The Determination of the Value of Rado’s Noncomputable Function $\Sigma(k)$ for Four-State Turing Machines

By Allen H. Brady

Abstract. The well-defined but noncomputable functions $\Sigma(k)$ and $S(k)$ given by T. Rado as the “score” and “shift number” for the $k$-state Turing machine “Busy Beaver Game” were previously known only for $k \leq 3$. The largest known lower bounds yielding the relations $\Sigma(4) \geq 13$ and $S(4) \geq 107$, reported by this author, supported the conjecture that these lower bounds are the actual particular values of the functions for $k = 4$.

The four-state case has previously been reduced to solving the blank input tape halting problem of only 5,820 individual machines. In this final stage of the $k = 4$ case, one appears to move into a heuristic level of higher order where it is necessary to treat each machine as representing a distinct theorem. The remaining set consists of two primary classes in which a machine and its tape are viewed as the representation of a growing string of cellular automata. The proof techniques, embodied in programs, are entirely heuristic, while the inductive proofs, once established by the computer, are completely rigorous and become the key to the proof of the new and original mathematical results: $\Sigma(4) = 13$ and $S(4) = 107$.

“In any case, even though skilled mathematicians and experienced programmers attempted to evaluate $\Sigma(3)$ and $S(3)$, there is no evidence that any known approach will yield the answer, even if we avail ourselves of high-speed computers and elaborate programs. As regards $\Sigma(4)$, $S(4)$, the situation seems to be entirely hopeless at present.”—Tibor Rado, 1963.

1. Background and Introduction. The “Busy Beaver Game” was devised by Tibor Rado [8] for the purpose of illustrating the notion of noncomputability. Given the set of Turing machines of exactly $k$ states which operate with the minimum alphabet of two symbols (a space and a mark or 0 and 1) one considers the problem of behavior of these machines on a tape which is initially all blank (all 0’s). This is a finite set of machines, there being exactly $(4k + 1)^{2k}$ distinct machines, where, with two table entries per state, each table entry may consist of printing 0 or 1, moving right or left, branching to state 1, 2, . . . , $k$ or simply an entry to declare a halt.*

Since the input tape is blank, each machine faces one of two possible fates: either it eventually halts, or else it continues running forever. After a particular machine
halts, a certain number of 1's will be marked upon its tape. This will be the score of
that machine. The object of the "game" for a given \( k \) states is to find a machine
which will produce the largest possible score. This score is referred to as the "Busy
Beaver Number" for \( k \) states. It defines in the "strongest possible way" an integer
function \( \Sigma(k) \) which Rado demonstrated to be noncomputable.

Related to \( \Sigma(k) \) is another noncomputable function \( S(k) \) called the maximum
shift number: the greatest number of moves that a \( k \)-state machine with a blank input
tape can make before halting.

Rado and others became intrigued with the actual problems of determining as
mathematical fact the values of \( \Sigma(k) \) and \( S(k) \) for reasonably small \( k \). (See
Appendix A.) He also proposed this and related problems as important to artificial
intelligence [4].

Using computer programs, Lin [5] finally determined that \( \Sigma(3) = 6 \) and \( S(3) = 21 \).
(See also Lin and Rado [6].) Acceptance of that result in the ordinary manner of
seeing a mathematical proof presents difficulties. For this author it was a case of
making a completely independent and, as it turns out, a somewhat different attack
on the three-state problem [1].

The Four-State Problem. One formidable obstacle to solving the four-state case
appears to be the large number of four-state machines. From the expression
\[(4k + 1)^2k,\]
giving the number of all possible \( k \)-state machines, one may exclude the
set of \( k \)-state machines without any halts giving
\[
(4k + 1)^2k - (4k)^2k.
\]

Furthermore, one can also impose the requirement (cf. Lin and Rado [6]) that no
machine need be considered which does not have the initial entry (in state 1
scanning 0) to print 1, move right, and enter state 2 ("1 R 2"). This yields the
expression
\[
(4k + 1)^{2k-1} - (4k)^{2k-1}.
\]

While a reasonable number is now produced for \( k = 2 \) and 3 (217 and 122,458,
respectively), the situation still has a doubtful appearance for \( k = 4 \) (141,903,217)
and seems totally impossible for \( k = 5 \) (282,525,287,122).

The most significant reduction (see only after the fact) is obtained by extending
the argument for the quintuple \((1,0,1, R, 2)\) into a tree generation of all of the
quintuples for the machines to be considered (Brady [2]). With the tree generation
scheme (in current terminology a form of back tracking) coupled to a programmed
"solution" to the halting problem, little more than a minute of computer time was
required on a second generation computer, generating fewer than 4,000 three-state
machines and producing only 27 holdouts** for direct inspection. For \( k = 4 \),
approximately 550,000 four-state machines were generated and 5,820 holdouts
ultimately remained.

Among the four-state machines, two were discovered [2] which yield the score of
13. Their tables are as follows.

**A holdout [8] refers to a Turing machine for which the blank input tape halting problem remains
undecided after analysis by the computer.
In order to achieve the scores and shift numbers claimed, the halt entry must be replaced by a triple which *prints a mark*, moves, and branches to a zero state following Rado’s convention.)

