Foundations of Linguistic Geometry: Complex Systems and Winning Strategies

D R A F T

Vladimir Yakhnis
Mathematical Sciences Institute
Cornell University
409 College Avenue, Ithaca, NY 14850
E-mail: vlad@msiadmin.cit.cornell.edu

Boris Stilman
Dept. of Computer Science & Engineering
University of Colorado at Denver
Campus Box 109, Denver, CO 80217-3364
E-mail: bstilman@cse.cudenver.edu

Abstract

The Linguistic Geometry (LG) approach to discrete systems was introduced B. Stilman in early 80s. It employed competing/cooperating agents for modeling and controlling of discrete systems. The approach was applied to a variety of problems with huge state spaces including control of aircraft, battlefield robots, and chess. One of the key innovations of LG is the use of almost winning strategies, rather than truly winning strategies for the playing agents. There are many cases where the winning strategies have so high time complexity that they are not computable in practice, whereas the almost winning strategies can be applied and they beat the opposing agent almost guaranteed. Independently of LG the idea of competing/ cooperating agents was employed in the late 80s by A. Nerode, A. Yakhnis, and V. Yakhnis (NYY) within their approach to modeling concurrent systems and, more recently, within the "Strategy Approach to Hybrid Systems" developed for continuous systems by A. Nerode, W. Kohn, A. Yakhnis, and others. In order to enlarge the range of applications of Stilman's results as well as to understand why they work, we introduce a new notion of a multi-agent graph-game. Since the applications of Linguistic Geometry sometimes require modeling of agents making simultaneous moves, the new notion was designed to accommodate such behavior of players.

1. INTRODUCTION

1.1. Computational Power of Linguistic Geometry

The purpose of this paper is to investigate the foundations for the LG, so that the similarities and the differences between it and the other approaches exploiting the idea of "several competing agents" would be well understood. What distinguishes LG from other AI approaches is its tremendous computational power. Experiments comparing the AI systems based on LG and those utilizing other approaches (e.g. alpha-beta pruning) have shown the former to be superior (Stilman, 1994a, 1994b, 1995b). Some of the problems were not solvable by other approaches at all, whereas LG systems successfully solved them (Stilman, 1995a). For other problems both the branching factor and the computation time for the systems based on LG were several times smaller than those for the competition (Stilman, 1994a). Note that although LG has a discrete nature, it has been applied to some continuous systems as well through an ad hoc discretization of the latter (Stilman, 1994-1995).

The computational power of LG is based on a utilization of human expert heuristics which were highly successful in a certain class of complex control systems. However, in contrast to the other approaches based on the idea of "two competing agents", these heuristics made the mathematical essence of LG extremely complex and thus less obvious.

After the present investigation we expect to extend the application domain of LG to the area of verification of concurrent systems (through the NYY approach) and to the operating systems supporting persistent truly concurrent objects. Presently an approach to the latter area is being developed (Yakhnis-Yakhnis, in preparation). It is partially based of the NYY ideas. We also expect that the Linguistic Geometry would have more extensive applications to continuous systems via the "Strategy Approach to Hybrid Systems" developed by A. Nerode, W. Kohn, A. Yakhnis (NKY), and others (Kohn-Nerode, 1993; Nerode-Remmel-Yakhnis,1993; Nerode-Yakhnis, 1992). This is possible since the latter approach eventually converts continuous systems into discrete ones.

1.2. State Transition Systems and Their Control

A state transition system (STS) is a system involved in a process of changing its state in a discrete manner either until some *goal* is reached or ad infinitum. For example, chess (or checkers) may be viewed as a discrete state system with states as positions of the pieces on the board, state transitions as legal moves of the players, and the goal as a checkmate (or elimination of all the pieces of the opposition). Another example is a concurrent program where the state of the system is a combination of the states of the component processes plus the set of all submitted but yet unexecuted instructions plus the set of messages that were sent but not yet delivered (Nerode-Yakhnis-Yakhnis, 1992, 1993). The state transitions are legal executions of instructions or receptions of messages, and the goal is the set of states corresponding to the computational result described in the program specification.

Besides the goal, a desirable behavior of a system may include a *constraint*, i. e., a condition that the system states must always satisfy during the process of going through the state transitions. For example, in chess a constraint may be formulated as “avoid the stalemate”. For a concurrent program adherence to “mutual exclusion” is an example of a constraint.

Another aspect of STS is their *control*. The problem of systems control is the problem of how to influence the state transitions so that the goal would be reached, the constraint would always be satisfied, etc. Often there exists an explicit agent (we shall call it *Controlling-Agent*) given at least a partial influence on the state transitions and entrusted with the task of guiding the system through the state transitions from some initial state, so that eventually the system specification would be satisfied. For example, in chess the Controlling-Agent is the player we are rooting for. For a concurrent program the Controlling-Agent is not explicitly given. We may think that it is the whole team of programmers entrusted with writing and maintaining the program (Nerode-Yakhnis-Yakhnis, 1992).

1.3. “Competing Agents” View on Discrete State Systems

To solve the problem of systems control, we view the state transition process of a system as a contest/cooperation between several competing/ cooperating agents, one of which may be the aforementioned Controlling-Agent. When there are only two competing agents, we'll call the other agent the *Opposing-Agent.* For simplicity sake and also since the case of two competing agents is a common occurrence, we'll mostly limit our introductory discussion to that case.

Similar to the Controlling-Agent, the Opposing-Agent exercises at least a partial influence over the state transitions and, in contrast to the Controlling-Agent, its purpose is either to guide the controlled process to a point where the constraint would be violated or to prevent reaching the goal. Such formal view of STS control was introduced in the early 1980s by B. Stilman in (Stilman, 1981, 1985). A substantial later refinement of Stilman's approach was called in (Stilman, 1992) the “Linguistic Geometry” (LG). In the late 80s a similar view independently appeared within Nerode-A. Yakhnis-V.Yakhnis (NYY) approach to modeling concurrent systems (Yakhnis, A. 1989, Yakhnis, V. 1989, Nerode-Yakhnis-Yakhnis 1992, 1993), and, more recently, within the NKY approach.

The NYY approach regards a specification for a concurrent program as a winning condition in a certain two player game, whereas a program satisfying the specification is regarded as a strategy winning the game with respect to the aforementioned winning condition. Thus the algorithms solving this class of games become applicable for the concurrent program specification and, possibly, even for the extraction of programs from the specifications.

The NKY approach is a highly interesting approach to the control of continuous systems proposing a framework intended for extracting and verifying control programs for continuous plants. The approach regards such programs as finite state winning strategies in the games associated with the plants. This approach partially builds on the ideas developed within the NYY approach.

