Grammatical Inference Methodology for Control Systems

ABOUBEKEUR HAMDI-CHERIF [(1)], CHAFIA KARA-MOHAMMED (*alias* HAMDI-CHERIF)[(2)]
Computer Science Department
Qassim University
PO Box 6688 – 51452 Buraydah
SAUDI ARABIA

[(1),(2)] Permanent Address : Université Ferhat Abbas Setif (UFAS)
Faculty of Engineering
Computer Science Department
19000 Setif
ALGERIA
[(1),]email: shrief@qu.edu.sa , elhamdi62@gmail.com
[(2),]email: smhmd@qu.edu.sa , chafikmo@yahoo.com

Abstract: - Machine Learning is a computational methodology that provides automatic means of improving programmed tasks from experience. As a subfield of Machine Learning, Grammatical Inference (GI) attempts to learn structural models, such as grammars, from diverse data patterns, such as speech, artificial and natural languages, sequences provided by bioinformatics databases, amongst others. Here we are interested in identifying artificial languages from sets of positive and eventually negative samples of sentences. The present research intends to evaluate the effectiveness and usefulness of grammatical inference (GI) in control systems. The ultimate far-reaching goal addresses the issue of robots for self-assembly purposes. At least two benefits are to be drawn. First, on the epistemological level, it unifies two apparently distinct scientific communities, namely formal languages theory and robot control communities. Second, on the technological level, blending research from both fields results in the appearance of a richer community, as has been proven by the emergence of many multidisciplinary fields. Can we integrate diversified works dealing with robotic self-assembly while concentrating on grammars as an alternative control methodology? Our aim is to answer positively this central question. As far as this paper is concerned, we set out the broad methodological lines of the research while stressing the integration of these different approaches into one single unifying entity.

Key-Words: - Machine learning, robot control, grammatical inference, graph grammar, formal languages for control, self-assembly, intelligent control, emergent control technologies.

1 Introduction

For the purpose of the present work, Grammatical Inference (GI) is understood as the learning of the syntax of a given language. As a general computational method, GI is therefore the process whereby a language is *automatically* generated from positive and eventually negative examples of sentences [1]. On the hand, assembling geometrical shapes into whatever desired shape is a difficult control problem. More specifically, self-assembly is the process in which a disordered system of pre-existing shapes or components forms an organized structure or pattern as a consequence of specific, local interactions among the components themselves, without external direction. It is therefore a phenomenon in which a collection of particles spontaneously arrange themselves into a coherent structure. In nature, self-assembly is ubiquitous. For example, cell membranes, and tissues are self-assembled from smaller components in a decentralized fashion. It is common to encounter, in the natural world members of decentralized systems that self-organize in response to environmental stimuli and to each other to produce complex global behaviors. This is referred to as flocking. Birds and bacteria group behavior are among the most common examples. Flocking has been used as a metaphor for the study and development of artificial swarm intelligence-based systems. Self-assembly, as a facet of flocking is beginning to find its way into science and engineering, through various disciplines ranging from molecular application encountered in bioinformatics [24], to robot reconfiguration [25], and stochastic self-assembly [15], among others.

Assembling shapes into a given pattern can be seen as a “language” where the small shapes are the “words” and the obtained pattern correspond to a

"sentence" obeying some specific rules or "grammar" for generating grammatically correct sentences. The process of self-assembly can therefore be seen as the automatic generation of a language.

One of the central questions for robotic self-organized systems is to know whether it is possible to synthesize a set of local controllers that produce a prescribed global behavior that is sufficiently robust to uncertainties about the environmental conditions. Since assembling geometrical shapes into some desired shape can be viewed as a set of sentences of a language, it is therefore not surprising to address this issue from the standpoint of grammars. By grammar we mean the set of rules that are used to generate a given language. More precisely, we propose to make use of grammatical inference (GI). Ultimately graph grammars are used as an emerging field that we believe to be promising [17]. The paper is organized as follows. In Section 2, the issue of controlling robots is addressed with concentration on self-assembly using GI. Section 3 describes the methodological steps to follow in order to solve the self-assembly problem using grammars, as an ultimate result of the actual work. In Section 4, results and discussions are presented. Then we conclude.

