JOURNAL OF COMPUTER AND SYSTEM SCIENCES 30, 116–129 (1985)

Solitaire Automata*

RICHARD E. LADNER AND JEFFREY K. NORMAN

Department of Computer Science, University of Washington, Seattle, Washington 98195

Received May 16, 1984; revised July 9, 1984

The notion of a solitaire automaton is introduced to model the class of solitaire games. Space and time bounded solitaire automata with varying degrees of information are investigated. In general, solitaire automata have partial or imperfect information. The principal results are (i) $s(n)$-space bounded solitaire automata are equivalent to Turing machines which run in space exponential in $s(n)$ and (ii) $t(n)$-time bounded solitaire automata are equivalent to alternating Turing machines which run in time $t(n)$. The power of solitaire automata is also determined when the information is perfect or zero. ©1985 Academic Press, Inc.

1. INTRODUCTION

There is a natural correspondence between certain computing automata and certain classes of games. For example, an alternating Turing machine can be thought of as a two-person game of perfect information, where the two players correspond to the universal and existential states of the automaton. An initial configuration of the automaton corresponds to an initial position in the game and an accepting configuration to a winning position in the game. The existence of a winning strategy by one of the players from an initial configuration corresponds to acceptance of the input by the automaton. Reif [5] and, later, Peterson and Reif [4] carried this correspondence much further to two-person private alternating and multiple-person private alternating Turing machines which model games of partial information. Surprisingly, all these game-like automata are equivalent to different types of standard deterministic or nondeterministic Turing machines (cf. [1, 5, 4, 6]). In this paper we call the two-person private alternating automata of Reif simply *game automata.*

This paper introduces and investigates a special case of game automata called *solitaire automata* which naturally correspond to solitaire games like Mastermind, Twenty Questions, Adventure, Rubik's Cube, Klondike (sometimes called Canfield), and some computer games. A solitaire game has two players, 0 and 1. From an initial position Player 0 begins by playing a series of nondeterministic moves and once Player 1 enters the game, Player 0 must play deterministically. Player 1 is the real player in the solitaire game while Player 0 has an initial role as "card shuffler" and later as "witless responder" to Player 1's moves.

* This research was supported by the National Science Foundation under Grant MCS80-03337.

0022-0000/85 $3.00

TABLE I

Complexity Classes

Information	Solitaire automata	Game automata
Complete	Π_2-$SPACE(s(n))$	$ASPACE(s(n))$
Zero	$\bigcup NSPACE(2^{cs(n)})$	$\bigcup NSPACE(2^{cs(n)})$[R]
Partial	$\bigcup NSPACE(2^{cs(n)})$	$\bigcup ASPACE(2^{cs(n)})$[R]

Note. The classes correspond to $s(n)$-space bounded automata provided $s(n) \geq \log n$ and $s(n)$ is space constructible. The unions are over all $c > 0$.

A game may have varying degrees of information available to Player 1: *complete, partial, or zero information.* A game is of complete (perfect) information if Player 0's moves are entirely visible to Player 1, otherwise it is of partial (imperfect or incomplete) information. If Player 0's moves are entirely invisible to Player 1, the game is zero information (blindfold). In a zero information game the only information Player 1 gets from Player 0 is an indication that it is finished moving and it is Player 1's turn to move again. The notion of degree of information can be applied naturally to game automata and solitaire automata. Note that since acceptance corresponds precisely to the existence of a winning strategy for Player 1, the degree of information available to Player 0 is immaterial.

Game automata and more specifically solitaire automata can be space or time bounded. We characterize the relationship between space and time bounded solitaire automata and more conventional automata. Our results are summarized in Tables I and II, contrasted with corresponding results for game automata.

One reason game-like automata are important is they provide a tool for uncovering the computational complexity of natural problems, particularly problems involving games. Stockmeyer and Chandra used alternating automata to discover the exponential complexity of the game of PEEK and several other games based on propositional formulas [6]. Jones was the first to examine the complexity of zero information games although he did not conceive of game automata [3]. Reif defined game automata and used them to explore the complexity of partial infor-

TABLE II

Complexity Classes

Information	Solitaire automata	Game automata
Complete	Π_2-$TIME(t(n))$	$ATIME(t(n))$
Zero	Σ_2-$TIME(t(n))$	Σ_2-$TIME(t(n))$[PR]
Partial	$ATIME(t(n))$	$ATIME(t(n))$[PR]

Note. The classes correspond to $t(n)$-time bounded automata provided $t(n) \geq n$ and $t(n)$ is time constructible.

mation games such as variants of PEEK and some feline pursuit games [5]. A survey of results on the complexity of games was compiled by Johnson [2].

