A Multi-Agent Graph-Game Approach to Theoretical Foundations of Linguistic Geometry

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 by 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 participating 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.

Keywords: Linguistic Geometry, Search Algorithms, Multiagent Systems, State Transition Systems, Graph-Games, Winning Strategies

1. INTRODUCTION

1.1. "Competing Agents" View on State Transition 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 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 Nerode-Kohn-Yakhnis (NKY) approach [Kohn, Nerode 1993; Nerode, Remmel, Yakhnis 1993; Nerode, Yakhnis 1992].

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

* Supported by the U.S. Army Research Office Through the Mathematical Sciences Institute of Cornell University.

strategy guiding its behavior. All the above approaches concentrate on finding such strategies.

1.2. Computational Power of Linguistic Geometry

The purpose of this paper is to investigate the foundations for the Linguistic Geometry (LG) approach, so that the similarities and the differences between it and the other approaches exploiting the idea of "several competing/cooperating agents" would be well understood. Experiments comparing the AI search 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.

2. MULTI-AGENT GRAPH-GAMES

2.1. Origins of Multi-Agent Graph-Games

We are going to generalize the alternating 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". We represent the rules of the game via a labeled directed graph (which we call *multi-agent game-graph*.) and, therefore we call our games *multi-agent graph-games*. We gave a brief introduction into the *multi-agent graph-games* in our earlier work [Yakhnis, Stilman 1995], where the origins of the notion of the alternating two player graph games were discussed at some length. Here we would like to list the works we are building upon. They are [Berge 1957; Büchi 1981; Büchi, Landweber 1969; Gurevich, Harrington 1982; Yakhnis, A., Yakhnis V. 1990, 1993; McNaughton 1993; Zeitman 1994].

2.2. The Agents and Their Moves

We call the agents (i.e., players) A_0, ..., A_{m-1}. In case of two 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 two players Spot and Stripe, and for chess we'll use the traditional White and Black.

There is a disjoint collection of finite alphabets Σ_0, , Σ_{m-1}. Each symbol of the alphabets represents an action changing the state of the game. Each agent (i.e., player) A_i

is associated with the alphabet, Σ_i. We'll informally refer to the symbols from the alphabet associated with an agent as the *singleton-moves* of that agent. A *packet-move* is a vector f = (w_0, , w_{m-1}), where w_i is a vector of singleton-moves from Σ_i. The vector w_i is called the *component-move* of A_i in t. The game proceeds as a series of packet-moves as follows:

- The agents make a packet-move by each agent grabbing a (possibly empty) vector (i.e., finite sequence) of symbols from his/her alphabet and placing it into a packet-move, which, in turn, is placed to the right of the previous moves;
- The resulting sequence of packet-moves is called a play;
- The collections of symbols that a player is allowed to grab at any given state of the game are controlled by the game rules;
- A packet-move results in application to the game state of all the actions in the packet;
- The rules sometimes give an agent an option to skip a move even when a non-empty move is permitted for the agent;
- The rules are given by the game graph .

2.3. Multi-Agent Game-Graph

For the graph below we need two sets, PM, "packet-moves" for labeling edges, and PR, "packet-rules" for labeling vertices.

- PM = $\mathbb{V}\Sigma_0 \times \ldots \times \mathbb{V}\Sigma_{m-1}$, where $\mathbb{V}\Sigma_i$ is the set of all vectors (i.e., finite sequences) from Σ_i for i = 0, ..., m–1. PM is the set of all possible packet-moves. We do not differentiate between the empty set and the null finite sequence and we designate the empty packet-move (, ...,) by ε;
- PR = $\mathbb{P}^+\mathbb{V}\Sigma_0 \times \ldots \times \mathbb{P}^+\mathbb{V}\Sigma_{m-1}$, where $\mathbb{P}^+X$ is the set of all non-empty subsets of X. We call the elements of PM the packet-rules. We designate the empty packet-rule ({ }, ..., { }) by Λ. This rule forbids making moves from the respective vertices.