2 Problem Formulation

2.1 Preliminaries

Based on our experience, gained in developing computer-aided control systems environments [7], [8] and in integrating various computer-aided engineering methods [9], we take, as point of departure of this research, an updated version of the *Inductive Learning System for Grammatical Inference*, or the so-called *ILSGInf*; a system capable of heuristically inferring, from positive examples of sentences, any regular language and some context-free languages (CFLs) [11]. *ILSGInf* classifies the negative examples correctly but does not take them into account for improving the grammar it generates. In other words, the positive examples help *ILSGINf* in improving the generated grammar, but the negative ones do not contribute to this improvement.

At the outset, we study grammatical inference (GI) as a special case of the larger problem of inductive machine learning. Then we describe graph grammars with a special emphasis on robotic self-assembly applications. We further propose a three-task methodology for robot control ending with graph grammatical inference. Concerning implementation, we discus the application of GI to a context-free language (CFL) as a prelude to a grammatical-based control.

2.2 Modes of self-assembly

Self-assembly, as defined above, comes in two modes, passive and active. In passive self-assembly, particles interact according to their geometry or surface chemistry and stay in a thermodynamic equilibrium, once this steady-state is reached. Particles behavior in chemical reactions can be classified in this mode. The geometrical patterns in the natural world give a clear indication that self-organized systems are omnipresent, from leaves to snowflakes, all governed by emergence of global patterns based on smaller patterns or fractals.

In active self-assembly, each particle may expend energy to accept some interactions with other particles while rejecting others, according to a controlling program. Typical examples are multi-robot systems, where small groups of robots determine the outcome of encounters according to their internal programming [17]. In our work, we will concentrate on this latter mode of self-assembly.

2.2 Self-assembly central issue

As stressed above, the main question in programmed self-organization concerns the ability to design rules that govern the global behaviour of a system by means of local rules. In a wide variety of settings, we can design local rules that yield a specified behaviour, with the ability to reason about the correctness of the result. In some circumstances, we can provide algorithms that automatically generate such a set of rules. Recent results are obtained in diverse areas ranging from algorithmic self-assembly of DNA [24], to the formation stabilization of multiple agents using decentralized navigation functions [12]. These results indicate that the emergent behaviour of a self-organizing system can be precisely predicted and controlled, although there is much work to be done to understand the physics, dynamics, and implementation of self-organization.

Progress in this area promises to open up new vistas for a completely new era of bottom-up engineering of systems, ranging from programmable nano-scale molecular machines to controlled swarms of interacting autonomous robots [16].

2.3 Grammatical tools for self-assembly

We consider the issue of programming active self-assembling systems at the level of interactions among particles in the system. To demonstrate the

approach, we propose to simulate the control of robots and consider illustrative examples from the literature. We follow two complementary directions:

(i) Use of GI as a control methodology.
(ii) Use of inference in graph grammars.

As far as this paper is concerned, only GI as an alternative control methodology is investigated. Special reference is given to machine dives to establish a link between formal languages and control systems.

2.4 Grammatical inference (GI)

2.4.1 Formal grammar

A formal string grammar or simply *grammar G* has four components [3]:

- A set of symbols V_N called non-terminals.
- A set of symbols V_T, called terminals with the restriction that V_T and V_N are disjoint.
- A special non-terminal symbol *S*, called a start symbol.
- A set of production rules *P*.

In other words, a formal string grammar is a set of rules that says whether a string of characters (*e.g.* a sentence in a natural language), constructed from the starting symbol, can be expressed in the form of terminals (words) *i.e.* the general form of a sentence.

2.4.2 Inference

Inference is defined as the process of reasoning from premises or conditions to consequent or conclusion. Inductive inference is a generalization process which attempts to identify a hidden function, given a set of its values. As mentioned above, in formal languages settings, learning the syntax of a language is usually referred to as *grammatical inference* or *grammar induction* (GI); an important domain for both cognitive and psycholinguistic domain as well as science and engineering. We are concerned with the problem of constructing a grammar from some given data. These latter, whether sequential or structured, are composed from a finite alphabet, and may have unbounded string-lengths.

In relatively simple situations, induction considers a deterministic finite-state automaton, or DFA, that takes strings of symbols as input, and produces a binary output, indicating whether that string, or sentence, is a part of the DFA's encoded language. In this particular example, GI builds a model of the hidden DFA internal structure, based only on pairs of input sentences and classifications, or outputs. From a control theory point of view, this is a classical input-output *identification* problem. The most successful GI algorithms produced so far are heuristic in nature; and *ILSGInf* is one of these [11]. An overview of some of these algorithms can be found in [1].

