THEORY OF COMPUTATION
A Language for String Computations - 18

Prof. Dan A. Simovici

UMB

We introduce for each $n > 0$ a programming language $\mathcal{S}_n$ designed for string calculations on an alphabet with n symbols.

The instructions of $\mathcal{S}_n$ are:

$V \leftarrow \sigma V$	place symbol σ at the left of V
$V \leftarrow V^{-}$	delete the final symbol of the string that is the value of V; if the value is 0 leave it unchanged
$V \leftarrow V$	do nothing instruction
IF V ENDS σ GOTO L	if the value of V ends in σ then execute the first instruction with label L; otherwise proceed with next instruction

Example

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

Instr.	Effect
$x \leftarrow s_2 x$	$s_2 s_3 s_2 s_2 s_1$
$x \leftarrow x^-$	$s_3 s_2 s_2$
$x \leftarrow x$	$s_3 s_2 s_2 s_1$
IF x ENDS s_2 GOTO L	no effect
IF x ENDS s_1 GOTO L	jump to$L$

Also the instructions of $\mathcal{S}_n$ 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

$$\begin{array}{l} \text{IF } V \text{ ENDS } s_1 \text{ GOTO } L \\ \text{IF } V \text{ ENDS } s_2 \text{ GOTO } L \\ \vdots \\ \text{IF } V \text{ ENDS } s_n \text{ GOTO } L \end{array}$$

The macro $V \leftarrow 0$ has the expansion

$$\begin{array}{ll} [A] & V \leftarrow V^- \\ & \text{IF } V \neq 0 \text{ GOTO } A \end{array}$$

The macro

$$\text{GOTO } L$$

has the expansion

$$\begin{array}{l} Z \leftarrow 0 \\ Z \leftarrow s_1 Z \\ \text{IF } Z \text{ ENDS } s_1 \text{ GOTO } L \end{array}$$

The block of instructions

$$\begin{array}{l} \text{IF } V \text{ ENDS } s_1 \text{ GOTO } B_1 \\ \text{IF } V \text{ ENDS } s_2 \text{ GOTO } B_2 \\ \vdots \\ \text{IF } V \text{ ENDS } s_n \text{ GOTO } B_n \end{array}$$

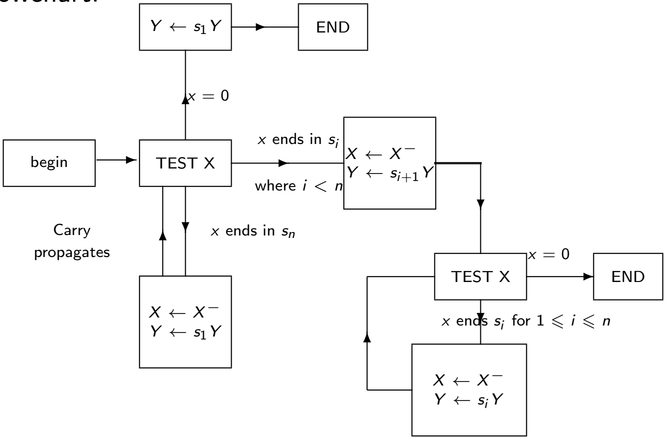
is abbreviated as

$$\text{IF } V \text{ ENDS } s_i \text{ GOTO } B_i (1 \leqslant i \leqslant n)$$

The macro $V' \leftarrow V$ for non-destructive copying of V into V' has the expansion:

$$
\begin{array}{ll}
 & Z \leftarrow 0 \\
 & V' \leftarrow 0 \\
[A] & \text{IF } V \text{ ENDS } s_i \text{ GOTO } B_i (1 \leqslant i \leqslant n) \\
 & \text{GOTO } C \\
[B_i] & V \leftarrow V^- (\text{This group of 4 repeated for } 1 \leqslant i \leqslant n) \\
 & V' \leftarrow s_i V' \\
 & Z \leftarrow s_i Z \\
 & \text{GOTO } A (\text{end group}) \\
[C] & \text{IF } Z \text{ ENDS } s_i \text{ GOTO } D_i (1 \leqslant i \leqslant n) \\
 & \text{GOTO } E \\
[D_i] & Z \leftarrow Z^- \\
 & V \leftarrow s_i V \\
 & \text{GOTO } C
\end{array}
$$

The function $x + 1$ is computable in $\mathcal{S}_n$, as shown by the following flowchart.



