Multiple-Person Alternation

Gary L. Peterson and John H. Reif
Department of Computer Science
The University of Rochester
Rochester, NY 14627

Current Address of 2nd Author:
Aiken Computation Laboratory, Harvard University

Abstract

We generalize the alternation machines of Chandra, Kozen and Stockmeyer [1] and the private alternation machines of Reif [14] to model multiple person (team) games of incomplete information. The resulting classes of machines are "multiple person alternation machines".

The characterization of certain time and space bounded versions of these machines demonstrate interesting relationships between ordinary time and space hierarchies (Table 1).

Our results are applied to relative succintness and power questions of finite state machines and to complexity questions of parallel finite state machines. Other machine variants, including private alternating pushdown store automata and Markovian alternation machines, are discussed.

I. Introduction

The alternation machines of Chandra, Kozen and Stockmeyer [1] are natural extensions of non-deterministic machines to include both existential and universal choices. These choices then correspond to moves by opposing players in a two person game of perfect information. Reif [14] has introduced a new class of alternation machines which correspond to two player games of incomplete information. Both types of machines have provided insights into the relationships between time and space bounded computations. The generalization of alternation machines to correspond to multiple person (team) games of incomplete information is presented. These "multiple person alternation machines" are used to demonstrate relationships between time and space hierarchies (Table 1).

Machine Type	Basis	Bounds
Private Alternating (PA_k-Tm)	Space $S(n) \geq \log n$	$\bigcup_{c>0} DTIME(2^{2^{\cdot^{\cdot^{2^{cS(n)}}}}}{}_{k+1})$
	Time $T(n) \geq n^2$	$k=1$ $\subseteq DSPACE(T(n)) \supseteq NSPACE(\sqrt{T(n)})$
		$k\geq 2$ $\bigcup_{c>0} NTIME(2^{cT(n)})$
Blindfold Alternating (BA_k-Tm)	Space $S(n) \geq \log n$	$\bigcup_{c>0} NSPACE(2^{2^{\cdot^{\cdot^{2^{cS(n)}}}}}{}_{k})$
	Time $T(n) \geq n$	$k=1$ $\sum 2^{T(n)}$
		$k\geq 2$ $\bigcup_{c>0} NTIME(2^{cT(n)})$

Characterizations of space and time bounded multiple person alternation machines
Table 1

String	Language	Class of Languages
Concrete game	Game	Game type
A Kriegspiel endgame	Kriegspiel	Two-player, incomplete information

Analogies between games and languages, with example
(Kriegspiel = Blindfold Chess)
Table 2

Games are a simple model of computation. The fundamental question of a concrete game, the outcome problem, is closely related to the membership question of languages and machines. (The outcome problem is to determine if a given player (team) has a winning strategy over the opposing player (team).) A set of concrete games for which the outcome answer if positive form the (general) game. A game corresponds to a language. Similarly, types of games can be clustered into classes analogous to language classes. Table 2 presents these relationships, along with an example. It is not surprising that different types of games correspond to different modes of computation (Table 3). Of particular interest to note is that the least interesting game corresponds to our most natural notion of computation, while the most interesting game corresponds to a novel, abstract notion of computation.

Deterministic	Solitaire, perfect information, unique next move
Non-deterministic	Solitaire, perfect information, open next move
Alternation	Two-player, perfect information
Private Alternation	Two-player, incomplete information
Multiple-person Alternation	Multiple-player, incomplete information

Comparing Computation and Games
Table 3

The results obtained from studying space bounded multiple person alternation machines give a clearer understanding of hyper-exponential complexity classes. In particular, the elementary recursive languages are characterized by a class of linear space bounded multiple person alternation machines. This allows us to exhibit natural problems with super-complexity.

The basic perfect information alternation "game" problem of QBF [18],[1] has been very useful in demonstrating that natural games are PSPACE-Complete (or Hard) [7],[4],[11]. Similarly, we expect that natural problems in complexity of logical theories may be resolved via multiple person games of incomplete information and the characterizations we provide. We note that upper bounds for logical theories frequently involve games (e.g. [12]).

The basic outcome problem of two player games is: "Does Player 1 have a winning strategy over Player 2?" A winning strategy is defined to be a set of legal moves from current information into positions such that Player 1 will always be guaranteed to win regardless of Player 2's (legal) countermoves. For games of complete information, the current information need only be the current "board position". Hence Player 1's strategy is a mapping from possible positions to new positions. For two person games of incomplete information, not all of the current position may be visible to Player 1, so its strategy will depend on the visible part of the history of positions.

The generalization to more than two players is to have two teams, A and B. Team A will always be the team of preference. In all of the models we will present, Team B need only be one player, hence the size of Team A (k) will be the determinate of the difficulty of the game. The outcome question for team games is: "Do the players of Team A each have a strategy which together will defeat Team B under all circumstances?" Team games of perfect information are no more difficult to analyze than two player games since only one player is needed for each team. This player, since it can always see the visible positions of each player, can make the moves for its fellow players. However, we shall see that team games of incomplete information are much more difficult than two player games.

We take a very general view of games of incomplete information so as to include as many variants as possible. Some important facts about these games are listed below.

(1) Players need not take turns in a round-robin fashion. The rules will dictate whose turn is next.
(2) Players on the same team are allowed to communicate only in the manner allowed by the rules.
(3) A player can include, as part of its history

of positions, changes to visible portions of its position which occur between its turns.

(4) A player may not know who changes its visible portion of the position.

(5) A player may not know how many turns were taken by other players between its turns.

These facts are incorporated into our definitions of machines and acceptance. Of course, we do not allow the players to "cheat", that is no player can make a move which is based upon information it does not have, i.e. it cannot "peek" at other player's private positions.

Other variant views of multiple person games include limiting strategies to be based on current visible positions alone (which will be discussed), introducing randomness (as done by Reif [14]), and allowing players to compete individually to maximize their own "winnings" as in mathematical game theory [13]. The latter does not have a simple machine representation since the outcome problem is no longer a "yes/no" question.

"Reasonable games" [14] (which are similar to the standard linear games of Chandra and Stockmeyer [2]) are games whose positions do not grow with time and the legality of each move can be checked in polynomial time. From [2] we know that the outcome problem for two person reasonable games of perfect information is $DTIME(2^{cn})$-Complete (all complexity classes which use "c" are taken to be over the union for all c>0, completeness is with respect to log-space reductions unless mentioned otherwise). Reif [14] showed that the outcome problem for two person reasonable games of incomplete information is $DTIME(2^{2^{cn}})$-Complete. Our basic result is that the outcome problem for three or more person reasonable games of incomplete information is undecidable in general. However, when restricted to "hierarchical" games where the players of Team A are ordered so that Player 1 knows the positions of Players 2 through k, Player 2 knows the positions of Players 3 through k, etc., the outcome problem is now $DTIME(2^{2^{\cdot^{\cdot^{\cdot^{2^{cn}}}}}}_{k+1})$-Complete when Team A is of size k. These hierarchical games form the basis for the characterizations in Table 1.

For time bounded games, we do not obtain hierarchies of increasing complexity as we add players. Instead, the limit is reached with but two players on Team A. Blindfold games are defined to be games where no communication is allowed from Team B to players on Team A, i.e. those players are "blindfolded". (Communication to inform players of turns is still allowed.) For the hierarchical versions of blindfold games we obtain the space hierarchies indicated in Table 1. As before, time bounded games do not increase in difficulty with more players.

The situation for constant space (fsa) multiple person alternation machines is also explored. This leads directly into the analysis of complexity of decision problems for parallel finite state machines, analogous to Ladner [5]. The basic results are similar to our space bounded results, undecidable in general and decidable (but non-elementary) in the hierarchical case.

II. Definitions

II.1 Notations

Given two tuples $S=(s_1,s_2,\ldots,s_n)$ and $S'=(s'_1,s'_2,\ldots,s'_n)$, S differs from S' on the j'th element if $s_j \neq s'_j$, and S differs from S' only on the j'th element if in addition $s_i=s'_i$ for $i\neq j$.

