Theory of Computation

Prof. Michael Mascagni

Florida State University
Department of Computer Science

A Programming Language for String Computations

We introduce, for each $n > 0$, a programming language $\mathscr{S}_n$, which is specifically designed for string calculations on an alphabet $A = \{s_1, s_2, \ldots, s_n\}$ of n symbols.

- Language $\mathscr{S}_n$ has the same input, output, and local variables as $\mathscr{S}$, except that we now think of them as having values in the set A^*.
- Variables not initialized are set to 0, the empty string.

Instructions of $\mathscr{S}_n$

$V \leftarrow \sigma V$	Place the symbol σ to the left of the string which is the value of V. (For each symbol $\sigma \in A$, there is such an instruction.)
$V \leftarrow V^-$	Delete the final symbol of the string which is the value of V. If $V = 0$, leave it unchanged.
IF V ENDS σ GOTO L	If the value of V ends in the symbol σ, execute next the first instruction labeled L; otherwise proceed to the next instruction.

An m-ary partial function on A^* which is computed by a program in $\mathscr{S}_n$ is said to be *partially computable* in $\mathscr{S}_n$. If the function is total and partially computable in $\mathscr{S}_n$, it is called *computable* in $\mathscr{S}_n$.

Macros in $\mathscr{S}_n$

IF $V \neq 0$ GOTO L has the expansion

$$\begin{array}{l} \text{IF } V \text{ ENDS } \sigma_1 \text{ GOTO } L \\ \text{IF } V \text{ ENDS } \sigma_2 \text{ GOTO } L \\ \ldots \\ \text{IF } V \text{ ENDS } \sigma_n \text{ GOTO } L \end{array}$$

$V \leftarrow 0$ has the expansion

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

GOTO L has the expansion

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

Macro $V \leftarrow V'$ has the expansion . . .

$$
\begin{array}{ll}
 & Z \leftarrow 0 \\
 & V' \leftarrow 0 \\
[A] & \text{IF } V \text{ ENDS } \sigma_1 \text{ GOTO } B_1 \\
 & \vdots \\
 & \text{IF } V \text{ ENDS } \sigma_n \text{ GOTO } B_n \\
 & \text{GOTO } C \\
[B_i] & V \leftarrow V^- \\
 & V' \leftarrow s_i V' \\
 & Z \leftarrow s_i Z \\
 & \text{GOTO } A \\
[C] & \text{IF } Z \text{ ENDS } \sigma_1 \text{ GOTO } D_1 \\
 & \vdots \\
 & \text{IF } V \text{ ENDS } \sigma_n \text{ GOTO } D_n \\
 & \text{GOTO } E \\
[D_i] & Z \leftarrow Z^- \\
 & V \leftarrow s_i V \\
 & \text{GOTO } C
\end{array}
$$

Two Theorems

Theorem 3.1. A function is partially computable if and only if it is partially computable in $\mathscr{S}_1$. □

Theorem 3.2. If a function is partially computable, then it is also partially computable in $\mathscr{S}_n$ for each n. □

Post-Turing Programs

The Post-Turing language $\mathscr{T}$ is yet another programming language for string manipulation.

- Unlike $\mathscr{S}_n$, the language $\mathscr{T}$ has no variables. All of the information being processed is placed on one linear tape.
- The tape is thought of as infinite in both directions. Each step of a computation is sensitive to just one symbol on the tape, the symbol on the square being "scanned".

Instructions of $\mathscr{T}$

PRINT σ	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 immediately to the right of the square presently scanned.
LEFT	Scan the square immediately to the left of the square presently scanned.

Blanks

When dealing with string functions on the alphabet $A = \{s_1, s_2, \ldots, s_n\}$, an additional symbol, written s_0 and called the *blank*, is used as a punctuation mark. Often we write B for the blank instead of s_0.
To compute a partial function $f(x_1, \ldots, x_m)$ of m variables on A^*, we place the m strings $x_1, \ldots, x_m$ on the tape initially; they are separated by single blanks.

$$\overset{\downarrow}{B} \;\; x_1 \; B \; x_2 \; \ldots \; B \; x_m \; B$$

Computability in $\mathscr{T}$

Let $f(x_1, \ldots, x_m)$ be an m-ary partial function on the alphabet $A = \{s_1, \ldots, s_m\}$. The program $\mathscr{P}$ in the Post-Turing language $\mathscr{T}$ is said to *compute* f if when started in the tape configuration

$$\overset{\downarrow}{B} \;\; x_1 \; B \; x_2 \; \ldots \; B \; x_m \; B$$

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 $s_1, \ldots, s_n$.

