A Progr. Language for String Computations

4. The macro \( V' \leftarrow V \) has a complicated expansion.

So let us first introduce the description

\[
\text{IF } V \text{ ENDS } s_i \text{ GOTO } B_i \quad (1 \leq i \leq n)
\]

to stand for

\[
\text{IF } V \text{ ENDS } s_1 \text{ GOTO } B_1 \\
\text{IF } V \text{ ENDS } s_2 \text{ GOTO } B_2 \\
\vdots \\
\text{IF } V \text{ ENDS } s_n \text{ GOTO } B_n
\]

This is also called a filter.

Finally, let us look at two useful functions, namely

\( f(x) = x + 1 \) and \( g(x) = x - 1 \).

We want to show that these functions are computable in \( L_n \) by writing programs that compute them.

Example 1: An \( L_n \) program that computes \( f(x) = x + 1 \)

\[
\begin{align*}
[B] & \quad \text{IF } X \text{ ENDS } s_i \text{ GOTO } A_i \quad (1 \leq i \leq n) \\
 & \quad \text{GOTO } E
\end{align*}
\]

\[
\begin{align*}
[A] & \quad X \leftarrow X' \\
 & \quad Y \leftarrow s_i Y : \\
 & \quad \text{GOTO } C : \\
[A_i] & \quad X \leftarrow X' \\
 & \quad \text{IF } X = 0 \text{ GOTO } C_2 \\
 & \quad \text{GOTO } E
\end{align*}
\]

\[
\begin{align*}
[C] & \quad Y \leftarrow s_i Y \\
 & \quad \text{GOTO } B \\
[C_2] & \quad \text{IF } X = 0 \text{ GOTO } C_2 \\
 & \quad \text{GOTO } E
\end{align*}
\]

\[
\begin{align*}
[D] & \quad X \leftarrow X' \\
 & \quad Y \leftarrow s_i Y : \\
 & \quad \text{GOTO } C : \\
&D_i & \quad \text{GOTO } E
\end{align*}
\]

Example 2: An \( L_n \) program that computes \( g(x) = x - 1 \)

\[
\begin{align*}
[B] & \quad \text{IF } X \text{ ENDS } s_i \text{ GOTO } A_i \quad (1 \leq i \leq n) \\
 & \quad \text{GOTO } E
\end{align*}
\]

\[
\begin{align*}
[A] & \quad X \leftarrow X' \\ & \quad Y \leftarrow s_i Y : \\
 & \quad \text{GOTO } C : \\
[A_i] & \quad \text{IF } X = 0 \text{ GOTO } C_2 \\
 & \quad \text{GOTO } E
\end{align*}
\]

\[
\begin{align*}
[C] & \quad Y \leftarrow s_i Y \\
 & \quad \text{GOTO } B \\
[C_2] & \quad \text{IF } X = 0 \text{ GOTO } C_2 \\
 & \quad \text{GOTO } E
\end{align*}
\]

The Languages \( L \) and \( L_n \)

Now that you know the language \( L \) and the different languages \( L_n \), do you think that they are equivalent, i.e. their programs can compute the same functions?

Well, you probably have the feeling that they are. And they are indeed.

This means that a function \( f \) is partially computable if and only if it is partially computable in each \( L_n \). Therefore, \( f \) is partially computable in any one \( L_n \) if and only if it is partially computable in all of them.
### The Languages $L$ and $L_n$

**Theorem 3.1:** A function is partially computable if and only if it is partially computable in $L_1$.

**Proof:** It is easy to see that the languages $L$ and $L_1$ are really the same.

That is, the numerical effect of the instructions $V \leftarrow s \cdot V$ and $V \leftarrow V$ in $L_1$ is the same as that of the corresponding instructions in $L$: $V \leftarrow V+1$ and $V \leftarrow V-1$.

And obviously, the condition $V \text{ ENDS } s_1$ in $L_1$ is equivalent to the condition $V \neq 0$ in $L$.

Since $s_1$ is the only symbol in $L_1$, ending in $s_1$ is equivalent to being different from the null string.

**Theorem 3.2:** If a function is partially computable, then it is also partially computable in $L_n$ for each $n > 0$.

**Proof:**

Let the function $f$ be computed by a program $\varphi$ in the language $L$.

We translate $\varphi$ into a program in $L_n$ by replacing each instruction of $\varphi$ by a macro in $L_n$.

We are using the fact that, as proven before, $x + 1$ and $x - 1$ are both computable in base $n$, and so they can be used to define a macro in $L_n$.

Obviously, the new program computes in $L_n$ the same function $f$ that $\varphi$ computes in $L$.

---

### Post-Turing Programs

We will now introduce the Post-Turing language $\mathcal{T}$, which is yet another language for string manipulation. This language is named after its inventor, Emil Post, and Alan Turing, who first analyzed computation processes of this kind.

Its main difference to $L_n$ is that $\mathcal{T}$ does not use any variables.

Instead, it uses an infinite tape to store and read symbols.

This tape is divided into squares, each of which can hold a single symbol.