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 PR and each edge is labeled by an element of PM;
- each vertex labeled by Λ is a leaf;
- for any vertex v labeled by a packet-rule (t_0, ..., t_{m-1}) $\neq \Lambda$:
  - each edge outgoing from v is labeled by a unique packet-move from $t_0 \times \ldots \times t_{m-1}$;
  - for each packet-move from $t_0 \times \ldots \times t_{m-1}$ there is a unique edge outgoing from v labeled by it;
  - if ε $t_0 \times \ldots \times t_{m-1}$, then the edge labeled by ε points to v.

PROPOSITION A vertex is a leaf if and only if it is labeled by Λ.

For alternating games it is sufficient to label each vertex by the player making moves from it. For example the game-graph on Figure 1 represents a strictly alternating two-player game without skips. The vertices are labeled by spots or stripes, the alphabet for Spot is {a, y, z} and the alphabet for Stripe is {b, w}.

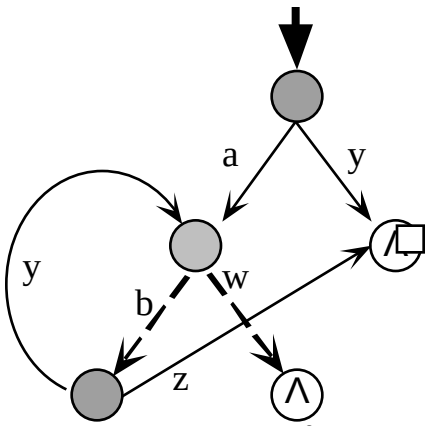

Fig. 1. A Game Graph for Spot playing with Stripe

2.4 Playing the Game as Running a State Transition System (STS)

Now we can give a rigorous definition of the game on a multi-agent game-graph. At each moment of the game exactly one vertex would be declared as "current" (we call it the current state of the game). The game starts at an initial vertex (i.e., it is declared "current" at the start). Assume that the current vertex v is labeled by a packet-rule $(t_0, ..., t_{m-1}) \neq \Lambda$. Each agent A_i chooses an element $w_i \ t_0$ (i.e., w_i would be a collection of singleton moves from Σ_i) as a component-move, thus forming a packet-move $p = (w_0, ..., w_{m-1})$. After this packet-move is completed, the new current state of the game is defined as the vertex pointed to by the unique edge d outgoing from v such that d is labeled by p. Then the process is repeated, either until a leaf is met or ad infinitum.

The winner of the play is selected according to the winning conditions associated with the agents. 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 vertices, the final states are leafs, the transition functions are packet-moves and the transition rules are formed as follows. Given a packet-move, say f, its applicability set is the set of all the vertices having an outgoing edge labeled by f. The above process of playing the game corresponds to forming the system run.

2.5 Formal Plays, Game Trees and Compatible Game Graphs

The sequence of all the packet-moves made during the play associated with an initial vertex v_0 constitute a formal play. For example, for an alternating two player game 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 packet-moves 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 (associated with an initial vertex v_0) onto the game graph with the root mapped into v_0.

Proof: Map the empty string into v_0. 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. □

We call such a homomorphism the game graph covering map (GGC). 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*.

2.6. The Winning Condition

Formally speaking, a play is just a path in a game tree (either infinite or ending by a leaf). If T is the game tree, we'll designate the set of all its paths as Path(T). To indicate when an agent Ω wins, it is enough to identify the set of all such paths in Path(T), say W, called the winning set for Ω, where we assign the victory to Ω. Within the theory of two player games, it is usually assumed that the winning sets of players are complementary (Gale-Stewart 1953). Such games are called win-loose games. For general multi-agent graph games it is usually not the case (e.g., in chess no-win situation is possible). Thus, in our games each agent A_i is associated with its winning set W_i. We call the list $(A_0, W_0; ...; A_{m-1}, W_{m-1})$ the combined winning condition. Finally, in our terminology a game is a pair $\langle G, AW \rangle$, where AW is the combined winning condition.