The program $\mathscr{P}$ is said to compute f strictly if, in addition,

1. no instruction in $\mathscr{P}$ mentions any symbol other than $s_0, s_1, \ldots, s_m$;
2. whenever $\mathscr{P}$ halts, the tape configuration is of the form

$$\ldots \; B \; B \; B \; \overset{\downarrow}{B} \;\; y \; B \; B \; \ldots$$

where the string y contains no blanks.

Simulation of $\mathscr{S}_n$ in $\mathscr{T}$ and simulation of $\mathscr{T}$ in $\mathscr{S}$

Theorem 5.1. If $f(x_1, \ldots, x_m)$ is partially computable in $\mathscr{S}_n$, then there is a Post-Turing program that computes f strictly. $\square$

Theorem 6.1. If there is a Post-Turing program that computes the partial function $f(x_1, \ldots, x_m)$, then f is partially computable. $\square$

Turing Machines

Informally, a Turing consists of a finite set of internal states $q_1, q_2, \ldots$, an finite set of symbols $s_0, s_1, s_2, \ldots$ that can appear on the tape (where $s_0 = B$ is the "blank"), and and a finite set of quadruples representing all possible transitions operating on a linear tape. The quadruple is in one of the following three forms:

1. $q_i \ s_j \ s_k \ q_l$
2. $q_i \ s_j \ R \ q_l$
3. $q_i \ s_j \ L \ q_l$

with the intended meaning that,

1. when in state q_i scanning symbol s_j, the device will print s_j and go into state q_l;
2. when in state q_i scanning symbol s_j, the device will move one square to the right and then go into state q_l;
3. when in state q_i scanning symbol s_j, the device will move one square to the left and then go into state q_l.

Turing Machines, Continued

A *deterministic Turing machine* satisfies the additional "consistency" condition that no two quadruples begin with the same pair $q_i\ s_j$.

The alphabet of a given Turing machine $\mathscr{M}$ consists of all of the symbols s_i which occur in quadruples of $\mathscr{M}$ except s_0.

A Turning machine always begins in state q_1. It halts if it is in state q_i scanning s_j and there is no quadruple that begins with $q_i\ s_j$.

Computations by Turing Machines

Using the same convention with Post-Turing programs, it should be clear what it means to say that some given Turing machine $\mathscr{M}$ *computes* a partial function f on A^* for a given alphabet A.

We further say that $\mathscr{M}$ computes a function f *strictly* if

1. the alphabet of $\mathscr{M}$ is a subset of A;
2. starting with the initial configuration $\overset{q_1}{B}\ x$, whenever $\mathscr{M}$ halts, the finial configuration has the form $\overset{q_i}{B}\ y$, where y contains no blanks.

Turing Machines, Examples

Writing $s_0 = B, s_1 = 1$, and considering the Turning machine $\mathscr{M}$ with alphabet $\{ \ 1 \ \}$ and the following transitions:

$$
\begin{array}{r}
q_1 \ B \ R \ q_2 \\
q_2 \ 1 \ R \ q_2 \\
q_2 \ B \ 1 \ q_3 \\
q_3 \ 1 \ R \ q_3 \\
q_3 \ B \ 1 \ q_1
\end{array}
$$

What does $\mathscr{M}$ compute?

Three Theorems

Theorem 1.1. Any partial function that can be computed by a Post-Turing program can be computed by a Turing machine using the same alphabet. □

Theorem 1.2. Let f be an m-ary partially computable function on A^* for a given alphabet A. Then there is a Turing machine $\mathscr{M}$ that computes f strictly. □

Theorem 1.4 Any partial function that can be computed by a Turing machine can be computed by a Post-Turing program using the same alphabet. □