On the Multiplying Ability of Two-Way Automata

J. R. H. DEMPSTER

Computing Center, University of British Columbia, Vancouver, British Columbia

Received June 27, 1968

ABSTRACT

It is shown that multiplication and square root extraction can be performed by the two-way automata of Kreider and Ritchie, thus answering questions raised by those authors. Square root extraction is straightforward (yielding an integer root and a remainder), but multiplication is achieved only by conversion of the input data from binary to ternary notation. In effect the Kreider-Ritchie machine can be and is here used as a weak linear bounded automation (Turing machine with access limited to \( k_1 + k_0 \) tape cells when the input length is \( l \) cells) with \( k_1 = 1.5 \), whereas it was intended to have \( k_1 = 1 \). The possibility of multiplication in the strict \( k_1 = 1 \) case remains unknown.

INTRODUCTION

The classification of numerical functions on the basis of the types of automata able to compute them is a problem of considerable interest. The simple operation of multiplying raises difficulties in this regard. Thus Kreider and Ritchie [1], discussing the properties of a class of automata with severely restricted working storage, leave the question open whether such a machine can compute the product of two given numbers (though if the three numbers \( x, y, z \) are given, it is possible to determine whether \( yd = x \cdot y \)).

The difficulty is with the limitation of storage. The two-way automaton of Kreider and Ritchie is a finite automaton which is allowed not only to scan its input tape in both directions, but also to write on it (they use the alphabet \( \{0, 1, B\} \)). It is not, however, allowed to expand its tape beyond the length of the input string, except perhaps for convenience by a specified finite number \( k_0 \) of cells.\(^1\)

\(^1\) In spite of this limitation, Kreider and Ritchie are able to show that this class of machines contains a machine universal over the class. However the class of computable functions is not easily characterized independently of machines, largely because it is not closed under such a basic operation as identification of variables. For example, \( g(x) = f(x, x) \) may not be computable even though \( f(x, y) \) is. As this paper will show, \( f(x, y) = x \cdot y \) (multiplication) is an example.
Equivalently, it may be regarded as a Turing machine which is restricted to using at most \( k_0 \) cells of its tape outside the input string. To compute the function

\[
y = f(x_1, x_2, \ldots, x_n),
\]

the automaton is given an input tape containing binary representations of the numbers \( x_1, x_2, \ldots, x_n \) separated by cells holding the "blank" symbol \( B \). If \( k_0 \) extra "blank" cells are added at the end, the input tape is described by the string

\[
Bx_1Bx_2B \cdots Bx_nBB^{k_0}.
\]

It is required to output a binary number \( y \) in the form

\[
B^{r}yB^s \quad (r, s \geq 1).
\]

Clearly, many functions cannot be computed by such an automaton simply because the output would exceed the input in length. The function \( y = x^2 \) is a simple example. This is not the case for the product function \( z = x \cdot y \). Indeed, if \( x, y \) require \( m, n \) bits respectively, \( z \) requires just \( m + n \) bits. However, familiar methods of carrying out a multiplication (e.g., an iterated shift and add) will require \( m + 2n \) (or \( 2m + n \)) bits of working storage throughout the computation. The question arises whether this is an intrinsic limitation, or whether it can be overcome by a change in method.

This note will show that multiplication can in fact be carried out on the two-way automaton as defined above. However, the method to be used violates the spirit if not the letter of the restrictions imposed on the machine. In effect, the machine described by Kreider and Ritchie is not strictly in the class of two-way automata, but is rather a low-ranking member of the class of linear bounded automata. The question of whether multiplication is possible on a strictly interpreted two-way automaton is unanswered, as is that of defining precisely what is meant by "strictly interpreted two-way automaton." However the present result does show clearly that more precise definitions are needed if a proof of impossibility is to be achieved.

**EXTRACTION OF SQUARE ROOT**

Before elaborating these comments, we note one or two other questions that can be answered affirmatively for strict two-way automata. To bypass the restriction that output length may not exceed input length, Kreider and Ritchie focus attention on the use of their automaton to recognize relations between given numbers. They note that many natural relations of number theory can be handled, but speculate that \( \sqrt{y} = x \) and "\( x \) is a perfect square" may be exceptions. In fact, however, these are both possible in a straightforward way, as is evaluation of the function \( y = x^{1/2} \), yielding an integer root and a remainder. Thus there are still no known examples of simple relations beyond the capability of a two-way automaton recognizer.
The square root calculation can use essentially the well-known longhand method. The essential element in a proof is to show that no more than the allowed storage is used at any stage. The following proof is generalized to an arbitrary number base. Suppose that \( x \), represented in the base \( b \), is a number of \( 2n \) or \( 2n - 1 \) digits. Then its square root \( y \) will have \( n \) digits:

\[
y = \sum_{i=1}^{n} q_i b^{n-i}, \quad 0 < q_i < b - 1.
\]