The remaining 5,820 machines were all run for at least 500 moves with none observed to halt, and a 100 machine sample was inspected for a general picture of machine behavior. It was then *conjectured* [2] that $\Sigma(4) = 13$ and $S(4) = 107$.

The remainder of this paper describes the final reduction of the 5,820 holdouts to only 218 through the use of new heuristic computer programs especially devised for the problem. Direct inspection of each of the 218 holdouts from these programs reveals that none will ever halt, and it can be stated as known that $\Sigma(4) = 13$ and $S(4) = 107$.

2. Heuristic Classification of the Machines.

*Proofs by Induction.* Among the 27 “holdout” machines of three states the following two machines represent in a simple way their general classes of behavior:

\[
\begin{array}{c|cc}
0 & 1 & 0 L 3 \\
1 & 1 R 2 & 0 L 3 \\
2 & 1 L 2 & 1 R 1 \\
3 & 1 L 4 & 0 L 2 \\
\end{array}
\quad \quad 
\begin{array}{c|cc}
0 & 1 & 0 L 3 \\
1 & 1 R 2 & 0 L 3 \\
2 & 0 L 1 & 0 R 2 \\
3 & 1 L 1 & \_
\end{array}
\]

3-State Holdout No. 6 (Xmas Tree) 3-State Holdout No. 2 (Counter)

Machine no. 6 demonstrates an oscillatory pattern as it sweeps across the written tape (Figure 1), growing at each end as it continues the extension of its pattern. S. Lin (private conversation) refers to such machines as “Christmas Trees,” an appropriately graphic term which we will use throughout our discussion.

A proof by mathematical induction that the Xmas Tree pattern of machine no. 6 grows continually is not at all difficult, but the behavior is fairly obvious. Machine no. 2 demonstrates a pattern which is considerably less apparent. Its history of operation may be used to establish an inductive proof based upon the observations of its actual behavior shown in Figure 2 and the hypothesis that after $3 \cdot 2^{n-1} - 2n - 7$ moves the machine will arrive at the configuration

\[
\begin{array}{cccccccc}
0 & 0 & 1 & 1 & 1 & \cdots & 1 & 1 & 0 \\
3 & & & & & & & & \underline{2n-1 \text{ marks}}
\end{array}
\]

having started on a blank segment of tape of length $2n + 1$ squares.
After the inductive proof was devised it was observed [1] that machine no. 2 is in essence a binary counter. The observation was triggered by the similarity of the expression $3 \cdot 2^{n-1} - 2n - 7$ to the computed rate of growth of a simple binary counter which had been constructed as an exercise.

Studying the tape configurations in Figure 2 one can see that a “zero” is represented by the code pair (0 0), while a “one” is represented by (1 0) as the machine counts continuously, adding “one” on the right and accumulating to the left. During the carry sweep (to the left), a “one” goes through an intermediate code (11), finally being converted to “zero” on the return sweep to the right as seen in the 49th “move” (Figure 2):

$$\begin{align*}
(0 & 0) (0 1) (0 0) (0 1) (1 1) (0 0) \\
2
\end{align*}$$

It appeared that what was needed for handling this problem was some sort of “mechanical induction” capability. Techniques involving symbolic and algebraic manipulation were considered, but the algebraic analysis of machines did not seem to be a sufficient approach. Algebraic characteristics did, however, turn out to yield a fruitful method for first order identification.

License or copyright restrictions may apply to redistribution; see https://www.ams.org/journal-terms-of-use
**Figure 2. Behavior of failure number two (Counter)**

*Rates of Growth of the Scanned Tape.* For the simple “sweep across” or Xmas Tree pattern, the rate of growth is inversely proportional to the tape excursion, \( y \), plus a constant,

\[
dy/dx = 1/(ky + c_1),
\]

yielding the relation

\[
x = ay^2 + by + c.
\]

For a counter, experience has shown that a sequence of the form

\[
a2^n + bn + c
\]

identifies the characteristic extremum. In particular, for a simple binary counter (one bit code) it can be seen that the sequence identifying the start of counting on an exact power of two will have the relationship

\[
S_n = 2S_{n-1} + 2.
\]

Through integration of the differential equation

\[
dx/(x + 2) = dy,
\]

suggested by the difference equation

\[
S_n - S_{n-1} = S_{n-1} + 2,
\]
one obtains

\[ x = a^y - 2. \]

(The same form can be obtained using known difference equation methods.)

From our derived expressions it can be seen that the characteristic (or necessary feature) identification is readily discernable through a table of differences formed from the sequence of maximum excursions occurring on the left (at \( l_i \) moves) and on the right (\( r_i \) moves) during machine operation.

If the second differences are constant, the maximum excursions follow a parabolic sequence (1) which is indicative of the Xmas Tree. A table of differences formed from (2) would yield

\[ D^1 = (a^{2n+1} + b(n + 1) + c) - (a^{2n} + bn + c) = a^{2n} + b \]

and

\[ D^2 = (a^{2n+1} + b) - (a^{2n} + b) = a^{2n}. \]

Therefore, if the second differences yield a simple geometric sequence progressing by a power of two, a binary counter is indicated (but not demonstrated) so long as the extreme on the opposite side is constant.

