Peg-Solitaire, String Rewriting Systems and Finite Automata

B. Ravikumar
Department of Computer Science
University of Rhode Island
Kingston, RI 02881
ravi@cs.uri.edu

Abstract

We consider a class of length-preserving string rewriting systems and show that the set of encodings of pairs of strings $< s, f >$ *such that* f *can be derived from* s *using the rewriting rules can be accepted by finite automata. As a consequence, we show the existence of a linear time algorithm for determining the solvability of a given* $k \times n$ *peg-solitaire board, for any fixed* k*. This result is in contrast to the recent results of [UEHA] and [AVIS] that the same problem is NP-hard for* $n \times n$ *boards. We look at some related string rewriting systems and find conditions under which the encodings of the pairs* $< s, f >$ *where* f *can be derived from* s *is regular.*

1 Introduction

Peg Solitaire is one of the most popular solitaire board games. Its history dates back to at least seventeenth century. It has been sold as a board game in various shapes, sizes and names. A complete chapter of the well-known work on mathematical games due to Berlekamp et al. [BERL] is devoted to peg-solitaire. The most comprehensive treatment of this game is the book by Beasley [BEAS]. For the sake of completeness, let us review the rules of the game. The solitaire board consists of holes which can hold pegs. The only rule of the game is the following: if A, B and C are three holes in a row and if A and B contain pegs but C does not, then the pegs in A and B can be replaced by a single peg in C. It is common to denote the holes with (without) pegs by 1 (0). The move is sometimes called a *jump*; A jumps over B and lands on C and the jumped peg is removed. The standard board on which this game is played in shown in Figure 1. The figure on the right is the result of making a move on the board shown on the left. [BEAS], [BERL], Martin Gardner's Scientific American column [GARD], and most other accounts of peg-solitaire primarily deal with this board. The game starts with

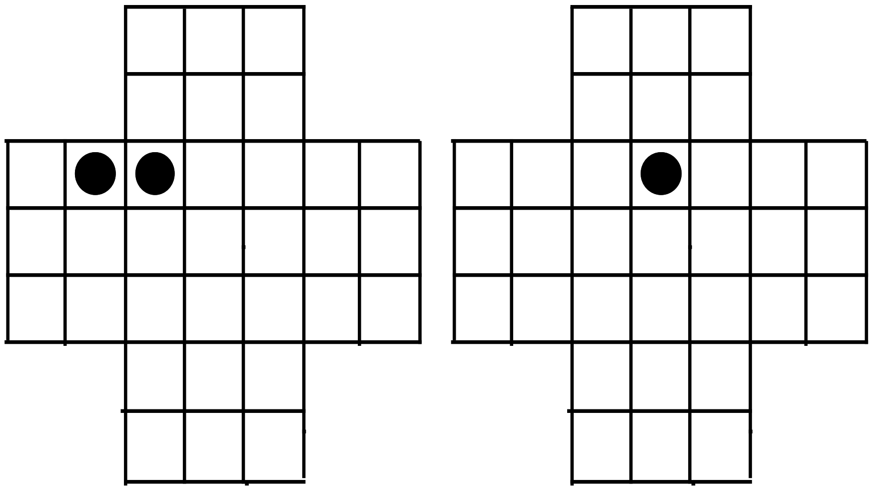

Figure 1: Peg-solitaire board

pegs in some holes (this position is called the starting position) and ends when no more jumps are possible. The starting position is said to be *solvable* if it is possible to make a sequence of jumps so that only one peg is left on the board. A central algorithmic question is the complexity of deciding if a given position is solvable. This is the main problem discussed in this paper.

