XVII. MECHANICAL TRANSLATION*

V. H. Yngve
J. R. Applegate
A. N. Chomsky (Absent)
W. S. Cooper
U. C. Dickman
D. A. Dinneen
E. S. Klima
K. C. Knowlton
G. H. Matthews
R. A. Rothstein
J. Viertel

A. COMPACT ENCODING OF REFERENCE DATA

In some computer applications it is essential that the program have quick access to large quantities of reference data. The data might be a list of employees' names and their payroll numbers, or possibly a catalog of manufactured items with their prices. The mechanical translation of languages is such an application, for a large dictionary of some sort must be stored in a rapid-access memory medium, no matter how sophisticated the translation program is. In these situations it is desirable to store the data in as compact a form as possible – to compress it, without losing any useful information. Various compression schemes have been devised; most of them are applicable only to the argument part of an item of data, and not to the function-value part. These methods usually introduce a risk of look-up error, which increases as higher rates of compression are demanded. In this report we summarize a system that is confined to the argument part of each item of data and introduces no risk of look-up error, even though it provides a high degree of compression. The system requires only that the item that is to be looked up shall be accounted for by the data, and that the arguments shall be arranged in numerical order.

For simplicity, let us consider a list of binary arguments listed in ascending numerical order (see List 1). In every row of such a list except the first, we single out the leftmost bit that differs from the neighbor immediately above it. All such bits are, per force, 1-bits. In List 1 these 1-bits have been underlined. The compression system exploits the fact that knowledge of these bits alone is sufficient to deduce the position of any item to be looked up in the list. That is, for look-up purposes we could reduce List 1 to a second list (List 2).

This meager list (List 2) contains by implication a surprising amount of information about List 1. To show that it contains enough information for look-up purposes, we reconstruct as much as possible of List 1, using only the information shown in List 2 and utilizing the properties of the underlined 1-bits. As an example, consider the neighbor immediately below an underlined 1-bit. This must itself be a 1-bit, provided that it lies to the left of the underlined 1-bit in its own row. For the latter underlined 1-bit, by definition, indicates that all bits to its left in its row are identical with their upstairs neighbors. Continuing by induction, we can reason that an underlined 1-bit is the topmost bit of a column of 1-bits which extends downward through all successive rows whose own underlined bits lie farther to the right than the column. When the

*This work was supported in part by National Science Foundation.

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 1 0 0 1
0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 1 0
0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0 1 0 0 0 1 1 1 0
0 0 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 0 1 1 0 0 1 0
0 0 0 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 1 0 0 0 1
0 1 0 1 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 1 1 1 0 0
0 1 0 1 1 1 0 0 1 1 0 0 1 0 0 1 0 1 1 0 1 1 0 0

List 1.

- -
- - - - - - - - - - - - - - - - - - - 1 - - - - -
- - - - - - - - 1 - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - 1 - - - - - - - -
- 1 -
- 1
- - - - - - - - - - 1 - - - - - - - - - - - - -
- - - - - - - - - - 1 - - - - - - - - - - - - -
- - - 1 -
- - - - - - - - - - - - - - 1 - - - - - - - - -
- 1 -
- - - - - - - - 1 - - - - - - - - - - - - - - -

List 2.

- 0 - 0 - - - - 0 - - - - - - - - - 0 - - - - -
- 0 - 0 - - - - 0 - - - - - - - - - 1 - - - - -
- 0 - 0 - - - - 1 - 0 0 - - 0 - - - - - - - - -
- 0 - 0 - - - - 1 - 0 0 - - 1 - - - - - - - 0 -
- 0 - 0 - - - - 1 - 0 0 - - 1 - - - - - - - 1 0
- 0 - 0 - - - - 1 - 0 0 - - 1 - - - - - - - 1 1
- 0 - 0 - - - - 1 - 0 1 - - - - - - - - - - - -
- 0 - 0 - - - - 1 - 1 - - - - - - - - - - - - -
- 0 - 1 - - - - - - - - - - 0 - - - - - - - - -
- 0 - 1 - - - - - - - - - - 1 - - - - - - - - -
- 1 - - - - - - 0 - - - - - - - - - - - - - - -
- 1 - - - - - - 1 - - - - - - - - - - - - - - -