The “two competing agents” view requires us to always identify (or create) the competing agents, even if the original system does not include their explicit definition. For example, in chess the Opposing-Agent is the player we are not rooting for. For a concurrent program, similar to the Controlling-Agent, the Opposing-Agent is not explicitly given. We may think that in some sense it is all the hardware involved in running the program (Nerode-Yakhnis-Yakhnis, 1992). The major benefit of the “two competing agents” view is a possibility to formulate for each competing agent an explicit strategy guiding its behavior. All the above approaches concentrate on finding such strategies.

1.4. Summary of Results

- Rigorous foundations of the LG complex systems via state transition systems (STS).
- Introduction of multi-agent graph-games (involving several competing/cooperating agents), capable of covering simultaneous moves of several agents.
- Theorem of Coincidence of the classes of STS generated by the LG complex systems and by the multi-agent graph-games.
- Transference of Linguistic Geometry's algorithms computing “almost winning” strategies to several approaches to systems control utilizing the idea of competing/cooperating agents.

2. THE LG COMPLEX SYSTEMS AS STATE TRANSITION SYSTEMS

2.1. State Transition Systems

In order to conveniently discuss the Linguistic Geometry approach, we would like to introduce the notion of *state transition systems* (STS). STS are very much like deterministic finite state automata (Hopcroft-Ullman, 1979), but viewed with the following three distinctions. First, similar to Büchi-Landweber (1969), we admit both finite and infinite input sequences. Second, similar to Rabin (1969), we concentrate on the automata *runs* (sometimes called *system runs* or *transition sequences*) which are, roughly speaking, sequences of states through which the automaton passes while accepting an input sequence. Just as the input sequences, runs may be finite or infinite. Third, similar to Stilman (1981, 1985), we treat the input symbols as partial unary operations on the automaton states (we call them *transition functions*), and we treat the automaton transition table as a collection of transition rules, as explained below. Thus, an STS, say Γ, is a quintuple

$\Gamma =$ **S, $\mathbf{S_{in}}$, $\mathbf{S_{fin}}$, TF, TR** , where

•	**S**	the (finite) *space of states*;
•	$\mathbf{S_{in}}$	the set of *initial states*;
•	$\mathbf{S_{fin}}$	the set of *final states*;
•	**TF**	the set of input symbols, also called *transition functions*;
•	**TR**	the transition table represented as a set of *transition rules*.

All the above sets are usually finite, although the definitions below are applicable to infinite sets as well. Now we'll describe the five sets in more detail. **S** is a collection of things called states. In a concrete system they may have a specific structure. Both $\mathbf{S_{in}}$ and $\mathbf{S_{fin}}$ are subsets of **S**. $\mathbf{S_{in}}$ and $\mathbf{S_{fin}}$ represent, respectively, the set of states from which the system is allowed to initiate a series of transitions (called a system run), and the set of states where the system run may be legally terminated. If a system run would never encounter a final state then it must be infinite.

Each transition function f from **TF** is a partial function of the form f: $\mathbf{S} \rightarrow \mathbf{S}$. Each transition rule is a pair of the form (S_f, f), where f is a transition function and S_f is a subset of **S**, called the *applicability* set for f. We also say that a transition rule (S_f, f) (or a transition function) is *applicable* to a state s, if s is in the set S_f.

We call **TR** *tight* if for any transition function f there is a unique transition rule (S_f, f) for some set of states S_f. Using tight transition rules is a matter of convenience. There are also two crucial properties that the transition systems must satisfy in order to correctly represent finite state automata, namely, *validity* and *completeness*:

- We call **TR** valid if for each transition rule (S_f, f),

$$S_f \quad \text{Dom}(f);$$

- We call **TR** complete if

$$\mathbf{S} = (\mathbf{S_{fin}} \quad (\quad {}_{f \quad \mathbf{TF}} S_f))$$

We now can formally define a system run for Γ. It is a finite or infinite sequence of states from **S**, written from left to write and such that:

- the leftmost state is an initial state;
- if the sequence contains a final state, the sequence is finite and the final state is the rightmost state;
- if the sequence does not contain a final state, the sequence is infinite;
- each state in the sequence, except the initial state, is the result of applying one of the transition functions to the previous state according to the transition rules as follows. Suppose that s_i and s_{i+1} are two adjacent states in a transition sequence. Then there is a transition rule (S_f, f) applicable to s_i and such that $s_{i+1} = f(s_i)$.

If the transition rules were invalid, then, while building a system run, we may have reached a state, say s, and choose a transition (S_f, f) applicable to s but such that f(s) would be undefined. On the other hand, if the transition rules are incomplete, then we may have reached a state not in $\mathbf{S_{fin}}$, say s, such that there would be no transition rules applicable to s. So the system run would be terminated illegally.

We would like to make a note about parametrizing the transition rules. Sometimes it is convenient to write them in the form

$$(S_f(y_0, ..., y_{n-1}), f(y_0, ..., y_{n-1})),$$

where $(y_0, ..., y_{n-1})$ are some parameters pertaining to some elements within the intrinsic structure of a state. When the parameters are replaced by concrete values, then the parametrized formula above would be converted to a concrete transition rule. For example the following is a real life parametrized transition rule for chess (for the convenience sake, we denoted the first player as 0 and the second player as 1):

- $S_f(i, y, z, q_0, q_1)$ is the set of all states satisfying "q_0 is a Bishop for a player i, q_0 is placed on a location y, z is a location where a bishop could be directly moved from y, there are no any pieces between the locations y and z, a piece q_1 of the player 1-i occupies the location z";
- $f(i, y, z, q_0, q_1)$ removes the piece q_1 from z and relocates the piece q_0 from y to z.

Finally, STS are intended to be used for modeling those real life systems which operate by switching from a state to a state according to some rules in a manner similar to system runs for STS. As was discussed in the introduction, real life systems are usually associated with one or more agents which may have some (possibly conflicting) requirements on the properties of the system run. The agents try to fulfill their goals by influencing the selection of the next input (i. e., a transition function) for the system. This selection is limited to the set of all transition functions applicable to the current system state. The purpose of systems control is to provide the agents with strategies helping them to select applicable transition functions so that their requirements on the system run would be eventually satisfied. The above "bare-bones" definition does not have any structure for expressing strategic behavior. The LG complex systems provide such a structure.

2.2. An intuitive View on the Linguistic Geometry

Linguistic Geometry (LG) is an approach to control of state transition systems with a particular structure of states. Each state is represented as a disposition of certain *pieces* (also called *elements*) on a *board* consisting of a collection of *locations* (also called *points*). In this context a piece is placed on a location (as in chess or checkers). The state transitions correspond to *moving* pieces, i. e., relocating pieces within the board, removing from, and placing pieces on the board. Each piece imposes a geometrical structure of *immediate reachability* on the board by defining for each given location a collection of all locations where the piece could be relocated in one step from the given location (hence, the “Geometry” portion of the name).

In addition, the pieces are divided between several competing/cooperating agents. Each agent can influence the state transitions by choosing the moves of its pieces within some framework of rules. In general, a state transition of the system consists of simultaneous movements (i. e., relocations, and/or removals, and/or additions) of pieces for several opponents, although some frameworks of rules specify that for each transition only one agent has control over the relocation of pieces.