---

**Theorem 3.1**

**Theorem 3.2**

**Post-Turing Programs**

You can think of Post-Turing programs being interpreted by a machine that uses a tapehead to read and write symbols.

At any given moment during the computation, this tapehead is positioned on one definite square.

Each step of a computation is sensitive to only one symbol on the tape, namely the one under the tapehead.

The $\mathcal{T}$ programs that control the tapehead are written in a way similar to $L$ programs.
Post-Turing Programs

Instruction | Interpretation
--- | ---
PRINT $\sigma$ | Replace the symbol under the tapehead by the symbol $\sigma$.
IF $\sigma$ GOTO L | If the symbol under the tapehead is $\sigma$, then go to the first instruction in the program labeled L; otherwise, proceed with the next instruction.
RIGHT | Move the tapehead one position to the right.
LEFT | Move the tapehead one position to the left.

Post-Turing Programs

How do we execute the computation by our programs? First of all, in addition to our alphabet $A = \{s_1, \ldots, s_n\}$, we introduce another symbol $s_0$, which serves as a punctuation mark. This symbol is called the blank and is often written as the letter B.

Initially, all squares of the tape contain the blank, except for a finite section that contains the input to the program.

Post-Turing Programs

Since the tape itself is infinite in both directions, we only show its (usually) finite section containing all of the nonblank squares. We also need to show the current position of the tapehead. For example, we write $s_2 s_1 B s_2$ to indicate a tape consisting of the string $s_2s_1Bs_2$ with blanks to the left and right of it and the tapehead pointing at the symbol $s_1$.

The entire tape looks like this:

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

Post-Turing Programs

Example 1: We want to compute a function $f(x_1, x_2, x_3)$, where $A = \{s_1, s_2\}$, $x_1 = s_1$, $x_2 = s_1s_2s_1$, and $x_3 = s_2s_2$. Then the initial tape configuration is as follows:

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

Example 2: We want to compute a function $f(x_1, x_2, x_3)$, where $A = \{s_1, s_2\}$, $x_1 = s_1$, $x_2 = 0$, and $x_3 = s_2s_2$. Then the initial tape configuration is as follows:

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

Notice that it is impossible to distinguish this initial tape configuration from that with two inputs $x_1 = 0$ and $x_2 = s_2s_2$. Therefore, the number of arguments must be provided externally.

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:

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

Post-Turing Programs

We speak of a tape configuration as consisting of the tape contents and the position of the tapehead.

Now, if we want to compute a partial function $f(x_1, \ldots, x_m)$ of $m$ variables on $A^*$, we have to place the strings $x_1, \ldots, x_m$ on the tape initially.

We do this by separating the strings by single blanks. The initial tape configuration looks like this:

```
B x_1 B x_2 B x_3 \ldots B x_m
```

So the symbol initially scanned is the blank immediately to the left of $x_1$. 
Now let us look at the execution of a very simple Post-Turing program. The arrow in the program indicates the next instruction to be executed.

\[
\rightarrow \quad \text{PRINT } s_1 \\
\text{LEFT} \\
\text{PRINT } s_2 \\
\text{LEFT} \\
B \ x \\
\uparrow
\]

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 \( \varphi \) in the Post-Turing language \( \mathcal{T} \) is said to **compute** \( f \) if when started in the tape configuration

\[
B \ x_1 \ B \ x_2 \ B \ x_3 \ \ldots \ B \ x_m \\
\uparrow
\]

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 from the tape by ignoring all symbols except \( s_1, \ldots, s_n \).
Post-Turing Programs

Notice that this definition allows $\phi$ to contain instructions that mention symbols other than $s_1, \ldots, s_n$.

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

Do you remember our simple example program?
It turned the tape configuration

$$
\begin{array}{c}
\text{B x} \\
\uparrow
\end{array}
$$

into the configuration

$$
\begin{array}{c}
\text{B s_2 s_1 x} \\
\uparrow
\end{array}
$$

So what does it compute?
It strictly computes the function $f(x) = s_2s_1x$.

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

$$
\begin{array}{l}
\text{[A]} \quad \text{RIGHT} \\
\quad \quad \text{IF B GOTO E} \\
\quad \quad \quad \text{PRINT M} \\
\text{[B]} \quad \text{RIGHT} \\
\quad \quad \text{IF s_1 GOTO B} \\
\text{[C]} \quad \text{RIGHT} \\
\quad \quad \text{IF s_1 GOTO C} \\
\quad \quad \quad \text{PRINT s_1} \\
\text{[D]} \quad \text{LEFT} \\
\quad \quad \text{IF s_1 GOTO D} \\
\quad \quad \quad \text{IF B GOTO D} \\
\quad \quad \quad \quad \text{PRINT s_1} \\
\quad \quad \quad \quad \quad \text{IF s_1 GOTO A}
\end{array}
$$

