A mathematical model of character string manipulation

by SAKTI PRAMANIK
Indiana University—Purdue University at Indianapolis
Indianapolis, Indiana

INTRODUCTION

Every insert and delete operation on a string of characters causes it to expand or contract in memory. If on the other hand, these commands are saved, the character string can be updated at the end of the editing session.¹² The advantage of doing this is to move any character at the most once in memory. This is possible because the final position of a character in the string can be determined from the saved edit functions. In Figure 1 below it is shown that a “Delete character in position 3” and then an “Insert a character \( \text{X} \) before the 3rd character position” can be combined into a single edit function, “Replace the character in position 3 by the character \( \text{X} \).”

Combining the edit functions

<table>
<thead>
<tr>
<th>Initial Character string</th>
<th>After Delete</th>
<th>After Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDE</td>
<td>ABCDE</td>
<td>ABXDE</td>
</tr>
<tr>
<td>Delete</td>
<td></td>
<td>Insert X</td>
</tr>
</tbody>
</table>

Figure 1—Combining the effect of delete and insert functions into a single edit function.

One difficulty of combining several edit functions as above is that they are issued relative to the character position in the current string while the combined edit function has to work on the characters in the initial string. For example, deleting the 4th character of the initial string and then deleting the 4th character of this updated string mean deleting the 4th and the \((i+1)\)th characters of the initial string. It seems that the above combining process requires the transformation of the editorial point in the current string into a corresponding editorial point in the initial string. But this conversion may not always be possible because the mapping may not exist.

Instead of defining the editorial point on a reference frame which is always changing a static frame of reference is created. This is done by defining a sequence of character slots. Characters of the string are stored in these slots in sequence. Three microscopic edit operators are defined which operate on the contents of these slots rather than on the character direct by its position in the string.

DEFINITION OF THE OPERATORS AND THEIR ALGEBRAIC PROPERTIES

Let \( S_i \) denote the \( i \)th character slot. So \( S_i \) also represents the slot location of the \( i \)th character in the current string. Thus, \( S_i \) denotes the slot containing the left most character in the string. The three basic micro operators, \( B_i, F_i, \) and \( R_i \), are then defined as follows:

\[
B_i = S_i \rightarrow \text{X}
\]

i.e., the character \( \text{X} \) is moved into the \( i \)th slot and the previous character in that slot is lost.

\[
F_i = S_i \rightarrow S_{i+1} \rightarrow \text{C}(S_{i+1})
\]

i.e., the character in the \( i \)th slot is moved into the \((i+1)\)th slot and the previous character in that \((i+1)\)th slot is lost. Then the character \( \text{X} \) is moved into the \( i \)th slot. Similarly,

\[
R_i = S_{i+1} \rightarrow S_i \rightarrow \text{C}(S_{i+1})
\]

the content of the \((i+1)\)th slot is moved into the \( i \)th slot and the previous character in the \( i \)th slot is lost. A character \( \text{X} \) is then moved into the \((i+1)\)th slot.

In the above definition, \( \text{X} \) represents any single character; for blank characters however, no superscript will be used. For example, \( B_i \) will move a ‘blank’ character into the \( i \)th slot. The following equivalence relations hold good for the above operators. They indicate that the operators on the right hand side of a relation eventually result in the same characters in the slots as those operators on the left hand side; but the ones on the right hand side achieve this by a minimum amount of character movement. The convention that the operators in a string gets executed in sequence from right to left, is assumed. For example, in the operator string \( B_i, F_i, \) \( F_i \) is executed first, and then \( B_i \).
DEFINITION OF INSERT AND DELETE FUNCTIONS

Function to insert a character \( X \) before the \( i \)th character in a character string of length \( N \) is represented by the operator string

\[
F_i^X = F_{i+1}^X \ldots F_{N-1}^X F_N^X
\]

where the \( F_i^X \)s are performed in a sequence from right to left. So \( F_{N-1}^X \) moves the character in the \( N \)th slot of the \( (N-1) \)th slot and then moves the character \( X \) into the \( N \)th slot. Then \( F_{N-2}^X \) moves the character in the \( (N-1) \)th slot to \( N \)th slot and then the character \( X \) into the \( (N-1) \)th slot, and so on.

The function to delete the \( i \)th character in a character string of length \( N \) is represented as follows:

\[
R_i = R_{i+1} \ldots R_{N-2} R_{N-1} R_i
\]

Similarly, the \( R_i \)s are performed from right to left in the operator string above. We can now represent a sequence of insert and delete functions by concatenating the strings of \( F_i^X \)s and \( R_i \)s. The temporal sequence of the edit functions are preserved by concatenating the operator string of the next edit function to the left of the string of the previous edit functions. The advantages of representing a sequence of edit functions by a concatenated operator string is that this string can be simplified considerably by using the relations 1 through 14. For example, the delete and the insert functions of Figure 1 is represented by the operator strings \( R_6 R_3 \) and \( F_3 F_4 F_5 \) respectively. To represent the combined effect of the two edit functions, we concatenate the operator strings as follows:

\[
F_3^X F_4^X R_3 R_6
\]

By using relation 1, \( F_4^X R_3 \) is reduced to \( B_4^X \). This \( B_4^X \) in turn can be merged with \( F_5^X \). The above operator string now becomes \( F_3^X R_5 \) which by using Relation 1 again becomes \( B_5^X \). Thus the combined effect of the two edit functions is the micro-operator \( B_5^X \) which imply moving the character \( X \) into slot 3. It should be noted that slot 3 is also the position of the third character in the initial character-string. In other words, \( B_5^X \) replaces the third character in the initial character-string by a character \( X \). A systematic approach to combine a sequence of edit functions to produce an optimized operator string is presented in the following section.