The competing agents have opposing goals, whereas the cooperating agents have similar goals. For each agent the LG approach provides for strategies guiding its behavior so that its goal would be reached despite the efforts of the opposition. These strategies utilize a hierarchy of formal grammars (hence, the “Linguistic” portion of the name) based on the trajectories on the board, networks of the trajectories (called zones), and networks of zones. We call these strategies “almost winning” since they do not guarantee reaching the goal against all possible behavior of the opponent. However, experiments have shown that they provide a high likelihood of reaching the goal. Moreover, often they allow an agent to reach its goal when other AI approaches fail to do so and when the truly winning strategies are impractical because of their extremely high time complexity (e. g., for chess or for the problem of several robotic vehicles in space). An intuitive reason for this is utilization of human expert heuristics which were highly successful in a certain class of complex control systems. In order to understand better the reasons for a high likelihood of winning for the LG strategies, we would like to look deeper into the mathematical foundations of the LG complex systems and the foundation of the notion of winning. We'll start from the LG complex systems.

2.3. The LG Complex Systems

2.3.1. The Eight-Tuple

Within the LG approach a practical implementation of STS was called *Complex Systems* (Stilman 1981, 1985, 1992). We'll first discuss the Complex Systems in the way they were originally defined and then we'll slightly adjust this definition to accommodate our present theoretical view. A Complex System, say Φ, is the following eight-tuple:

$$\Phi = \mathbf{X, P, R, ON, v, S_{in}, S_{trg}, TR}$$

where

- **X** the space of locations (the board);
- **P** the set of pieces;
- **R** the reachability relation, where **R** **P**×**X**×**X**;
- **ON** the *placement* notation for describing *placement* equations of the form **ON**(p) = x, meaning “a piece p placed at a location x”;
- **v** a function assigning to each piece a numerical value;
- $\mathbf{S_{in}}$ the set of initial states;
- $\mathbf{S_{trg}}$ the set of target states;
- **TR** the set of transition rules.

2.3.2. Locations, Pieces and Reachability

Note that the reachability relation **R** imposes some structure on the board **X** and the pieces **P**. In addition, **X** may have some other intrinsic structure. For example, for chess (where locations are sometimes called squares) each square not on the border has eight adjacent squares corresponding to eight possible directions of movement, the board is divided into two disjoined subsets, the White squares and the Black squares, etc. Also, for a complex system describing robotic vehicles, the board may be called Terrain and it may include the subsets Forest, Swamp, Highway. It is assumed that we have a list of names for all the locations, e. g., x_0, ..., x_{k-1}.

The set of pieces **P** is a disjoint union of collections of pieces P_0, ..., P_{m-1} assigned, respectively, to agents A_0, ..., A_{m-1}:

$$\mathbf{P} = \ _{0\le i<m} P_i$$

Similar to locations, the pieces **P** may have some additional structure and it is assumed that we have a list of names for all the pieces for each agent A_i, e. g., $p_{i,0}$, ..., p_{i,n_i}. For example, for robotic vehicles we may have a subset Tanks with pieces Putton (for agent USA), Centurion (for agent UK), and T-72 (for agent Russia), and a subset Cars with pieces Ford (for agent USA), and Volvo (for agent Sweden).

R(q, y, z) means that y and z are distinct locations and that the piece q may be relocated from the location y to the location z in one move, provided that there are no additional obstacles. For example, in chess if q is not a Knight then any piece placed between y and z serves as an obstacle. There are no obstacles for Knights. The statements describing the absence of obstacles are included within the applicability conditions for the transition rules.

2.3.3. The STS embedded in Φ

We will now define which transition system $\Gamma =$ **S, $\mathbf{S_{in}}$, $\mathbf{S_{fin}}$, TF, TR** is being modeled by the eight-tuple above. We can immediately identify two components, namely, $\mathbf{S_{in}}$ and **TR**, that have exactly the same meaning and notation in Φ and in Γ. However, we'll still have to discuss their implementation within the LG framework. In addition, there is a third component, namely, $\mathbf{S_{fin}}$, which has the same meaning and notation in Φ and in Γ. $\mathbf{S_{fin}}$ was not explicitly included in the eight-tuple, although it was implicitly present. A historical reason for such implicit inclusion was avoiding confusion with $\mathbf{S_{trg}}$ which has entirely different meaning.

The space of states **S** for Φ is a subset of the space DISP of all possible functions D (called *disposition functions*) of the form D: X$\rightarrow$PowerSet(P). Given a disposition function D, it is convenient for us to slightly abuse notation by sometimes writing D(x, p) instead of p D(x). Both notations mean “a piece p is placed at a location x within the disposition D”. We would like to adjust some of the original notation for LG complex systems to this view. We'll write D(x, p) instead of “ON(p) = x in a state D”. Just as $\mathbf{S_{fin}}$, **S** was implicitly included in the LG complex systems. In a little while we'll describe how **S** DISP was implemented.

Now we may describe **TF**. We'll usually call transition functions for complex systems *moves*. Given a disposition D, a move may change D via one or several relocations, removals, or placements of pieces. Moves are inherently partial functions since one cannot remove a piece p from a location x in relation to a disposition function D if p is not in D(x). In addition to some other moves, **TF** contains all possible moves relocating a piece q from a location y to a location z for q, y, z satisfying the reachability condition **R**(q, y, z).

2.3.4. Implementation of the Elements of the Embedded STS

Now that we understand how the five elements of STS (i. e., **S, $\mathbf{S_{in}}$, $\mathbf{S_{fin}}$, TF, TR**) are embedded within Φ, we'll describe how they are implemented. A move f **TF** is implemented as a sequential application of two operations, removal of some pieces from the board, and placement of some pieces on the board. The first one is defined via a *Removal list*, and the second one is defined via a *Placement list*. Each list contains several placement equations of the form D(x, p) (ON(p) = x in the original notation). For example, relocating a piece q from a location y to a location z is defined as follows:

- **Removal list:** D(y, q);
- **Placement list:** D(z, q).

The sets of states (i. e., the subsets of DISP) will be described as follows. We'll define a first order language with a unique variable D associated with elements of DISP (quantifications over DISP would not be allowed) and some other variables associated with other elements of the structure of Φ (i. e., **X**, **P**, etc.) After that any formula ϕ(D) in that language, with no occurrences of free variables other than D, would represent the set of all concrete elements D_0 of DISP such that $\phi(D_0)$ holds. A formula containing occurrences of free variables other than D would represent a parametrized subset of DISP with the free variables other than D serving as parameters.