In [CHAN], peg-solitaire (on the standard board) was studied as a programming problem. The goal was to write a program to determine if a given starting position is solvable, and if so, to find a sequence of moves leading to the solution. A time limit of 10 minutes was considered reasonable, and the program was expected to solve most (if not all) the starting positions in this approximate time frame. It appears from the concluding section of [CHAN] that none of the participants could achieve the intended goal. During the class discussion, some related questions about peg-solitaire were raised to get a better understanding of the problem. One of them led to a speculation that the set of solvable starting positions (when suitably encoded as strings) on the $k \times n$ board can be accepted by a finite automaton for every fixed k. This conjecture was motivated by an exercise in Manna's book [MANN] that the set of solvable patterns in the 1-dimensional case (namely $1 \times n$ board) is regular. One of the main results presented in this paper is a proof of this conjecture.

The 1-dimensional peg-solitaire is easy enough that a regular expression for the solvable patterns can be explicitly constructed, see e.g. [CHAN]. Such an explicit construction seems virtually impossible for general k. Further the regular expression approach does not easily generalize. For example, suppose in the 1-dimensional case, the board is changed to a circular one. This is just one of many possible variations presented in the next paragraph. Now the regular expression may be entirely different from the one for the linear case. What we need is a framework that works for a wide variety of such problems. Presenting such a unified approach is the main contribution of this paper.

Many variations of peg-solitaire have evolved during its long history. We will present some of them below. In one variation, a peg is specially designated, which can't be removed. Thus the result of a move sequence leading to a solution will leave the special peg as the lone survivor. In another variation, one of the holes will be designated and the last peg is required to land there. We can also combine the two variations so that one peg and one hole are designated; the goal is to move the special peg to the selected hole, removing all other pegs in the process. A third variation is to specify a starting position s and a final position f (which may have more than one peg in it) and ask if f is reachable from s by a sequence of moves. Numerous other variations are obtained by modifying the rule slightly. For example, we may add a new rule: If A, B and C are three holes in a row, and if A and C have pegs but B does not, then remove pegs from A and C and introduce one in B. This version seems to have some relevance to the work of Avis and Deza [AVIS]. In fact, many other board games can be defined by such string rewriting rules.[1] We can also create new variations (in an orthogonal direction) by making the board shape triangular, or toroidal etc. The technique and the results presented in this paper hold for all these variations. In fact, our result holds for a general class of string rewriting systems.

It has been shown in [UEHA] that the set of solvable positions in the two dimensional rectangular peg-solitaire is NP-complete. Avis and Deza [AVIS] considered a closely related problem (in which the jumping rule is the same but more than one peg can be placed in a hole) and showed it to be NP-complete as well. It is therefore of interest to know if the problem becomes tractable when we restrict one of the dimensions to a constant. Recently, fixed parameter complexity of many problems has been studied by bounding one of the input parameters of the problem by a constant (see e.g. [DOWN]). Our result that the $k \times n$ peg-solitaire is solvable in linear time for every fixed k has the same flavor. (Of course, the constant factor is exponential in k.)

The rest of the paper is organized as follows. In section 2, we introduce the notion of a length-preserving, change-bounded string rewriting systems and show that the languages associated with them are regular. In section 3, we show that monotonicity (another property of rewriting systems) implies change-boundedness. As a consequence, the set of solvable positions in $k \times n$ peg-solitaire and the variations presented above are regular. In section 4, we discuss some string rewriting systems motivated by the results of section 3 and present general conditions under which the associated languages are regular. In section 5, we present some open problems and possible ways to extend this work.

2 String Rewriting Systems

We begin with some definitions. As indicated above, the peg-solitaire can be viewed as a string rewriting system. (For example, the 1-dimensional case cor-

[1] For example, the *All 1's Problem* studied by Sutner [SUTN] involves complementing all the bits in the neighborhood of a vertex of a graph.