REDUCTION ALGORITHM

This algorithm starts by merging the operator strings of the first and the second edit functions. The string thus obtained is now merged with the operator string of the third edit function. This process of merging two operator strings at a time continues until the operator strings of all the edit functions have been merged. The merging is done by taking one operator from the left string and comining toward right through the operators of the right string until it combines with an operator or finds a place in the right string where it can not be combed any further to the right. The combing is essentially done by commuting the operator successively with its right neighbor in the string. Combining two operators involves finding the previous relation whose left hand side corresponds to these two operators; then replacing them by the operators of the right hand side of the relation. These operators may now be combined with other operators of the right string. The following example shows how the operator strings for the three edit functions have been merged into a single operator string. The edit functions, in the order they are issued, are: (1) insert a character \( X \) before the 9th character; (2) insert a character \( Y \) before the 9th character; (3) delete the 6th character.

Assuming an initial character string of length 10, the operator strings for the above edit functions are: (1) \( F_9^X F_{10}^Y \); (2) \( F_9^Y F_{10}^Y F_{11}^Y \); (3) \( R_9 R_9 R_9 R_8 R_8 \); respectively.

The merging process starts with the operator string of the first edit function, i.e., \( F_9^X F_{10}^Y \). The second operator string is now merged with this from left as follows:

\[
F_9^X F_{10}^Y \xrightarrow{\text{merge with}} F_9^Y F_{10}^X
\]

Producing \( F_9^Y F_{10}^Y F_9^X F_{11}^Y F_{10}^X \).

See that \( F_{11}^Y \) of the left string has commuted with \( F_{10}^X \) of the right string. This resulting string is now merged with the 3rd operator string as follows:

\[
R_9 R_9 R_9 R_8 R_8 \xrightarrow{\text{merge with}} F_9^Y F_{10}^Y F_9^X F_{11}^Y F_{10}^X \quad R_6 \text{ combs all the way through the right string and so does}
\]
Mathematical Model of Character String Manipulation

Let \( R_7 \) producing

\[
R_1 R_9 R_7 R_8 \xrightarrow{\text{merge with}} F_7 F_{10}^t F_4^t F_{11}^t F_6^t R_7 R_6
\]

\( R_8 \) now combines with \( F_9^t \), thus both of them are replaced by \( B_9^t F_9 \) as follows:

\[
R_1 R_9 R_7 R_8 \xrightarrow{\text{merge with}} F_8^t F_9 F_{10}^t F_4^t F_{11}^t F_6^t R_7 R_6
\]

The new operator \( B_9^t \) is now combed through its right neighbors and eventually combines with \( R_7 \) to produce

\[
R_1 R_9 R_7 R_8 \xrightarrow{\text{merge with}} F_9^t F_{10}^t F_4^t F_{11}^t F_6^t R_7 R_6
\]

Now \( R_8 \) combines with \( F_{10} \) to produce \( B_{10} \), and this \( B_{10} \) combines with \( F_{10}^t \) to produce \( F_{10} \). Continuing this process we get the final merged string

\[
F_9^t F_{10}^t R_7 R_6
\]

The resulting merged string consists of a disjoint set of substrings of only \( F_s \), or only \( R_s \), or only \( B_s \). For example, the merged string above consists of the disjoint substrings \( F_9^t F_{10}^t \) and \( R_7 R_6 \). These two substrings are disjoint because \( R_7 R_6 F_9^t F_{10}^t \) is equivalent to \( F_9^t F_{10}^t R_7 R_6 \). The relative order of these disjoint substrings is important, however, for the merging algorithm discussed above because it may result in an incomplete merge if the substrings are not properly ordered. For example, a fourth edit function “Delete the 8th character” is issued after the three edit functions, discussed above. The operator string for the fourth edit function is \( R_9 R_7 R_8 R_9 \). If the right string is \( R_9 R_7 R_8 F_9^t F_{10}^t \) then the merged string will be \( R_9 R_7 R_8 F_9^t F_{10}^t \). This cannot be reduced any further because \( R_6 \) does not commute with \( R_7 \). On the other hand if the right string is \( F_9^t F_{10}^t R_9 R_7 R_8 \), then the merged string is reduced considerably to \( R_8 R_6 \). This is because \( R_8 \) combines with \( F_9^t \), and so on.

It can be shown that the merging algorithm will always produce a completely reduced string when the disjoint substrings of the right string are kept in descending order from left to right; the ordering is done by their highest subscript.

A computer program has been written to implement the above merging algorithm. For random input data it is found that the number of commutations (required for combing) increases very rapidly with the increasing length of the character string. The number of commutations can be reduced considerably if combing is done for a group of operators at a time rather than taking only one operator at a time from the left string. This reduction is possible because the substrings of the operators within a group are sequential. The following table gives the merged substring in terms of the

<table>
<thead>
<tr>
<th>( i )</th>
<th>( j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 \leq i &lt; j )</td>
<td>( 1 \leq j &lt; i )</td>
</tr>
<tr>
<td>( 1 \leq i )</td>
<td>( j &lt; 1 )</td>
</tr>
<tr>
<td>( i &lt; 1 )</td>
<td>( j \geq 1 )</td>
</tr>
</tbody>
</table>

The merged operator string works as an efficient mapping between the original character string and the updated character string. For example, a text file on a tape unit can be considered as a continuous string of characters and all the update information about this text string can be maintained through a merged operator string. The merging process requires a fair amount of string manipulation. This, however, remains bounded when the number of inserts and deletes are evenly distributed over time and space.

ACKNOWLEDGMENT

I would like to thank professors Edgar T. Irons and Alan J. Perlis of Yale University for many helpful conversations.

REFERENCES