The purpose of this paper is to present the basic theory of solitaire automata without any applications. It is our hope that some applications of our results will be discovered, but until that time we can only present some intuitive justification that solitaire game-like phenomena actually exist in computing practice. For example, an interactive program like the "executive" in a computer system is a solitaire game of sorts. The system can be in many possible states when you "log in" modeling the initial phase of a solitaire game. Once you are "logged in" the executive responds deterministically to what you enter at your keyboard. Whether you have a winning strategy in the solitaire game could be interpreted as whether you have a strategy to break the security of the system.

2. Definitions

In this section we give the formal definitions of the variants of game automata. Unfortunately, the definitions are long and complicated, but this seems to be necessarily so because the formal definition of partial information games is itself long and complicated. In the proofs later we will refer to these formal definitions for guidance, but they will not bog us down.

Game Automata

A k-tape *game automaton* with p private tapes is a Turing machine of the form

$$M = (V, P, \Sigma, \Gamma, \delta_0, \delta_1, F, s, B),$$

where:

V is a finite set of visible states,
P is a finite set of private states,
Σ is a finite input alphabet, $\Sigma \subseteq \Gamma$,
Γ is a finite tape alphabet,
F is the set of accepting states, $F \subseteq V$,
s is the start state, $s \in V \cap P$,
B is the blank symbol, $B \in \Gamma - \Sigma$,
δ_0 is the transition function for Player 0,

$$\delta_0 \colon V \times P \times \Gamma^{2k+2-p} \to 2^{V \times P \times \{0,1\} \times (\Gamma - \{B\})^{2k-p} \times \{R,L,S\}^{2k+2-p}}.$$

δ_1 is the transition function for Player 1,

$$\delta_1 \colon V \times \Gamma^{k+1-p} \to 2^{V \times \{0,1\} \times (\Gamma - \{B\})^{k-p} \times \{R,L,S\}^{k-1-p}}.$$

Figure 1 describes a 2-tape game automaton with 1 private tape.

The automaton M has $k+1$ semi-infinite tapes, with tape 0 being the read-only

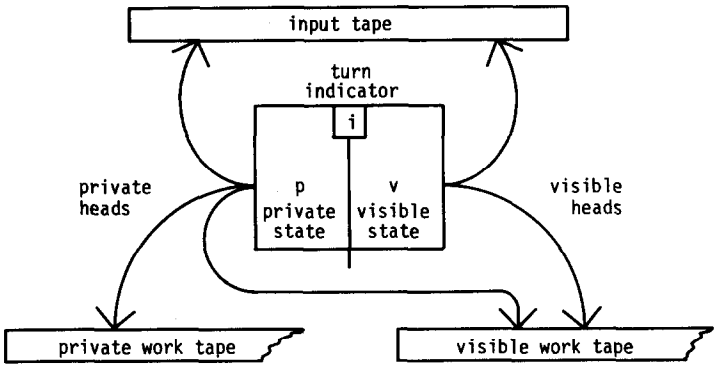

FIG. 1. A 2-tape game automaton with 1 private tape.

input tape and tapes $k-p+1$ through k being the private tapes. Tapes 0 through $k-p$ have two heads, a *visible head*, which can be moved and observed by either player, and a *private head*, belonging to Player 0, which cannot be monitored or located by Player 1. The heads and contents of tapes $k-p+1$ through k are visible only to Player 0. The state of M is a 3-tuple (v, p, i), where $v \in V$, $p \in P$, and $i \in \{0, 1\}$. Both v and i are visible to Player 1 while p is private to Player 0 and not visible to Player 1. The component i indicates it is Player i's turn to move. Player 0 can read the contents of the cells under all the tape heads and both the visible and private states, then move by changing both the visible and private states, change whose move it is, change the contents of the tape cells under the tape heads (except the input tape), and move the tape heads individually right, left, or not at all. Player 1 can read the contents of the cells under its tape heads of the visible work tapes and its input tape head and the visible state, then change the visible state, change whose move it is, change the contents of the tape cells under its visible work tape heads, and move its tape heads of the visible work tapes and input tape individually right, left, or not at all.

Acceptance Condition

To define what it means for a game automaton to accept an input requires formally defining what it means for Player 1 to have a winning strategy. This will take some development.