A Heuristic Filter. Observations of a small sample of Xmas Trees and Counters will make apparent the fact that the table of differences is applicable only to an enveloping sequence, since machine “gyrations” on the growing edge of the traversed region of tape can be quite complex.

Various techniques for extracting the envelope were considered, including even numerical curve fitting. Ultimately, the following simple algorithm for extracting an enveloping ascending sequence was devised: Start with a sequence of maximum excursions (for instance, \( r_i \)) and form a table of first differences \((D_i = r_{i+1} - r_i)\).
Then trace forward through the table of first differences extracting a strictly increasing sequence while discarding all the rest. The extracted sequence can then be used to form the table of first and second differences which we want to inspect. A flowchart for the algorithm is shown in Figure 3.

The filtering scheme and difference computations were embodied in a program along with some other parameter dependent heuristics and applied to the 5,820 4-state holdouts. This computer program separated the holdouts into three tentative or probable classes: XMAS (4,849); COUNTER (402); and UNKNOWN (569).

3. The Cellular Representation of Xmas Trees.

The Basic Picture. Initial attempts to handle the Xmas Trees using a tediously constructed program were unsatisfactory, so after the failure of this first program to make a significant reduction, a careful study was made of the resistant holdouts of the Xmas Tree sample. From the thought that had already gone into the Counter problem, a cellular picture had evolved in “hardware” terms. It then became apparent that (1) a cellular representation for Xmas Trees also had merit, and (2) the inductive pattern should be sought relative to the growing chain of perceived cells and not fixed absolutely to the tape of the Turing machines.

A Xmas Tree can be represented by a chain of tape segments (cells) which go through metamorphic changes in the process of reproducing themselves. At some point in the metamorphic and reproductive cycle, say, when the machine has reached a new right extremum in its tape excursion, we observe that the tape is made up of a left segment, a right segment, and a sequence of identical interior segments:

\[ / \quad \text{L} \quad / \quad / \quad \text{X} \quad / \quad / \quad \text{X} \quad / \quad / \quad \text{X} \quad / \quad / \quad \text{R} \quad / \\]

\[ \text{scan position of TM} \]

(initial configuration)

As we will learn later, it is not so important that we identify the L and R cells as it is for us to identify the X cell. (The reader may find it helpful to observe the behavior of the four-state example no. 1 in Appendix B.)

If we operate the machine until it reaches the next major right extremum (determined by the filtering process) we will see that a new X cell has been produced:

\[ / \quad \text{L} \quad / \quad / \quad \text{X} \quad / \quad / \quad \text{X} \quad / \quad / \quad \text{X} \quad / \quad / \quad \text{X} \quad / \quad / \quad \text{R} \quad / \\]

\[ \text{scan} \]

(final configuration)

What happens in the general case is that the L and R segments act as parent cells and generate partial cells Z₁ and Z₂ which, when juxtaposed, form an X cell.

Starting with the initial configuration, as the machine proceeds to process the right segment, it eventually emerges, sweeping from right to left converting the X cells to Y cells, their metamorphic successors.

\[ / \quad \text{L} \quad / \quad / \quad \text{Y} \quad / \quad / \quad \text{Y} \quad / \quad / \quad \text{Y} \quad / \quad / \quad \text{R} \quad / \\]

\[ \text{scan} \]

The machine then enters the left segment, takes it through its reproductive transformation and emerges from where it entered, sweeping from left to right converting
the Y cells to Z cells. The machine enters the right segment and eventually reaches a
new extreme at which point we observe the chain

\[
\begin{array}{c}
\text{L}_1 \rightarrow \text{Z}_1 \rightarrow \text{Z}_2 \rightarrow \text{Z}_3 \rightarrow \text{Z}_4 \rightarrow \text{Z}_5 \rightarrow \text{R} \\
(\text{alternate view of final configuration})
\end{array}
\]

The Z cells, which were the metamorphic successors to the Y cells, are in fact
complementary images of the original X cells, so that the cell wall boundaries simply
shift if (as is usually the case)

\[
\begin{array}{c}
\text{Z}_1 \rightarrow \text{Z}_2 = \text{X}_1 \rightarrow \text{X}_2 \\
\end{array}
\]

It is apparent by inductive extension (since the X and Z cells are the same size)
that a chain of n Z cells,

\[
\begin{array}{c}
\text{Z}_1 \rightarrow \text{Z}_2 \rightarrow \ldots \rightarrow \text{Z}_n \rightarrow \text{Z}_{n+1} \\
\end{array}
\]

bounded by a Z₁ cell on the left and a Z₂ cell on the right is identical to a chain of
n + 1 X cells:

\[
\begin{array}{c}
\text{X}_1 \rightarrow \text{X}_2 \rightarrow \ldots \rightarrow \text{X}_{n+1} \\
\end{array}
\]

The actual situation, however, is made much more complicated by the fact that
the metamorphic transformation of a cell is intertwined with the transformation of
its neighboring cells. As will be seen, this interdependence is (fortunately) not
inextricable.

Furthermore, the transformations of the L and R segments are frequently not
simply described, but problems with their behavior are easily circumvented by
incorporating several X cells into these end segments to act as buffers.