With this in mind, a parametrized transition rule would be of the form (ϕ(D, w_0, ..., w_{n-1}), RL(w_0, ..., w_{n-1}), PL(w_0, ..., w_{n-1})), where w_0, ..., w_{n-1} are parameters, ϕ is a formula with no free variables other than D, w_0, ..., w_{n-1}, and RL(w_0, ..., w_{n-1}) and PL(w_0, ..., w_{n-1}), respectively, are parametrized Removal and Placement lists. Moreover, it is assumed that both the Removal and Placement lists contain only such placement equations D(y, q) where y and q are either parameters or some concrete locations or pieces.

Now we would like to show how to establish the validity and completeness of **TR**. First we associate RL(w_0, ..., w_{n-1}) with a formula CRL(w_0, ..., w_{n-1}) representing the conjunction of all the placement equations in the list. One of the necessary conditions of validity is the formula ϕ(D, w_0, ..., w_{n-1}) CRL(w_0, ..., w_{n-1}). If this formula is not true, then the transition function may attempt to remove a piece from a location in a state where the location does not contain that piece.

Another necessary condition of validity is to prove for every placement equation D(z, q) in RL(w_0, ..., w_{n-1}) that

(CRL(w_0, ..., w_{n-1}) (z_1 D(z_1, q)))
(ϕ(D, w_0, ..., w_{n-1}) $\neg$(z_1 D(z_1, q))).

If this formula is not true, then the transition function may attempt to place a piece on the board in a state where the board already contains that piece. This, in turn, would violate the definition of a state. Concrete complex systems may have other requirement necessary to check in order to establish validity. For example, in chess or checkers a location may contain at most one piece.

Finally in order to establish completeness, one has to prove that the disjunction of the formula representing the set of final states $\mathbf{S_{fin}}$ with the disjunction of all the formulas of the form (w_0, ...,

$w_{n-1}\phi(D, w_0, ..., w_{n-1}))$, where $\phi(D, w_0, ..., w_{n-1})$ is the applicability condition from a (parametrized) transition rule, follows from the formula representing the space of states **S**.

2.3.5. The Formal Language Associated with Φ

- **Sorts**: **X**, **P**, DISP, **N**, where **N** is the set of nonnegative integers.
- **Relations**: **R** **P**×**X**×**X**; $P_0, ..., P_{m-1}$ **P**; Apply DISP×**X**×**P**, and some additional intrinsic relations on **X** and **P**.
- **Functions**: **v**: **P**→**N**; Agent: **P**→**N** and some additional intrinsic functions on **X** and **P**.
- **Constants**: $x_0, ..., x_{k-1}$; $p_{i,0}, ..., p_{i,n_i}$ for i = 0, ..., m–1.
- **Variables**: w for any sort; y, z for **X**; q for **P**; i, j for **N**; D for DISP; Indices 0, 1, ... are allowed for all the above variables except for D.
- **Restrictions**: no quantifiers over D.
- **Abbreviations**: Apply(D, x, p) is abbreviated as D(x, p).
- **Axioms**: **X** = $\{x_0, ..., x_{k-1}\}$; $P_i = \{p_{i,0}, ..., p_{i,n_i}\}$ for i = 0, ..., m–1; $P_0, ..., P_{m-1}$ are disjoint;

  P = $_{0\le i<m}P_i$; D does not allow the same piece at more than one locations; Agent($p_{i,j}$) = i for i = 0, ..., m–1 and j = 0, ..., n_i.

To illustrate the above, we'll formalize the example of a parametrized applicability condition for chess given in section “State Transition Systems”. Recall that for chess we have two agents, A_0 and A_1. We also need two relations from the chess structure, a unary relations Bishop and a ternary relation Between. Bishop(q) holds if and only if q is a bishop and Between(y, y_1, z) holds if y and z are on the same row, or column, or diagonal and y_1 is strictly between y and z. Now we are ready to formalize the example:

- $S_f(i, y, z, q_0, q_1)$ =
  (Bishop(q_0)   i = Agent(q_0)   D(y, q_0)   R(q_0, y, z)   (y_1 (Between(y, y_1, z)   ¬(q_2 D(y_1, q_2))))   D(z, q_1)   (1-i) = Agent(q_1));
- f(i, y, z, q_0, q_1):
  - **Removal list:** D(y, q_0), D(z, q_1);
  - **Placement list:** D(z, q_0).

2.3.6. Using Registers

In addition to the STS discussed above, an LG complex system may have one or more auxiliary STS representing some simple and distinctly separate aspect of the system. We call them *registers*. Now it is convenient to call the STS discussed above the *main* STS (MSTS). Registers have the same transition functions as those of MSTS (i. e., moves). We combine all the registers and MSTS into one global STS (GSTS) as follows.

- The state space of GSTS is a Cartesian product of the state spaces of the components.
- For every move f its GSTS applicability condition is a Cartesian product of the applicability conditions for f for the components.

In practice, instead of making all these Cartesian products, it is enough to run the component STS in parallel and to allow a move if and only if all the STS allow it simultaneously.

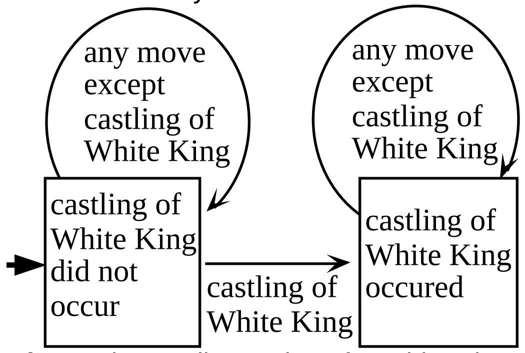
any move except castling of White King
any move except castling of White King
castling of White King did not occur
castling of White King
castling of White King occured

Fig. 1. The castling register for White King

2.3.7. The Winning Conditions and (Almost) Winning Strategies

The last elements left for discussion are $\mathbf{S}_{trg}$ and **v**. The set of target states represents all the final states that are desirable to reach for some agent. This is called the winning condition for that agent. The agent, though not explicit in the eight tuple, is implicitly associated with $\mathbf{S}_{trg}$ by the complex system. Note that it was also implied that $\mathbf{S}_{trg}$ $\mathbf{S}_{fin}$.

The function **v** is used within the LG almost winning strategies. Intuitively, those enemy pieces within reach that have larger **v**-value are targeted for elimination. Moreover, the LG strategies project the total **v**-value lost by the opponent(s) and by the friendly agent(s) during a potential skirmish. If the friendly losses are larger, the strategy would not initiate the skirmish.

Now we would like to ask the following two questions.

- Are there other winning conditions of interest that the LG winning strategies are either able to meet now, or may be enabled to meet in the future?
- What is the relation between the LG almost winning strategies and other computable winning strategies?

In order to answer these questions, in the following sections we will investigate the graph-games.

Meanwhile, we would like to fulfill our promise about a slight adjustment in the definition of the LG Complex Systems.

Firstly, we would like to explicitly include **S**, $\mathbf{S_{fin}}$, and **TF** within the definition.