responds to the rewriting rules $110 \rightarrow 001$ and $011 \rightarrow 100$.) Let Σ be a finite alphabet. For simplicity, we will take $\Sigma = \{0,1\}$, although the results hold for any finite alphabet. A rewrite rule is of the form $x \rightarrow y$ where $x, y \in \Sigma^*$. A rewriting system R is a finite collection of rewrite rules. String rewriting systems have been extensively studied, see e.g. [BOOK]. Let R be a string rewriting system. With respect to R, we say that a string w derives z in one step (denoted $w \Rightarrow_R z$) if there exists a rewrite rule $x \rightarrow y$, and strings w_1, w_2 such that (i) $w = w_1 x w_2$ and (ii) $z = w_1 y w_2$. We say that w derives y (denoted $w \Rightarrow_R^* y$) if there is a sequence w_1, ..., w_k such that $w = w_1$, $y = w_k$ and $w_i \Rightarrow_R w_{i+1}$ for all i. Note in the above definition that the rules are used in one direction only, i.e., only the left-side string is replaced by the right-side string, not vice-versa. When there is no confusion about R, we may omit the subscript.

A rewriting system R is called *length-preserving* if it satisfies the following condition: for each rule $x \rightarrow y$ in R, $|x|$ = length of x, is equal to $|y|$. Let R be a length-preserving rewriting system. For a string x, let $x^{(i)}$ denote its i-th symbol from the left. A string x is said to be k-bounded if the number of i's such that $x^{(i)} \neq x^{(i+1)}$ is at most k. R is said to be *change-bounded* (with change-bound d) if the following condition holds: Suppose x_1, x_2, ..., x_m is any sequence of strings (all of the same length) such that x_i can be derived from x_{i-1} in a single step using R. Then for any k, $1 \leq k \leq |x_1|$, the string $x_1^{(k)} x_2^{(k)} ... x_m^{(k)}$ is d-bounded. Intuitively, change-boundedness means the following. For each position in the string, the number of times the symbol in that position changes is bounded by constant d, independent of the length of the string.

Two points should be noted in the above definitions. The first one concerns length-preservedness. This condition does not require that all rules have the same LHS length. It only requires that for each rule, the lengths of LHS and RHS be the same. Regarding change-boundedness, it should be noted that this term is meaningful only when applied to length-preserving rewriting systems.

We will define two languages associated with R. The first language is the encoding of pairs of strings $< x, y >$ where y can be derived from x. For the second language, the target strings are specified explicitly.

1. $L(R) = \{[a_1, b_1][a_2, b_2]...[a_m, b_m] \mid a_1 ... a_m \Rightarrow_R{}^* \; b_1 ... b_m \}$.
2. Let T be a language over the same alphabet Σ.
   $$L(R,T) = \{x \mid x \Rightarrow_R^* y \text{ for some } y \in T \}$$

Finally, we define a *monotonic* string-rewriting system as follows. R is called *monotonic* if there is some symbol $a \in \Sigma$ such that for every rule $x \rightarrow y$, the number of occurrences of a in y is strictly less than the number of occurrences of a in x. Note that the above inequality should hold w.r.to the same symbol a for all rules. Clearly, the rewriting system that defines the 1-dimensional peg-solitaire is monotonic.

In the rest of this section, we will show that if R is length-preserving and change-bounded, then $L(R)$ is regular. If, in addition, T is also regular then

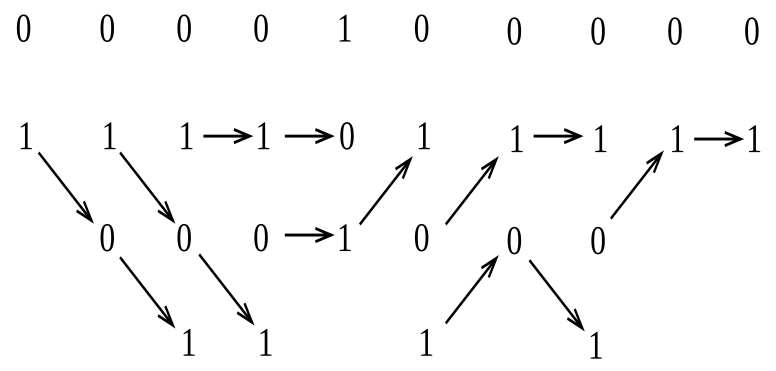

Figure 2: A partial history graph

so is $L(R, T)$. In Section 3, we will show that monotonicity implies change-boundedness (but not vice-versa). Thus the results of this section hold under the stronger hypothesis of monotonicity as well. As a consequence of this latter result, we deduce a linear time algorithm for determining the solvability of a given position in $k \times n$ peg-solitaire.