The Detailed Metamorphic Cycle. Assume, then, that at least one X cell is in the
right end of the left segment, L₀, and at least one X cell is in the left end of the right
segment, R₀:

\[
\begin{array}{c}
\text{Left (L₀)} \rightarrow \text{Right (R₀)} \\
\text{X}_1 \rightarrow \text{X}_2 \rightarrow \ldots \rightarrow \text{X}_n \rightarrow \text{X}_{n+1} \\
\end{array}
\]

Let the controlling Turing machine be in state q₀ at the extreme right. Operate the
machine until it emerges from the right segment to the left. Call the emerging state
qₐ:

\[
\begin{array}{c}
(qₐ) \rightarrow \text{X}' \\
\end{array}
\]
The $X$ cell on the left end will now be in an intermediate metamorphic stage, $X'$. We must next demonstrate that the machine will operate on a two cell segment

$$\begin{array}{c}
\underline{X} \\
\underline{X'} \\
\underline{q_a}
\end{array}$$

resulting in the transformation to the segment

$$\begin{array}{c}
\underline{X'} \\
\underline{Y} \\
\underline{q_a}
\end{array}$$

with the machine emerging on the left in the same state $q_a$.

If this has occurred as described, then obviously by extension we can see that any chain of $X$ cells entered on the right in the same state $q_a$ with an $X'$ cell to the right of the chain

$$\begin{array}{c}
\underline{X} - \underline{X} - \ldots - \underline{X} - \underline{X'} \\
\underline{q_a}
\end{array}$$

must undergo transformation into a string of $Y$ cells with an $X'$ cell on the left and the controlling machine emerging in state $q_a$:

$$\begin{array}{c}
\underline{X'} - \underline{Y} - \ldots - \underline{Y} - \underline{Y} \\
\underline{q_a}
\end{array}$$

In other words, a chain of $X$ cells of any length is transparent to the passage of the controlling machine as it travels from the right segment to the left segment.

As our next step in the inductive proof, we operate on a partially changed left segment (its $X$ cell converted to an $X'$ cell) to the left of the $X'$ cell in state $q_a$:

$$\begin{array}{c}
\underline{X'} \\
\underline{q_a}
\end{array}$$

We run the machine until it emerges to the right. Call the emerging state $q_b$. The cell on the right end will now be in an intermediate metamorphic stage $Y'$:

$$\begin{array}{c}
\underline{Y'} \\
\underline{q_b}
\end{array}$$

We must next show that a chain of $Y$ cells is transparent to passage of the machine to the right. Forming a segment consisting of a $Y'$ and a $Y$ cell, we demonstrate that

$$\begin{array}{c}
\underline{Y'} - \underline{Y} \\
\underline{q_b}
\end{array}$$
becomes
\[
\frac{Z}{q_b} - \frac{Y'}{q_b}
\]

by simulating the controlling machine under the implied restrictions.

As the final simulation step in the establishment of our proposition, we operate on a partially changed right segment (the right segment as we left it but with its Y cell converted to a Y' cell) positioned to the right of its Y' cell in state \(q_b\):
\[
\text{(right segment)}
\]

\[
\frac{Y'}{q_b}
\]

We run this configuration until the machine reaches the new extremum on the right in state \(q_0\):
\[
\text{(right segment, } R_0')
\]

\[
\frac{Z}{q_0}
\]

The metamorphosed left segment with its Y' cell converted to a Z cell (which we have proved will happen) we will call \(L'_0\). The right segment above we will call \(R'_0\). We determine that under these transformations "birth" has taken place. I.e.,
\[
\frac{L'}{0} = \frac{L}{0} - \frac{Z}{1}
\]

and
\[
\frac{R'}{0} = \frac{Z}{2} - \frac{R}{0}
\]

—the \(L'_0\) segment is identical to the original \(L_0\) with a partial cell \(Z_1\) on the right, while the \(R'_0\) segment is identical to the original \(R_0\) with a partial cell \(Z_2\) on the left.

As already mentioned, we must finally show that
\[
\frac{Z}{1} - \frac{Z}{2} = \frac{X}{1} - \frac{X}{2}
\]

The original premise is the existence of the initial configuration. Since this was already established by the feature extraction process, the inductive proof for the particular machine is then complete.

It should be pointed out that should any particular step fail there is no proof. There are also heuristically established limits placed upon any simulation to prevent the possible occurrence of a loop. Wherever we have in effect said "until anticipated condition arises" it should be assumed that we mean "until anticipated condition arises or limiting condition occurs." The occurrence of a limiting condition implies failure of the proof procedure for a given Turing machine.
Heuristic Extraction of Xmas Tree Features. Extraction of the features of a Xmas Tree is a simple process which was arrived at after several experiments.

The machine is run for a thousand moves or so, and the enveloping sequences of extreme (max. and min.) excursions are filtered out. The last extreme point (on the right) occurs at what we shall refer to as $i_2$ moves (shifts). The preceding filtered extreme occurs at $i_1$ moves.

We rerun the machine from scratch to the point $i_1$. We then “snip” the tape (Figure 4) in the middle of its range of excursion creating and saving the segments $L_0$ and $R_0$. We then run the machine from $i_1$ to $i_2$. We match $L_0$ from the left on the new tape configuration at $i_2$ and $R_0$ from the right. (If no match occurs, the test fails at this point.) The residual segment in the center of the tape after removal of $L_0$ and $R_0$ is taken to be the $X$ cell.

