in a consecutive block of cells.
Theorem

If \( f(x_1, \ldots, x_m) \) is a partially computable function in \( S_n \), then there is a Post-Turing program that computes \( f \) strictly.

Proof.

Let \( \mathcal{P} \) be a program in \( S_n \) that computes \( f \) using \( \ell = m + 1 + k \) variables that include the input variables \( X_1, \ldots, X_m \), the output variable \( Y \), and the local variables \( Z_1, \ldots, Z_k \).

The list of variables

\[
X_1, \ldots, X_m, Y, Z_1, \ldots, Z_k
\]

will be denoted by

\[
V_1, \ldots, V_\ell.
\]
Proof.

Let $Q$ be a Post-Turing program that simulates $P$ step by step. We must allocate space on the tape to accommodate the values of the $\ell$ variables. At the beginning of each simulated step the tape configuration is

\[
B \ x_1 \ B \ x_2 \ B \ \cdots \ B \ x_m \ B \ z_1 \ B \ \cdots \ z_k \ B \ y
\]

where $x_1, \ldots, x_m, z_1, \ldots, z_k, y$ are the current values of $X_1, \ldots, X_m, Z_1, \ldots, Z_k, Y$. \hfill \square
Proof cont’d

Note that the initial tape configuration

\[ B \ x_1 \ B \ x_2 \ B \ \cdots \ B \ x_m \]

\[ \uparrow \]

is already in correct form because the remaining variables are initialized to 0.
Next, we show how to program the effect of each instruction in \( S \) in \( \mathcal{T} \).
Proof cont’d

Simulating Instructions in $S_n$ by Post-Turing Programs:

- every simulation of an instruction of $S_n$ begins and ends on the first blank;
- the value of $V_i$ is written between the $i^{th}$ blank and the $i + 1^{st}$ blank;
- if $V_i$ is 0 we have two consecutive blanks: the $i^{th}$ blank and the $i + 1^{st}$ blank.
Proof cont’d

Simulation of \( V_j \leftarrow s_i V_j \):

To place \( s_i \) at the left of the \( j^{th} \) variable on the tape, the values of \( V_j, \ldots, V_\ell \) must be all moved one square to the right to make room.

After \( s_i \) was inserted, the head must go back at the left of the value of \( V_1 \) to be ready for the next simulated instruction.

RIGHT TO NEXT BLANK [\( \ell \)]
MOVE BLOCK RIGHT [\( \ell - j + 1 \)]
RIGHT
PRINT \( s_i \)
LEFT TO NEXT BLANK [\( j \)]
Proof cont’d

Simulation of $V_j \leftarrow V_j^-$: difficulty is that if the value is 0 we need to leave it unchanged. By moving one square to the left we find two consecutive blanks.

RIGHT TO THE NEXT BLANK $[j]$
LEFT
IF B GOTO C
MOVE BLOCK RIGHT $[j]$
RIGHT
GOTO E
[C] LEFT TO NEXT BLANK $[j-1]$
Proof cont’d

Finally, to simulate

IF $V_j$ ENDS $s_i$ GOTO L

we use

RIGHT TO NEXT BLANK $[j]$
LEFT
IF $s_i$ GOTO C
  GOTO D
[C] LEFT TO NEXT BLANK $[j]$
  GOTO L
[D] RIGHT
  LEFT TO NEXT BLANK $[j]$
When simulation ends the tape configuration is

\[ \cdots \mathcal{B} \mathcal{B} \mathcal{B} x_1 \cdots x_n \mathcal{B} z_1 \mathcal{B} \cdots z_k \mathcal{B} y \mathcal{B} \mathcal{B} \cdots \]

\[ \uparrow \]

At the end of the computation we need to have the tape configuration

\[ \cdots \mathcal{B} \mathcal{B} \mathcal{B} y \mathcal{B} \mathcal{B} \mathcal{B} \cdots \mathcal{B} \mathcal{B} \cdots \]

\[ \uparrow \]

To reach this configuration we put at the end of the Post-Turing program the following:

\[ \text{ERASE A BLOCK } [\ell - 1] \]

Thus, the program computes the function \( f \) strictly.
Consider the following statements:

1. \( f \) is partially computable;
2. \( f \) is partially computable in \( S_n \);
3. \( f \) is strictly computed by a Post-Turing Program;
4. \( f \) is computed by a Post-Turing program.