Gurevich and Harrington (GH) (1982) introduced a very convenient notation for the winning sets in terms of game trees. We'll restate their notation in terms of game graphs. Let T be the game tree, G be the game graph, and $h: T \to G$ be the GGC. Let Q be a subset

of the set of all vertices of G. (Below we will abbreviate such statements by saying that Q is a subset of G.) Then [Q] designates the set of all paths in T intersecting with $h^{-1}(Q)$ infinitely often. In addition, in (Yakhnis, A., 1990; Yakhnis, V., 1990), the set of all paths in T intersecting with $h^{-1}(Q)$ at least once was designated as (Q). Now, let $Q_1,...,Q_n$ be subsets of the game graph G. GH considered winning sets in the form of a Boolean combination of the sets $[Q_1],...,[Q_n]$. A set of the form (Q) may often be represented as [Y] for some Y. However, when the finite plays are allowed, it is convenient to explicitly add to the winning set some Boolean combinations of $(Q_1),...,(Q_n)$. We will still refer to such combined winning sets as GH winning sets and we will call the sets $Q_1,...,Q_n$ 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 $(Q)^C$, where Q is some subset of the game graph G and "^{C}" is the complementation operator in Path(G).

2.7. 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 A_i be an agent associated with its winning set W_i and a set of moves Σ_i be its set of moves. We'll call a vertex which label includes A_i an A_i-vertex and we'll designate the set of all such vertices as $Ver(A_i)$. We'll call a function $f:Ver(A_i) \to \Sigma_i$ a *deterministic* A_i-*strategy* if for every vertex v $Ver(A_i)$ 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 $Ver(A_i)$ then all children of v are consistent with f;
- if v is consistent with f and v $Ver(A_i)$ 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 A_i *wins* G, W using f if every play consistent with f is in W. If A_i wins the game using f, we call f a winning A_i-strategy. Finally, we say that A_i *wins* G, W if there is a winning A_i-strategy. However since for multi-agent graph-games the usual determinacy theorem s do not apply, we are more concerned with winning individual plays, rather than the whole game.

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.

2.8. State-Strategies

We'll show how to deal with potentially infinite 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 A_i is an input-output automaton accepting the packet-moves of as an input and outputting moves of A_i . 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 A_i during the play.

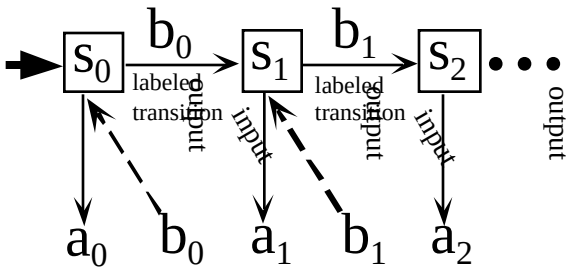

Fig. 2. Application of a state-strategy.

A state-strategy for alternating two player games is illustrated on Figure 2.

As was stated in [Yakhnis, Stilman 1995], the "complex systems" defined in [Stilman 1981, 1985, 1992] are in fact multi-agent graph-games and the strategies defined there the state-strategies intended to win individual plays (they are not the winning strategies in the above sense). Thus, our long-range intention is to study those strategies. In many practical situations the winning strategies may have too high time complexity for a successful usage.

2.9. 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.

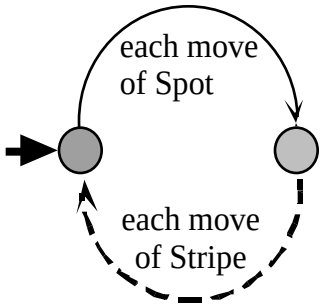

Fig. 3. The Turn Register for Spot playing with Stripe

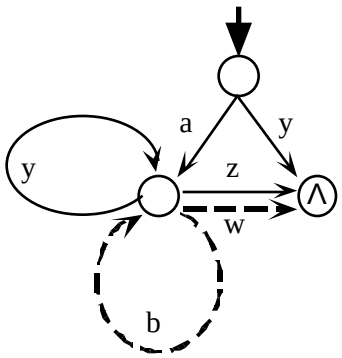

Fig. 4. The main graph for Spot playing with Stripe

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, A., Yakhnis, V., 1993], would give us a game graph compatible with the original game graph. In addition, registers (such as Castling Register) were used in practical applications of the LG Complex Systems developed by B. Stilman in 1980s.

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 4 with the Turn Register on Figure 3 would give us the game graph on Figure 1.

