Now we combine the fractions in the expression for $f(x)$ to get

(6) $$f(x) = P(x)/[x(x + 1) \ldots (x + 2k)]$$

and observe that these negative roots are also zeros of $P(x)$, since the factors in the denominator of (6) cannot be zero at these values of x. But the degree of $P(x)$ is $2k$. Therefore, $P(x)$ possesses one more zero, and this is then the r obtained in Section 2. Q.E.D.

Remark: The branch of the curve, skipped in the above argument, then does not cut the x-axis at all.

4. THE PSI FUNCTION

The psi function, denoted by $\Psi(x)$, is defined by some authors [2, p. 241] by means of

(7) $$\Delta^{-1}\left(\frac{1}{x}\right) = \Psi(x) + C,$$

where C is an arbitrary periodic function. This is the analog for defining $\ln(x)$ in the elementary calculus by means of

$$\int \frac{1}{x}\,dx = \ln(x) + c.$$

We employ (7) to obtain

$$f(x) = 2\Psi(x + k) - \Psi(x) - \Psi(x + 2k + 1).$$

This provides us with an iteration method for the calculation of r, starting with $r_1 = k^2$.

REFERENCES

1. T. J. Bromwich. *An Introduction to the Theory of Infinite Series.* London: Macmillan, 1947. Pp. 522 ff.
2. L. M. Milne-Thomson. *The Calculus of Finite Differences.* London: Macmillan, 1951.
3. I. J. Schwatt. *An Introduction to the Operations with Series.* Philadelphia: University of Pennsylvania Press, 1924. Pp. 165 ff.
4. E. T. Whittaker & G. N. Watson. *A Course of Modern Analysis.* New York: Macmillan, 1947. Pp. 246 ff.

*****

RECOGNITION ALGORITHMS FOR FIBONACCI NUMBERS

DENNIS C. SMOLARSKI

University of Illinois, Urbana, IL 61801

LEONARD F. KLOSINSKI

University of Santa Clara, Santa Clara, CA 95053

A FORTRAN, BASIC, or ALGOL program to generate Fibonacci numbers is not unfamiliar to many mathematicians. A Turing machine or a Markov algorithm to recognize Fibonacci numbers is, however, considerably more abstruse.

A Turing machine, an abstract mathematical system which can simulate many of the operations of computers, is named after A. M. Turing who first described such a machine in [2]. It consists of three main parts: (1) a finite set of states or modes; (2) a tape of infinite length with tape reader; (3) a set of instructions or rules. The tape reader can read only one character at a time, and, given the machine state and tape symbol, each instruction gives us information consisting of three parts; (1) the character to be written on the tape, (2) the direction in which the tape reader is to move; (3) the new state the machine is to be in.

A Turing machine can be described by either a diagram or a table. An example of a Turing machine that adds two numbers is shown in Fig. 1. The figure shows both the table form and the diagram form of this Turing machine.

Character \ State	*A*	*B*
1	1 *R A*	¢ Halt
b	1 *R A*	
¢	¢ *L B*	

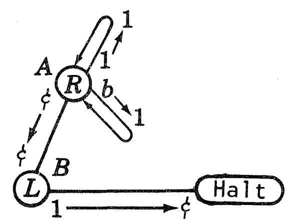

Fig. 1

Let us now consider the tape shown in (1). This tape shows a

(1) 1 1 *b* 1 1 1 ¢

two represented by two ones, a blank space represented by *b*, a three shown by three ones, and the ¢ which will mean the end of the information. The Turing machine shown in Fig. 1, when started in State A at the leftmost character of the tape in (1) will produce the following tape which shows a five, the sum of two and three.

(2) 1 1 1 1 1 ¢ ¢

The above table is read in the following way. The first row represents the states that the machine can be in, and the first column shows the characters that the machine can read. Let us, for example, look at the entry under State *A* and Character *b*. That entry, 1 *R A*, like every entry, save one, consists of three parts. The first part of the entry, 1, means change the character that is being read, *b* in this case, to a 1; the *R*, the second part of of the entry, means move one space to the right on the tape; and the *A*, the third part of the entry, says that the machine is to be in State *A* before reading the next character. Thus, if the machine is in State *A* and sees ¢, the table says that it changes the ¢ to ¢, moves left one place, and goes into State *B*.