Example

Start with the string $s = s_2 s_1 s_1 s_3$. The numerical values is 208. Strings produced by the algorithm are:

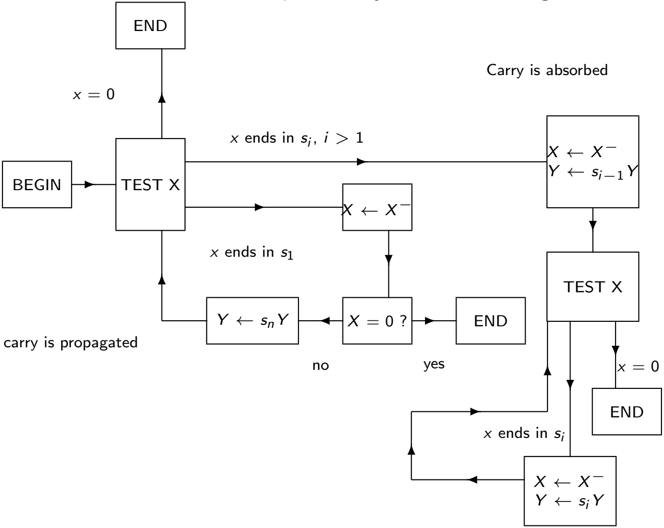
X	Y
$s_2 s_1 s_1 s_3$	s_1
$s_2 s_1 s_1$	$s_2 s_1$
$s_2 s_1$	$s_1 s_2 s_1$
s_2	$s_2 s_1 s_2 s_1$
0	

The initial value of X is $2 \cdot 3^3 + 1 \cdot 3^2 + 1 \cdot 3 + 3 = 69$; the final value of Y is $2 \cdot 3^3 + 1 \cdot 3^2 + 2 \cdot 3 + 1 = 70$.

The previous flowchart corresponds to the program

$$
\begin{array}{ll}
[B] & \text{IF } X \text{ ENDS } s_1 \text{ GOTO } A_i (1 \leqslant i \leqslant n) \\
 & Y \leftarrow s_1 Y \\
 & \text{GOTO } E \\
[A_i] & X \leftarrow X^- \text{ (This group of 3 repeated for } 1 \leqslant i \leqslant n) \\
 & Y \leftarrow s_{i+1} Y \\
 & \text{GOTO } C \\
[A_n] & X \leftarrow X^- \\
 & Y \leftarrow s_1 Y \\
 & \text{GOTO } B \\
[C] & \text{IF } X \text{ ENDS } s_i \text{ GOTO } D_i (1 \leqslant i \leqslant n) \\
 & \text{GOTO } E \\
[D_i] & X \leftarrow X^- \text{ (This group of 3 repeated for } 1 \leqslant i \leqslant n) \\
 & Y \leftarrow s_i Y \\
 & \text{GOTO } C
\end{array}
$$

The function $x \dot{-} 1$ is computed by the following flowchart:



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:

X	Y
$s_3 s_2 s_1 s_1$	0
$s_3 s_2 s_1$	s_3(carry is propagated)
$s_3 s_2$	$s_3 s_3$
s_3	$s_1 s_3 s_3$(carry is absorbed)
0	$s_3 s_1 s_3 s_3$

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

In either $\mathcal{S}$ or $\mathcal{S}_n$ 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 $\mathcal{S}_1$.

Proof.

Note that the languages $\mathcal{S}$ and $\mathcal{S}_1$ are the same. Indeed, the effect of the $\mathcal{S}_1$ instructions

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

is identical to the effect of the $\mathcal{S}$ instructions

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

The condition V ENDS s_1 in $\mathcal{S}_1$ is equivalent to $V \neq 0$ in $\mathcal{S}$. $\square$

Thus, the results involving $\mathcal{S}_n$ can be specialized to $n = 1$ to give results about $\mathcal{S}$.