Given a sequence of elements from a set: $S=s_1,s_2,\ldots,s_n$, let collapse(S) be the same sequence but with adjacent identical elements collapsed into a single element.

II.2 Game Definitions

We first informally define multiple person (team) games of incomplete information. The type of games presented here will be modelled by our multiple person alternation machines.

A multiple person (team) game consists of k+1 players, {0,1,2,...,k} divided into two teams, Team A and Team B, with Team A our team of preference. (One property of these games is that the size of Team A will be the determinate of the complexity of the game and therefore Team B need only be one player. We will then identify Players 1 through k as Team A and Player 0 as Team B.)

Each player will have private and common positions. A private position is not visible to any other player, while common positions are shared with one or more other players. (Positions common to two or more players are not necessarily visible to all other players.)

The winning positions for Team A is a subset of all the possible combined positions of all players.

Each player has a next move relation from its private and common positions to new positions. (A player can only change positions which are visible.)

There is a public common resource which will be used only to indicate which player is to take its turn. (However, it is not considered a part of the position so no player can use the value of the turn resource in its strategy. The reason for allowing this was pointed out in Section I.)

From a given initial set of positions for all players, we define a game tree to be the set of all reachable positions from those initial positions, with the arcs of the tree denoting a possible move by a player.

We define an equivalence relation on nodes: "$\tilde{}_i$". For two nodes in the game tree, n_1 and n_2, $n_1 \tilde{}_i n_2$ if the sequence of the private and common positions of Player i from the initial node to n_1 and the sequence from the initial node to n_2 are identical.

An accepting subtree is a finite subset of the game tree, where all nodes with no successors are winning for Team A; all nodes which are moves by players on Team B have all the successors of the original tree; all nodes which are moves by players on Team A have only one successor; and for any two nodes n_1 and n_2 which are moves for the same player i on Team A such that $n_1 \tilde{}_i n_2$, the positions private and common to the player of the successors of each node are identical.

A concrete game is a game plus a set of initial positions for all players. Team A has a winning strategy for a concrete game if there exists an accepting subtree whose initial node is the initial positions of the players.

II.3 Machine Definitions

We define a machine model that corresponds to multiple person (team) games of incomplete information. We will label the states of the machines by tuples, each element of the state tuple will denote information that can be read or written by players plus a turn indicator. In addition, the t tapes are allocated among the players so that they might store private information. Hence in representing a game, information private to a player will be denoted by state elements and tapes. Information common to two or more players will be denoted by state elements and tapes shared by those players.

The machine

Definition. A k+1 player, t tape, s state element, multiple person (team) alternation machine (a MPA_k-Tm) M is a tuple: $M = (T, I, b, Q_L, Q_G, q_o, Q_F, \delta)$

where:

T is the set of tape symbols,
I is the set of input symbols, $I \subseteq T$,
b is the blank symbol not in I,
Q_L is the set of local states,
Q_G is the set of global states of the form $(i, q_1, q_2, \ldots, q_s)$ with a turn element $i \in \{0, \ldots, k\}$ and state elements $q_j \in Q_L$,
q_o in Q_G is the initial state,
Q_F is the set of accepting states, $Q_F \subseteq Q_G$,
δ is the state transition function:

$$\delta : Q_G \times T^t \to P(Q_G \times T^t \times \{\text{Left}, \text{Right}, \text{Same}\}^t).$$

Basic machine moves

Definition. A configuration of M is a t+1 tuple of the form $(q, x_1, x_2, \ldots, x_t)$ with $q \in Q_G$ and each $x_j \in T^* \mid T^*$ where $\mid$ is a symbol not in T that denotes the head position of the j'th tape. The initial configuration of M on input X is $(q_o, \mid X, \mid b, \ldots, \mid b)$. An accepting configuration is any $(q, x_1, x_2, \ldots, x_t)$ such that $q \in Q_F$.

We will identify players with teams, and label them both by player number in the machine, and by type labels. Let player 0 be Team B and identified as the $\forall$-player. Players 1 through k are Team A, and each are labelled as the $\exists_i$-player.

Definition. A state $q \in Q_G$ is a play for player i, denoted $q \in \text{play}(i)$, if the first element of q is i. A configuration $C = (q, x_1, x_2, \ldots, x_t)$ is a play for player i, denoted by $C \in \text{play}(i)$, if $q \in \text{play}(i)$.

Operations on visible resources

The set of tapes visible to a given player are those which the player may either write upon or read. Reading a tape is defined to be the state transition function being different for two configurations that differ only on the symbol under the head of that tape. The set of state elements visible to a player are similarly defined. (We could have defined visible resources explicitly, in which case the following definition would be modified so that it ensures proper access to visible resources.)

Definition. The j'th tape is visible to player i, denoted $j \in vt(i)$, if there exists configurations $S = (q, t_1, t_2, \ldots, t_t) \in Q_G \times T^t$ and $S' = (q', t'_1, \ldots, t'_t) \in \delta(S)$ and $t_j \neq t'_j$ or $d_j \neq \text{Same}$;

or there exists $S = (q, t_1, \ldots, t_t)$, $S' = (q, t_1, \ldots, t'_j, \ldots, t_t) \in Q_G \times T^t$ (i.e. S and S' differ only on the j'th tape symbol) such that $\delta(S) \neq \delta(S')$.

The j'th state element is visible to player i, denoted $j \in vs(i)$ if there exists $q_1, q_2 \in Q_g$ which differ only on the j'th state element, and $t_1, t_2, \ldots, t_t$ in T such that

$$\delta(q_1, t_1, \ldots, t_t) \neq \delta(q_2, t_1, \ldots, t_t)$$

or there exists $S = (q, t_1, \ldots, t_t) \in Q_G \times T^t$ and $S' = (q', t'_1, \ldots t'_t, d_1, \ldots, d_t) \in \delta(S)$ where q and q' differ on the j'th state element.

A multiple person alternation machine appears to be quite similar to a multiple process automata. The above definition is nearly the same definition for read/write access to shared variables in a parallel system, e.g. [9] and elsewhere.

Definition. A configuration $C'=(q',x'_1,x'_2,\ldots,x'_t)$ follows from a configuration $C=(q,x_1,x_2,\ldots,x_t)$, denoted $C=>C'$, if there exists $d_1,d_2,\ldots,d_t \in \{Left,Right,Same\}$ such that $(q',s'_1,\ldots,s'_t,d_1,\ldots,d_t) \in \delta(q,s_1,s_2,\ldots,s_t)$ (where s_i is the symbol under the i'th tape head, i.e. $x_i=u_i\uparrow s_i w_i$, $u_i,w_i \in T^*$ and $s_i \in T$, similarly for s'_i's). Also, for each x_i,x'_i one of the following is true:

(a) d_i=Right then for i) $x_i=w\uparrow s_i u$, $x'_i=ws'_i\uparrow u$, or ii) $x_i=w\uparrow s_i$, $x'_i=ws'_i\uparrow b$.

(b) d_i=Same then for $x_i=w\uparrow s_i u$, $x'_i=w\uparrow s'_i u$.

(c) d_i=Left then for i) $x_i=us\uparrow s_i w$, $x'_i=u\uparrow ss'_i w$, or ii) $x_i=\uparrow s_i w$, $x'_i=\uparrow bs'_i w$.

Furthermore, for all x_1, x'_1, $s_1=s'_1$, i.e. the input head is read-only.

We define the visible part of a position of a player to be those state elements and tapes that are visible to it. It is the sequence of visible positions that is the only information available to a player in making a move.

Definition. The visible position of a configuration C of player i, denoted visible(C,i), is a s+t tuple $(q_1,\ldots q_s,x_1,\ldots,x_t)$ where q_j is the j'th state element of the state of C if $j \in vs(i)$ and "-" otherwise, and x_j is the j'th tape configuration of C if $j \in vt(i)$ and "-" otherwise.