A *configuration* of M on input x is a $3k+5-p$-tuple

$$(v, p, t, h_0^0, \ldots, h_k^0, h_0^1, \ldots, h_{k-p}^1, \alpha_1, \ldots, \alpha_k),$$

where $(v, p, t)(v \in V, p \in P, t \in \{0, 1\})$ is the current state of M, h_i^j is the position of Player j's head on tape i, and $\alpha_1, \ldots, \alpha_k$, all members of $(\Gamma - \{B\})^*$, are the current nonblank contents of tapes 1 through k, respectively. The configuration is

composed of two parts, the *visible* and *private*. Let $C = (v, p, t, h_0^0,..., h_k^0, h_0^1,..., h_{k-p}^1, \alpha_1,..., \alpha_k)$

$$\text{visible}(C) = (v, t, h_0^1,..., h_{k-p}^1, \alpha_1,..., \alpha_{k-p}),$$

$$\text{private}(C) = (p, h_0^0,..., h_k^0, \alpha_{k-p+1},..., \alpha_k).$$

Also, define $\text{player}(C) = t$. It is straight forward but tedious to define the *transition relation* for M on input x, $\vdash_M^x$, between configurations. We will simply use $\vdash$ when no confusion would arise. It is important to remember that Player 0 is unrestricted while Player 1 can only read and modify the visible configuration. Hence, if $\text{player}(C) = 1$, $\text{visible}(C) = \text{visible}(D)$ and $C \vdash E$ then $D \vdash E'$, where $\text{visible}(E') = \text{visible}(E)$ and $\text{private}(E') = \text{private}(D)$. The *initial configuration* for M on input x, $(s, s, 0, 1,..., 1, \lambda,..., \lambda)$, is denoted by $\text{init}(M, x)$ (λ denotes the empty string). An *accepting configuration*, a winning configuration for Player 1, is one where the visible state is in F.

Informally, M accepts an input x if in the game described by M Player 1 has a winning strategy. Before we formally define acceptance we make a series of definitions. A *history* for M on input x is a sequence $C_0, C_1,..., C_n$ such that $C_j \vdash C_{j+1}$ for $0 \leqslant j \leqslant n-1$ and $C_0 = \text{init}(M, x)$. For each history H define $\text{last}(H) = C_n$. For each history H we define its visible part, $\text{visible}(H)$. If H is a single configuration then $\text{visible}(H)$ is already defined. For histories of length 2 or greater we define the visible part inductively. Let H, D be a history where H is itself a history.

$$\begin{aligned} \text{visible}(H, D) &= \text{visible}(H) && \text{if } \text{visible}(\text{last}(H)) = \text{visible}(D), \\ &= \text{visible}(H), \text{visible}(D) && \text{otherwise.} \end{aligned}$$

A *visible history* for M on input x is the image under the function visible of a history for M on x. Intuitively, the visible history is all that Player 1 has an opportunity to see. Player 1 cannot witness a move of Player 0 unless Player 0 moves the visible head or alters the contents of one of the visible tapes or modifies the visible state. A *strategy* for Player 1 in M on input x is a function σ mapping visible histories into visible configurations with the property that if H is a history with $\text{player}(\text{last}(H)) = 1$ then H, C is a history where $\text{visible}(C) = \sigma(\text{visible}(H))$ and $\text{private}(C) = \text{private}(\text{last}(H))$.

A *computation tree* of M on input x is a finite labeled tree with the properties:

(a) each node η of the tree is labeled with a configuration $l(\eta)$ of M on x,

(b) the root of the tree is labeled $\text{init}(M, x)$,

(c) if η is an internal node of the tree and $\text{player}(l(\eta)) = 0$ then for each C such that $l(\eta) \vdash C$ there is a child θ of η, where $l(\theta) = C$,

(d) if η is an internal node of the tree and $\text{player}(l(\eta)) = 1$ then η has exactly one child θ and $l(\eta) \vdash l(\theta)$.

A strategy σ is a *winning strategy* for Player 1 in M on input x if there is a computation tree of M on input x with the properties:

(i) each leaf of the tree is labeled with an accepting configuration,

(ii) if η is an internal node of the tree with player$(l(\eta))=1$ and H the sequence of labels on the path from the root to η and θ is the child of η, then $\sigma(\text{visible}(H))=\text{visible}(l(\theta))$.

The input x is *accepted* by M if Player 1 has a winning strategy in M on input x. The language accepted by M, denoted by $L(M)$ is the set of all inputs accepted by M.

Space and Time Bounded Game Automata

Space and time bounded game automata are defined in a natural way. If $x \in L(M)$ then x is *accepted in space* s if there is a winning strategy for Player 1 in M on input x such that while Player 1 is playing the winning strategy no more than s distinct tape cells are scanned on any work tape. More formally, there is a winning strategy for Player 1 in M on input x whose computation tree T has the property that whenever $(v, p, t, h_0^0,..., h_k^0, h_0^1,..., h_{k-p}^1, \alpha_1,..., \alpha_k)$ is the label of a node in T then $|\alpha_j| < s$ for $1 \leqslant j \leqslant k$. If $x \in L(M)$ then x is *accepted in time* t if there is a winning strategy for Player 1 in M on input x such that while Player 1 is playing the winning strategy the number of moves made by both players combined before Player 1 wins is less than t. More formally, there is a winning strategy for Player 1 in M on input x whose computation tree T has the property that no root to leaf path in T is longer than t. Let f be a function from the nonnegative integers to the nonnegative integers. The language $L(M)$ is accepted in space (time) $f(n)$ if for all $x \in L(M)$, x is accepted by M in space (time) $f(|x|)$.