Theorem

If a function is partially computable, then it also partially computable in $\mathcal{S}_n$ for each n.

Proof.

Suppose f is computed by $\mathcal{P}$ in $\mathcal{S}$. $\mathcal{P}$ is translated into a program in $\mathcal{S}_n$ by replacing instructions in $\mathcal{P}$ by a macro in $\mathcal{S}_n$:

- $V \leftarrow V + 1$ is replaced by the macro $V \leftarrow V + 1$ in $\mathcal{S}_n$;
- $V \leftarrow V - 1$ is replaced by the macro $V \leftarrow V \dot{-} 1$ in $\mathcal{S}_n$;
- IF $V \neq 0$ GOTO L by the macro IF $V \neq 0$ GOTO L in $\mathcal{S}_n$.

$\square$

$\mathcal{T}$ 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.

$\cdots$	B	B	a_2	B	a_3	a_1	B	$\cdots$
					$\uparrow$			

This is indicated by writing

$$\begin{array}{cccc} a_2 & B & a_3 & a_1 \\ & & \uparrow & \end{array}$$

There are four types of instructions in the Post-Turing Language:

PRINTσ	replace the symbol on the square being scanned by σ
IF σ GOTO L	goto the first instruction labeled L if the symbol currently scanned is σ; otherwise continue to the next instruction.
RIGHT	scan the square to the right of the current square.
LEFT	scan the square to the left of the current square.

To compute a partial function $f(x_1, \ldots, x_m)$ of m variables we start with the initial tape configuration

$$\begin{array}{l} B\ x_1\ B\ x_2\ \cdots\ x_m \\ \uparrow \end{array}$$

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 $x_1 = s_1 s_1$, $x_2 = s_1$. Then, the initial configuration is:

$$\begin{array}{lllll} B & s_1 & s_1 & B & s_1 \\ \uparrow & & & & \end{array}$$

Example

$n = 2$, $x_1 = s_1 s_2$, $x_2 = s_2 s_1$. The initial configuration is

$$\begin{array}{llllll} B & s_1 & s_2 & B & s_2 & s_1 \\ \uparrow & & & & & \end{array}$$

Example

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

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

Example

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

$$\begin{array}{lllllll} B & s_1 & s_2 & B & s_2 & s_1 & B \\ \uparrow & & & & & & \end{array}$$

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 $s_2 s_1 x$ is

$$\begin{array}{l} \text{PRINT}s_1 \\ \text{LEFT} \\ \text{PRINT}s_2 \\ \text{LEFT} \end{array}$$

The program starts with

$$\begin{array}{ll} B & x \\ \uparrow & \end{array}$$

and ends with

$$\begin{array}{llll} B & s_2 & s_1 & x \\ \uparrow & & & \end{array}$$

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

$$\begin{array}{l} B\ x \\ \uparrow \end{array}$$

the program needs to halt with the tape configuration

$$\begin{array}{l} B\ x\ s_1\ s_1 \\ \uparrow \end{array}$$

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

$$
\begin{array}{ll}
[A] & \text{RIGHT} \\
 & \text{IF } s_1 \text{ GOTO } A \\
 & \text{IF } s_2 \text{ GOTO } A \\
 & \text{IF } s_3 \text{ GOTO } A \\
 & \text{PRINT} s_1 \\
 & \text{RIGHT} \\
 & \text{PRINT} s_1 \\
[C] & \text{LEFT} \\
 & \text{IF } s_1 \text{ GOTO } C \\
 & \text{IF } s_2 \text{ GOTO } C \\
 & \text{IF } s_3 \text{ GOTO } C
\end{array}
$$

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 Bs are ignored.