THEOREM 2.1 *Let R be a length-preserving and change-bounded string rewriting system. Then $L(R)$ is regular. Further, if T is also regular, then so is $L(R, T)$.*

Proof. We will show the latter claim. The former can be shown by an almost identical proof technique. Let d be the change-bound of R (independent of the input length). We will define a language L' over the alphabet $\Sigma_2 = \cup_{i=1}^{d} {\Sigma_1}^i$ where $\Sigma_1 = \Sigma \times \{0, 1, 2, ..., d\}$. We will show that L' is regular. There is a homomorphism h such that $h(L') = L(R, T)$. Since regular languages are closed under homomorphism, the theorem follows. The formal description of L' in the general case is somewhat complicated. So we will first present an example.

Let $\Sigma = \{0, 1\}$. The example we consider is an extension of peg-solitaire. i.e., $R = \{110 \rightarrow 001, 011 \rightarrow 100, 101 \rightarrow 010\}$. Let T be the set of strings with exactly one 1 in it. Clearly, T is regular. Let x be the string 10111101110. x is solvable since there is a sequence of moves that transform x into 00001000000. Start by writing this final target string 00001000000 as the first row string of a string structure that we create. Consider the moves in reverse. The last move involved replacing the string 001 by 110. Record this rewriting move by writing 110 just below 001. Further, to indicate that 110 is a group, link them up by an arrow going from the first 1 to the second 1, and from the second one to 0. Suppose the second to last move changed 00110000000 into 00101100000. Again, we record this by writing 011 just below the corresponding 100 and making a chain of symbols in 011. Continuing like this, we end up creating the structure shown in Figure 2.

This structure has the following properties: Concatenation of the last row symbol of each column gives the start string x. The string in the first row is the target string y. It contains a partial history of the moves made. The exact order in which the moves were made is not retained. But there is enough information to provide "witness" to the fact that the start string can derive the target string. It should be noted that there can be more than one 'partial history graph' associated with an (input, output) pair. We will now convert the above structure into a string over Σ_2 as follows. Each column will define a super-symbol containing many 'tracks' - one for each row. Since R is d change-bounded, the number of tracks in the super-symbols is at most d. The mapping is defined as follows: The first (track) entry of the i-th super-symbol will be $< a, 0 >$ if a is the i-th symbol in the first row of the structure. The other track entries are defined as follows: Suppose the i-th column r_i-th row symbol a has an arrow to $i+1$-st column r_{i+1} th row symbol, then the r_i-th track of the i-th super-symbol will be the pair $< a, r_{i+1} >$. If the symbol is the last symbol in a chain, its next symbol field will be 0. Thus, each track of the super-symbol is a member of Σ_1, and so the super-symbol itself is in Σ_2. It is not difficult to see that there is a 1-1 mapping between the structure we described above and the corresponding string over the alphabet Σ_2. To summarize, for each string $x \in L(R,T)$, there is at least one partial history graph that witnesses its membership in $L(R,T)$ and this graph can be mapped (in a 1-1 manner) to a string in Σ_2. The first six symbols of the string over Σ_2 thus obtained from the partial history graph of Figure 1 is shown below.

$$\begin{pmatrix} <0,0> \\ <1,3> \end{pmatrix} \begin{pmatrix} <0,0> \\ <1,3> \\ <0,4> \end{pmatrix} \begin{pmatrix} <0,0> \\ <1,2> \\ <0,4> \\ <1,0> \end{pmatrix} \begin{pmatrix} <0,0> \\ <1,2> \\ <0,3> \\ <1,0> \end{pmatrix} \begin{pmatrix} <1,0> \\ <0,0> \\ <1,2> \end{pmatrix} \begin{pmatrix} <0,0> \\ <1,0> \\ <0,2> \\ <1,3> \end{pmatrix}$$