The above diagram, which is equivalent to the table, can be most easily explained by considering only a portion of it. The states of the

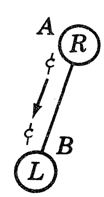

Fig. 2

machine are shown on the outside of the circles; the direction of the move is shown inside the next circle; and the character change is shown along the line connecting the circles. Thus, Fig. 2 says that a machine in State A and seeing ¢, changes ¢ to ¢, moves left one place, and goes into State B.

The Turing machine diagram which appears in Fig. 3 exhibits a machine which will halt only when presented with a string of consecutive ones, whose length is a Fibonacci number. If the total number of consecutive ones is not a Fibonacci number, the machine will loop endlessly. A basic assumption is that the string of ones is bounded on each side by at least one zero.

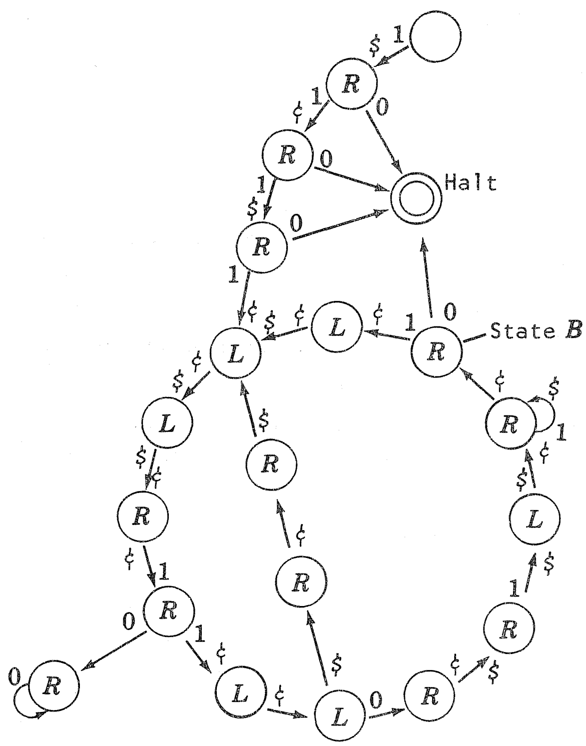

Fig. 3

The machine depicted examines the string of ones, starting at the left end, and repeatedly builds larger and larger Fibonacci numbers within this string. It keeps track of its place, and of previously constructed Fibonacci numbers, by slowly changing the ones to a series of dollar signs and cent signs as it moves through the string of ones. Each time the machine reaches the states labeled B in Fig. 3, the segment of the tape which has been examined has been changed to a string of dollar signs with the exception of a cent sign in the F_n place (which is the place immediately to the left of the tape digit being read while in State B), and a second cent sign in the F_{n-1} place.

After the machine finishes constructing a Fibonacci number within the string of ones, that is, each time it reaches State B, it checks to see if the next digit on the tape is zero or not. If so, the number of ones in the original string is a Fibonacci number and the machine halts. If, however, the next digit is a one, the machine attempts to build the next larger Fibonacci number within the string of ones (and, at this point, dollar and cent signs). If it encounters a zero on the tape before completing the construction of this next Fibonacci number, the machine goes into an endless loop. Thus, it halts only when the original number of consecutive ones is a Fibonacci number.

As an example, suppose the initial input string was ...001111111100... . By the time the Turing machine reaches State B, the string would be changed to $\ldots 00\$\$\text{¢}\$\text{¢}\underset{\wedge}{1}1100\ldots$ with the first cent sign replacing the third "1" (3 being a Fibonacci number) and the second cent sign replacing the fifth "1" (5 being the next Fibonacci number), and the "tape reader" would be "reading" the first remaining "1" in the string, as indicated. The next time around the major loop the string would be changed to $\ldots 00\$\$\$\$\text{¢}\$\$\text{¢}\underset{\wedge}{0}0\ldots$ (the first cent sign replacing the fifth "1," and the second cent sign replacing the eighth "1" in the original string). Since the tape reader now reads a zero, the Turing machine moves to the Halt state and stops.

A Markov algorithm provides an alternate but equivalent approach to having a recognition algorithm for Fibonacci numbers. A Markov algorithm, like the Turing machine, operates on a string of elements over a given alphabet and consists of a sequence of rules which specify operations on the given string. Each rule ends with a number indicating the number of the next rule to be executed. If that rule is inapplicable, then the next rule in order is taken. The algorithm starts with rule number zero and each rule is applied to the leftmost occurrence of the element in the string. A rule ending with a period indicates a terminating rule, after which the algorithm is completed.

The Markov algorithm given below operates in a manner similar to the Turing machine given above. Both the Markov algorithm and the Turing machine generate Fibonacci numbers inside the given string of 1's and check to see if the constructed string and the given string are equal.