Let \( u_k \) represent the first \( k \) digits of \( y \),

\[
u_0 = 0, \quad u_k = \sum_{i=1}^{k} q_i b^{k-i} < b^k,
\]

and define

\[
x_k = x - (u_k b^{n-k})^2.
\]

Then \( u_1, u_2, \ldots, u_n = y \) can be calculated by the iteration:

\[
x_k = x
\]

\[
x_{k+1} = x_k - q_{k+1} b^{2(n-k-1)} (2bu_k + q_{k+1})
\]

\[
u_{k+1} = bu_k + q_{k+1}
\]

where at each step \( q_{k+1} \) is the largest possible value that does not make \( x_{k+1} < 0 \) (this will of course satisfy \( q_{k+1} \leq b - 1 \)). Hence

\[
x_k < (q_{k+1} + 1) b^{2(n-k-1)} (2bu_k + q_{k+1} + 1)
\]

\[
< 2b^{2n-k}.
\]

Thus \( x_k \) requires at most \( 2n - k + 1 \) digits, while \( u_k \) of course requires \( k \) digits. Consequently the digit storage required does not at any stage exceed \( 2n + 1 \) digits, and the computation can be carried out by a two-way automaton with at most a fixed number \( k_0 \) of additional tape cells. The final result will be \( u_n = y \) and \( x_n = x - y^2 < 2y + 1 \), where \( x_n = 0 \) if and only if \( x \) is a perfect square.

**Multiplication**

The scheme for multiplication is based on a suggestion by Ritchie [2] that the calculation might be possible by using the marker character \( B \) to help encode the information more compactly, provided a way could be found to do this without risk of "falling off the end of the tape." This can in fact be done, by the following device. Every pair of tape cells, each containing one of the three symbols 0, 1, \( B \), has nine
MULTIPLICATION BY TWO-WAY AUTOMATA

423

possible configurations. Let these represent (in any order) the octal digits 0, 1,..., 7 and a marker symbol $M$. Then rearrange the input tape so that it has the form

$$M^rMxMyM$$

where $\bar{x}$, $\bar{y}$ are octal representations of the input numbers $x$, $y$. This amounts to a recoding in which each 3 cells of the binary representations of $x$ and $y$ are compressed into two cells in the "ternary coded octal" representation, while the three $B$'s are expanded into the two-cell combinations $M$.

If $\bar{x}$ requires $m$ and $\bar{y}$, $n$ octal digits, the digit space required for the new representation is $2(m + n)$ tape cells. However the binary representations of the input used $3(m + n)$ cells. Thus, assuming $m \leq n$, the recoding has released $m + n \geq 2m$ tape cells, or at least $m$ double cells. Provided a few additional cells are made available for markers, the total of $2m + n$ octal cells is sufficient for multiplication by the familiar "shift and add" method.\(^2\)

The recoding is easily carried out by the two-way automaton (a possible program is given in an appendix) and the final result can be restored to binary form without further difficulty.

CONCLUSION

The result is remarkably unsatisfying. One feels intuitively that it has been achieved by an unfair trick, that ought somehow to be ruled illegal. Even so, it has only barely been achieved at all. Because $3^3 = 2^3 + 1$, the ternary encoding was just able to provide the necessary marker $M$ in addition to the octal digits, and the resulting compression of numbers was in exactly the 3:2 ratio needed to allow multiplication in the usual way. (It is easily checked that a corresponding trick would have failed if the original number base had exceeded 2.) One is unhappy that a result of possible significance should hinge on such an accident of the encoding.

The uncertainty is to some degree resolved by the realization that the machine of Kreider and Ritchie is not truly a two-way automaton, in the intuitive sense. Their intention was to define the latter in such a way that, for input strings of length $l$, the storage available for computation does not exceed $l + k_0$ cells for some fixed $k_0$. (One could as well insist on $k_0 = 0$, with the extra information remembered by a

\(^2\) More precisely, the binary representation of the input, including three end markers, uses at least $(3m - 2) + (3n - 2) + 3 = 3(m + n) - 1$ cells compared to $2(m + n + 3)$ cells for the ternary representation. The multiplication must begin with a product area of $m + 1$ octal digits and (conveniently) three extra two-cell markers besides the input. The same space serves throughout the multiplication, as increases in the product length are exactly compensated by discarding successive digits of the multiplier $\bar{y}$ as they are used. Thus the total space required is $2(m + n + 3) + 2(m + 1) + 6 = 4m + 2n + 14 < 3(m + n) + 14$, provided the larger number is used as multiplier, and the process is always possible with the addition of $k_0 = 15$ cells. (This number can be reduced, but many more internal states are then needed.)
finite increase in the number of internal states of the automaton.) The linear bounded automaton of Myhill [3] is in the same situation allowed up to $k_1$ storage cells for computation, for some fixed $k_1$. It can be visualized as having a tape of fixed length but with $k_1$ channels, of which only one is used for input. Equivalently, if its input alphabet has $b$ symbols it may use the same cells for computation but with an alphabet of $b^{k_1}$ symbols.