The above construction can be readily generalized to any R and T. For any string $x \in L(R,T)$, let s be an associated witness string in Σ_2. Define the language:

$$L' = \{s \mid s \textit{ is a witness string for some } x \textit{ in } L\}$$

We will now present an informal proof that L' is regular. Each super-symbol b in Σ_2 is a column vector of pairs $< a, t >$ where $a \in \Sigma$ and $t \in \{0, 1, ..., d\}$. If this pair appears as the i-th row entry of b, a will be called the i-th basic symbol in b.

It is not difficult to see that a string $s \in \Sigma_2$ is in L' if and only if:

1. The concatenation of the basic symbols of the first row gives a string in T.
2. The concatenation of the last row basic symbols of each column (from left to right) is the input string x.
3. For every chain in the structure, if the string defined by the chain is p, and if q is the string obtained by concatenating the symbols immediately above the chain, then $p \to q$ is in R.

We will informally describe an NFA M' that can verify the above conditions. Condition (1) can be verified by simulating a DFA for language T. There is nothing to verify in condition (2). To verify (3), the NFA scans the input from left to right. Basically, when a chain starts, M' guesses the rule number i and also remembers the length of the chain processed so far. (Initially, this length is 0.) After reading a prefix of the input, the NFA is in a state that has the information about all the chains whose verification is in progress in the form $< i, j, k >$ where i is the rule number, j is the prefix of the chain processed so far, and k is the position of the next symbol in the chain in the next column. After reading the next symbol, the following updates are made: First, it is checked that the chains do not "cross". It is also checked that the pair of symbols in two successive rows corresponding to a chain are correct (w.r.to the guess made) and the length information is updated (i.e., decreased by 1). Finally, if the next members of the chains of two adjacent row elements are r_i and r_{i+1}, a new chain is started for all the intermediate row numbers, namely $r_i + 1$, $r_i + 2$, ..., $r_{i+1} - 1$. (For example, consider Figure 2. In column 3, the neighbors of the successive rows 2 and 3 in column 4 are 2 and 4 respectively. The intermediate row number 3 in the next column must be the beginning point of a new chain.) The next state is updated nondeterministically to include all these factors. If any of the attempted checks fails, the input is rejected. If the NFA can successfully complete the verification of the entire input as described above, it accepts. A formal construction can be found in the full version of the paper. It is not difficult to show that $L(M') = L'$.

To conclude the proof of the main theorem, define the homomorphism h : $\Sigma_2 \to \Sigma$ defined as: $h\left(\begin{pmatrix} < a_1, b_1 > \\ < a_2, b_2 > \\ \cdots \\ < a_k, b_k > \end{pmatrix}\right) = a_k$, i.e., each super-symbol is mapped to the last row basic symbol. Clearly, $h(L') = L(R, T)$. Since regular languages are closed under homomorphism, the result follows. ♣

How big is the NFA accepting L'? A rough upper bound is presented below. Let r be the number of rules, s the maximum length of the LHS (and RHS) of the rewrite rules, and let d be change-bound. Then the size of the NFA can be shown to be upper bounded by $(rsd)^d$. In the case of 1 dimensional peg-solitaire, r is 2, s is 3 and d can be shown to be at most 5. Even in this simple case, the size of our NFA is extremely high. It is claimed in [CHAN] that there is a DFA with about 17 states for the set of solvable strings in the 1-dimensional case. It should be possible to optimize the construction in specific cases by using further properties of the rewriting rules.

3 Application to Peg Solitaire and its Variations

In this section, we will present some applications of Theorem 2.1 to peg-solitaire problems presented in Section 1. Consider any hole in the peg-solitaire board. It is in one of the two states - 1 (has a peg) and 0 (no peg). We will show that the number of state changes that any hole undergoes in the two-dimensional board (even in the infinite case) is bounded by a constant. As a consequence, we will show that the set of solvable positions in the $k \times n$ board (k constant), suitably encoded as strings over $\{0, 1\}$, is regular.