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

$$
\begin{array}{l}
\text{[A]} \quad \text{RIGHT} \\
\quad \quad \text{IF B GOTO E} \\
\quad \quad \quad \text{PRINT M} \\
\text{[B]} \quad \text{RIGHT} \\
\quad \quad \text{IF s_1 GOTO B} \\
\text{[C]} \quad \text{RIGHT} \\
\quad \quad \text{IF s_1 GOTO C} \\
\quad \quad \quad \text{PRINT s_1} \\
\text{[D]} \quad \text{LEFT} \\
\quad \quad \text{IF s_1 GOTO D} \\
\quad \quad \quad \text{IF B GOTO D} \\
\quad \quad \quad \quad \text{PRINT s_1} \\
\quad \quad \quad \quad \quad \text{IF s_1 GOTO A}
\end{array}
$$

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

$$
\begin{array}{l}
\text{[A]} \quad \text{RIGHT} \\
\quad \quad \text{IF B GOTO E} \\
\quad \quad \quad \text{PRINT M} \\
\text{[B]} \quad \text{RIGHT} \\
\quad \quad \text{IF s_1 GOTO B} \\
\text{[C]} \quad \text{RIGHT} \\
\quad \quad \text{IF s_1 GOTO C} \\
\quad \quad \quad \text{PRINT s_1} \\
\text{[D]} \quad \text{LEFT} \\
\quad \quad \text{IF s_1 GOTO D} \\
\quad \quad \quad \text{IF B GOTO D} \\
\quad \quad \quad \quad \text{PRINT s_1} \\
\quad \quad \quad \quad \quad \text{IF s_1 GOTO A}
\end{array}
$$

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

$$
\begin{array}{l}
\text{[A]} \quad \text{RIGHT} \\
\quad \quad \text{IF B GOTO E} \\
\quad \quad \quad \text{PRINT M} \\
\text{[B]} \quad \text{RIGHT} \\
\quad \quad \text{IF s_1 GOTO B} \\
\text{[C]} \quad \text{RIGHT} \\
\quad \quad \text{IF s_1 GOTO C} \\
\quad \quad \quad \text{PRINT s_1} \\
\text{[D]} \quad \text{LEFT} \\
\quad \quad \text{IF s_1 GOTO D} \\
\quad \quad \quad \text{IF B GOTO D} \\
\quad \quad \quad \quad \text{PRINT s_1} \\
\quad \quad \quad \quad \quad \text{IF s_1 GOTO A}
\end{array}
$$
Post-Turing Programs

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

$[A]$ RIGHT $B\ M\ s_1$
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]**: \(\text{RIGHT } B m s_1 B s_1\)
  - IF B GOTO E
  - PRINT M
- **[B]**: \(\text{RIGHT } if s_1 GOTO B\)
- **[C]**: \(\text{RIGHT } if s_1 GOTO C\)
  - PRINT \(s_1\)
  - \(\rightarrow\)
  - 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]**: \(\text{RIGHT } B m s_1 B s_1\)
  - IF B GOTO E
  - PRINT M
- **[B]**: \(\text{RIGHT } if s_1 GOTO B\)
- **[C]**: \(\text{RIGHT } if s_1 GOTO C\)
  - PRINT \(s_1\)
  - \(\rightarrow\)
  - IF \(s_1\) GOTO D
  - IF B GOTO D
  - PRINT \(s_1\)
  - IF \(s_1\) GOTO A
Post-Turing Programs

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

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

Post-Turing Programs

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

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

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

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

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

```
[A] RIGHT B s_1 MB s_1
   IF B GOTO E
   PRINT M
   \uparrow
[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\}):  

\[ A \rightarrow \text{RIGHT} \quad B \; s_1 \; M \; B \; s_1 \; B \] 

IF B GOTO E 
PRINT M 

\[ B \rightarrow \text{RIGHT} \quad \text{IF } s_1 \text{ GOTO B} \] 

\[ C \rightarrow \text{RIGHT} \quad \text{IF } s_1 \text{ GOTO C} \] 

PRINT \( s_1 \) 

\[ D \rightarrow \text{LEFT} \quad \text{IF } s_1 \text{ GOTO D} \] 

\[ \text{IF } B \text{ GOTO D} \] 

PRINT \( s_1 \) 

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

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

\[ A \rightarrow \text{RIGHT} \quad B \; s_1 \; M \; B \; s_1 \; B \] 

IF B GOTO E 
PRINT M 

\[ B \rightarrow \text{RIGHT} \quad \text{IF } s_1 \text{ GOTO B} \] 

\[ C \rightarrow \text{RIGHT} \quad \text{IF } s_1 \text{ GOTO C} \] 

PRINT \( s_1 \) 

\[ D \rightarrow \text{LEFT} \quad \text{IF } s_1 \text{ GOTO D} \] 

\[ \text{IF } B \text{ GOTO D} \] 

PRINT \( s_1 \) 

\[ \text{IF } s_1 \text{ GOTO A} \]
Another example (A = \{s_1\}): 

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

- **[A]** Right
  - 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 $A$

This program computes the function $f(x) = xx$.

Or, if we interpret $x$ as a number, $f(x) = 2x$.}

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

- **[A]** Right
  - 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 $A$