Secondly, since "**ON**" is a notational device and does not represent a set, function, etc., we would like to exclude it from the definition. The same goes for "D", the suggested above replacement for "**ON**".

Thirdly, we would like to replace $\mathbf{S_{trg}}$ by **AW**, a list of agents and their winning conditions. **AW** would have the following form:

$$A_0, W_0; A_1, W_1; ...; A_{m-1}, W_{m-1},$$

where A_i is an agent and W_i is her winning condition for i = 1, ..., m.

Finally, we would like to rearrange the entities in the tuple by grouping together all the entities pertaining to the embedded MSTS and combining the latter and the registers into GSTS. Thus we will represent Φ as the following six-tuple:

$$\Phi = \mathbf{GSTS}, \mathbf{X}, \mathbf{P}, \mathbf{R}, \mathbf{AW}, \mathbf{v}$$

3. GRAPH-GAMES

3.1. Origins of Graph-Games

The earliest definition of games on graphs that we know of is in (Berge, 1957). Berge, however, focused primarily on games where each player attempts to amass the largest cash value, whereas for us the games of interest are those where each player attempts to satisfy his/her winning condition. In addition, Berge's graphs appear to be inconvenient for our purposes since their edges were not labeled by the moves of the players, thus making the latest move and the state of the game inseparable. We remove these disadvantages in our definition of graph-games.

Graph-games are based on the idea of encoding potentially infinite plays into states of an automaton one step further. Büchi and Landweber (e. g., see Büchi, 1981; Büchi-Landweber, 1969) pioneered work in this area. They were first to show that all the reasoning about each game from a large class of infinite games can be done in terms of the transition table of a suitable finite automaton. However, they worked with very complicated automata (Büchi automata) accepting the infinite plays corresponding to winning conditions.

Independently of Büchi, Gurevich and Harrington (1982) introduced certain equivalence relations on infinite game trees, thus implicitly converting them into more general (possibly finite) directed graphs. Later this conversion was made explicit in (Yakhnis, A. 1990; Yakhnis V. 1990; Yakhnis-Yakhnis, 1990, 1993). As a result of such explicit conversion (called game automata and later graph-games) the algorithms for finding winning strategies presented in (Yakhnis-Yakhnis, 1990,1993) were significantly more transparent than those developed in (Büchi, 1981; Büchi-Landweber, 1969).

Building on ideas developed by Büchi-Landweber, Gurevich-Harrington, and Yakhnis-Yakhnis, McNaughton (1993) developed a slightly different version of graph games. For his class of games McNaughton developed a new and very transparent algorithm for finding winning strategies which was significantly less complex than that of Büchi-Landweber's algorithm. Independently of McNaughton, Zeitman (1994) simplified the algorithms for finding winning strategies presented in (Yakhnis-Yakhnis, 1990) and considerably improved their exposition. She generalized the notion of game automata to infinite game automata and introduced the term "graph-games".

In order for material of this paper to be accessible by wider audience, we'll include below a summary of graph-games. Although a slightly different treatment of similar material appeared (e. g., Yakhnis-Yakhnis, 1990, 1993) in relation to games defined on trees, in the papers devoted to graph-games similar material is usually presented somewhat sketchy.

3.2. A Summary of Graph-Games

3.2.1. The Players and Their Moves

We start with the two player games and later will generalize them to multi-player games. Following Gurevich-Harrington (1982) (GH), we shall sometimes call the players *0* and *1*. In this case, if Ω is a player then $1-\Omega$ is the opponent. In some examples we'll call the players Spot and Stripe, and for chess we'll use the traditional White and Black.

Each player is associated with a finite alphabet, A for 0 (or Spot) and B for 1 (or Stripe). A player makes a move by grabbing a symbol from his/her alphabet and placing it to the right of the previous moves. We'll informally refer to the symbols from the alphabet associated with a player as the "moves" of that player.

3.2.2. The Game Graph

We'll now introduce the game graph. We follow Yakhnis A. (1990), Yakhnis-Yakhnis (1993), and Zeitman (1994) with some modifications.

- the game graph is a finite or infinite directed graph with a unique initial vertex;
- each vertex is labeled by a unique player;
- the initial vertex is labeled by the player making the first move;
- vertices with no outgoing edges are called leaves;
- for any vertex v which is not a leaf and which is labeled by a player Ω:
  - each edge outgoing from v is labeled by a unique move of Ω;
  - no two edges outgoing from v have the same label;

- each edge outgoing from v points at an edge labeled by 1-Ω.

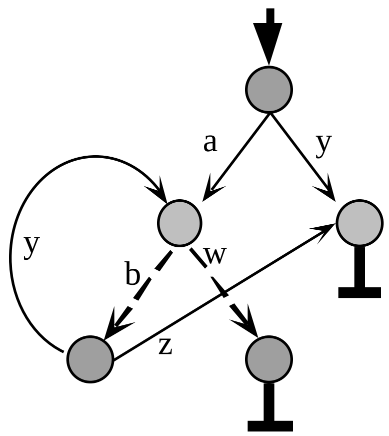

Fig. 2. A Game Graph for Spot playing with Stripe

3.2.3. Playing the Game as Running an STS

The game starts at the initial vertex by the player associated with the vertex. He/she makes a move by choosing one of the outgoing edges and moving along the edge toward the next vertex. Any such choice is considered to be legal. We also think that the move was made by the label of the chosen edge. Then the other player takes over, chooses an edge and moves to the next vertex. And so on, either until a leaf is met or ad infinitum. By describing all the legal moves, the game graph encodes the game rules.

The winner of the play is selected according to the winning condition associated with the game. Below we'll discuss how the winning conditions are formalized.

The game graph is associated with the following STS. Its states are the graph's vertices, the set of initial states consists of the initial vertex, the final states are leafs, the transition functions are moves and the transition rules are formed as follows. Given a move, say f, its applicability set is the set of all the vertices having an outgoing edge labeled by f. The process of playing the game corresponds to forming the system run.

3.2.4. Formal Plays, Game Trees and Compatible Game Graphs

The sequence of all the moves made during the play constitute a formal play. For example, it may look as a_0, b_1, a_2, ..., a_n, b_{n+1}, where a_0, a_2, ..., a_n are the moves of 0 and b_1, b_3, ..., b_{n+1} are the moves of 1. A string of symbols forming a prefix (i.e. finite initial segment) of a formal play is called a *position* of the game. Due to Gurevich-Harrington (1982), we call the set of all possible legal positions of the game the *game tree*. The tree structure is imposed by the usual properties of strings of symbols. For example, the *root* of the game tree is the empty string. It is easy to see that the game tree is a particular form of the game graph.

PROPOSITION. There is a unique homomorphism (of labeled directed graphs) from the game tree onto the game graph.

<u>Proof</u>: Map the empty string into the initial vertex. For each legal position of length one, say f there is a unique edge, say d, labeled by f. So, map f into the vertex pointed to by d. Etc., etc.