Definition. The visible position of a sequence of configurations $S=C_1,C_2,\ldots,C_m$ to player i, denoted visible(S,i), is collapse(visible(C_1,i), visible(C_2,i), ...,visible(C_m,i)).

Acceptance

We are at last able to define acceptance of a string. The above formalisms allow us to exclude computation trees which do not correspond to legal sequences of moves by the players.

Definition. $S=C_0,C_1,\ldots,C_m$ is an accepting sequence of configurations for M on input X if C_0 is the initial configuration of M on X, C_m is an accepting configuration, and for all i, $0\leq i\leq m-1$, $C_i=>C_{i+1}$.

Definition. T={S|S is an accepting sequence of configurations of M on X} (and finite) is an accepting subtree of M on X if:

(a) For all $S=C_0,C_1,\ldots,C_n,\ldots,C_m$ in T, C_n not accepting and in play(0), then for all C'_{n+1} such that $C_n=>C'_{n+1}$, $C_0,C_1,\ldots,C_n,C'_{n+1}$ is a prefix for a sequence in T. (I.e. the $\forall$-player can make all moves.)

(b) For all proper prefixes of sequences in T, $S_p=C_0,C_1,\ldots,C_n$, there is exactly one C_{n+1}, $C_n=>C_{n+1}$, such that $C_0,C_1,\ldots,C_n,C_{n+1}$ is a prefix of a sequence in T if $C_n \in play(i)$, $i>0$. (I.e. all $\exists$-players can make only one move.)

(c) For all proper prefixes of sequences in T, $S_p=C_0,\ldots,C_n$ and $S'_p=C'_0,\ldots,C'_{n'}$, where C_n and $C'_{n'}$ are in play(i), $i>0$, and $visible(S_p,i)=visible(S'_p,i)$ then for all prefixes S_p,C_{n+1} and $S'_p,C'_{n'+1}$ of sequences in T, $visible(C_{n+1},i)=visible(C'_{n'+1},i)$. (I.e. all $\exists$- player's strategies must depend on visible information alone.)

Definition. M accepts X if there exists an accepting subtree for M on X. Let L(M) denote the set of strings accepted by M.

Private and Blind Machines

A MPA_1-Tm corresponds to PA-Tm's in [14]. A MPA_1-Tm with vs(1)=vs(0) and vt(1)=vt(0) corresponds to an A-Tm [2] (not the newer version which includes "not" operations [1]). A MPA_1-Tm with unique next moves for the $\forall$-player is a non-deterministic Tm. If all players have unique next moves, then it is a deterministic Tm. Hence it is easy to see that MPA_k-Tm's accept the r.e. languages since they include ordinary Tm's, similarly acceptance by MPA_k-Tm's is r.e. since one can enumerate all possible accepting subtrees and check recursively whether each tree is a true accepting subtree.

Our primary objective in introducing MPA_k-Tm's is to derive a characterization of the full elementary recursive hierarchy. However, we discovered that even MPA_2-Tms with constant space accept all r.e. languages. This resulted in variant definitions which give a more natural extension of the space and time results of [14].

Definition. A k+1 player private alternation machine (PA_k-Tm) is a MPA_k-Tm with $vs(i)\leq vs(i-1)$ and $vt(i)\leq vt(i-1)$ for all i, $2\leq i\leq k$. Hence there is a hierarchical ordering of the $\exists$-players.

Definition. A k+1 player blindfold alternation machine (BA_k-Tm) is a PA_k-Tm which for all i, $1\leq i\leq k$, and configurations C, $C\varepsilon play(i)$, there does not exist C', C=>C', such that for some $j\varepsilon vs(i)$ the state of C differs from the state of C' in the j'th state element or for some $j\varepsilon vt(i)$ the j'th tape configuration of C differs from the j'th tape configuration of C'. (I.e. the $\forall$-player cannot communicate to any $\exists$-player.)

It is clear from the definitions that a PA_1-Tm is also a MPA_1-Tm (and therefore also a PA-Tm [14]). Similarly, a BA_1-Tm is also a BA-Tm [14].

II.4 Time and Space Definitions

Informally we define the space complexity of a machine on an input to be the maximum total non-input tape used in any configuration in a given accepting subtree. Time is taken to be the depth of the accepting subtree. These definitions are now formally applied to our machine definition. Note that these definitions are exactly the same as those of a non-deterministic machine when the latter's accepting subtree being a single path is taken into account.

Definition. A configuration $C=(q,x_1,x_2,\ldots,x_t)$ has space complexity S if $|x_2|+|x_3|+\ldots+|x_t|\leq S$. A machine M accepts X in space S if there exists an accepting subtree T such that all configurations C in T have space complexity no greater than S.

Definition. A language L is in MPA_k-SPACE(s(n)) (PA_k, BA_k resp.) if there is a MPA_k-Tm (PA_k, BA_k resp.) M such that L(M)=L and M accepts all X in L in space $s(|X|)$.

Definition. A machine M accepts X in time t if there is an accepting subtree T of depth $\leq t$, i.e. every sequence in T is at most t+1 configurations long.

Definition. A language L is in MPA_k-TIME(t(n)) (PA_k, BA_k resp.) if there is a MPA_k-Tm (PA_k, BA_k resp.) M such that L(M)=L and M accepts all X in L in time $t(|X|)$.

The main results of our paper, as given in Table 1, can now be given. The next section proves the results of the space bounded machines, and section IV gives the time bounded characterizations.

III. Space Bounded Machines

III.1 Two person games revisited

In Reif [14], the complexity of space bounded machines that represent two person games of incomplete information was derived using alternation machine complexity. In this paper, we will directly derive our results by comparison to ordinary Tm's. It will be important that we re-derive the characterization of two person games first. The multiple person games will then follow as nearly trivial extensions of these new proofs. It should be obvious that the PA_1-Tm's (=MPA_1-Tm's) and BA_1-Tm's of this paper are equivalent to the PA-Tm's and BA-Tm's of the previous paper. The theorem for space bounded machines is now restated.

Theorem 1 (Reif). For constructable $S(n)\geq \log n$, $PA_1\text{-SPACE}(S(n))=\text{DTIME}(2^{2^{cS(n)}})$ and $BA_1\text{-SPACE}(S(n))=\text{DSPACE}(2^{cS(n)})$. (Recall that unions over all c>0 are taken.)

An important construction. In order to derive the lower bounds (that is the classes contained by space bounded PA_1/BA_1-Tm's) we first point out a crucial method for simulating a task that deterministically requires a large amount of space with a game that requires far less space. The most important of these tasks is counting from 0 to $2^{2^n}-1$. This deterministically requires 2^n bits, but we will present a game which simulates the task with only n bits. The $\exists$-player in the game will be sending (via a common state element) a sequence of groups of characters corresponding to counting on 2^n bits, a character at a time, i.e. $\#0^{2^n}\#0^{2^n-1}1\#\ldots\#1^{2^n}\#$ and halts. The $\forall$-player's job is to find a flaw in the $\exists$-player's sequence as it comes in. The $\exists$-player can win only if the $\forall$-player fails to find a flaw. Given that the $\forall$-player has only n tape cells, the player can only store a small fraction of a number and therefore

must use a different strategy. However, what information the ∀-player decides to save can be kept hidden from the ∃-player. The ∀-player's counter-strategy is therefore to indeterministically choose to do one of the following verifications:

(1) The first group is in 0^* and the last is in 1^*.
(2) Choose any group and verify (secretly) that it is length 2^n (using n bits).
(3) Choose any bit of any group, remember it and its location (using n bits) and check it against the same position in the next group. (This requires a simple check of the following bits for a possible carry in.)

Since the what and where of the ∀-player's checking is secret, in order to win, the ∃-player must give the first group as 0^{2^n}, the second as $0^{2^n-1}1$, etc. If the ∀-player's tape were visible to the ∃-player, then the ∃-player could cheat and give too short groups, wrong additions, etc. Note that the above game is blindfold, the ∃-player receives only turn information. If the ∀-player can send information to the ∃-player, then the technique generalizes to counting, i.e. the ∀-player can ask the ∃-player to count both up and down.