So far we proved the implications

\[(1) \Rightarrow (2) \Rightarrow (3) \Rightarrow (4).\]

We are about to prove \((4) \Rightarrow (1)\) thereby showing that all statements are equivalent.
**Theorem**

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

Let \( P \) be a Post-Turing program that computes \( f \). We need to construct a program \( Q \) in the language \( S \) that computes \( f \). \( Q \) consists of three sections:

BEGINNING
MIDDLE
END

- BEGINNING arranges the input in \( Q \) in the appropriate format for MIDDLE.
- MIDDLE simulates \( P \) in a step-by-step manner.
- END extracts the output.
The Post-Turing program makes use of $B$ and perhaps some additional symbols $s_{n+1}, \ldots, s_r$ in this order:

$$s_1, \ldots, s_n, s_{n+1}, \ldots, s_r, B$$

$Q$ simulates $P$ by using the numbers that strings on this alphabet represent in base $r + 1$ as codes for corresponding strings. $B$ represents the number $r + 1$. For this reason, we will write $B$ as $s_{r+1}$.

The tape configuration at a stage of $P$ is tracked by $Q$ using three numbers $L$, $H$, and $R$:

- the value of $H$ is the numerical value of the symbol currently scanned
- the value of $L$ is the numerical value in base $r + 1$ of a string $w$ such that the content of the tape at the left of the head is $\cdots B B w$;
- the value of $R$ is the numerical value in base $r + 1$ of a string $z$ such that the content of the tape at the right of the head is $z B B \cdots$. 
Example

For the tape configuration

\[ \cdots B B B B s_2 s_1 B s_3 s_1 s_2 B B \cdots \]

with \( r = 3 \) and the base 4, we have

\[
\begin{align*}
H & = 3, \\
L & = 2 \cdot 4^2 + 1 \cdot 4 + 4 = 40 \\
R & = 1 \cdot 4 + 2 = 6.
\end{align*}
\]
An instruction PRINT $i$ is simulated by $H \leftarrow i$.

An instruction IF $s_i$ GOTO $L$ is simulated by

$$\text{IF } H = i \text{ GOTO } L$$
An instruction RIGHT is simulated by

\[ L \leftarrow \text{CONCAT}_{r+1}(L, H) \]
\[ H \leftarrow \text{LTEND}_{r+1}(R) \]
\[ R \leftarrow \text{LTRUNC}_{r+1}(R) \]
\[ \text{IF } R \neq 0 \text{ GOTO } E \]
\[ R \leftarrow r + 1 \]
An instruction LEFT is simulated by

\[
R \leftarrow \text{CONCAT}_{r+1}(H, R) \\
H \leftarrow \text{RTEND}_{r+1}(L) \\
L \leftarrow \text{RTRUNC}_{r+1}(L) \\
\text{IF } L \neq 0 \text{ GOTO } E \\
L \leftarrow r + 1
\]

The section MIDDLE of \( Q \) can be obtained by replacing each instruction by its simulation.
The BEGINNING and END section must deal with the fact that $f$ is a function of $m$ arguments on $\{s_1, \ldots, s_n\}^*$.

- Initial values of $X_1, \ldots, X_m$ for $Q$ are numbers that represent the input strings in base $n$.

- The BEGINNING section calculates the initial values of $L, H, R$ that correspond to the tape configuration

$$B \ x_1 \ B \ x_2 \ B \ \cdots \ B \ x_m$$

where the numbers $x_1, \ldots, x_m$ are represented in base $n$ notation.
the BEGINNING section is:

\[
\begin{align*}
L & \leftarrow r + 1 \\
H & \leftarrow r + 1 \\
Z_1 & \leftarrow \text{UPCHANGE}_{n,r+1}(X_1) \\
Z_2 & \leftarrow \text{UPCHANGE}_{n,r+1}(X_2) \\
& \quad \vdots \\
Z_m & \leftarrow \text{UPCHANGE}_{n,r+1}(X_m) \\
R & \leftarrow \text{CONCAT}_{r+1}(Z_1, r + 1, Z_2, r + 1, \ldots, r + 1, \ldots, Z_m)
\end{align*}
\]
The END section consists of:

\[ Z \leftarrow \text{CONCAT}_{r+1}(L, H, R) \]
\[ Y \leftarrow \text{DOWNCHANGE}_{n,r+1}(Z). \]

This concludes the description of the program \( Q \).