Example cont'd

Example

$$
\begin{array}{ll}
[C] & \text{RIGHT} \\
 & \text{IF } B \text{ GOTO } E \\
 & \text{IF } s_2 \text{ GOTO } A \\
 & \text{IF } s_1 \text{ GOTO } C \\
[A] & \text{PRINT} B \\
 & \text{IF } B \text{ GOTO } C
\end{array}
$$

The function computed by this program satisfies

$$
\begin{aligned}
f(s_2 s_1 s_2) &= s_1, \\
f(s_1 s_2 s_1) &= s_1 s_1.
\end{aligned}
$$

Example

The previous program achieves the following computation:

$$
\begin{array}{llll}
B & s_1 & s_2 & s_1 \\
\uparrow & & & \\
B & s_1 & s_2 & s_1 \\
 & \uparrow & & \\
B & s_1 & s_2 & s_1 \\
 & & \uparrow & \\
B & s_1 & B & s_1 \\
 & & \uparrow &
\end{array}
$$

ending with Bs_1Bs_1B on the tape.

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

$$\begin{array}{l} B\ u \\ \uparrow \end{array}$$

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

$$\begin{array}{llll} B & u & B & u \\ & & \uparrow & \end{array}$$

Definition

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

$$\underset{\uparrow}{B}\ x_1\ B\ \cdots\ B\ x_m$$

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

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

$$\begin{array}{c} \cdots\ B\ \underset{\uparrow}{B}\ y\ B\ \cdots \end{array}$$

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

Thus, when $\mathcal{P}$ computes f strictly, the output y is available in a consecutive block of cells.

Theorem

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

Proof.

Let $\mathcal{P}$ be a program in $\mathcal{S}_n$ 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 $\mathcal{Q}$ be a Post-Turing program that simulates $\mathcal{P}$ step by step. We must allocate space on the tape to accommodate the values of the ℓ variables. At the begining of each simulated step the tape configuration is

$$\begin{array}{l} B\ x_1\ B\ x_2\ B\ \cdots B\ x_m\ B\ z_1\ B\ \cdots z_k\ B\ y \\ \uparrow \end{array},$$

where $x_1, \ldots x_m, z_1, \ldots, z_k, y$ are the current values of $X_1, \ldots, X_m, Z_1, \ldots, Z_k, Y$. □

Proof cont'd

Note that the initial tape configuration

$$\underset{\uparrow}{B}\ x_1\ B\ x_2\ B\ \cdots B\ x_m\ ,$$

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 $\mathcal{S}$ in $\mathcal{T}$.

Proof cont'd

We discuss a number of macros in $\mathcal{T}$:

- GOTO L
- RIGHT TO NEXT BLANK
- LEFT TO NEXT BLANK
- MOVE BLOCK RIGHT
- ERASE A BLOCK

The $\mathcal{T}$ macro GOTO L has the expansion

$$\begin{array}{l} \text{IF } s_0 \text{ GOTO } L \\ \text{IF } s_1 \text{ GOTO } L \\ \vdots \\ \text{IF } s_n \text{ GOTO } L \end{array}$$

Proof cont'd

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

$$\begin{array}{ll} [A] & \text{RIGHT} \\ & \text{IF } B \text{ GOTO } E \\ & \text{GOTO } A \end{array}$$

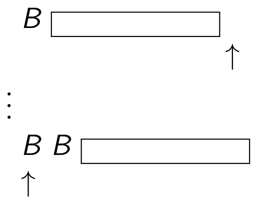
Similarly, LEFT TO NEXT BLANK has the expansion

$$\begin{array}{ll} [A] & \text{LEFT} \\ & \text{IF } B \text{ GOTO } E \\ & \text{GOTO } A \end{array}$$

Proof cont'd

The macro MOVE BLOCK RIGHT has the expansion

$[C]$ LEFT
IF s_0 GOTO A_0
IF s_1 GOTO A_1
$\vdots$
IF s_n GOTO A_n
$[A_i]$ RIGHT (This group of 4
PRINTs_i
LEFT
GOTO C repeated for $1 \leqslant i \leqslant n$)
$[A_0]$ RIGHT
PRINTB
LEFT



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. It expansion is

$$
\begin{array}{ll}
[A] & \text{RIGHT} \\
 & \text{IF } B \text{ GOTO } E \\
 & \text{PRINT} B \\
 & \ \text{GOTO } A
\end{array}
$$