![Figure 4. Extraction of Xmas Tree cells](image)

It is important to note that while the success of the extraction process is necessary to the proof, it is not sufficient. The precise demonstration of the independence of the $L_0$ and $R_0$ transformation must be made along with the demonstration of the transparency of the chain of $X$ cells to passage of the machine back and forth through them. These demonstrations, as already described, are then made step by step from move $i_1$ to move $i_2$.

Xmas Tree Variants. Two variants of Xmas Trees occur which were easily accommodated by minor modifications in the extraction heuristics and the mechanical proofs. These we will refer to as Leaning Trees and Trees with a Shadow. (See Appendix B.) In both of these variants, the segment $R_0$ gives birth to an extra $X$ cell while the $L_0$ segment consumes a portion of an $X$ cell. The Tree with a Shadow in addition remains attached to a trail of skeletons left behind.

The other variant is the Xmas Tree with a double sweep, which we will refer to as the Alternating Xmas Tree. This variant was handled in a manner similar to that used for the standard Xmas Tree. The Alternating Xmas Tree becomes a generalization of the Xmas Tree, and the mechanical proof for Alternating Xmas Trees is applicable to Xmas Trees.

The minor variants analogous to the Leaning Trees and the Trees with Shadow show up in the Alternating Xmas Trees. Minor changes were made in the programs to accommodate the (alternating) Leaning Trees. Only eight “Alternating Trees with Shadow” appeared in the 218 final holdouts.
From the set of 4,829 holdouts tentatively separated into XMAS, 4,841 were proved never to halt, while 8 holdouts remained. From the set of 569 holdouts tentatively separated as UNKNOWN, 389 were proved never to halt, while 180 holdouts remained.


The Hardware Analog. For a Counter one can picture a semi-infinite chain of registers, initially all in the ZERO state,

\[
\ldots \rightarrow \text{E} \rightarrow \text{0} \rightarrow \text{0} \rightarrow \text{0} \rightarrow \ldots
\]

driven from an END cell or register labeled \(E\). Carry and return signals are propagated through this chain. When a carry signal enters a cell or register in the ZERO state, the cell goes into the ONE state and sends back a return signal:

\[
c \rightarrow \text{0} \Rightarrow r \leftarrow \text{1}
\]

When a carry enters a cell in the ONE state, the cell goes into the transition or \(1'\) state and sends the carry on:

\[
c \rightarrow \text{1} \Rightarrow \text{1'} \rightarrow c
\]

When a return signal enters a cell (in the \(1'\) state), the cell goes into the ZERO state and the return signal is continued back:

\[
\text{1'} \leftarrow r \Rightarrow r \leftarrow \text{0}
\]

Finally, when the return signal is detected by the END cell, the carry is regenerated and sent into the register chain:

\[
\text{E} \leftarrow r \Rightarrow \text{E} \rightarrow c
\]

Another possible cell state (which does arise) is the blank or unwritten state. In such a cell, the code for zero is not all Turing machine 0's. In this case, a blank (B) cell is first converted to a ONE:

\[
c \rightarrow \text{B} \Rightarrow r \leftarrow \text{1}
\]

For input signals of interest we may now look at the complete state diagram of a cell with state B, 0, 1', and 1 shown in Figure 5(a).

Heuristic Extraction of Counter Components. The initial difficulty in analyzing a counter rests completely in the recognition of its components. Cell size is taken to be the increase in tape excursion from one filtered extreme to the next. At an extreme
point we have a blank cell, $B$, being converted into a ONE cell:

$$(\text{i}_1 \text{ moves}) --$$

$$\overline{E} - \overline{1} - \overline{1} - \ldots - \overline{1} - \overline{B} - \overline{B} - \ldots$$

$q_c$

At the first relative minimum point (prior to reaching $i_2$, the next extreme of carry propagation) the end cell, $E$, is in the process of regenerating (or reflecting) a carry signal:

$$\overline{E} - \overline{0} - \overline{0} - \ldots - \overline{0} - \overline{1} - \overline{B} - \ldots$$

$q_m$

From the tape configuration at this point we attempt to extract the code for a ONE and for a ZERO, based upon cell-size alignment and the unfiltered excursion extreme occurring between $i_1$ moves and this point of operation. From alignment of the configuration at $i_1$, we extract the code for $1'$, and tentatively for $E$. We augment the tentative end cell with two $1'$ cells for a margin of safety. (Once a counter is in operation, it is evident that the addition of a fixed number of counter cells to the END cell does not change its defined operation as depicted in the state diagram of Figure 5(b).)