We now apply the above technique to our lower bound proofs. We assume the reader is familiar with the corresponding lower bound proof of [1].

Lemma 1. $DSPACE(2^{S(n)}) \subseteq BA_1\text{-}SPACE(O(S(n)))$ and $DTIME(2^{2^{S(n)}}) \subseteq PA_1\text{-}SPACE(O(S(n)))$, for constructable $S(n) \geq \log n$.

Proof: (a) For BA_1-Tm's, we modify the above counting technique so that the ∃-player is sending a sequence of configurations $(C_0, C_1, \ldots, C_m)$ of a given N-Tm which uses tape at most $2^{S(n)}$. (The conversion to a D-Tm affects only the constant in the exponent. Later on, the fact we are simulating a N-Tm will become important.) The BA_1-Tm will accept the input iff there is an accepting sequence of configurations for the N-Tm on that input, i.e. the input is also accepted by the N-Tm. The ∀-player's counter-strategy is much the same, it checks the first against the input, the last as accepting, or the length of any configuration, or that corresponding tape cells are the same except for the head motion, etc. Since the ∀-player needs to only write down the length of a configuration, it will need only space O(S(n)).
(b) For PA_1-Tm's, recall that the [1] proof uses only a constant amount of space plus two counters. Hence we can replace their counters by the game counters, and instead of simulating a $DTIME(2^{S(n)})$ Tm with O(S(n)) tape, we can simulate a $DTIME(2^{2^{S(n)}})$ Tm with the same amount. The details are easy to fill in. (Note that the ∃-player is both counting and making the existential moves of the [1] proof.) []

We now turn to the upper bound proofs. An important fact to realize about games of incomplete information is that it is not so much the history directly that should be used in forming a strategy, but the indirect information it gives as to the private position of the opponent. Hence in a two person game, if the set of possible private positions of the opponent and the current visible positions repeats, then that player can cause the cycle to recur indefinitely. Or more importantly, the ∃-player must be able to find a win before such a cycle occurs or it can never win. This leads directly into a variation of the subset construction proof for fsa's.

Lemma 2. $PA_1\text{-}SPACE(S(n)) \subseteq DTIME(2^{2^{cS(n)}})$ and $BA_1\text{-}SPACE(S(n)) \subseteq DSPACE(2^{cS(n)})$, for constructable $S(n) \geq \log n$.

Proof: Using the above idea, we will consider a modified game tree where each node corresponds to a visible configuration of the ∃-player and the set of possible private configurations of the ∀-player based upon the history. The nodes are labeled with a tuple $(C_\exists, SC_\forall)$ where $C_\exists$ is a configuration visible to the ∃-player, and $SC_\forall$ is a set of possible configurations private to the ∀-player. This modified game tree is of size $2^{2^{O(S(n))}}$. (There are $2^{O(S(n))}$ possible private configurations of the ∀-player and therefore the power set of that set is an exponential higher.) An arc exists from each node to all nodes which are the result of one move from a reconstructed position which includes the ∃-player position and one of the ∀-player positions.
(a) For the PA_1-Tm's, the modified game tree will be explored for an accepting subtree in time $2^{2^{O(S(n))}}$. A series of sweeps is made through the tree (assumed to be written completely down on a tape) marking accepting nodes and their predecessors until the initial node is marked (accept) or no new nodes are marked (reject). Initially, all nodes that have the property that all reconstructed configurations are accepting are marked. Then a pass is made marking all node whose reconstructed configurations for the ∃-player have a corresponding arc leading to a marked node and configurations for the ∀-player have all corresponding arcs leading to a marked node. There will be at most $2^{2^{O(S(n))}}$ passes of duration $2^{2^{O(S(n))}}$ each, giving the result.
(b) For the BA_1-Tm result, the tree is too large to

write down entirely, however the path through the tree for the ∃-player does not branch since there is no incoming information, it can be effectively explored by a N-Tm using a non-deterministic search. (Again we note the the non-determinism affects only a constant in the exponent, by applying Savitch's result [16].) The N-Tm explores the tree by guessing the move of the BA_1-Tm when it is at a purely ∃-player node (all reconstructed configurations are for the ∃-player) and for mixed or purely ∀-player nodes, the ∀-player configurations are pushed forward until a ∃-player configuration is reached (or a loop occurs, and that part is dropped). The N-Tm only needs to store information on the order of the size of label of a node, so the whole algorithm uses space $2^{2^{O(S(n))}}$. []

Note that the last strategy cannot work for PA_1-Tm's since information sent to the ∃-player may differ, and each corresponding path must be explored, requiring a stack, etc. The results of section III.3 will follow closely the above techniques.

III.2 Space and Multiple Person Machines

Since a MPA_1-Tm is by definition also a PA_1-Tm, from the previous section (and [14]) we have:

Theorem 2. $MPA_1\text{-SPACE}(S(n))=\text{DTIME}(2^{2^{cS(n)}})$, for constructable $S(n)\geq \log n$.

For more than one player on Team A, we find that our machines are very powerful, indeed space bounds do not limit their power.

Theorem 3. $MPA_2\text{-SPACE}(\text{constant})=\text{r.e. languages}$.

Proof: Given a Tm M, we will construct a MPA_2-Tm M' which accepts the same set of strings. The game of the machine will be based on having the ∃-players each find a sequence of configurations of M on an input that leads to acceptance. Hence, each ∃-player will give to the ∀-player on request the next character of its sequence of configurations (secretly from the other). The configurations will in the form: $\#C_0\#C_1\#\ldots\#C_m\#$, where C_0 is the initial configuration of M on the input and C_m is an accepting configuration.

The ∀-player will choose to verify the sequences in one of the following ways:

(1) Check one of the first configurations against the input and make sure it is in the correct form. Check the last configuration for accepting state. (This implies that the input tape is private to the ∀-player.)

(2) Check that the players are giving the same sequence by alternating turns between them.

(3) Run one of the players ahead to the next # and check that player's C_i against the others C_{i-1} for proper head motion, change of state, tape cells change or remain the same according to the transition rules, etc.

Note that the ∀-player is only dealing with incoming information, so it does not have to store anything on a tape, it can remember all information in its private state. (That all languages accepted are r.e. has already been pointed out.) []

In order to eliminate, by restriction, the over-generality of MPA_k-Tms, we considered several interesting variants. For instance, it might seem that making the input tape public to all would help the problem, but it is trivial to show the following corollary.

Corollary 1. Assuming the input tape is public to all players, $MPA_2\text{-SPACE}(\log n)=\text{r.e. languages}$.

Membership for MPA_2-SPACE(constant) Tms is still undecidable (via the halting problem on blank tape).

III.3 Space and Private and Blindfold Machines

The PA_k-Tm's and BA_k-Tm's were specifically designed to avoid the non-recursiveness of space bounded computation of the previous section. They are clearly (in terms of space characterizations) natural extensions of A-Tm's, PA-Tm's, and BA-Tm's. We state what can be considered the main result of the paper.

Theorem 4. $PA_k\text{-SPACE}(S(n))=\text{DTIME}(2^{2^{\cdot^{\cdot^{2^{cS(n)}}}}}\Big\}k+1)$

and $BA_k\text{-SPACE}(S(n))=\text{DSPACE}(2^{2^{\cdot^{\cdot^{2^{cS(n)}}}}}\Big\}k)$, for constructable $S(n)\geq \log n$.