Convention: a non-negative number between brakets 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 $\mathcal{S}_n$ begins and ends on the first blank;
- the value of V_i is written between the i^{th} blank and the $i+1^{\mathrm{st}}$ blank;
- if V_i is 0 we have two consecutive blanks: the i^{th} blank and the $i+1^{\mathrm{st}}$ 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 $V_j, V_{j+1}, \ldots, V_\ell$ must be all moved one square to the right tp 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
PRINTs_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

```
     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

$$\begin{array}{l}\cdots B\ B\ \underset{\uparrow}{B}\ x_1\ \cdots x_n\ B\ z_1\ B\ \cdots z_k\ y\ B\ B\ \cdots\end{array}$$

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

$$\cdots B\ B\ \underset{\uparrow}{B}\ y\ B\ B\ B\ \cdots B\ B\ \cdots$$

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

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

Thus, the program computes the function f stricly.

Consider the following statements:

1. f is partially computable;
2. f is partially computable in $\mathcal{S}_n$;
3. f is stricly 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 tp construct a program $\mathcal{Q}$ in the language $\mathcal{S}$ that computes f. $\mathcal{Q}$ consists of three sections:

BEGINNING
MIDDLE
END

- BEGINNIG 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:

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

Note that the blank represents the number $r+1$, so blank will represent the number $r+1$. For this reason, we will write B as s_{r+1}.

- $\mathcal{Q}$ simulates $\mathcal{P}$ by using the numbers that strings on this alphabet represent in base $r+1$ as codes for corresponding strings.
- The tape configuration at a stage of $\mathcal{P}$ is tracked by $\mathcal{Q}$ 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

$$\cdots B\ B\ B\ B\ s_2\ s_1\ B\ \underset{\uparrow}{s_3}\ s_1\ s_2\ B\ B\ \cdots$$

with $r = 3$ and the base 4, we have

$$\begin{aligned} H &= 3, \\ L &= 2 \cdot 4^2 + 1 \cdot 4 + 4 = 40 \\ R &= 1 \cdot 4 + 2 = 6. \end{aligned}$$

- An instruction PRINTi 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

$$
\begin{array}{l}
L \leftarrow \mathrm{CONCAT}_{r+1}(L, H) \\
H \leftarrow \mathrm{LTEND}_{r+1}(R) \\
R \leftarrow \mathrm{LTRUNC}_{r+1}(R) \\
\text{IF } R \neq 0 \text{ GOTO } E \\
R \leftarrow r + 1
\end{array}
$$

- An instruction LEFT is simulated by

$$
\begin{aligned}
& R \leftarrow \mathrm{CONCAT}_{r+1}(H, R) \\
& H \leftarrow \mathrm{RTEND}_{r+1}(L) \\
& L \leftarrow \mathrm{RTRUNC}_{r+1}(L) \\
& \text{IF } L \neq 0 \text{ GOTO } E \\
& L \leftarrow r + 1
\end{aligned}
$$

The section MIDDLE of $\mathcal{Q}$ 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 $\mathcal{Q}$ 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

$$\begin{array}{l} B\ x_1\ B\ x_2\ B\ \cdots B\ x_m \\ \uparrow \end{array}$$

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

the BEGINNING section is:

$$
\begin{aligned}
& L \leftarrow r+1 \\
& H \leftarrow r+1 \\
& Z_1 \leftarrow \mathrm{UPCHANGE}_{n,r+1}(X_1) \\
& Z_2 \leftarrow \mathrm{UPCHANGE}_{n,r+1}(X_2) \\
& \vdots \\
& Z_m \leftarrow \mathrm{UPCHANGE}_{n,r+1}(X_m) \\
& R \leftarrow \mathrm{CONCAT}_{r+1}(Z_1, r+1, Z_2, r+1, \ldots, r+1., Z_m)
\end{aligned}
$$

The END section consists of:

$$\begin{aligned} Z &\leftarrow \text{CONCAT}_{r+1}(L, H, R) \\ Y &\leftarrow \text{DOWNCHANGE}_{n,r+1}(Z). \end{aligned}$$

This concludes the description of the program $\mathcal{Q}$.