<u>EOP</u>

Obviously, two game graphs corresponding to identical game trees define the same game and encode the same game rules. We will call such game graphs *compatible*.

3.2.5. The Winning Condition

Formally speaking, a play is just a path in a game graph (either infinite or ending by a leaf). If G is the game graph, we'll designate the set of all its paths as Path(G). To indicate a winning condition for a player Ω, it is enough to identify the set of all such paths, say W (i. e., W Path(G)), where we assign the victory to Ω. Within the theory of two player games, it is usually assumed that the winning conditions of players are complementary. Thus, if W is the winning condition for Ω, then Path(G)–W (or W^c in another notation) is the winning condition for 1–Ω. Finally, in our terminology a game is a triple G, Ω, W (or, equivalently, G, 1–Ω, W^c), where W is the winning condition for the player Ω, the player making first move and W is his/her winning condition. We'll now discuss how to specify the winning conditions.

Gurevich and Harrington (GH) introduced the following notation in (Gurevich-Harrington, 1982). Let X be a subset of the set of all vertices of the game graph G. (Below we will abbreviate such statements by saying that X is a subset of G.) Then [X] designates the set of all paths in G (i. e., plays) intersecting with X infinitely often. In addition, in (Yakhnis, A. 1990 and Yakhnis, V. 1990), the set of all paths in G intersecting with X at least once was designated as (X). Now, let X_1,...,X_n be subsets of the game graph G. GH considered winning conditions in the form of a Boolean combination of the sets [X_1],...,[X_n]. Although a set of the form (X) may be designated as [Y] for some Y, when the finite plays are allowed it is convenient to explicitly add to the winning condition some Boolean combinations of (X_1),...,(X_n). We will still refer to such combined winning conditions as GH winning conditions and we will call the sets X_1,...,X_n their GH kernels. We expect that specifications of most of the practical systems would be covered by the above winning conditions.

For example, in chess if CheckMateB is the set of all board states where the Black King is under checkmate, then the winning condition for White is (CheckMateB). Suppose now that we have two processes, P_0 and P_1, and that we would like to write a monitor running these processes in perpetuity without starving either of them. If $ExecP_i$ is the set of all system states where a new instruction from P_i has completed its execution (for i = 0, 1), then $[ExecP_0] \cap [ExecP_1]$ is the winning condition for the monitor. Finally, recall from the introduction the discussion of a kind of winning conditions called "constraint". Each constraint has a form $(X)^c$, where X is some subset of the game graph G and "^{c}" is the complementation operator in Path(G).

3.2.6. Strategies

We'll need the following convenient notation. If v is a vertex on a game graph G, we designate as G(v) the set of *children* of v in G (i. e., all the vertices w with an edge going from v to w). In addition, if b is the label of an edge d outgoing from v, then G(v, b) designates the vertex at which d is pointing.

Let Ω be a player and Σ be its set of moves. We'll call a vertex labeled by Ω an Ω-vertex and we'll designate the set of all such vertices as Ver(Ω). We'll call a function f:Ver(Ω)→Σ a *deterministic Ω-strategy* if for every vertex v Ver(Ω) which is not a leaf, f(v) is a label of an edge outgoing from v. The set of vertices *consistent* with f is defined as follows.

- the root e is consistent with f;
- if v is consistent with f and v Ver(1–Ω) then all children of v are consistent with f;
- if v is consistent with f and v Ver(Ω) then G(v, f(v)) is consistent with f.

We say that a play is consistent with f if all its prefixes are consistent with f. We say that Ω *wins* A, B, G, Ω, W using f if every play consistent with f is in W. If Ω wins the game using f, we call f a winning Ω-strategy. Finally, we say that Ω *wins* A, B, G, Ω, W if there is a winning Ω-strategy.

In addition to deterministic strategies, it is sometimes convenient to use *nondeterministic strategies*, as was first demonstrated in (Gurevich-Harrington,1982) and later in (Yakhnis-Yakhnis, 1990). However, nondeterministic strategies are beyond the scope of this paper.

3.2.7. State-Strategies

We'll show how to deal with potentially infinitary nature of plays without having to memorize the entire prehistory of the play. The state strategies were first introduced in several works of Büchi and Büchi-Landweber (e. g., see Büchi, 1981; Büchi-Landweber, 1969). Independently, Gurevich-Harrington (1982) introduced "strategies with restricted memory" which fulfilled a similar purpose. Büchi and GH were first to show that for any game with GH winning condition there is a winning strategy with restricted memory. This result is sometimes called "restricted memory determinacy for two player games with GH winning conditions". These strategies were further developed in (Yakhnis, A., 1989; Yakhnis V., 1989; Yakhnis-Yakhnis, 1990, 1993; Nerode-Yakhnis-Yakhnis, 1992, 1993) where nondeterministic state-strategies and "strategies with restraints" were first introduced.

We will use here a modification of Büchi's strategies from (Yakhnis, A., 1989; Yakhnis V., 1989). A state strategy for a player Ω is an input-output automaton accepting moves of 1-Ω as an input and outputting moves of Ω. The input is used for memorizing some information about the play (limited by the memory of the automaton), whereas the output is used to guide the behavior of Ω during the play. This is illustrated on Figure 3, where Ω is treated as the player making the first move.

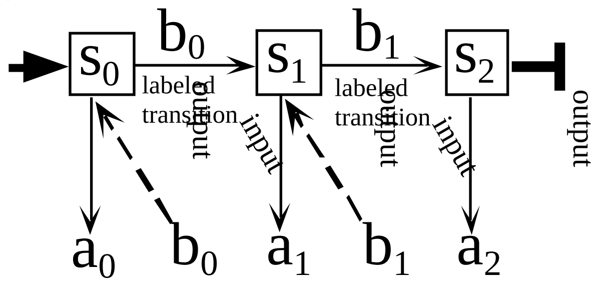

Fig. 3. Application of a state-strategy.

Practical state-strategies were also used in (Stilman, 1981, 1985, etc.) without developing their general theory. A significant difference in the usage of state-strategies between Büchi, GH, NYY on the one hand and Stilman on the other hand was that, whereas the former were looking for state-strategies always winning against every possible behavior of the opponent, the latter considered strategies with a significant likelihood of winning. In many practical situations the former strategies may have too high time complexity for a successful usage.

3.3. Registers

In (Yakhnis, A. 1990 and Yakhnis-Yakhnis 1993), it was shown how to combine smaller game graphs encoding different aspects of the game into one game graph by viewing them as automata (or STS in our present view). For now it is enough to know that intuitively such combination is equivalent to running the component STS simultaneously. Note that we already discussed similar combination within the context of the LG complex systems.