Proof (lower bounds): The lower bound proofs extend the ideas of section III.1 to counting to very high values with n bits. Using k players and n bits we can count up to $2^{2^{\cdot^{\cdot^{2^{n}}}}}\Big\}k+1$. Note that the earlier technique used n bits for counting alone (for lengths of numbers, positions within numbers, etc.) and furthermore we need only to count up. Hence we can recursively apply the technique. The $\exists_k$-player will supply the count sequence on $2^{2^{\cdot^{\cdot^{2^{n}}}}}\Big\}k$ bits, which will be checked

using the count sequence on $2^{2^{\cdot^{\cdot^{2^n}}}}\Big]_{k-1}$ bits being supplied by the $\exists_{k-1}$-player. The $\forall$-player can also decide to check that count against the sequence being supplied by the $\exists_{k-2}$-player, etc. Note that these are hierarchical machines, implying that the $\exists_{k-1}$-player can know that it is being used to check the $\exists_k$-player's sequence, but doesn't know if it is being checked by the $\exists_{k-2}$-player. If the order were reversed, then the $\exists_1$-player would know where it was being checked and could cheat. Note also that these are blindfolded players, they only receive turn information. If the $\exists$-players were not blind, then the $\forall$-player can instruct the $\exists_k$-player to count both up and down. So again, the [1] technique can be adapted, using the new very large counters to simulate a $\mathrm{DTIME}(2^{2^{\cdot^{\cdot^{2^{S(n)}}}}}\Big]_{k+1})$ Tm with a space $O(S(n))$ PA_k-Tm. For the blindfold case, the $\exists_k$-player sends out the configuration sequences of size $2^{2^{\cdot^{\cdot^{2^{S(n)}}}}}\Big]_{k}$, with the $\exists_{k-1}$ through $\exists_1$-players sending out the counts. The $\forall$-player uses the counts to check other counts and uses the largest to check the length of the configurations, etc. []

Proof (upper bounds): Again we extend the constructions of section III.1. For PA_k-Tm's, we extend the "subset construction" idea to form configurations that represent serially eliminating players from the game. The $\exists_1$-player knows about all other players except the $\forall$-player, therefore we create intermediate configurations which combine the current state of the $\exists_1$-player and the set of possible states of the $\forall$-player. There will be at most $2^{2^{O(S(n))}}$ of these configurations. In the next step, we create new configurations which represent the $\exists_2$-player's private state and the set of possible states of the $\exists_1$-player (which includes the set of possible states of the $\forall$-player). There will be at most $2^{2^{2^{O(S(n))}}}$ of these configurations. The process is repeated until we have only the $\exists_k$-player's private state and the set of possible states of the $\exists_{k-1}$-player which includes the set of possible states of the $\exists_{k-2}$-player, etc. A modified game tree can now be built, and will be of size $2^{2^{\cdot^{\cdot^{2^{O(S(n))}}}}}\Big]_{k+1}$. It can be search as before in time the order of its size by a D-Tm. For BA_k-Tm's, the game tree can be explored in space proportional to the size of a configuration by a N-Tm. Again, the conversion to determinism does not affect the order of growth. []

The above proofs bring to mind methods of Stockmeyer and Meyer [17] such as "rulers" to derive results in super-exponential complexity.

It is a little surprising that such natural extensions to PA-Tm's and BA-Tm's should exist. The well ordering of the generated hierarchies indicates that space and time must complement each other very well. The reader might be wondering about the precise relation between PA_k-Tm's and BA_k-Tm's (any BA_k-Tm is by definition a PA_k-Tm). We can show that for any PA_{k-1}-Tm there is an equivalent BA_k-Tm, which uses the same space and a little more time. We leave it as a puzzle (=solitaire game of incomplete information) to the reader.

III.4 An Example

We give an intuition of the inherent complexities of natural looking games by presenting a concrete example of a simple game. We call our game "TEAM-PEEK". The game is an extension of the games of PEEK [2] and PRIVATE-PEEK [14]. Consider a box containing a stack of slide out plates. The plates slide partially in one direction and may be either "in" or "out". Holes have been punched in some order in the plates. The teams stand on opposite sides of the box, and are separated by curtains as in voting booths. A player sees in front of him/her a subset of the plates with knobs. We can assume here that the play goes round-robin style in some order. During its turn, a player can decide to move any or all of the plates visible to it either in or out. All players know the initial positions of the plates and the layout of the holes. A team wins, if at the end of a turn by one of its players, there is a sequence of holes lined up through the box allowing one to "peek" from top to bottom. If both teams consist of single players, then it is PRIVATE-PEEK, if both players can also see all plates, then it is PEEK. We now cite the obvious complexities of the outcome problem for these versions of PEEK. (All are based on the plates representing clauses of formulas, and so on. Our results follow simply from [2].)

Theorem 5. PEEK is DTIME(2^{cn})-Complete [2]. PRIVATE-PEEK is DTIME($2^{2^{cn}}$)-Complete [14]. TEAM-PEEK is undecidable with two or more players on Team A.

If the Team A side is restructured so that player 1 is looking over the shoulders of all the other players, player 2 is looking over the shoulders of player 3 through player k (but may not see any of player 1's plates), etc. we get Hierarchical TEAM-PEEK.

Theorem 6. Hierarchical TEAM-PEEK is DTIME($2^{2^{\cdot^{\cdot^{\cdot^{2^{cn}}}}}}$ $\Big]$ $k+1$)-Complete when Team A has k players.

As a general rule, we can generate games which for unbounded number of players are non-elementary to decide. Hence we have natural problems with both elementary and non-elementary complexity (in addition to undecidable).

IV. Time Bounded Machines

IV.1 More two person games

Unlike space bounded computations, time bounded multiple person alternation games do not form a hierarchy. There is a clear division between two person games and three or more person games. From [14] we know the following theorem.

Theorem 7 (Reif). $BA_1\text{-TIME}(T(n))=\Sigma_2^{T(n)}$, for countable $T(n)\geq n$.

($\Sigma_2^{T(n))}$ is the notation for the second existential step in the T(n) time based hierarchy [8].)

The corresponding theorem in [14] for PA_1-Tm's (=MPA_1-Tm's) is in error. We present the correction and sketch its proof.

Theorem 8. $PA_1\text{-TIME}(T(n)) = MPA_1\text{-TIME}(T(n)) = A\text{-TIME}(T(n))$.

Proof: By virtue of definition, any A-Tm (without "not's" as in [2]) is also a PA_1-Tm. So we only need to prove that any PA_1-Tm can be converted into an A-Tm which has the same order of time complexity. We create a two person game of complete information (an A-Tm) which simulates the two person game of incomplete information (the PA_1-Tm). In the first stage of the game, the $\forall$-player of the A-Tm will make at any time, any move that could ever be possibly made by the simulated $\forall$-player of the PA_1-Tm, legal or otherwise. (By move we mean any action visible to the $\exists$-player, it performs no private actions.) The $\exists$-player of the A-Tm responds to every move by the $\forall$-Tm as if it were simulating the $\exists$-player of the PA_1-Tm. Both players write down (in public of course) the moves they make. This stage of the game takes order T(n) time. In the next stage of the game, the $\forall$-player checks whether the simulated $\forall$-player of the PA_1-Tm could have really made that sequence of moves. The $\exists$-player is no longer used so this checking can be done in public. The second phase takes order T(n) time. If the $\exists$-player has a winning strategy for a legal sequence of moves by the $\forall$-player, then it will be used in the simulation for that sequence. If it has no winning strategy for a legal sequence of moves, then it loses. The $\forall$-player loses all plays of games where it takes an illegal sequence of moves. []

From [2] and [1] we have the following final result.

Corollary 2. For countable $T(n)\geq n$, $PA_1\text{-TIME}(T(n))\subseteq DSPACE(T(n))$ and $NSPACE(T(n))\subseteq PA_1\text{-TIME}(T(n)^2)$.

The reader is referred to [10] for the case of lower bounds for PA_1-Tm's (A-Tm's) with less than n^2 time.

IV.2 An important formula game: DQBF

It turns out, that for all our machines with three or more players, the time bounded complexity results are identical. We found an analog to QBF ([18], [2]) extended to multiple person games to be particularly helpful. QBF (quantified Boolean formulas) were used in [2], [1] to demonstrate the complexity of time bounded A-Tm's. We present DQBF (dependency QBF) to analyze time bounded MPA_k-Tm's.