What has happened is that the availability of a 3-symbol alphabet with only binary coding of the input data makes the Kreider-Ritchie machine a linear bounded automaton with $k_1 = \log 3/\log 2 = 1.57$ instead of $k_1 = 1$. (The need for a marker symbol in fact reduces $k_1$ to exactly 1.5.) The same would of necessity be true of any automaton allowed the unrestricted use of marker symbols over and above those used for input coding. However if the input is coded in base $b > 2$ and only one marker is used, the effective memory expansion is less than

$$k_1 = \frac{\log(b + 1)}{\log b}$$

which is very near to unity.

Perhaps this suggests that there will be no sharp distinction between functions computable by a strict two-way automaton and those computable by a "weak" linear bounded automaton. If the contrary is true, definitions will have to be further sharpened before the distinction can be revealed. As regards multiplication, the question remains, in the form "Is multiplication possible on any linear bounded automaton with $k_1 < 1.5$?"

**APPENDIX: A TWO-WAY AUTOMATION FOR OCTAL-TERNARY ENCODING**

A machine for the encoding problem is here specified in terms of a state-input transition table (Table I). Each entry gives three items: a character (0, 1, or B) to be written, a motion (L or R) of the tape head, and a new state. For illustration, we describe the encoding of only one binary number, using a 22-state machine. (Two numbers can be encoded with two more states and a few extra transitions.) The initial configuration is taken to be

$$BxOB \cdots BBB$$

where $x$ is binary, its right-hand marker has already been changed from B to 0, four (or more) extra blank cells have been added to the input tape, and the machine is in state $S_0$ on the third blank cell from the right.

A configuration during the encoding might be

$$Bx_kOB \cdots B0x_kBB$$

where $x_k$ is binary, its right-hand marker has already been changed from B to 0, four (or more) extra blank cells have been added to the input tape, and the machine is in state $S_1$ on the third blank cell from the right.
where $\bar{x}_k$ is the coded part and $x_k$ the uncoded part of $x$. The states $S_1$ and $S_2$ scan to the left for the next digits of $x_k$ ($S_1$ recognizing and erasing the 0 marker which follows them). States $U_0, U_1, V_0, V_1, V_2, V_3$, and $W_0, W_1, ..., W_7$ record the value of the next three bits (erasing them and restoring a 0 marker in their place) and one of the $W$'s carries this information back to the right.

The unit ternary digit is placed by the $W$'s in the location of the 0 marker preceding $\bar{x}_k$, and states $T_0, T_1, T_2$, place the “three's” digit. (The ternary encoding used for the octal digits 0 — 7 is the natural one using $B$ as equivalent to 2. The combination $BB = 8$ is used for the marker $M$.)

Termination of the process may occur in two slightly different ways, depending on whether or not the number of bits in $x$ is an exact multiple of three. If so, the state $S_2$ will eventually encounter the initial blank cell, and the state $M_1$ will be entered directly. If instead one of the states $U_i$ or $V_j$ encounters the initial blank cell a temporary 1 marker is placed there and the final converted digit is stored. On the next cycle $S_1$ detects the 1 marker and enters state $M_1$. The states $M_1$ and $M_2$ place the initial $BB$ marker. Thus the final configuration is

$$0B ... BB\overline{BB}.$$  

Conversion of the product from ternary back to binary offers no greater difficulty. Once sufficient storage space has been made available, multiplication by the familiar shift and add procedure is straightforward.

### TABLE I

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$B$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$L$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$R$</td>
</tr>
<tr>
<td>$U_i$</td>
<td>$W$</td>
</tr>
<tr>
<td>$V_j$</td>
<td>$W$</td>
</tr>
<tr>
<td>$W_{2k}$</td>
<td>$R$</td>
</tr>
<tr>
<td>$W_{2k+1}$</td>
<td>$R$</td>
</tr>
<tr>
<td>$W_{2k+2}$</td>
<td>$R$</td>
</tr>
<tr>
<td>$M_1$</td>
<td>$M$</td>
</tr>
<tr>
<td>$T_0$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T_1$</td>
<td>$T$</td>
</tr>
<tr>
<td>$T_2$</td>
<td>$T$</td>
</tr>
<tr>
<td>$M_2$</td>
<td>$E$</td>
</tr>
<tr>
<td>$E$</td>
<td>exit</td>
</tr>
</tbody>
</table>
REFERENCES