The next result applies to peg-solitaire on a $n \times m$ board for any $n, m \geq 1$.

THEOREM 3.1 *In $n \times m$ peg-solitaire, the number of times any hole changes its state in every possible move sequence is bounded by 26.*

Proof. The proof idea is based on the well-known Conway's trick of assigning a weight to each cell of the board. In fact, the above claim even holds for a board that extends to infinity in all four directions. Label the holes by a pair of integers $< i, j >$. Since all cells are identical in the infinite board, without loss of generality, we will show the claim for the hole $< 0, 0 >$. Assign a weight of $t^{|i|+|j|}$ to the cell $< i, j >$ where t is the positive root of the equation $x^2 + x - 1 = 0$. t is thus $\frac{(\sqrt{5}-1)}{2}$ which is the reciprocal of the golden ratio $\frac{(\sqrt{5}+1)}{2}$. Clearly $0 < t < 1$.

A board position is defined by the set of occupied holes, i.e., the holes with a peg in it. The total weight of a position is defined as the sum of the weights of positions containing the peg. Let w be the weight of a board position and w' the weight of a board position derived from the former by a sequence of legal peg-solitaire moves. Then we observe that $w' \leq w$. This is seen by considering two cases:

case i: the move does not involve a cell on the X or Y axis as the middle cell. In this case, a left or down jump does not change the weight. (Old weight includes $t^{i+1} + t^{i+2}$ for some i and the new weight includes t^i but not the other two weights. The change of weight is thus $t^{i+1} + t^{i+2} - t^i = 0$ by the choice of t.) A right or down jump results in a weight reduction since the net change in weight will be $t^i + t^{i+1} - t^{i+2}$.

case ii: the move involves a cell on the X or Y axis as the middle cell. Clearly the total weight is seen to decrease by an amount equal to the weight of the middle cell. Thus in every case, the total weight does not increase. Now consider the hole $< 0, 0 >$. When this hole changes from 1 to 0, the total weight decreases by at least 1. (There are two ways in which the change can occur. In one, the weight decreases by 1, and in the other, it decreases by $2t$ which is greater than 1.) Thus after every two changes of state of the hole $< 0, 0 >$, the total weight of the configuration decreases by at least 1. The initial weight of the infinite board is easily seen to be less than 13. Thus the maximum number of state change possible to the cell $< 0, 0 >$ is 26. This concludes the proof. ♣

As a consequence of Theorem 3.1, we will prove the main result of this paper.

THEOREM 3.2 *Let L be the set of solvable positions in the $k \times n$ peg-solitaire board, each encoded as a string over the alphabet $\{0, 1\}$ using column major order (by concatenating the column strings from left to right). Then L is regular.*

Proof. The proof is a simple consequence of Theorems 2.1 and 3.1. We can express the moves of $k \times n$ peg-solitaire in this encoding as a collection R

of rewriting rules described below: $110 \to 001$, $011 \to 100$, $\{1w1x0 \to 0w0x1 \mid w, x$ are strings of length $k-1\}$, $\{0w1x1 \to 1w0x0 \mid w, x$ are strings of length $k-1\}$. (The first two rules describe a column jump, while the rest of them describe all possible row jumps.) These rules are length-preserving. By Theorem 3.1, the number of state changes in any cell is bounded by 26. Thus R is length-preserving and change-bounded. Therefore, by Theorem 3.1, $L(R)$ is regular. ♣

Using similar ideas, we can show the following results regarding the variations of peg-solitaire presented in Section 1. We will skip the details.