3. THE LINGUISTIC GEOMETRY (LG) APPROACH

3.1. An intuitive View on the LG

Linguistic Geometry (LG) [Stilman 1981, 1985, 1992, etc.] is an approach to finding strategies likely to win individual plays for multi-agent graph-games using the following three features:

- each state has a rich geometrical structure such as board, pieces, reachability relations, etc. (hence, the "Geometry" portion of the name);
- certain formal grammars are utilized for convenient building of state-strategies (hence, the "Linguistic" portion of the name);
- formalization of search heuristics of the highly-skilled human experts as a major mechanism for forming state-strategies. This promotes the decomposition of a complex control system into a dynamic hierarchy of subsystems, thus solving intractable problems by dramatically reducing the search.

In addition, since the process of playing a multi-agent graph-games may be understood as running the corresponding STS, LG may also be viewed as an approach to control the behavior of STS with a particular structure of states. In our previous work [Yakhnis, Stilman 1995] we discussed the origins of LG and provided a slightly more advanced description (in comparison with [Stilman 1981, 1985, 1992]) of the geometrical structure of the current state of the game called "complex system". In this paper we are recapturing the definition of the complex systems from [Yakhnis, Stilman 1995]; in our future papers we will provide a new outlook on the formal grammars from [Stilman 1981, 1985, 1992, etc.] using such notions as first order theories and algebras with explicit types, STS, and objects (see [Yakhnis, V., Yakhnis, A., to appear]). Also, in our next paper we will discuss the notion of "shortest trajectories" as the simplest human heuristics from LG, leaving more complicated heuristics for a future work.

3.2. The LG Complex Systems

3.2.1. The Quadruple

An LG Complex System, say Φ, is the following quadruple:

$$\Phi = \mathbf{E}, \Gamma, \mathbf{Reg}, \mathbf{AW}$$

where

- **E** The environment algebra, where the board **X** and the set of pieces **P** are among the sorts and the reachability relation **R** and the "worth" of pieces function **v** are in the signature;
- Γ the STS built from **E** as described in the following sections;
- **Reg** the register (see Section 2.9);
- **AW** the winning condition of the 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 his/her winning condition for i = 0, ..., m–1.

We will show that in combination, Γ and **Reg** define a multi-agent game graph G. Thus a Complex System contains a multi-agent graph game G, **AW** as a component.

3.2.2. The Environment Algebra E

The algebra **E** contains at least the following sorts:

- **X** the space of locations $\{x_0, ..., x_{k-1}\}$ for placing the pieces upon, also called the *board*;
- **A** the set of agents $A_0, ..., A_{m-1}$;
- **P** the set of all pieces represented as a disjoint union of collections of sets of pieces $\mathbf{P_0}, ..., \mathbf{P_{m-1}}$ assigned, respectively, to agents $A_0, ..., A_{m-1}$;
- $\mathbf{P_i}$ the set of all pieces assigned to agent A_i, where i = 1, ..., m–1;
- $\mathbb{Z}$ the set of all integers;
- **DISP** the set $\{d: \mathbf{X} \to \mathbb{P}\mathbf{P} \mid x,y \in \mathbf{X}. x \neq y \Rightarrow d(x) \cap d(y) = \emptyset\}$ of all possible placements of the pieces on the board, where a piece may occupy at most one location. It serves as the set of states for Γ and we will call its elements "the states of the board".
- **S** the set of states for **Reg**. We do not have any assumption about its structure.

The algebra **E** contains at least the following relations and functions:

- **R** $\subseteq \mathbf{P}\times\mathbf{X}\times\mathbf{X}$ the reachability relation, where **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 z is unoccupied and if q is not a Knight then any piece placed between y and z serves as an obstacle. The statements describing the absence of obstacles are included within the applicability conditions for the transition rules;
- **v**: $\mathbf{P} \to \mathbb{Z}$ the "worth" of pieces function assigning to each piece an integer value;
- **Agent**: $\mathbf{P} \to \mathbf{A}$ the function assigning to each piece its agent;
- **Apply**: $\mathbf{DISP}\times\mathbf{X} \to \mathbb{P}\mathbf{P}$ the application of elements of **DISP** function, where **Apply**(d, x) = d(x). This function allows to treat **DISP** as a set of functions while preserving the first order properties of **E**. We would usually abbreviate **Apply**(d, x) as d(x);

The algebra **E** contains at least the following constants:

- names for all the locations: $x_0, ..., x_{k-1}$;
- names for all the agents: $A_0, ..., A_{m-1}$;
- names for all the pieces: $p_{i,0}, ..., p_{i,n_i}$, where i = 1, ..., m–1.