Degree of Information

The game automaton M may vary on the *degree of information* provided to Player 1. The unrestricted automaton is said to be one of *partial information.* If there is nothing private to Player 0 then M is a *complete information* automaton. More formally, if $P=\{s\}$, $p=0$, and Player 0 only moves the visible tape heads then M is a complete information automaton. If everything possible is private to Player 0 then M is said to be a *zero information* automaton. That is, Player 0 never changes the visible state, moves the visible tape heads, nor modifies the contents of the visible work tapes. More formally, to make M have zero information we modify the δ_0 function to be

$$\delta_0 \colon V \times P \times \Gamma^{2k+2-p} \to 2^{P \times \{0,1\} \times (\Gamma - \{B\})^p \times \{R,L,S\}^{p-1}}.$$

The definition of the transition relation must also be changed accordingly.

Solitaire Automata

A *solitaire automaton* is a game automaton which starts with a phase where Player 0 plays alone nondeterministically. Once Player 1 enters the game the second phase begins where Player 0 must play deterministically. Formally we add a component D to the specification of M. The set D, which is a subset of P, is the set of deterministic private states. We have the additional restrictions on δ_0:

$$\delta_0\colon V \times (P-D) \times \Gamma^{2k+2-p} \to 2^{V\times(P-D)\times\{0\}\times(\Gamma-\{B\})^{2k-p}\times\{R,L,S\}^{2k+2-p}} \cup 2^{V\times D\times\{1\}\times(\Gamma-\{B\})^{2k-p}\times\{R,L,S\}^{2k+2-p}},$$

$$\delta_0\colon V \times D \times \Gamma^{2k+2-p} \to 2^{V\times D\times\{0,1\}\times(\Gamma-\{B\})^{2k-p}\times\{R,L,S\}^{2k+2-p}}.$$

Finally, if $(v, d, x_1, \ldots, x_{2k+2-p}) \in V \times D \times \Gamma^{2k+2-p}$ then $\delta_0(v, d, x_1, \ldots, x_{2k+2-p})$ is either empty or a singleton. Solitaire automata, like game automata, can also have varying degrees of information, partial, complete, or zero.

Complexity Classes

There are two types of automata, game and solitaire, three degrees of information, partial, complete, and zero, and two types of resources, space and time. So, altogether there are twelve complexity class types to define:

$$\begin{Bmatrix} P \\ C \\ Z \end{Bmatrix} \begin{Bmatrix} G \\ S \end{Bmatrix} - \begin{Bmatrix} SPACE \\ TIME \end{Bmatrix} (f(n)) = \text{languages accepted with degree of information}$$

$$\begin{Bmatrix} \text{partial} \\ \text{complete} \\ \text{zero} \end{Bmatrix} \text{by automata type} \begin{Bmatrix} \text{game} \\ \text{solitaire} \end{Bmatrix} \text{in resource bounded by} \begin{Bmatrix} \text{space} \\ \text{time} \end{Bmatrix} cf(n)$$

for some c. For example, $CG - TIME(t(n))$ is the class of languages accepted by a game automaton with complete information in time $t(n)$. Coincidentally, $CG - TIME(t(n)) = ATIME(t(n))$ because an alternating Turing machine is merely a complete information game automaton.

Using the standard technique of augmenting the tape alphabet by encoding fixed-size blocks of symbols into new characters, it can be shown that space bounded solitaire automata differing by a constant factor are equivalent. At this time we do not know if solitaire time can have constant factor speed-ups like solitaire space. To make the statements of the results simpler we, by definition, force our complexity classes to be invariant under constant factors.

General Complexity Notions

We are assuming all along that the reader is familiar with alternating Turing machines [1]. We also refer to *alternation bounded* alternating Turing machines, Σ_k- and Π_k-Turing machines. By a Σ_k-Turing machine (Π_k-Turing machine) we

mean an alternating Turing machine which begins in an existential (universal) state and makes less than k alternations along any computation path. The complexity classes Σ_k-$SPACE(s(n))$ and Σ_k-$TIME(t(n))$ (and the corresponding Π_k classes) simply refer to the class of sets accepted by $s(n)$-space and $t(n)$-time bounded Σ_k-Turing machines (Π_k-Turing machines), respectively.

Our theorems have some technical assumptions about space and time constructibility which we clarify here. A function $s(n)$ is *space constructible* if there is a Turing machine which for each n and input of length n marks exactly $s(n)$ work tape cells, then halts. A function $t(n)$ is *time constructible* if there is a Turing machine which for each n and input of length n runs for exactly $t(n)$ steps, then halts.