Consider the following typical QBF formula (F is some function over Boolean variables, the captitalized variables denote tuples of Boolean variables):

$$\forall X_1 \exists Y_1 \forall X_2 \exists Y_2 \; F(X_1,X_2,Y_1,Y_2)$$

Note that selecting Y_1 depends on the value of X_1 but not on X_2. Selecting Y_2 depends on both X_1 and X_2. We can denote these cases by $Y_1(X_1)$ and $Y_2(X_1,X_2)$. The order of the variables can now be changed to:

$$\forall X_1, X_2 \exists Y_1(X_1), Y_2(X_1,X_2) \; F(X_1,X_2,Y_1,Y_2)$$

We call formulas that are in the above form (with universal variables, existential variables with dependencies, followed by a Boolean function) DQBF formulas. While all QBF formulas have a succinct (i.e. no more than squaring in size) DQBF formula, the reverse is apparently not true. Consider:

$$\forall X_1, X_2 \exists Y_1(X_1), Y_2(X_2) \quad F(X_1,X_2,Y_1,Y_2)$$

We will show later that this formula is not likely to have a succinct QBF formula. We note that all DQBF formulas can be put into a functional form, as we can do with the last formula.

$$\exists G_1, G_2 \forall X_1, X_2 \quad F(X_1,X_2,G_1(X_1),G_2(X_2))$$

However, the size of G_1 and G_2 can be exponential in the size of their inputs, hence this is not nearly as succinct.

We are able to show that DQBF formula are more succinct than QBF formula if we assume there is a language in NEXPTIME which is not in PSPACE. (NEXPTIME=NTIME(2^{cn}).) Recall that validity of QBF formulas is PSPACE-Complete [18]. We now present the corresponding result for DQBF.

Theorem 9. DQBF validity is NEXPTIME-Complete.
Proof: (a) DQBF validity is in NEXPTIME. Our proof is based on the above functional interpretation of DQBF formula. In the first phase, the "truth table" (i.e. functional dependencies) for all the existential variables are guessed. This can take no more than non-deterministic exponential time. In the second phase, the function is evaluated for all assignments of the universal variables, looking up in the "truth tables" the values of the existential variables to use in the evaluation. Each evaluation takes at most exponential time, and there at most exponential evaluations. The total time is then non-deterministic exponential.
(b) Given a single tape, non-deterministic exponential time bounded N-Tm, and an input string, we construct (in log-space) an O(n) length DQBF formula which is valid iff the input is accepted by the N-Tm. Hence, DQBF is NEXPTIME-Hard. The DQBF formula will have the general form:

$$\forall T_1, T_2 \exists Y_1(T_1), Y_2(T_2) \quad F(T_1,T_2,Y_1,Y_2)$$

This formula corresponds to a three person game where the universal player asks each of the existential players for a complete description of the activities of the N-Tm at two times (T_1 and T_2) on a accepting sequence of moves on the input of their choosing. T_1 and T_2 are each sets of O(n) Boolean variables that can therefore represent up to exponential time. The Y_i's are tuples of variables giving the information: (old-state$_i$, new-state$_i$, old-symbol$_i$, new-symbol$_i$, position$_i$, last-time$_i$, motion$_i$). This information is suppose to reflect a complete description of the activities of the head at that time if the N-Tm is to accept the input. old-state$_i$ represents the previous state of the machine and new-state$_i$ represents the new state. Similarly old-symbol$_i$ and new-symbol$_i$. All of the above need a constant number of variables to represent. position$_i$ is the head position and last-time$_i$ is the last time the head was at that position (0 for the first visit). Both need O(n) variables to represent. motion$_i$ is the direction the head takes at that time (-1=left, 0=same, 1=right).
The universal checking of the -player is represented by the set of conjunctive clauses forming the function F:

(a) If $T_1=T_2$ then $Y_1=Y_2$ and the state, symbol, motion transition is allowed. (Both players choose the same non-deterministic set of moves.)
(b) If $T_1=1$, then old-state$_1$ is the initial state, and position$_1$=1, etc. (The players start off correctly)
(c) If position$_1$=position$_2$ and $T_1>T_2$ then last-time$_1 \leq T_2$, and if $T_1=T_2$ then old-symbol$_1$=new-symbol$_2$. (Whenever the two players give the same head location, the time given by the first for the last time at that position cannot be any earlier than the time for the second. Also, if the times are equal, the symbol written by the second is the symbol read by the first.)
(d) If last-time$_1$=0 then old-symbol$_1$ is the position$_1$'th input symbol if position$_1 \leq$n and blank otherwise. (The first visit to a cell gives the old symbol according to the initial contents of the tape.)
(e) If $T_1+1=T_2$ then position$_2$=position$_1$+motion$_1$, and old-state$_2$=new-state$_1$. (The motion of the machine is proper and the state is preserved.)
(f) If $T_1=T(n)$ (the exponential time limit) then new-state$_1$ is accepting.

Hence, we are existentially choosing the complete operation of the N-Tm without writing it all down, and it is being universally checked. []

IV.3 General time results

In a game with two existential players and a universal player, we can simulate the evaluation of the lower bound formula in the DQBF proof. The universal player first sends to the $\exists_1$-player a time, and gets the Y_1 tuple as a response. Similarly for the $\exists_2$-player. The $\forall$-player then checks the responses as in the lower bound proof. Hence in T(n) time we can verify acceptance of a $NTIME(2^{cT(n)})$ N-Tm. Note that this game is also hierarchical since the $\exists_1$-player is asked first and doesn't know what the $\exists_2$-player will be asked.

Corollary 3. $NTIME(2^{cT(n)})$ is contained in PA_k-TIME(T(n)) (MPA_k-TIME(T(n))) for countable $T(n)\geq n$ and $k\geq 2$.

For BA_2-Tm's, there is a tricky way for the $\forall$-player to get around the blindfold rule and send information to the $\exists_1$-player (and get its response) and then to the $\exists_2$-player. The $\exists_2$-player is initially cycling through two states on its first 2T(n) moves. The $\forall$-player can advance the $\exists_2$-player to one of those states, advance the $\exists_1$-player, which will see the $\exists_2$-player's state and a bit has been sent. This process is repeated until all T(n) bits have been sent. The $\exists_1$-player now sends the Y_1 tuple to the $\forall$-player. The $\forall$-player again sends T(n) more pieces of information to the $\exists_1$-player who relays it back to the $\exists_2$-player. The $\exists_2$-player does not know when the $\exists_1$-player makes its move so it doesn't know what time was sent to the $\exists_1$-player. Similarly, the $\exists_1$-player has already sent its response when the $\exists_2$-player's time is received.

Corollary 4. BA_k-TIME(T(n)) contains $NTIME(2^{cT(n)})$, for countable $T(n)\geq n$ and $k\geq 2$.

Since a MPA_k-Tm is the most powerful of our machines, we need only show that MPA_k-TIME(T(n)) is contained in $NTIME(2^{cT(n)})$ to complete our results.

Lemma 3. MPA_k-TIME(T(n)) is contained in $NTIME(2^{cT(n)})$, for countable $T(n)\geq n$ and $k\geq 2$.

Proof: We follow in general terms the upper bound proof for DQBF. We first modify the machines so that the $\exists$-players' strategies need only depend on current visible information. This is accomplished by having each player record on extra tapes the complete history of visible information up to that point. This will take O(T(n)) time per player. (The machine is now Markov, see section V.4.) We simulate the running of this machine by first guessing a correct strategy for each player. This is a "truth table" arrangement where what each $\exists$-player is suppose to do in each situation is recorded. This takes (non-deterministic) time $2^{O(T(n))}$. Each player's strategy is written on a separate tape in time order (time=length of history). The machine is now simulated deterministicly. The $\forall$-player is represented by a set of states as in Section III. (A set of configurations for the $\forall$-player takes no more than $2^{O(T(n))}$ space to write down.) The $\exists$-players' moves are simulated by looking up on the tapes the appropriate move in each situation. Each simulated move will take exponential (in T(n)) time to make for either the $\forall$-player or the $\exists$-players. There are T(n) simulated moves, hence the total time is (non-deterministic) exponential in T(n). []

Corollary 5.