Just as for the LG complex systems, it is sometimes convenient to decompose the game graph into a combination of the "main" graph (which is the transition table of the "main" STS) and one or more *registers*. By a register we mean a simple automaton recording some distinctly separate aspect of the game. Combining back the main graph and the registers, as was done in (Yakhnis, A., 1990; Yakhnis-

Yakhnis, 1993), would give us a game graph compatible with the original game graph.

The most common register is Turn Register which tells whose turn it is to move. Informally, both the register and the main automaton are running during the play. The former tells whose turn it is to move whereas the latter tells which moves are available for this player. When the Turn Register is used, the definition of the main graph is the same as the one for the game graph, except that the requirement "each vertex is labeled by a unique player" may be omitted. This view allows the main graph to be significantly smaller than the game graph. For example, a combination of the main graph on figure 5 with the Turn Register on figure 4 would give us the game graph on figure 2.

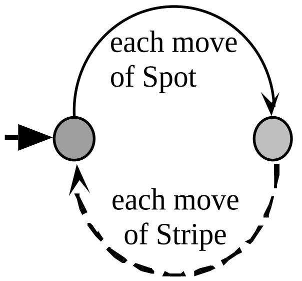

Fig. 4. The Turn Register for Spot playing with Stripe

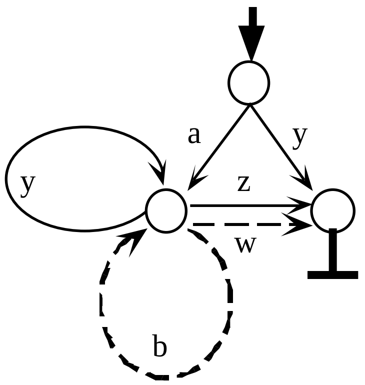

Fig. 5. The main graph for Spot playing with Stripe

3.4. Multi-Agent Graph-Games

We are going to generalize the two player graph games by allowing arbitrary number of players, simultaneous moves, and skipping of moves by players. In addition, we'll allow the players to have not mutually excluding winning conditions. Allowing simultaneous moves significantly changes the game environment. For example, the game determinacy result for two player games fails when simultaneous moves are allowed. To underline this difference, we will call the players "agents".

Now, assume that the agents A_0, ..., A_{m-1} are associated, respectively, with the alphabets Σ_0, ..., Σ_{m-1}. We will introduce two sets, LE, "labels for edges", and LV, "labels for vertices".

- LE = $(\Sigma_0\ \{skip_0\})\times \dots \times(\Sigma_{m-1}\ \{skip_{m-1}\}) - \{(skip_0, ..., skip_{m-1})\}$, where $skip_i$ is not in Σ_i for i = 0, ..., m–1;
- LV is the set of all nonempty increasing subsequences of 0, ..., m–1 with the addition that any integer in the subsequence may be enclosed in the brackets [ ]. For example, (0, 3, 5) or (2, [3], [5], 6) are allowed. Below the variables w and t will represent either bracketed or unbracketed integers.

Now we may define the multi-agent game graph as follows.

- it is a finite or infinite directed graph with a one or more initial vertices;
- each vertex is labeled by an element of LV;
- vertices with no outgoing edges are called leaves;
- for any vertex v which is not a leaf and which is labeled by a sequence $\sigma = w_0, ..., w_{k-1}$:
  - if t is in σ and t is unbracketed, then the label of each edge outgoing from v may not contain $skip_t$;
  - if i is in {0, ..., m–1} and neither i nor [i] is in σ, then the label of each edge outgoing from v must contain $skip_i$;
  - no two edges outgoing from v have the same label.

It is easy to understand the limitation on the behavior of the agents imposed by the above graph. They are confined to producing the system runs associated with the graph.

PROPOSITION. Each two-player game graph G may be converted into a two-agent graph G´ in such a way that the two graphs would define the same game.

Proof: For each edge d of G labeled by w do the following. If w is a move of 0, change the label into (w, $skip_1$). Otherwise, change the label into ($skip_1$, w). The result is G´.

EOP

THEOREM. The class of STS generated by the LG complex systems coincides with the class of STS generated by multi-agent game graphs, up to an isomorphism.

Proof: It is easy to show that a graph CS of a STS associated with an LG complex system may be relabeled by indices of agents and transitions to conform to the above definition while preserving the same behavior.

Let G be a multi-agent game graph. We'll define the following STS of an LG complex system. The board is the set of all vertices of G. Every agent is given a unique piece. At a move along an edge d labeled by σ, every nonskipping (according to σ) agent places his/her unique piece at the vertex pointed to by d. The skipping (according to σ) agents do nothing. The pieces placed at the previous move are removed.

EOP

4. CONCLUSION

We would like to answer the questions formulated in the section “The Winning Conditions and (Almost) Winning Strategies”. Firstly, we note that the graph-games and the LG complex systems are closely connected via the multi-agent graph-games, and that the GH winning conditions are, probably, the most general winning conditions solvable by computable winning strategies. Therefore we would like progress by enabling the LG methods to solve increasingly complicated winning conditions within the framework of the GH winning conditions.

At the moment the LG solves the winning conditions of the form “reach a set of states S_{trg}”, where S_{trg} is defined by a formula describing certain relations between the locations of the pieces on the board. Thus in our formal notation the winning condition is (S_{trg}). It appears that LG systems may also satisfy the constraints in the form $(X)^c$, where X is also defined by a formula describing states of the board, see (Botvinnik, 1984). (Recall that fulfilling $(X)^c$ means “you must never reach a state from the set X”.) The subject for a future investigation is to achieve more complicated Gurevich-Harrington winning conditions in the form [X].

Secondly, we note that both LG and several methodologies solving GH winning condition employ state-strategies. There are some other tantalizing similarities between some winning strategies and LG, although they were developed completely independently. For example, LG employs state-strategies based on the shortest paths, which is similar to Gurevich-Harrington's DecreaseRank strategies (see Gurevich-Harrington 1982 and Yakhnis-Yakhnis 1990). In addition, LG employs state-strategies based on a hierarchy of formal grammars which has a certain similarity in structure to the functional computing winning strategies in (Yakhnis-Yakhnis 1990). In a future work these similarities will be further investigated.

ACKNOWLEDGMENTS

We would like to express our gratitude to Professor Anil Nerode for his encouragement and support. Many thanks to Alex Yakhnis for discussing the subject matter of this paper and for suggesting several improvements.

REFERENCES

Berge, C. (1957) Topological Games with Perfect Information (Contributions to the Theory of Games 3), *Annals of Math. Studies*, Vol. 39, p. 165, 1957.

Botvinnik, M.M. (1984) *Computers in Chess: Solving Inexact Search Problems.* Springer Series in Symbolic Computation, Springer-Verlag: New York, 1984.

Büchi, J. R. (1981) Winning State-Strategies for Boolean-F_S Games, a manuscript, 1981.

Büchi, J. R. (1983) State-Strategies for Games in F_{sd}«G_{ds}, *The Journal of Symbolic Logic*, Vol. 48, No 4, Dec. 1983.