List 3.

column arrives at a row whose underlined 1-bit lies to the left of the column bits, it is uncertain whether the column of 1-bits continues down into this row or not. So if we confine ourselves to what can be deduced with certainty, the column of 1-bits terminates just above this uncooperative row. Such columns display all the 1-bits that are contained in or implied by List 2. Next, notice that the definition of an underlined 1-bit implies that the neighbor immediately above any underlined 1-bit must be a 0-bit. Reasoning as before, we see that such a 0-bit must be the bottom bit of a column of 0-bits which extends upward past all underlined 1-bits that lie to its right, and that the column, in fact, extends into the first row it encounters whose underlined 1-bit lies leftward of the column. Such columns display all of the 0-bits that are implied by List 2.

We have been able, a priori, to augment the second list sufficiently to form List 3. This list has the surprising property that any row differs from any other row in at least one place; that is, for two arbitrary rows, some 0-bit in the upper row will lie directly above a 1-bit in the lower row. To see this, consider the rows between the two rows in question, and also the lower row itself. One of these rows must contain an underlined 1-bit that lies farther toward the left than the underlined 1-bits of the other rows; this 1-bit must be at the top of a column of 1-bits which extends down into the lower row in question, and this 1-bit must also lie under a column of 0-bits which extends into the upper row in question. Thus any row of List 3 is distinguishable from any other row, in spite of the large proportion of unknown bits. This proves that any item that could be looked up in List 1 could also be looked up in List 3, for it would be distinguishable from all but one member of the third list.

The look-up routines need not be as forbiddingly involved as List 3 might suggest, for this list actually contains some superfluous information. The item to be looked up corresponds, in reality, to some particular row of List 3 – viz., the row that lies in the same position in List 3 as does the row in List 1 which matches the item being looked up. We regard this row in List 3 as the "correct" row, and we require a look-up program to point out its position. The item does not correspond to any row above the correct row, and a distinction will always be apparent between the item and such a (higher) row, for the row must contain a 0-bit in at least one place where the item contains a 1-bit. Similarly, the item can be differentiated from any row lying below the correct row, for a lower row must contain at least one 1-bit where the item contains a 0-bit. Observe now that the 0-bits shown in the third list are not needed, for it is sufficient that the item be distinguishable from all rows lying below the correct row. Therefore, for the purposes of a look-up program, the correct row may be defined as that row for which all succeeding rows contain at least one 1-bit in the same place that the item contains a 0-bit. With this definition, the look-up routine need reconstruct only columns of 1-bits, and this is a one-way procedure whereby successive rows are generated as the routine works downward through the list. A still simpler algorithm avoids the

formation of parts of some columns of 1-bits; only one column is essential in any given row, provided only one item is being looked up. These look-up procedures and their variations are more fully described in Mechanical Translation (1).

The second list, which displays only the underlined 1-bits, comprises all of the information that needs to be stored. A simple way to code this information would be to number the columns of the list, and then store for each row the column number of its underlined 1-bit. With this system, N-bit arguments could be compressed to a length of $[\log_2 N]$ bits, where the square brackets indicate that the quantity within must be increased until it is an integer. Should the computer happen to operate with a base $b \neq 2$, then the compression system can be modified in a straightforward manner, so that an N character argument compresses to $[1 + \log_b N]$ characters. These compression rates could be improved a little by utilizing certain weak interdependencies among the underlined digits themselves, but the improvement would be slight and would entail a more complex look-up procedure.

Here is an illustration of the capabilities of this sort of compression system: The name-and-address part of every listing in a city telephone directory could be replaced by a two-letter code in such a way that a user could still look up the phone number of any friend whose name and address he knew exactly .

W. S. Cooper

References

1. W. S. Cooper, The storage problem, MT, Vol. II, No. 5 (in press).