Outline

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. Simulating Instructions in $S_n$ by Post-Turing Programs
7. Simulation of $S$ in $T$
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>
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 $X \leftarrow s_iX$ 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

\[
\begin{align*}
\text{IF } V \text{ ENDS } s_1 \text{ GOTO } L \\
\text{IF } V \text{ ENDS } s_2 \text{ GOTO } L \\
\vdots \\
\text{IF } V \text{ ENDS } s_n \text{ GOTO } L
\end{align*}
\]
The macro $V \leftarrow 0$ has the expansion

$$[A] \quad V \leftarrow V^-$$

IF $V \neq 0$ GOTO A
The macro

GOTO \( L \)

has the expansion

\[ Z \leftarrow 0 \]
\[ Z \leftarrow s_1 Z \]
\[ \text{IF } Z \text{ ENDS } s_1 \text{ GOTO } L \]
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

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

```plaintext
begin
    TEST X

    Y ← s₁ Y
    X ← X⁻
    Y ← sᵢ Y

    x = 0
    TEST X
    x ends sᵢ, i < n
    X ← X⁻
    Y ← sᵢ+₁ Y

    Carry propagates
    x ends sₙ

END
```
Example

Start with the string $s = s_2 s_1 s_1 s_3 s_1$. 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>$s_2 s_1 s_1 s_3$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2 s_1 s_1$</td>
<td>$s_3 s_2$</td>
</tr>
<tr>
<td>$s_2 s_1$</td>
<td>$s_1 s_3 s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_1 s_1 s_3 s_2$</td>
</tr>
<tr>
<td>0</td>
<td>$s_2 s_1 s_1 s_3 s_2$</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

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

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

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

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

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

Start with the string $s = s_2 s_1 s_1 s_3 s_2$ having the numerical value 209. The successive values of $X$ and $Y$ are:

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_2 s_1 s_1 s_3$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_2 s_1 s_1$</td>
<td>$s_3 s_1$</td>
</tr>
<tr>
<td>$s_2 s_1$</td>
<td>$s_1 s_3 s_1$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_1 s_1 s_3 s_1$</td>
</tr>
<tr>
<td>0</td>
<td>$s_2 s_1 s_1 s_3 s_1$</td>
</tr>
</tbody>
</table>

The numerical equivalent is 208.
Example

Let $s = s_3 s_2 s_1 s_1$ 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>$s_3 s_2 s_1$</td>
<td>0</td>
</tr>
<tr>
<td>$s_3 s_2 s_1$</td>
<td>$s_3$ (carry is propagated)</td>
</tr>
<tr>
<td>$s_3 s_2$</td>
<td>$s_3 s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_1 s_3 s_3$ (carry is absorbed)</td>
</tr>
<tr>
<td>0</td>
<td>$s_3 s_1 s_3 s_3$</td>
</tr>
</tbody>
</table>

The numerical equivalent of $s_3 s_1 s_3 s_3$ is 102.
The previous flowchart corresponds to the program

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

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

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

[C_2]  $Y \leftarrow s_n Y$
    GOTO B

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

[D_i]  $X \leftarrow X^{-}$ (This group of 3 repeated for $1 \leq i \leq n$)
    $Y \leftarrow s_i Y$
    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 $S_n$. 
\( \mathcal{T} \) 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:

$$
\begin{array}{c}
B & s_1 & s_1 & B & s_1 \\
\uparrow
\end{array}
$$
Example

$n = 2, x_1 = s_1 s_2, x_2 = s_2 s_1$. 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