MARKOV ALGORITHM TO RECOGNIZE FIBONACCI NUMBERS

0:	$1 \to \alpha$, 1	first 1 converted to α
1:	$\alpha 1 \to \beta\alpha$, 3	first α changed to β, next available 1 to α
2:	$\Lambda \to \Lambda$.	nothing changed and Markov algorithm stops
3:	$\alpha \to \delta$, 3	repeated step, α's to deltas
4:	$\Lambda \to \gamma$, 5	gamma inserted at beginning of string
5:	$\gamma\beta \to \beta\gamma$, 9	gamma shifted right one through β's
6:	$\gamma \to \Lambda$, 7	delete gamma
7:	$\delta \to \beta$, 7	repeated step, deltas to β's
8:	$\Lambda \to \Lambda$, 3	dummy step—if rule 7 is nonapplicable, do nothing and skip to rule 3
9:	$1 \to \alpha$, 5	change next available 1 to an α
10:	$\gamma \to \Lambda$, 11	delete gamma
11:	$\alpha \to 1$, 13	change first α back to a 1
12:	$\Lambda \to \Lambda$.	nothing changed and Markov algorithm stops
13:	$\alpha \to 1$, 13	repeated step, α's to 1's
14:	$1 \to 1$, 14	does nothing, endless loop which occurs if original string is NOT a Fibonacci number

Lambda is the null symbol. Thus, rules 2 and 12 say "do nothing and stop." Rule 4 says to insert a gamma at the beginning of the string, and rule 6 says to delete the first gamma.

This Markov algorithm works as follows: it converts a given string of 1's into a string of β's and α's that represent F_i and F_{i+1} within the string of 1's. At the end of a loop, the α's are changed to deltas and more 1's are changed into α's to correspond to the number of β's which begin the string. The deltas are then changed to β's. Thus, after one loop, the number of α's has changed from F_i to F_{i+1}, and the number of β's has changed from F_{i+1} to

$$F_i + F_{i+1} = F_{i+2}.$$

If there are no more 1's to be changed at the end of a loop, the Markov algorithm stops at rule 12, indicating that the original string of 1's was a Fibonacci number. If, however, the string was not a Fibonacci number, the Markov algorithm jumps out of the loop in midstream of changing 1's to α's and goes into an endless loop at rule 14 after changing the α's back to 1's.

REFERENCES

1. J. E. Hopcroft & J. D. Ullman. *Formal Languages and Their Relation to Automata.* Reading, Mass.: Addison-Wesley, 1969.
2. A. M. Turing. "On Computable Numbers with an Application to the Entscheidungsproblem." *Proc. London Math. Soc.* 2-42:230-265.

*****

ON SOME CONJECTURES OF GOULD ON THE PARITIES OF THE BINOMIAL COEFFICIENTS

ROBERT S. GARFINKEL
Management Science Program, University of Tennessee, Knoxville, TE 37916
STANLEY M. SELKOW
Computer Science Dept., University of Tennessee, Knoxville, TE 37916

In studying the parities of the binomial coefficients, Gould [1] noted several interesting relationships about the signs of the sequence of numbers

$$(-1)^{\binom{n}{0}},\ (-1)^{\binom{n}{1}},\ \ldots,\ (-1)^{\binom{n}{n}}.$$

Further interesting relationships may be discovered by converting each such sequence to a binary number, $f(2, n)$, by

$$f(x, n) = \sum_{k=0}^{n} x^k \frac{1 - (-1)^{\binom{n}{k}}}{2} \tag{1}$$

and then comparing the numbers of the sequence $f(2, 0)$, $f(2, 1)$, $f(2, 2)$, ... The following conjectures were then proposed by Gould.

Conjecture 1: $f(2, 2^m - 1) = 2^{2^m} - 1.$

Conjecture 2: $f(2, 2\ ) = 2^{2^m} + 1.$

Conjecture 3: $f(x, 2n + 1) = (x + 1)f(x, 2n).$

We will prove these conjectures and present some related results.

The following lemma provides a convenient recursive scheme for generating the sequence of numbers $f(x, 0)$, $f(x, 1)$, ... We use the notation $(.)_x$ to denote the representation of a number to the base x.

Lemma 1: The sequence $f(x, n)$ may be defined by $f(x, 0) = 1$, and if

$$f(x, n - 1) = (a_{n-1}, \ldots, a_0)_x$$

for $n > 0$, then