ECE 664
Lecture 11
Avi Kak
Purdue Univ.

The Post-Turing Language for String Processing

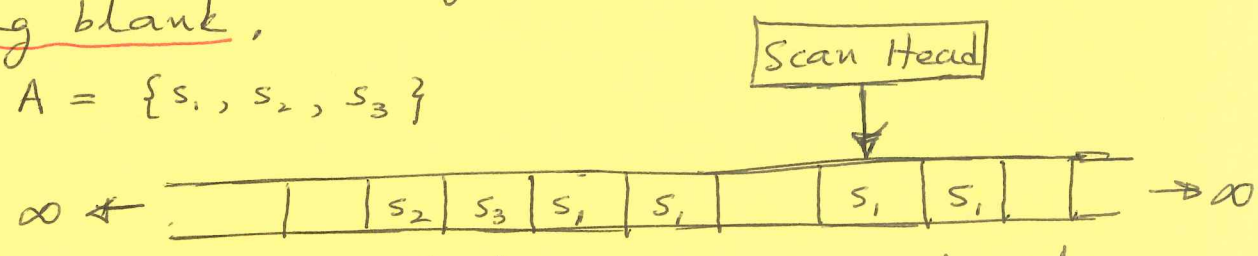
"Post" is for Emil Post — one of the greats of the first half of the last century. He was one of the founders of computability theory

- You already know the language $\mathcal{S}_n$ for string processing.
- So why do we need to learn another language for the same thing?
- ANSWER: From the previous lecture you know that if a function is partially computable in $\mathcal{S}$, then it is also partially computable in $\mathcal{S}_n$ for every n. Now we want to prove the opposite: That if a function is partially computable in $\mathcal{S}_n$ for any n, then it is also partially computable in $\mathcal{S}$. This opposite proof is best carried out with the help of the Post-Turing language $\mathcal{T}$.

The Post-Turing Language $\mathcal{T}$

- There are <u>no</u> variables in $\mathcal{T}$
- The strings to be processed are placed on a linear tape that is divided into squares with each square holding one symbol from the alphabet A, or therwise remaining blank.

$$A = \{s_1, s_2, s_3\}$$



- The computation consists of scanning the tape one square at a time and taking one of the following actions

The instruction set of $\mathcal{T}$

PRINT σ	: replace the symbol in the square by $\sigma \in A$. We are also allowed to say PRINT B where B stands for blank
IF σ GOTO L	: for any $\sigma \in A$. σ can also be B
RIGHT	: move scan head one square to the right
LEFT	: move scan head one square to the left.

- A program in $\mathcal{T}$ is a finite sequence of the above instructions.
- During the execution of a program in $\mathcal{T}$, the state of the tape will be shown as

tape configuration

$$s_2\ s_3\ s_1\ B\ s_1\ s_1\ B\ B\ s_2$$

implied that all squares to the left of the first word are blank

scan head's current position

implied that all squares to the right of the last word are blank

- To compute the partial function $f(x_1, x_2, \ldots, x_m)$, we start with the tape configuration:

$$\boxed{B\ x_1\ B\ x_2\ B\ \cdots\cdots B\ x_m}$$

Example of a Post-Turing program for computing $f(x) = 2x$ on the alphabet $A = \{s_1\}$:

```
[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
```

```
B s1 s1 B B B
B s1 s1 B B B
B M  s1 B B B
B M  s1 B B B
B M  s1 B B B
B M  s1 B B B
B M  s1 B s1 B
B M  s1 B s1 B
B M  s1 B s1 B
B s1 s1 B s1 B B
B s1 s1 B s1 B B
B s1 M  B s1 B B
B s1 M  B s1 B B
B s1 M  B s1 s1 B
B s1 s1 B s1 s1 B
```

The solution

- Now we will define what we mean by <u>computing</u> a function with a Post-Turing program $\mathcal{T}$.