In addition to the elements of signature introduced above, **E** may have some additional structure:

- **Board:** In chess (where locations are sometimes called squares) each square not on the border of the board 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.
- **Pieces:** For robotic vehicles we may have a subset Tanks with pieces Patton (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).

The algebra **E** satisfies at least the following axioms:

- $P_i = \{p_{i,0}, ..., p_{i,n_i}\}$ for i = 0, ..., m–1;
- $\mathbf{P_0}, ..., \mathbf{P_{m-1}}$ are disjoint;

- **P** = $_{0\le i<m}\mathbf{P_i}$;
- **X** = $\{x_0, ..., x_{k-1}\}$;
- d **DISP** x,y **X**. x≠y d(x)∩d(y) = ;
- **Agent**($p_{i,j}$) = A_i for i = 0, ..., m–1 and j = 0, ..., n_i.

3.2.3. The STS Γ and the Register Reg

3.2.3.1. The States

Both Γ and **Reg** are finite state machines, but, whereas **Reg** has very simple structure and is intended to keep track of some simple events, like turns or castling of Black and White Kings, Γ includes a multitude of information about the board, the pieces, etc. The state structure of Γ and **Reg** is as follows:

	Γ	**Reg**
The Set of States	**DISP**	**S**
The Set of Initial States	**DISP$_{in}$** **DISP**	**S$_{in}$** **S**
The Set of Final States	**DISP$_{fin}$** **DISP**	**S$_{fin}$** **S**

3.2.3.2. The Singleton-Moves, the Component-Moves, and the Packet-Moves

A singleton-move of an agent is the smallest individual action allowed for the agent to perform on the board. The collection of all possible singleton-moves of an agent corresponds to the alphabet associated with an agent in terms of graph-games. For the LG Complex Systems, there are the following categories of singleton-moves:
- removal of a piece from a location;
- placement of a piece in a location;
- changing the state of a piece (when pieces are viewed as state machines);
- changing the state of a location (when locations are viewed as state machines).

In this paper we will discuss only examples of the removal and placement of pieces, leaving the two other kinds of singleton-moves for the future works. Designating the current state of the board as D, we will represent the removal of a piece p from a location x as the assignment `D(x) := D(x) – {p}`, and the placement of a piece q in a location y as the assignment `D(y) := D(y) {q}`. By their definition, the singleton-moves may be regarded as partial unary operations on **DISP**. We assume that they may be also regarded as partial unary operations on **S**.

A component-move is a (possibly empty) finite sequence of singleton-moves. Given a component-move w = (h_0, ..., h_{k-1}), we associate it with a partial unary operation $h_{k-1}\circ ...\circ h_0$ (or h_0; ...; h_{k-1} in terms of the ";"-connector). Assume that we have two agents, Stripe and Spot. We would like to write down the following component-move g of Stripe "relocate a Stripe's piece q_0 from a location y_0 to a location y_1 and simultaneously remove a Spot's piece q_1 from the location y_1". It could look as the following sequence of singleton-moves:

```
D(y0) := D(y0) – {q0};
D(y1) := D(y1) – {q1};
D(y1) := D(y1)   {q0}.
```

A packet-move of the agents A_0, ..., A_{m-1} is a vector f = (w_0, ..., w_{m-1}), where t_i is a component-move of the agent A_i. We assume that somehow f may be regarded as a partial unary operation on **DISP** and on **S** constructed via operations w_0, ..., w_{m-1} and

that f(D) and f(s) are defined if and only if $w_i(D)$ and $w_i(s)$ are defined for all i = 0, ..., m-1.