3. Space Bounded Solitaire Automata

In this section we characterize the language recognition capability of space bounded solitaire automata in terms of more familiar automata.

THEOREM 1. $ZS\text{-}SPACE(s(n)) = PS\text{-}SPACE(s(n)) = \bigcup NSPACE(2^{cs(n)})$, *provided* $s(n) \geqslant \log n$ *and* $s(n)$ *is space constructible.*

Proof. The proof is divided into two parts, the first showing $\bigcup NSPACE(2^{cs(n)}) \subseteq ZS\text{-}SPACE(s(n))$ and the second showing $PS\text{-}SPACE(s(n)) \subseteq \bigcup NSPACE(2^{cs(n)})$. The result follows because trivially $ZS\text{-}SPACE(s(n)) \subseteq PS\text{-}SPACE(s(n))$. The proof of (1) below is a slight variant on Reif's proof that $\bigcup NSPACE(2^{cs(n)}) \subseteq ZG\text{-}SPACE(s(n))$ [5].

$$\bigcup NSPACE(2^{cs(n)}) \subseteq ZS\text{-}SPACE(s(n)). \tag{1}$$

Let M be a 1-tape nondeterministic Turing machine accepting L in space $2^{cs(n)}$ for some $c > 0$. Informally, we construct a solitaire automaton M' in which Player 1's only winning strategy on input x is to produce successive characters of a string representing an accepting computation of M on input x. For the 1-tape Turing machine represent a configuration as a string of the form uqv, where u and v are strings of tape symbols and q is a state. Given input x of length n, M accepts x if and only if there exists a string $w = \# C_0 \# C_1 \# \cdots \# C_m \#$ with the following properties: (i) C_0 describes the initial configuration, (ii) C_m contains an accepting state, (iii) for $0 \leqslant i \leqslant m$, $|C_i| = 2^{cs(n)}$, and (iv) for $0 \leqslant i \leqslant m-1$, $C_i \vdash C_{i+1}$.

Player 1 moves by successively writing characters of w on the first cell of the visible tape. Player 0 attempts to refute Player 1's string w by privately deciding to refute (i), (ii), (iii), or (iv). Conditions (i) through (iii) are easily checked deterministically. Condition (iv) is checked by initially guessing a number j, $1 \leqslant j \leqslant 2^{cs(n)}$, then deterministically verifying that for some i, $0 < i \leqslant m$, that the $(j-1)$th, jth, and $(j+1)$th characters of C_i is not consistent with the $(j-1)$th, jth, and $(j+1)$th characters of C_{i-1} according to the rules of M. The guess of j is made during Player

0's initial moves before Player 1 joins the game. Because $s(n)$ is space constructible Player 0 can lay out $cs(n)$ space, then write the number j in binary in that space. Player 0 remembers j permanently on its private tape. For each i, $0 \leqslant i \leqslant m$, Player 0 records the $(j-1)$th, jth, and $(j+1)$th characters of C_i, then checks to see if they are consistent with the $(j-1)$th, jth, and $(j+1)$th characters of C_{i-1}. If they are not consistent then Player 0 immediately wins. Otherwise, the $(j-1)$th, jth, and $(j+1)$th characters of C_{i-1} are forgotten and the game proceeds to the next configuration produced by Player 1. If the entire string w produces no inconsistency in the $(j-1)$th, jth, and $(j+1)$th characters of the configurations then Player 1 wins. The space required for the game is logarithmic in $2^{cs(n)}$ and hence linear in $s(n)$. Thus, $x \in L$ if and only if x is accepted by M' in space linear in $s(n)$.

$$PS\text{-}SPACE(s(n)) \subseteq \bigcup NSPACE(2^{cs(n)}). \tag{2}$$

Let $M = (V, P, D, \Sigma, \Gamma, \delta_0, \delta_1, F, s, B)$ be a solitaire automaton accepting L in space $s(n)$. We will describe a nondeterministic algorithm M' that accepts L which runs in space $2^{cs(n)}$ for some c. Let x be an input of length n.

Because the algorithm M' will simulate the moves of M in a certain way we will make some assumptions about M that will make the explanation of M' easier. First, we assume that if M halts it always does so when it is Player 1's turn to move. Second, we assume that whenever M is in a configuration C with player$(C) = 0$ it is not possible for M to run forever from configuration C without allowing Player 1 to make a move. The machine M can be modified to satisfy the first assumption easily. To modify M to satisfy the second assumption we use the fact that $s(n)$ is space constructible to build a $s(n)$-space bounded counter that can count up to the number of possible configurations of M, $2^{cs(n)}$ for some c. Whenever Player 0 begins its turn the counter is set to zero and while Player 0 is continuing its turn the counter is incremented by one for each move made during that turn. Should the counter ever overflow then the modified machine terminates without accepting the input. The input would never have been accepted by the original machine anyway because Player 0 could continue to play forever without allowing Player 1 to make a move.