DEF: We say that a program $\mathcal{P}$ in $\mathcal{T}$ computes a function $f(x_1, \ldots, x_m)$ if when started in the tape configuration

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

it eventually halts iff $f(x_1, \ldots, x_m)$ is defined (analytically) for those arguments and if, on halting, the output can be read off the tape by ignoring all symbols other than $s_1, s_2, \ldots, s_n$

DEF: A program $\mathcal{P}$ in $\mathcal{T}$ is said to compute $f(x_1, \ldots, x_m)$ **strictly** if two additional conditions are satisfied:

1) No instruction in $\mathcal{P}$ mentions any symbols other than $s_1, \ldots, s_n$ and B (sometimes denoted s_0)
2) When $\mathcal{P}$ halts, the tape contains only the output value (without any intervening blanks).

These definitions lead to the following important theorem:

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

For a formal proof of this theorem, we first define the following <u>macros</u> for programming in $\mathcal{T}$:

① **GOTO L**

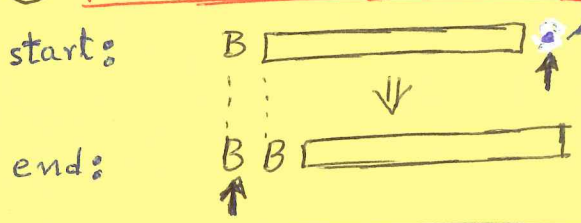
```
IF B GOTO L
IF s1 GOTO L
 :
IF sn GO TO L
```

② **RIGHT TO NEXT BLANK**

```
[A] RIGHT
    IF B GOTO E
    GO TO A
```

③ **LEFT TO NEXT BLANK**

④ **MOVE BLOCK RIGHT**



If this square is non-blank, its value gets overwritten when the block moves one square to the right.

⑤ **ERASE A BLOCK**

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

In addition to the macros, we also need the following convention regarding the configuration of the tape. We will assume the tape is in the following configuration after executing each statement of the program:

$$\underset{\uparrow}{B}\ \underbrace{V_1\ B\ V_2\ B \cdots B\ V_m}_{\text{input vars}}\ B\ \underbrace{V_{m+1}\ B \cdots B\ V_{m+k}}_{\text{local vars}}\ B\ \underbrace{V_{m+k+1}}_{\text{output var}}\ B$$

scan head

Read this first ⟹ Note that in order to prove the theorem stated above our goal is to simulate a program $\mathcal{P}$ in $\mathcal{S}_n$ with a program Q in $\mathcal{T}$. The variables named on the tape are those that show up in $\mathcal{P}$.

- With the tape-configuration convention and the macros in place, we can now easily show how to simulate the 3 instructions of $\mathcal{S}_n$ in $\mathcal{T}$:

$V_j \leftarrow s_i V_j$:

```
RIGHT TO NEXT BLANK [l]
MOVE BLOCK RIGHT [l-j+1]
RIGHT
PRINT si
LEFT TO NEXT BLANK [j]
```

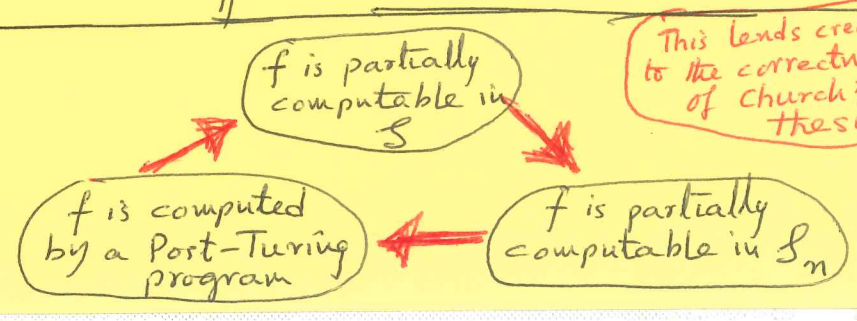
(means l times; $l = m+k+1$)

$V_j \leftarrow V_j^-$:

```
RIGHT TO NEXT BLANK[j]
LEFT
IF B GO TO C
MOVE BLOCK RIGHT[j]
RIGHT
GOTO E
[C] LEFT TO NEXT BLANK
    [j-1]
```

IF V_j ENDS s_i GOTO L: ⋮

That proves the theorem stated above. The next important theorem is to prove that $\mathcal{T}$ can be simulated in $\mathcal{S}$. With all these proofs in place we have the circle of equivalences shown at right:



This lends credibility to the correctness of Church's thesis