3.2.3.3. The Transition Rules **TR**

Now we'll define the set of transition rules **TR**. Each transition rule is a pair of the form (S_w, w), where w is a component-move and S_w is a subset of **DISP×S**, called the *applicability* set for w. Below we represent a current state from **DISP** as D and the current state from **S** as s. We also say that a transition rule (S_w, w) (or a component-move w) is *applicable* to a combined state (D, s), if (D, s) S_w. Finally, we say that a packet-move f is applicable to (D, s) if all its component-moves are applicable to (D, s).

For example, let us describe the applicability set for the component-move g in the previous subsection. Assume that we have a turn register and that a player Ω may make a move in a register's state s if and only if the predicate Turn (Ω, s) holds. Also assume that the predicate Obstacle(p, x, y) holds if and only if there is an obstacle for p between x and y. Then the set S_g may be defined by the following predicate:

$$\text{Turn}(\text{Stripe}, s) \quad q_0 \quad D(y_0) \quad \mathbf{R}(q_0, y_0, y_1) \quad \neg\,\text{Obstacle}(q_0, y_0, y_1) \quad q_1 \quad D(y_1)$$

We now can formally define a combined system run for Γ and **Reg**. It is a finite or infinite alternating sequence of pairs from **DISP×S** and the packet-moves of the form (D_0, s_0), f_0, (D_1, s_1), f_1, ..., (D_i, s_i), f_i, ... and such that:

- the leftmost pair is a pair of initial states;
- if a pair in the sequence contains a final state, the sequence is finite and the pair containing the final state is the rightmost pair;
- 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 the preceding transition function to the previous state according to the transition rules as follows. Suppose that (D_i, s_i), f_i, (D_{i+1}, s_{i+1}) are in the sequence. Then f_i is applicable to (D_i, s_i) and $D_{i+1} = f_i(D_i)$ and $s_{i+1} = f_i(s_i)$.

In order for the system run not to be abnormally terminated, we require that the transition rules must satisfy two crucial properties, namely, *validity* and *completeness*:

- Validity:

  for each transition rule (S_w, w), if (D, s) S_w, then D Dom(w) and s Dom(w);
- Completeness:

  if (D, s) is a reachable pair, then either one of (D, s) is a final state, or there is a transition rule, say (S_w, w), with non-empty w, such that (D, s) S_w.

3.2.4. The LG Complex Systems as Multi-Agent Graph-Games

In order to verify that the combined system runs correspond to plays in an multi-agent graph-game, we would like to show how to create vertices, edges, and labels for the game graph from the descriptions of Γ and **Reg**. We say that the pair (D, s) is a *reachable pair* if it belongs to some combined system run. The set of vertices of the game-graph G is exactly the set of all reachable pairs (D, s). Since, given a packet-rule $(t_0, ..., t_{m-1})$ labeling a vertex v of a game-graph, there is a 1-1 correspondence between the edges outgoing from v and $t_0 \times ... \times t_{m-1}$, we would completely define the game-graph G if:

1) given a pair (D, s) we would define its label $(t_0, ..., t_{m-1})$;
2) given f $t_0 \times ... \times t_{m-1}$, we would define f as a partial unary operation f: **DISP** → **DISP** and f: **S** → **S** and show that D Dom(f) and s Dom(f).

In addition, in order to comply with the definition of the system run given in the previous section, we would have to show that:

3) given a pair (D, s), it is labeled by Λ if and only if one of D, s is a final state.

The rest is obvious.

ACKNOWLEDGMENTS

We would like to express our gratitude to Professor Anil Nerode for his encouragement and support. Many thanks to Dr. Alex Yakhnis for helpful discussions.

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_σ Games, a manuscript, 1981.

Büchi, J. R. (1983) State-Strategies for Games in $F_{\sigma\delta} \cap G_{d\sigma}$, *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.

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.

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.

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 forHierarchical 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.
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 (June1989) Army Conference on Applied Mathematics and Computing*, pp.333-343, 1990.
Yakhnis, V., Stilman, B. (1995a) Foundations of Linguistic Geometry: Complex Systems and Winning Conditions, *Proceedings of the First World Congress on Intelligent Manufacturing Processes and Systems (IMP&S)*, February 1995.
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.