Define an *initial visible history* to be a visible history H such that player(last$(H)) = 1$ and for every proper prefix H' of H player(last$(H')) = 0$. In other words H is Player 1's point of view up to the point when it enters the game. The algorithm M' will check for each initial visible history H whether or not the game can be continued with Player 1 eventually winning.

Let H be an initial visible history and let S_H be the set of configurations C such that there is a history J where visible$(J) = H$ and last$(J) = C$. We can think of S_H as the set of configurations that M could be in given that Player 1 has seen the history H. Hence, all the configurations in S_H have the same visible part.

The algorithm M' will then maintain a family F_H of sets of configurations, where $F_H = \{S_H\}$ initially. The family F_H has the property that if $S \in F_H$, C, $D \in S$ then visible$(C) =$ visible(D) and player$(C) = 1$. Intuitively, a set in F_H represents all the

configurations the game could be in given what Player 1 has seen so far. The different sets within F_H represent a difference in what Player 1 has seen so far. We now describe how F_H is updated. Let $F_H = \{S_1, ..., S_p\}$ and $S_i = \{C_{i1}, ..., C_{iq_i}\}$. For each i, let V_i be the unique visible configuration such that $V_i = \text{visible}(C_{ij})$ for $1 \leqslant j \leqslant q_i$. First, the algorithm nondeterministically simulates Player 1's moves from the visible configuration V_i to a visible configuration U_i where either U_i is a halting visible configuration or a visible configuration with $\text{player}(U_i) = 0$. In the former case, if the halting configuration is accepting then the algorithm continues with the set S_i removed from F_H. If the halting configuration is not accepting then a new F_H cannot be computed, so the algorithm terminates by not accepting. In the latter case, we define the configuration D_{ij}, for $1 \leqslant j \leqslant q_i$, such that $\text{visible}(D_{ij}) = U_i$ and $\text{private}(D_{ij}) = \text{private}(C_{ij})$. Thus, the configuration D_{ij} is the configuration that Player 1 reaches from C_{ij}, where either it has won the game or it is Player 0's turn to move again.

In the case that it is Player 0's turn to move in configuration D_{ij} the algorithm computes deterministically the unique configuration E_{ij} reachable from the configuration D_{ij} and the subhistory H_{ij} of configurations that lead from D_{ij} to E_{ij}, where E_{ij} is the first configuration from D_{ij} in which it is Player 1's turn to move again. Define $S_i' = \{E_{ij}: 1 \leqslant j \leqslant q_i\}$ and the equivalence relation $\equiv_i$ on the domain S_i' by

$$E_{ij} \equiv_i E_{ik} \quad \text{if and only if} \quad \text{visible}(H_{ij}) = \text{visible}(H_{ik}).$$

So two E-configurations are equivalent if Player 1 cannot see that there is any difference between histories leading to each of them. Finally, we can define the new F_H maintained by the algorithm. For each i such that S_i' exists (i.e., S_i was not removed) the $\equiv_i$-equivalence classes are in F_H. If for each i, $1 \leqslant i \leqslant p$, S_i is removed from F_H then the algorithm proceeds to the next initial visible history. Figure 2 describes this process of going from one F_H to the next.

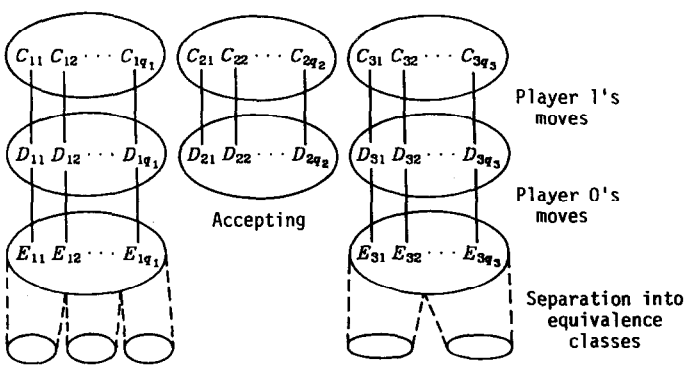

FIG. 2. Pictorial summary of how F_H is updated.

The following summarizes the algorithm M' running on input x.

begin
  for each initial visible history H **do**
  begin
    compute S_H;
    $F_H := \{S_H\}$;
    while F_H is not empty **do**
      compute the new F_H by the method described above
      (if a new F_H cannot be computed then terminate without accepting);
  end;
  accept
end.

There are two things left to show. First, that the set accepted by the algorithm M' is $L(M)$ and second, that the algorithm M' runs in the required amount of space, $2^{cs(n)}$ for some c.

Claim 1. The algorithm M' accepts x if and only if the original solitaire automaton accepts x.