$MPA_k/PA_k/BA_k$-TIME(T(n))=$NTIME(2^{cT(n)})$, for countable $T(n)\geq n$ and $k\geq 2$.

The general conclusion for multiple person alternation machines is that space is a much more powerful resource than time.

V. Other Machines

V.1 Finite state machines

If we were to simply define a PA_k-fsa (or BA_k-fsa) to be a constant space PA_k-Tm (BA_k-Tm), we would not have regular languages. If the input tape (since all players know the input initially, this means the input head position) were private to the $\forall$-player, it could be used as a size n (upward only) counter. Hence the techniques of Section III apply all over again so that we can count up to large numbers. A game with k $\exists$-players can count up to $2^{\left[2^{\cdot^{\cdot^{2^n}}}\right]^k}$ with a size n counter. (The $\forall$-player can only count up to n, not 2^n.) It is then a simple matter to rederive the results of Section III.

Theorem 10. PA_k-SPACE(constant)=DTIME($2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{cn}}}}}\right]_k}$) and BA_k-SPACE(constant)=NSPACE($2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{cn}}}}}\right]_{k-1}}$).

This is where the fact that BA_k-Tm's simulate and are simulated by N-Tm's becomes important. For k=1, the non-determinism cannot be eliminated without affecting the order of growth.

To get PA_k-Tm's that accept only the regular languages, we need to make the input tape a public resource.

Definition. A PA_k-fsa (BA_k-fsa) is a constant space PA_k-Tm (BA_k-Tm) with the input tape one-way and public to all.

Lemma 4. PA_k-fsa's and BA_k-fsa's accept only the regular languages.

Proof: The extended "subset construction" of Sections III.1 and III.3, when applied to PA_k-fsa's results in configurations in the modified game tree of constant size. The only non-constant size piece of information is the input head position which does not need to be put into the configurations since it is public to all. Hence the whole game tree is constant size and can be explored for acceptance by a non-deterministic fsa with $2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{O(n)}}}}}\right]_k}$ states where n is the number of states of the PA_k-fsa. []

Chandra and Stockmeyer [2] prove that A-fsa's (alternating fsa's) are exponentially more succinct, in some cases, than N-fsa's. We show that our PA_k-fsa's are even more succinct.

Consider the variant of the language in [2].

$$L_n = \{\{0,1\}^*w\{0,1\}^*\$w \mid w\in\{0,1\}^n\}$$

It is easy to show (via counting arguments) that L_n is not accepted by any D-fsa (with one-way input head) which has fewer than 2^{2^n}, or any N-fsa with fewer than 2^n states. It can be accepted by an O(n) state A-fsa (but by no fewer). Given a PA_k-fsa with O(n) states, we can again apply our counting methods again to count up to $2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{n}}}}}\right]_k}$. It is a simple matter to use this to accept L_n with a O(log n) state PA_1-fsa. However, not all L_n's can be represented by O(log log n) state PA_2-fsa's since there are more L_n's in a given range than there are O(log log n) state PA_2-fsa's. Some can be accepted by O(log log n) state PA_2-fsa's, e.g. when n is a power of two. Hence we have to phrase our main succinctness theorem accordingly.

Theorem 11. $L_{2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{n}}}}}\right]_k}}$ is accepted by an O(n) state PA_k-fsa (also a BA_k-fsa), but by no D-fsa with fewer than $2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{n}}}}}\right]_{k+2}}$ states. (The constant factor for the PA_k-fsa will include a c^k term.)

As a direct application of PA_k/BA_k-fsa's we consider the "(not) emptiness of language" question. Given a $2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{cn}}}}}\right]_k}$ SPACE N-Tm and an input, it is quite easy to construct an O(n) state BA_k-fsa which accepts only the accepting sequence of configurations of the N-Tm on the input, if it exists. The technique uses the ability to count to large numbers to check length of configurations, corresponding positions in adjacent configurations, etc. Similarly for PA_k-fsa's. The upper bounds follow from the extended "subset construction" for the fsa's which results in an N-fsa with $2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{O(n)}}}}}\right]_k}$ states. Hence only strings up to that length have to be checked, with the space to write down the string being the dominant factor in the analysis.

Theorem 12. The (not) emptiness of language question for PA_k-fsa's and for BA_k-fsa's is NSPACE($2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{cn}}}}}\right]_k}$) Complete. (Where n is the number of states. In terms of a representation of the machines, a log n factor is introduced using a standard form which reduces the top exponent of the lower bound by a log n factor.)

From [17] we know that the "not empty complement" question for extended regular expressions with nested complements at most k deep is NSPACE($2^{\left[2^{\cdot^{\cdot^{\cdot^{2^{cn}}}}}\right]_k}$) Complete (poly-lin reduction).

Hence "nesting" of players in a hierarchical PA_k-fsa is indirectly related to nesting of complements in extended regular expressions.

V.2 Private pushdown store automata

A large number of results based upon the alternation version of pushdown store automata (A-pda's) appeared in [6]. Characterizations of several versions resulted in well defined time and space complexity classes. We do not expect that studying "multiple person pda's" will provide any such insight as even a very simple form accepts all r.e. languages.

Definition. A MPA_k-pda is a MPA_k-Tm with only two tapes: a one way, read only input tape public to all players and the second is a pushdown store (i.e. it always prints blanks when going left).

Theorem 13. All r.e. languages are accepted by MPA_1-pda's.

Proof: The pushdown store is allowed to be private to the $\forall$-player. For a given single-tape Tm, the $\exists$-player of the game for that Tm has the task of sending to the $\forall$-player a character at a time, the sequence of configurations leading to acceptance on the input (if it exists). Every other configuration will be in reverse order: $C_0, C_1^R, C_2, \ldots, C_m^{(R)}$. The $\forall$-player checks against the input the first configuration to verify that it is indeed the initial configuration for that input. It also chooses to check two adjacent configurations to verify that one legally follows from the other. The first one it chooses is secretly placed on the pushdown store as it comes in. It is then popped as the second configuration comes in, and it verifies that corresponding tape cells match up right, head motion and symbol written are correct, etc. (This is why every other configuration is in reverse order, so that the corresponding tape positions can be properly checked.) It also checks that the last tape configuration holds an accepting state. []

V.3 Markov alternating machines

We briefly mentioned Markov alternation machines (MA_k-Tm's) in Section IV.3. These machines have the property that the $\exists$-player's strategies can only depend on their current visible positions, and not at all on their histories of visible positions. We return to Markov machines because we are interested in obtaining a non-deterministic (elementary recursive) hierarchy of machines to go along with our deterministic time and (non-deterministic) space hierarchies. While we obtain a non-deterministic time class with space bounded MA_1-Tm's, they do not generalize for more players. We first present the main theorem for time bounded MA_k-Tm's.

Theorem 14. $MA_k\text{-TIME}(T(n)) = NTIME(2^{cT(n)})$, for countable $T(n) \geq n$ and any $k \geq 1$.

Proof: From Section IV.3, we know that time bounded MA_k-Tm's represent PA_k-Tm's which have written their histories down. The upper bound then follows from Corollary 5. For the lower bound, we just need to show that a MA_1-Tm can play the DQBF lower bound game in $O(T(n))$ time. The $\forall$-player first sends to the $\exists$-player the first time, which it writes down. The $\exists$-player then responds with the appropriate Y_1 tuple and erases its tape. The $\forall$-player sends next the second time. The $\exists$-player has effectively forgotten the first time, and must respond as if it were the second player of the DQBF lower bound game. It must send the right Y_2 tuple and both tuples are checked by the $\forall$-player. []

Unlike the other machines, we find that "SPACE = TIME" for Markov machines.

Theorem 15. $MA_k\text{-SPACE}(S(n)) = NTIME(2^{cT(n)})$, for constructable $S(n) \geq \log n$ and any $k \geq 1$.