With the alignment of cells now determined, we snip the chain of the $i_1$ configuration at a cell boundary:

$$(\text{i}_1 \text{ moves}) --$$

$$\overline{E} - \overline{1} - \ldots - \overline{1} - \overline{1} - \overline{1} - \overline{B} - \overline{B} - \ldots$$

(snip)

$q_c$

and run the partial configuration on the right until the controlling Turing machine emerges at the cut:

$$\overline{1} - \overline{0} - \ldots - \overline{0} - \overline{1}$$

$q_r$

The emerging state at the cut, $q_r$, is assumed to be the return signal.
As the first step in our inductive proof, we now establish the premise that a counting state exists, i.e. the right segment is indeed a chain of ZEROS with a ONE on the right:

\[
\begin{array}{cccccccc}
| & \underline{0} & - & \underline{0} & - & \ldots & - & \underline{0} & - \underline{1} \\
| & \underline{0} & - & \underline{0} & - & \ldots & - & \underline{0} & - \underline{1} \\
\end{array}
\]

and the left segment is a chain of transitional cells (1') with the END cell on the left:

\[
\begin{array}{cccccccc}
| & \underline{E} & - & \underline{1'} & - & \underline{1'} & - & \ldots & - & \underline{1'} & - \underline{1'} \\
| & \underline{E} & - & \underline{1'} & - & \underline{1'} & - & \ldots & - & \underline{1'} & - \underline{1'} \\
\end{array}
\]

**Final Steps in the Proof.** We are finally left with the straightforward task of demonstrating the state conversions of our heuristically extracted cells.

First we run the Turing machine on the restricted configuration representing a 1' cell, starting on the right end in state \( q_r \) to demonstrate that the Turing machine emerges from the left end in state \( q_r \), and that the tape configuration is identical with the extracted code for ZERO:

\[
\begin{array}{ccc}
\underline{1'} & \Rightarrow & \underline{0} \\
\hat{q}_r & \hat{q}_r \\
\end{array}
\]

Next we demonstrate that

\[
\begin{array}{ccc}
\underline{E} & \Rightarrow & \underline{E} \\
\hat{q}_r & \hat{q}_c \\
\end{array}
\]

(This determines \( q_c \), the carry signal, but we must show that \( E \) is unchanged.)

Then we demonstrate that the carry will be propagated by ONE cells:

\[
\begin{array}{ccc}
\underline{1} & \Rightarrow & \underline{1'} \\
\hat{q}_c & \hat{q}_c \\
\end{array}
\]

Finally, we demonstrate that ZERO reflects the carry as it changes to a ONE:

\[
\begin{array}{ccc}
\underline{0} & \Rightarrow & \underline{1} \\
\hat{q}_c & \hat{q}_r \\
\end{array}
\]

We must also determine whether or not

\[
\underline{0} = \underline{B},
\]

and if the ZERO code differs from the BLANK code, we must also demonstrate that

\[
\begin{array}{ccc}
\underline{B} & \Rightarrow & \underline{1} \\
\hat{q}_c & \hat{q}_r \\
\end{array}
\]
The Counter proving program was run on the set of 402 machines which had been classified as counters. All but 73 machines were demonstrated never to halt.

A Modification to the Program. At this point it was no surprise to observe an interdependence of counter cells, i.e. the conversion from one cell state to the next can be dependent upon the preceding cell. Such behavior is exactly what occurs with what one could term “two-shot” carry and return signals.

Another difficulty which plagues counters is their rather slow rate of growth. It was not always possible to extract the features and establish existence of a counting state in only 1,000 moves!

The new cell conversion tests were incorporated into the program. The modified counter program was run with a 2,000 move limit on the 73 Counter holdouts. It found 26 more counters leaving 47 holdouts. But, additionally, the modified counter program was run on the miscellaneous holdouts from the Xmas Tree runs. It found 17 Counters. The Counter proofs were therefore effective in eliminating 372 machines. A total of 218 holdouts remained from all programs.

Other Counter Behavior. Several other counter variants were observed among the holdouts, but it was decided that further programming to devise the necessary cell extraction heuristics was not worth the trouble.

5. Final Steps in the Solution of the Four-State Problem.

The Computer Reduction Runs. A total of eight programs were used directly in the process of reducing the final set of four-state holdouts. (See Figure 6.) BBFILT was used to separate heuristically the 5,820 holdouts into “Xmas Trees”, “Counters,” and “Unknown,” while BBFXX, a modification of BBFILT separated the “Alternating Xmas Trees” from the “Unknown” set.

```
5.820 Original Holdouts
   ▼
   | 569 - [UNKNOWN?] |
   ▼
   | 173 - [No Effect] |
   ▼
   | 156 Holdouts |
   ▼
   | 173 - [BBALTX] |
   ▼
   | 373 Holdouts |
   ▼
   | 196 |
   ▼
   | 73 |
   ▼
   | 102 - [COUNTER?] |
   ▼
   | 404 |
   ▼
   | 17 |
   ▼
   | 8 Holdouts |
   ▼
   | 47 Holdouts |
   ▼
   | 73 |
   ▼
   | 472 |
   ▼
   | 569 |
   ▼
   | 849 |
   ▼
   | 4849 |
   ▼

Figure 6. The computer reduction runs
```
BBX2 was the Xmas Tree prover which used the cellular automata approach. BBSSHAD was a modification to handle “Trees with Shadow”, BBALTX was an extension of BBX2 to handle “Alternating Xmas Trees”, while BBALTX1 was a minor modification of BBALTX to handle double sweeps in which the extremum was reached on alternate sweeps only.