Proof. Instead of giving a completely formal proof, which would tax the readers' patience, we give an informal proof partially by example. Assume x is accepted by M. There is a winning strategy σ for Player 1 in M on input x. Hence, there is a computation tree T of M on input x with every leaf labeled with an accepting configuration. Furthermore if η is an internal node of T with player$(l(\eta)) = 1$, H is the sequence of configurations on the path from the root to η, and θ is the child of η, then visible$(l(\theta)) = \sigma$(visible(H)). Because M is a solitaire automaton the tree T must have the following structure. The top part of the tree is fully branching representing all the initial histories up to the point when Player 1 enters the game. Once a node is reached whose label indicates it is the first time it is Player 1's turn to play, the tree is unary branching below that node. It is unary branching because

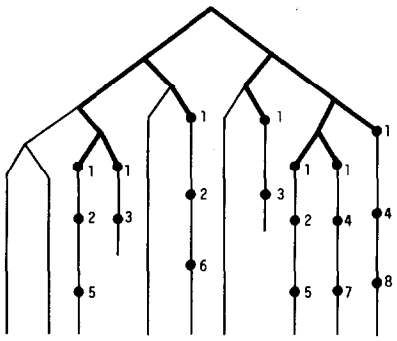

FIG. 3. Structure of the computation tree of a solitaire automaton.

either it is Player 0's turn and it is deterministic or it is Player 1's turn and it just makes the one move according to the winning strategy. Figure 3 illustrates the structure of a computation tree for a solitaire automaton.

By example let us trace what the algorithm M' can do with respect to the tree T. An initial visible history H defines the subtree of T consisting of all initial histories which have H as their visible part. This subtree is marked in bold in Fig. 3. The set of configurations at the leaves of this subtree is the set S_H. The set S_H is marked with "1" in Fig. 3. At this point F_H consists of exactly one set, namely S_H. Below those nodes marked "1" Player 1 moves according to the strategy σ, that is, Player 1's nondeterministic moves follow the strategy σ. After Player 1's turn Player 0 moves deterministically until it is Player 1's turn again. The nodes reached at this point are marked "2", "3", and "4" because we are assuming that there were three different visible histories along those paths. At this point F_H consists of three sets. Again, below the nodes marked "2", "3", and "4" Player 1 moves according to the strategy σ. In this case there are three different sequences of moves that follow the strategy σ. We assume that below the nodes marked "3" M accepted the input during Player 1's moves. Hence, the branches below the nodes marked "3" terminate. Below the nodes marked "2" and "4" Player 1's moves are followed by Player 0's moves until it is Player 1's turn again. The nodes reached in this case are marked "5", "6", "7", and "8" indicating that there were four different visible histories along those branches. At this point F_H consists of four sets. Eventually, each branch terminates in an accepting configuration. This corresponds to F_H becoming empty.

For each initial visible history H, if the algorithm M' follows the strategy σ during Player 1's turn then it will successfully stop with F_H being empty. Thus x is accepted by M'.

Assume that x is accepted by M'. To show x is accepted by M we have to show how to construct a winning strategy for Player 1 for the solitaire automaton M given an accepting computation of M' on input x. We will not go into the details except to say that a computation tree of M similar to the one in Fig. 3 and a winning strategy can be constructed from an accepting computation of M'.

Claim 2. The algorithm M' runs in space $2^{cs(n)}$ for some c.

Proof. It suffices to show that every object in the algorithm has length bounded by $2^{cs(n)}$ for some c. Because $s(n) \geqslant \log n$ the length of each configuration is bounded by $2^{cs(n)}$ for some c. By assumption, M does not run forever during a turn of Player 0. Hence, each initial history, Player 0's initial series of moves, has no configuration repeated. Thus, each initial visible history H has length bounded by $2^{cs(n)}$ for some c. The number of members of S_H is bounded by $2^{cs(n)}$ for some c. Hence, the total length of S_H is bounded by $2^{cs(n)}$ for some c. The sum of the cardinalities of the sets in F_H is at most the cardinality of the initial set S_H, and hence, is bounded by $2^{cs(n)}$ for some c. So, the total length of F_H is bounded by $2^{cs(n)}$ for some c. There are a number of temporary values that are computed during the computation of a new F_H, for example, the configurations D_{ij} and E_{ij} and the subhistories H_{ij}. Each of these has length bounded by $2^{cs(n)}$ for some c and there are no more than

$2^{cs(n)}$ (for some c) of them that have to be maintained at any one time. Hence the length of all the temporary values combined is bounded by $2^{cs(n)}$ for some c. ∎

A solitaire automaton of complete information is really just a Π_2-Turing machine so that $CS\text{-}SPACE(s(n)) = \Pi_2\text{-}SPACE(s(n))$, finally establishing the space results of Table I.