Büchi, J. R., Landweber, L. H. (1969) Solving Sequential Conditions by Finite State Strategies, *Trans. of the Amer. Math. Soc.*, Vol. 138, pp. 295-311, 1969.

Davis, M. (1964) Infinite Games of Perfect Information, *Annals of Mathematical Studies*, Vol. 52, pages 85-102, 1964.

Gale, D., Stewart, F. M. (1953) Infinite games with perfect information, *Contributions to the theory of games*, *Ann. of Math. Studies*, No. 28, Princeton Univ. Press, pp. 245-266, 1953.

Gurevich, Y., Harrington, L. (1982) Trees, Automata and Games, *Proc. of the 14th Annual ACM Symposium on Theory of Computing*, pp. 60-65, 1982.

Hopcroft, J., Ullman, J. (1979) *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

Kohn, W., Nerode, A. (1993) Models for Hybrid Systems: Automata, Topologies, Controllability, Observability, in *Hybrid Systems*, R. Grossman et al eds., *Lecture Notes in Computer Science 736*, Springer-Verlag, pp. 317-356, 1993.

Landweber, L. H. (1973) Exposition of [Büchi-Landweber 1969], in B.A. Trakhtenbrot and Ya. M. Barzdin, *Finite Automata*, pp. 113-126, North-Holland, 1973.

Martin, D. A. (1985) A purely inductive proof of Borel determinacy, *Proc. of Symp. in Pure Mathematics*, Vol. 42, 303-308, 1985.

McNaughton, R. (1993) Infinite Games Played on Finite Graphs, *Annals of Pure and Applied Logic*, Vol. 65, pp. 149-184., 1993.

Morris, J. M. (1979) A Starvation Free Solution to the Mutual Exclusion Problem, *Information Processing Letters*, Vol. 8, pp. 76-80, 1979.

Nerode, A., Remmel, J. B., Yakhnis, A. (1993) Hybrid System Games: Extraction of Control Automata with Small Topologies, Mathematical Sciences Institute, Cornell University, Technical Report 93-102, 61 pp., 1993.

Nerode, A., Yakhnis, A. (1992) Modeling Hybrid Systems as Games, *Proceedings of the 31st IEEE Conference on Decision and Control*, pp. 2947-2952, 1992.

Nerode, A., Yakhnis, A., Yakhnis, V. (1992) Concurrent Programs as Strategies in Games, in *Logic From Computer Science*, MSRI series (Y.

Moschovakis, ed.), pp. 405-479, Springer-Verlag, 1992.

Nerode, A., Yakhnis, A., Yakhnis, V. (1993) Distributed Concurrent Programs as Strategies in Games, in *Logical Methods* (J. N. Crossley et al, ed.), pp. 624-653, Birkhauser, 1993.

Peterson, J. L., Silberschatz, A. (1985) *Operating System Concepts*, Addison–Wesley, 1985.

Stilman, B. (1981) Formalization of PIONEER Method Employing Theory of Formal Grammars, Tech. Report, All-Union Research Inst. for Electrical Engineering, Moscow, Russia, 105 pp., (in Russian).

Stilman, B. (1985) A Hierarchy of Formal Grammars for Solving Search Problems, Artificial Intelligence. Results and Prospects, Moscow, pp. 63-72, (in Russian).

Stilman, B. (1992) A Linguistic Geometry of Complex Systems, Abstracts of the Second International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, FL, USA, Jan. 1992.

Stilman, B. (1993a) Network Languages for Complex Systems, *An International Journal: Computers & Mathematics with Applications*, Vol. 26, No. 8, pp. 51-80, 1993.

Stilman, B. (1993b) A Linguistic Approach to Geometric Reasoning, *An International Journal: Computers & Mathematics with Applications*, Vol. 26, No. 7, pp. 29-58.

Stilman, B. (1993c) A Formal Language for Hierarchical Systems Control, *Languages of Design*, Vol.1, No.4, pp.333-356,1993.

Stilman, B. (1994a) A Formal Model for Heuristic Search, Proc. of the 22nd Annual ACM Computer Science Conf., pp. 380-389, Phoenix, AZ, USA, March 8-12, 1994.

Stilman, B. (1994b) Heuristic Networks for Space Exploration, Telematics and Informatics, *An Int. Journal on Telecommunications & Information Technology*, Vol. 11, No. 4, pp. 403-428, 1994.

Stilman, B. (1994c) Translations of Network Languages, *An International Journal: Computers & Mathematics with Applications*, Vol. 27, No. 2, pp. 65-98, 1994.

Stilman, B. (1995a) Deep Search in Linguistic Geometry, Symposium on Linguistic Geometry and Semantic Control, *Proc. of the First World Congress on Intelligent Manufacturing: Processes and Systems*, Mayaguez, Puerto Rico, Feb. 1995.

Stilman, B. (1995b) Multiagent Air Combat with Concurrent Motions, Symposium on Linguistic Geometry and Semantic Control, *Proc. of the First World Congress on Intelligent Manufacturing: Processes and Systems,* Mayaguez, Puerto Rico, Feb. 1995.

Von Neumann, J., Morgenstern, O. (1944) *Theory of Games and Economic Behavior*, Princeton University Press, 1944.

Yakhnis, A. (1989) Concurrent Specifications and their Gurevich-Harrington Games and Representation of Programs as Strategies, *Transactions of the 7th (June 1989) Army Conference on Applied Mathematics and Computing*, pp. 319-332, 1990.

Yakhnis, A. (1990) Game-Theoretic Semantics for Concurrent Programs and Their Specifications, Ph. D. thesis, Cornell University, August 1990.

Yakhnis, A., Yakhnis, V. (1990) Extension of Gurevich-Harrington's Restricted Memory Determinacy Theorem: a Criterion for the Winning Player and an Explicit Class of Winning Strategies, *Annals of Pure and Applied Logic*, Vol. 48, pp. 277-297, 1990.

Yakhnis, A., Yakhnis, V. (1993) Gurevich-Harrington's Games Defined by Finite Automata, *Annals of Pure and Applied Logic*, Vol. 62, pp. 265-294, 1993.

Yakhnis, A., Yakhnis, V., A Model of Object-Oriented Analysis and Design Tailored Toward Stepwise Refinement, in preparation.

Yakhnis, V. (1989) Extraction of Concurrent Programs from Gurevich-Harrington Games, *Transactions of the 7th (June 1989) Army Conference on Applied Mathematics and Computing*, pp. 333-343, 1990.

Yakhnis, V. (1990) Concurrent Programs, Calculus of State-Strategies and Gurevich-Harrington Games, Ph. D. thesis, Cornell University, August 1990.

Yakhnis, V., Winning Semi-Finite Games with a Unique Finite State-Strategy, in preparation.

Zeitman, S. (1994) Unforgettable forgetful determinacy, *Logic and Computation*, Vol. 4, pp. 273-283, 1994.