BBC was the counter prover, while BBCM was a modification of BBC to handle “two-shot” carries and some cases of cell interdependence.

More than 18 other programs were written for various housekeeping purposes, simulating and displaying machine behavior, exploring other reduction and filtering possibilities, etc. In all, at least 53 files were created and maintained for the project. Keeping track of what resembled a large scientific experiment became a major task in itself.

While not all of the exploratory activities are reproducible, the runs shown in Figure 6 can be reproduced, so that by utilizing the techniques described in this paper the proof can be corroborated.

All of the programs utilized all or part of the filtering techniques of the program BBFILT. However, it must be remembered that the filtering was a heuristic technique based upon experimental observation. The reasons why it was not always effective were not explored in every case, since the ultimate proofs were mechanized independently of the heuristic separation. Furthermore, the heuristic separation was dependent upon such parameters as the length of behavior histories (number of moves or shifts). E.g., a 1,000 move limit picked up fewer Counters than a 2,000 move limit due to anomalous behavior in the early portion of the maximum excursion sequences.

The 218 Remaining Holdouts. The general behavior of some of the 218 holdout machines has already been described. Three of the machines escaped detection by programs which should have proved they would never halt. One of these machines was in fact a simple “left traveling loop,” but it required 422 moves to display a repetition in its sequence. Several machines showed up which are forms of ternary (base 3) and quaternary (base 4) counters.

A new generic type which may be described as “Tail-Eating Dragons” dominates the “Other” set. This type displays either a rapid growth approximating Xmas Trees or else an extremely slow growth approximately quaternary counters. Tail-Eating Dragons are almost the converse of “Trees with Shadow”. Starting with a long string of characters (their “tail”) these machines sweep back and forth like a Xmas Tree while nibbling a bit off their tail on each swing. Once the tail is consumed, they create a much larger tail and begin the process anew. The “rapid growth” machines add to the size of their tape on each swing, while the “slow growth” machines add to their scanned tape only when they create a new tail. Their behavior is precariously similar to that of the “champion” lower bound machines devised by Green [3].

All of the remaining holdouts were examined by means of voluminous printouts of their histories along with some program extracted features. It was determined to the author’s satisfaction that none of these machines will ever stop.

6. Conclusion. It was mentioned in the introduction that these results should be independently verified. Proofs of “correctness” of the programs used are not practical. Independent verification is the only means we currently have at our disposal.
This attack on the $k = 4$ problem has produced the bonus of creating the components of an "intelligent machine" which will accept any three-state binary Turing machine and render a decision on its blank input tape halting problem without any resort to an enumeration of specific machines supplied by its human creators. I.e., all of the 27 holdouts for $k = 3$ were recognized by the new programs as either Xmas Trees or Counters and were mechanically proved never to halt. An effort to solve the problem for $k = 5$ might similarly produce enhancements verifying the four-state problem.

By fitting an expression of the form $(ak + b)^c$ to the number of tree normal machines for $k = 2, 3, 4$ one obtains values for $a$, $b$, and $c$ of 1.72, -0.513, and 1.80, respectively, yielding a projection of 150,000,000 for $k = 5$. Using nothing more than a small computer one could probably generate this entire set and reduce the five-state case to perhaps a few million holdouts. These five-state holdouts should present an interesting and reasonable challenge to persons interested in mechanical proof techniques.

The final steps in the solution to this problem for $k = 4$ have resulted in a successful effort in applying a form of mechanized mathematical induction. (The computer programs could have been designed to print out individual proofs for each machine expressed in English and mathematical prose.) There are no doubt other classes of new and interesting automata problems which could be attacked successfully by similar methods. The pursuit of such techniques is important in bringing about the day when mathematical research may, as a matter of course, produce results using automated proofs with these results being communicated, understood, and accepted.

**APPENDIX A. Current Known Results.** The best known results for Rado's problem insofar as the author is able to determine are the following.

<table>
<thead>
<tr>
<th>$k$</th>
<th>$\Sigma(k)$</th>
<th>$S(k)$</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>= 1</td>
<td>= 1</td>
<td>T. Rado et al.</td>
</tr>
<tr>
<td>2</td>
<td>= 4</td>
<td>= 6</td>
<td>T. Rado et al.</td>
</tr>
<tr>
<td>3</td>
<td>= 6</td>
<td>= 21</td>
<td>S. Lin</td>
</tr>
<tr>
<td>4</td>
<td>= 13</td>
<td>= 107</td>
<td>A. Brady</td>
</tr>
<tr>
<td>5</td>
<td>$\geq 112$</td>
<td>$\geq 7,707$</td>
<td>D. Lynn</td>
</tr>
<tr>
<td>6</td>
<td>$\geq 117$</td>
<td>$\geq 13,488$</td>
<td>D. Lynn</td>
</tr>
<tr>
<td>7</td>
<td>$\geq 22,961$</td>
<td></td>
<td>M. Green</td>
</tr>
<tr>
<td>8</td>
<td>$\geq 3 \cdot (7 \cdot 3^9 - 1)/2$</td>
<td></td>
<td>M. Green</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n$</td>
<td>(cf. [3])</td>
<td></td>
<td>M. Green</td>
</tr>
</tbody>
</table>

The results for $k = 5$ and $k = 6$ have not been published. The results for $k = 5$ have been communicated to the author by Donald S. Lynn and are an extension of work beyond that reported in [7]. The results for $k = 6$ have been generated by the author using the 5-state machines discovered by Lynn.