4. Time Bounded Solitaire Automata

In this section we establish the characterization of time bounded solitaire automata.

THEOREM 2. (i) $PS\text{-}TIME(t(n)) = ATIME(t(n))$ *and* (ii) $ZS\text{-}TIME(t(n)) = \Sigma_2\text{-}TIME(t(n))$ *provided* $t(n) \geqslant n$ *and* $t(n)$ *is time constructible.*

Proof. Peterson and Reif, have shown under the hypothesis that $t(n) \geqslant n$ that $PG\text{-}TIME(t(n)) = ATIME(t(n))$ and $ZG\text{-}TIME(t(n)) = \Sigma_2\text{-}TIME(t(n))$ [4, 5]. Because trivially, $PS\text{-}TIME(t(n)) \subseteq PG\text{-}TIME(t(n))$ to obtain (i) it suffices to show that $ATIME(t(n)) \subseteq PS\text{-}TIME(t(n))$. Likewise, because $ZS\text{-}TIME(t(n)) \subseteq ZG\text{-}TIME(t(n))$ to obtain (ii) it suffices to show $\Sigma_2\text{-}TIME(t(n)) \subseteq ZS\text{-}TIME(t(n))$. The proof of (4) below is only a slight variant on the proof of Reif that $\Sigma_2\text{-}TIME(t(n)) \subseteq ZG\text{-}TIME(t(n))$ [4].

$$ATIME(t(n)) \subseteq PS\text{-}TIME(t(n)). \tag{3}$$

Let M be an alternating Turing machine running in time $t(n)$. Let d be the maximum number of distinct single moves possible from a configuration. To simulate M with a solitaire automaton Player 0 begins by writing a "choice sequence" for the universal moves of M on a private tape hidden from Player 1. Because $t(n)$ is time constructible the choice sequence, which is just a string $w \in \{1, \ldots, d\}^{t(n)}$, can be written down in time proportional to $t(n)$. Once w is written down Player 1 enters the game and the direct simulation of M begins. Player 1 directly simulates the existential moves of M. When a universal move of M is to be simulated one new character a of w is revealed by Player 0 and Player 0 deterministically makes the ath move from the current configuration of M. Player 1 wins if the simulation terminates with M in an accepting configuration. Player 1 has a winning strategy if and only if M accepts the input.

$$\Sigma_2\text{-}TIME(t(n)) \subseteq ZS\text{-}TIME(t(n)). \tag{4}$$

Let M be a Σ_2-Turing machine running in time $t(n)$. Let d and w be as above except the interpretation of the choice sequence w will be different in this case. Once Player 0 has written down w privately, Player 1 begins to directly simulate the initial existential moves of M. Once the first universal configuration of M is reached, Player 0 privately copies that configuration onto its private tapes and

begins to simulate privately the universal moves of M following the choice sequence w. So, Player 0 is behaving deterministically. Should the simulation lead to acceptance by M then Player 1 is declared the winner. Hence, Player 1 has a winning strategy if and only if M accepts the input. ▮

A solitaire automaton of complete information is really just a Π_2-Turing machine so that $CS\text{-}TIME(t(n)) = \Pi_2\text{-}TIME(t(n))$, finally establishing the time results of Table II.

5. Conclusion

We have presented some fundamental results concerning the power of solitaire automata. We hope that these results will prove useful in the future in uncovering the computational complexity of some natural problems.

There are a number of technical questions that seem worthy of attention. Are there constant factor speed-ups for time bounded solitaire automata? Considering that $PS\text{-}SPACE(s(n)) = ZG\text{-}SPACE(s(n))$, is there a direct simulation of partial information solitaire automata by zero information game automata and vice versa?

Acknowledgment

Thanks to Anne Condon for carefully reading and commenting on a preliminary version of the paper.

References

1. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer, Alternation, *J. Assoc. Comput. Mach.* **28**, No. 1 (1981), 114–133.
2. D. S. Johnson, The NP-completeness column: An ongoing guide, *J. Algorithms* **4**, No. 4 (1983), 397–411.
3. N. D. Jones, Blindfold games are harder than games with perfect information, *Bull. of the EATCS* **6** (1978), 4–7.
4. G. L. Peterson, and J. H. Reif, Multiple-person alternation, *in* "Proc. 20th Ann. Sympos. Found. Comput. Sci.," pp. 348–363, IEEE Computer Society, New York, 1979.
5. J. H. Reif, Universal games of incomplete information, *in* "Proc. 11th Ann. ACM Sympos. Theory of Computing," pp. 288–308, Assoc. for Comput. Mach., New York, 1979; The complexity of two-player games of incomplete information, *J. Comput. System Sci.* **29**, 274–301.
6. L. J. Stockmeyer, and A. K. Chandra, Provably difficult combinatorial games, *SIAM J. Comput.* **8** (1979), 151–173.