THEOREM 3.3 *The set of solvable positions in the $k \times n$ board is regular even in the following variations: (i) a special peg is designated which can't be removed from the board. (ii) in addition to the rules $011 \to 100$ and $110 \to 001$, include the rule $101 \to 010$.*

THEOREM 3.4 *The set of solvable positions in the $k \times n$ toroidal and triangular peg-solitaire, when suitably encoded, are regular.*

In fact, we can show a general result about string rewriting system from which the results about all the variations of peg-solitaire introduced in Section 1 can be deduced.

THEOREM 3.5 *Let R be a monotonic, length-preserving string rewriting system. Then $L(R)$ is regular. Further if T is regular, then so is $L(R, T)$.*

4 Generalizations that do and do not hold

An even more general question suggested by the results of Sections 2 and 3 is to characterize the rewrite systems such that the set of solvable strings is regular (given that the target set is regular). This seems to be difficult. However, we have made some preogress in answering this question. A complementary question is to find the simplest rewriting systems such that the set of solvable strings is **not** regular. We also present some negative results of this kind. Because of space limitation, the details of this section are omitted in this abstract. Complete proofs can be found in the full version of this paper.

5 Conclusions and Open Problems

This work was motivated by the $k \times n$ peg-solitaire for fixed k, but the main result of this work holds in a more general setting. It is surprising that an explicit polynomial time algorithm even for $3 \times n$ case is not obvious although we know, in principle, the existence of a linear time algorithm (which simulates the underlying DFA). Explicitly deriving such an algorithm would be of interest.

Although our techniques in this paper provide a way to understand several variations of peg-solitaire, there is one variation which seems to be beyond the reach our techniques, namely the *two-player version.* Even the 1-dimensional case presented below seems to be nontrivial. Two players play the following game taking turns alternately, starting with a binary string. At his/her turn, either player can replace some occurrence of 110 (011) by 001 (100). The player who makes the last move is the winner. What can we say about the set of starting strings for which the first player can force a win? Is it regular?

References

[AVIS] D. Avis and A. Deza 1996. *Solitaire Cones*, Technical Report, School of Computer Science, McGill University.

[BERL] E. Berlekamp, J. Conway and R. Guy 1982. *Winning Ways for Your Mathematical Plays: Vol. 1* Academic Press.

[BEAS] J. D. Beasley 1985. *Ins and Outs of Peg Solitaire*, Oxford University Press, Recreations in Mathematics Series.

[BOOK] R. Book and F. Otto 1993. *String Rewriting Systems* Springer-Verlag Graduate Texts in Computer Science.

[DOWN] R. Downey and M. Fellows, 1992. *Fixed-parameter Intractability.* Proceedings of 7th Annual Conf. on Structure in Complexity Theory, IEEE Computer Society Press, Los Alamitos, CA, 1992.

[FRAZ] M. Frazier and C.D.Page, 1993. *Prefix Grammars: An alternative characterization of regular languages*, Information Processing Letters. Vol. 51, pp. 67-71.

[GARD] M. Gardner, Peg Solitaire, in *Unexpected Hanging and other Mathematical Diversions*, Simon and Schuster, New York.

[HOPC] J. Hopcroft and J. Ullman 1979. *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley Inc., Reading, Mass. (1979).

[KARI] L. Kari, G. Paun, G. Rozenberg, A. Salomaa and S. Yu 1996. *DNA Computing, Matching Systems and Universality*, Turku Center for Computer Science, TUCS Technical Report No. 49.

[MANN] Z. Manna 1974. *Mathematical Theory of Computation*, McGraw-Hill book company.

[SUTN] K. Sutner 1989. *Linear Cellular Automata and the Garden-of-Eden*, Mathematical Intelligencer Vol. 11, No. 2, Springer-Verlag, New York.

[UEHA] R. Uehara and S. Iwata 1990. *Generalized Hi-Q is NP-complete*, Trans IEICE 73.

[CHAN] E. Chang, S. Phillips and J. Ullman 1991. *A Programming and Problem Solving Seminar*, Stanford University, Computer Science Department Technical Report No. STAN-CS-91-1350.