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

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

```
PRINT $s_1$
LEFT
PRINT $s_2$
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

\[ f(s_2 s_1 s_2) = s_1, \]
\[ f(s_1 s_2 s_1) = s_1 s_1. \]
Example

The previous program achieves the following computation:

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

ending with \( Bs_1 Bs_1 B \) on the tape.
Example

The next program uses three symbols: $s_1$ from the input alphabet $\{s_1\}$, $B$, and a \textbf{marker symbol} $M$. Beginning with the tape $B \ u$

\begin{align*}
\uparrow
\end{align*}

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

\begin{align*}
\uparrow
\end{align*}
Example cont’d

Example

[A] RIGHT
   IF B GOTO E
   PRINT M

[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
Definition

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

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

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 $\mathcal{P}$ computes $f$ strictly if two additional conditions are met:

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

$$\cdots \ B \ B \ y \ B \ \cdots$$

↑

where $y = f(x_1, \ldots, x_m)$.
Thus, when $\mathcal{P}$ 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 \( Z_1, \ldots, Z_k \). \( \square \)
Proof cont’d

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 $X_1, \ldots, X_m, Z_1, \ldots, Z_k, Y$. 

$\square$
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

We discuss a number of macros in $T$.
The $T$ macro $\text{GOTO } L$ has the expansion

\[
\text{IF } s_0 \text{ GOTO } L \\
\text{IF } s_1 \text{ GOTO } L \\
\vdots \\
\text{IF } s_n \text{ GOTO } L
\]
Proof cont’d

The $\mathcal{T}$ macro **RIGHT TO NEXT BLANK** has the expansion

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

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

$$[A] \text{ LEFT}
\text{ IF } B \text{ GOTO } E
\text{ GOTO } A$$
Proof cont’d

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

\[
\begin{align*}
[C] & \quad \text{LEFT} \\
& \quad \text{IF } s_0 \text{ GOTO } A_0 \\
& \quad \text{IF } s_1 \text{ GOTO } A_1 \\
& \quad \vdots \\
& \quad \text{IF } s_n \text{ GOTO } A_n \\
[A_i] & \quad \text{RIGHT (This group of 4} \\
& \quad \text{PRINT} s_i \\
& \quad \text{LEFT} \\
& \quad \text{GOTO } C \text{ repeated for } 1 \leq i \leq n) \\
[A_0] & \quad \text{RIGHT} \\
& \quad \text{PRINT} B \\
& \quad \text{LEFT}
\end{align*}
\]
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

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

- 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.
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 [$j$]
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 $[j - 1]$
Finally, to simulate

\[
\text{IF } V_j \text{ ENDS } s_i \text{ GOTO } L
\]

we use

\[
\begin{align*}
\text{RIGHT TO NEXT BLANK } [j] \\
\text{LEFT} \\
\text{IF } s_i \text{ GOTO } C \\
\text{GOTO } D \\
[C] \text{ LEFT TO NEXT BLANK } [j] \\
\text{GOTO } L \\
[D] \text{ RIGHT} \\
\text{LEFT TO NEXT BLANK } [j]
\end{align*}
\]
When simulation ends the tape configuration is

\[ \cdots B B B x_1 \cdots x_n B z_1 B \cdots z_k y B B \cdots \]

↑

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

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

↑

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

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 $\mathcal{P}$ be a Post-Turing program that computes $f$. We need to construct a program $\mathcal{Q}$ in the language $\mathcal{S}$ that computes $f$. $\mathcal{Q}$ consists of three sections:

- BEGINNING
- MIDDLE
- END

- BEGINNING arranges the input in $\mathcal{Q}$ in the appropriate format for MIDDLE.
- MIDDLE simulates $\mathcal{P}$ 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

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

↑

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

\[
H = 3,
\]

\[
L = 2 \cdot 4^2 + 1 \cdot 4 + 4 = 40
\]

\[
R = 1 \cdot 4 + 2 = 6.
\]
An instruction PRINT$i$ is simulated by $H ← i$.
An instruction IF $s_i$ GOTO $L$ is simulated by

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

\[
\begin{align*}
L & \leftarrow \text{CONCAT}_{r+1}(L, H) \\
H & \leftarrow \text{LTEND}_{r+1}(R) \\
R & \leftarrow \text{LTRUNC}_{r+1}(R) \\
\text{IF } R \neq 0 & \text{ GOTO } E \\
R & \leftarrow r + 1
\end{align*}
\]
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:

\[ L \leftarrow r + 1 \]
\[ H \leftarrow r + 1 \]
\[ Z_1 \leftarrow \text{UPCHANGE}_{n,r+1}(X_1) \]
\[ Z_2 \leftarrow \text{UPCHANGE}_{n,r+1}(X_2) \]
\[ \vdots \]
\[ Z_m \leftarrow \text{UPCHANGE}_{n,r+1}(X_m) \]
\[ R \leftarrow \text{CONCAT}_{r+1}(Z_1, r + 1, Z_2, r + 1, \ldots, r + 1, Z_m) \]
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 \( Q \).