**APPENDIX B. Machine Examples.** Since the generic machine types discussed are relatively rare among all four-state machines, some examples are given here which may be used to illustrate the discussion in the paper.
Among the proved set:

1. Xmas Tree-
   
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1R2 0L3</td>
<td>1R2 0L3</td>
</tr>
<tr>
<td>21</td>
<td>1L1 0R4</td>
<td>0L1 1R4</td>
</tr>
<tr>
<td>32</td>
<td>0L2 0L2</td>
<td>1L1 1L2</td>
</tr>
<tr>
<td>44</td>
<td>1R1</td>
<td>0R1</td>
</tr>
</tbody>
</table>

2. Leaning Tree-
   
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1R2 0L3</td>
<td>1R2 0L3</td>
</tr>
<tr>
<td>22</td>
<td>1L1 0R3</td>
<td>0L1 0R2</td>
</tr>
<tr>
<td>33</td>
<td>1L4 0R1</td>
<td>1R1 1L4</td>
</tr>
<tr>
<td>44</td>
<td>1L1</td>
<td>0R4</td>
</tr>
</tbody>
</table>

3. Shadow Tree-
   
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1R2 0L3</td>
<td>1R2 1L1</td>
</tr>
<tr>
<td>22</td>
<td>0L1 1R4</td>
<td>0L1 0R3</td>
</tr>
<tr>
<td>33</td>
<td>1L1 1L2</td>
<td>1R4 1R3</td>
</tr>
<tr>
<td>44</td>
<td>1L1</td>
<td>1L1</td>
</tr>
</tbody>
</table>

4. Alternating Tree-
   
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1R2 0L3</td>
<td>1R2 1L3</td>
</tr>
<tr>
<td>22</td>
<td>1L1 0R3</td>
<td>0L1 0R2</td>
</tr>
<tr>
<td>33</td>
<td>1L4 0R1</td>
<td>1R1 1L4</td>
</tr>
<tr>
<td>44</td>
<td>1L1</td>
<td>0R2</td>
</tr>
</tbody>
</table>

5. Counter (3-square cell)-
   
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1R2 0L3</td>
<td>1R2 1L3</td>
</tr>
<tr>
<td>22</td>
<td>1L1 0R3</td>
<td>0L1 1R4</td>
</tr>
<tr>
<td>33</td>
<td>1L4 0R1</td>
<td>1R1 0L4</td>
</tr>
<tr>
<td>44</td>
<td>1L1</td>
<td>1L1</td>
</tr>
</tbody>
</table>

6. Counter (Nonblank zero)-
   
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1R2 0L3</td>
<td>1R2 1L3</td>
</tr>
<tr>
<td>22</td>
<td>1L1 0R3</td>
<td>0L1 1R4</td>
</tr>
<tr>
<td>33</td>
<td>1L4 0R1</td>
<td>1R1 1L1</td>
</tr>
<tr>
<td>44</td>
<td>1L1 0R4</td>
<td>0L1 1R3</td>
</tr>
</tbody>
</table>

Among the holdouts:

7. Ternary Counter-
   
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1R2 0L3</td>
<td>1R2 0R4</td>
</tr>
<tr>
<td>22</td>
<td>1L1 1R1</td>
<td>1R3</td>
</tr>
<tr>
<td>32</td>
<td>0R1 0L4</td>
<td>0L1 1R1</td>
</tr>
<tr>
<td>44</td>
<td>1L3</td>
<td>1L3</td>
</tr>
</tbody>
</table>

8. Tailing-eating Dragon (fast growth)-
   
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1R2 0L3</td>
<td>1R2 1L1</td>
</tr>
<tr>
<td>22</td>
<td>1L1 1R1</td>
<td>1R3 0R4</td>
</tr>
<tr>
<td>32</td>
<td>0R1 0L4</td>
<td>0L1 1R1</td>
</tr>
<tr>
<td>44</td>
<td>1L3 1R4</td>
<td>0L1 1R4</td>
</tr>
</tbody>
</table>

9. Tailing-eating Dragon (slow growth)-
   
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1R2 0L3</td>
<td>1R2 1L1</td>
</tr>
<tr>
<td>22</td>
<td>1L1 1R1</td>
<td>1R3 0R4</td>
</tr>
<tr>
<td>32</td>
<td>0R1 0L4</td>
<td>0L1 1R1</td>
</tr>
<tr>
<td>44</td>
<td>1L3 1R4</td>
<td>0L1 1R4</td>
</tr>
</tbody>
</table>

Examination of the behavior of these machines without the aid of a computer is somewhat tedious. A simple Turing machine simulator written in a machine independent form of BASIC is available from the author upon request.

Acknowledgements. The author wishes to acknowledge the support of the University of Nevada System for that portion of this work carried out during a sabbatical leave. He also desires to thank Professor Harry E. Goheen of Oregon State University for his continuing interest and encouragement.

Department of Mathematics
University of Nevada
Reno, Nevada 89557