Proof: The lower bound follows from the previous theorem since a $S(n)$ space bounded machine has at least $S(n)$ time available to it. (The case for $S(n) < n$ is a simple extension of the previous proofs.) For the upper bound, we will first show that any MA_k-Tm with space $S(n)$ can be simulated by a MA_1-Tm with the same space. We use a technique hinted at in the previous proof which has one player making the moves for all. The $\forall$-player will usually keep all information of all players on its private tapes. When it is time for the $\exists_i$-player to make a move, the $\forall$-player sends to the $\exists$-player of the MA_1-Tm the information visible to that player and the player's number. The $\exists$-player must respond as if it were that player, sends the new visible position back to the $\forall$-player and clears its tapes and state. This effectively causes the $\exists$-player to forget what move it made for that player. It involves no increase in space. To simulate a MA_1-Tm with bounded space, we first note that any path of the game tree which is more than exponential in $S(n)$ deep, contains an infinite loop. For MA_k-Tm's, any repetition of configurations (an exponential in space number) causes an infinite loop, unlike PA_k-Tm's. If the $\exists$-player has a winning strategy, then it also must have one that corresponds to an accepting subtree whose depth is at most exponential in $S(n)$. Thus, we only have to explore the game tree to a depth of $2^{O(S(n))}$. The tree is explored as in the proof of Corollary 4. First, the complete set of moves to be made in any situation by the $\exists$-player is written

down, taking (non-deterministic) time $2^{cS(n)}$. Then the game tree is explored using the extended "subset construction" to represent the $\forall$-player's private configurations. There will be $2^{O(S(n))}$ steps, each taking time $2^{O(S(n))}$ each. (Before there were only T(n) steps of time $2^{O(T(n))}$ each.) The total time is therefore (non-deterministic) $2^{O(S(n))}$ time. []

VI. Parallel Finite State Automata

It is a very short step to the equation "player = process". This obvious relation between games and parallel systems has been applied by Devillers [3] and Ladner [5], and only in a context of perfect information. We have applied our results on the complexity of decision problems for multiple person fsa's to questions related to lockout/deadlock in parallel systems, as was done in [5]. We will briefly sketch our results so far as applied to parallel fsa's. A follow-up paper [15] discusses these questions in more detail, and presents a logic for multi-processing systems with incomplete information based upon dynamic logics.

It is perhaps surprising that although we restrict ourselves to finite machines, not all questions are decidable.

Our definitions are informal and we rely on the reader's intuition to understand them and how they are related to games of incomplete information (and their complexities). We consider a model related to Ladner's [5], but augmented with private states.

Definition. A system of parallel finite state machines consists of p processes, each having local (private) states and shared states representing shared variables. Communication is by "handshaking", that is a process changing a shared variable must wait until it is read before moving on. The transition rules within a process are not deterministic in order to represent branches caused by factors unknown to other processes.

(Note that Ladner uses a state matrix not unlike our concept of global states, and his matrix elements correspond to our local states.)

A basic question for parallel systems is "lockout avoidance". It asks whether a given process can avoid being locked out (that is, prevented from completing a task) regardless of the choices made in branching by the other processes. An application of this question is the situation where a process wishes to perform a task, but will not start it unless it can first prove that it will be able to find a way to complete it no matter what the other processes do. This is a natural game question. Ladner [5] treats this as a game of complete information. However, in most distributed systems, the state of the other processes is not known or not easily obtained. Similarly, the other processes may be communicating between themselves through variables not visible to the target process. In these situations, the question can be better understood as a game of incomplete information. This version of the lockout avoidance problem is thus a p player game, with Team A having one player and Team B the rest. We can easily apply the results of the previous sections. We just have to equate avoiding lockout (completing the task) with reaching the final state (or winning the game).

Theorem 16. The "one against all" lockout avoidance question requires $2^{2^{cn/\log n}}$ time to answer.

Ladner's result, based on perfect information, is one exponential lower. Again, the difference between counting states and representing an n state fsa appears. (The upper bound does not have a log n divisor for this problem or for Ladner's.)

If we had instead phrased the lockout avoidance question from the point of view of the other set of processes, we would not have obtained the exact complement of the question. The "guaranteed lockout" question is to determine if k processes have a way of locking out the k+1'st process regardless of its counter actions. While the lockout avoidance question is a "one against many" game, the guaranteed lockout question is "many against one". The asymmetry caused by being a game of incomplete information again reappears. Our proposed logic [15] distinguishes between both cases and we have:

Theorem 17. The guaranteed lockout question requires $2^{2^{cn}}$ time when k=1 and is undecidable for larger k.

VII. Conclusion

We feel that multiple person alternation machines have two distinguishing characteristics that indicates that further study will be productive:

(1) Multiple person alternation machines seem to lie between logical theories (via alternation) and regular expression word problems (via the PA_k-fsa connection). Perhaps new relations and better bounds can be found for problems in both areas.

(2) Consider languages which consist of a single n-character string. Any D-fsa, N-fsa, or A-fsa which accepts such a language requires n+1 states. However, some can be recognized by O(log n) state PA_1-fsa's, etc. These machines provide an interesting framework for studying succinctness of representaion.

Bibliography

[1] Chandra, A. K., D. C. Kozen, and L. J. Stockmeyer, "Alternation", IBM Research Report RC 7489, Yorktown Heights, N.Y., Jan. 1978.

[2] Chandra, A. K., and L. J. Stockmeyer, "Alternation", Proceedings 17th Annual Symposium on Foundations of Computer Science, (1976) pp. 98-108.

[3] Devillers, R., "Game interpretation of the deadlock avoidance problem", CACM 20, 10, (Oct. 1977) pp. 741-745.

[4] Fraenkel, A. S., M. R. Garey, D. S. Johnson, T. Schaefer, and Y. Yesha, "The complexity of checkers on an NXN board", Proceedings 19th Annual Symposium on Foundations of Computer Science, (1978) pp. 55-64.

[5] Ladner, R. E., "The complexity of problems in systems of communicating sequential processes", Proceedings 11th ACM Symposium on Theory of Computing, (1979) pp. 214-223

[6] Ladner, R. E., R. J. Lipton, and L. J. Stockmeyer. "Alternating pushdown automata", Proceedings 19th Annual Symposium on Foundations of Computer Science, (1978) pp. 92-106.

[7] Lichtenstein, D. and M. Sipser, "GO is PSPACE hard", Proceedings 19th Annual Symposium on Foundations of Computer Science, (1978) pp. 48-54.

[8] Meyer, A. R., and L. J. Stockmeyer, "The equivalence problem for regular expressions with squaring requires exponential space", Proceedings 13th Annual Symposium on Switching and Automata Theory, (1972) pp. 125-129.

[9] Miller, R. E., and C. K. Yap, "On formulating simultaneity for studying parallelism and synchronization", Proceedings 10th ACM Symposium on Theory of Computing, (1978) pp. 105-113.

[10] Paul, W. J., E. J. Praub, and R. Reischuk, "On alternation", Proceedings 19th Annual Symposium on Foundations of Computer Science, (1978) pp. 113-122.

[11] Peterson, G. L., "Press-Ups is PSPACE complete", University of Rochester, Jan. 1979.

[12] Rackoff, C., "On the complexity of the theories of weak direct products", Proceedings 6th ACM Symposium on Theory of Computing, (1974) pp. 149-160.

[13] Rapoport, A., N-Person Game Theory, University of Michigan Press, Ann Arbor, 1970.

[14] Reif, J. H., "Universal games of incomplete information", Proceedings 11th ACM Symposium on Theory of Computing, (1979) pp. 288-308.

[15] Reif, J. H., and G. L. Peterson, "A dynamic logic of multiprocessing with incomplete information" (in preparation).

[16] Savitch, W. J., "Relationships between nondeterministic and deterministic tape complexities", JCSS 4, (1970) pp. 177-192.

[17] Stockmeyer, L. J., "The complexity of decision problems in automata theory and logic", MIT Project MAC TR-133 (Ph.D. dissertation), July 1974.

[18] Stockmeyer, L. J., and A. R. Meyer, "Word problems requiring exponential time", Proceedings 5th ACM Symposium on Theory of Computing, (1973) pp. 1-9.