2.4.3 GI as a machine learning discipline

The objective of machine learning is to produce general hypotheses by induction, which will make predictions about future instances. The externally supplied instances are usually referred to as training set. Generally speaking, machine learning explores algorithms that reason from externally supplied instances, also called inputs, examples, data, observations, or patterns, according to the community where these appear. Because GI is considered as a sub-field of machine learning, it therefore inherits this fundamental characteristic.

To induce a hypothesis from a given training set, a learning system needs to make assumptions, or biases, about the hypothesis to be learned. A learning system without any assumption cannot generate a useful hypothesis since the number of hypotheses that are consistent with the training set is practically infinite and many of these are totally irrelevant [19]. One of the issues addressed in our work is to consider reasonable ways and means for reducing the computational obstacles as far as applied GI is concerned [10]. Others considered evolutionary methods to address the issue [21].

3 Problem Solution

We describe the main building blocks of the proposed solution in the form of a general scientific method. In the sequel, we make a top-down description of the proposed methodology, starting from graph grammars and ending with string grammars as applied to control drives. We propose to theoretically address the three tasks described below, but we will specifically develop only limited *ad hoc* orientations among these. Our ultimate goal is the application of inference to graph grammars in the context of self-assembly.

3.1 Graph grammars

3.1.1 What are graph grammars?

Graphical structures of various kinds, like graphs, diagrams, visual sentences are very useful to describe complex structures and systems in a direct and intuitive way. *Graph grammars* have been invented in the early seventies in order to generalize Chomsky's string grammars. This generalization consists in gluing graphs instead of concatenating strings. Graph grammars are evolving graphs from some starting graph, and whose evolution follows specified production rules.

A graph grammar is a pair (G_0, P) where:

- G_0 is called the *starting graph*
- P is a *set of production rules*.

A *graph* is a pair (V, E) where:

- V is a finite set called *vertices*
- E is a finite set with elements in $V \times V$, called *edges*.

Similarly to a language generated by string grammars, a language generated by a graph grammar is the *set of graphs* that can be derived from the starting graph and applying rules in P [23]. Mathematical accounts of graph grammars are based on algebraic representation [4].

3.1.2 Application of graph grammars in self-assembly

From the point of view of graphical programming languages, graph grammars are useful especially in the storage level. Thus, instead of storing all these graphical structures as individual objects, we store only their grammar for reasons of compact size and generative power. While earlier mathematical work focused on string grammars, more interest is recently based on tree and graph grammars [14].

In self-assembly applications, graph grammars are used to model the physics of the particles by describing the outcomes of interactions among them. When used to program the desirable outcomes of interactions among particles, a graph grammar represents a description of a communication protocol and is thus intended to be coupled with a physical model of the environment that mediates the interactions. In particular, a suitably designed graph grammar can precisely describe and direct the changing network topology of a self-organizing system [20].

3.2 GI and control systems

3.2.1 Motivation for using grammars to control of machine drives

Before embarking on self-assembly, we describe the interaction between the control problem and GI. As a simpler control problem we consider control of machine drives. It is an area where GI has been applied with various degrees of success [18]. Control of machine drives is a specialized subject in its own right, usually studied within traditional disciplines such as electrical and / or mechanical / industrial engineering. Based on mathematical models, this subject encompasses a tremendous body of knowledge since the early days of Wiener's cybernetics going back to the late 40's. To dynamically control a machine drive is to let it follow an imposed behavior, automatically calculated in real-time. The main methodology of dynamic control is therefore to produce the so-called prescribed feedback control law on the basis of output observations, as and when needed [13]. If the environment is unknown, we use adaptive control.

For the purpose of this work, we are only concerned with control, using grammars as a methodology. So far, by GI, we intend only deterministic finite automata DFA, equivalent to regular grammars, on the one hand and some context-free grammars (CFGs), on the other hand. If we refer to Chomsky hierarchy, only type-3 and subclasses of type-2 grammars, respectively, are concerned [10]. Now, in order to control drives, these classes of grammars are not sufficient. We need to include larger classes of grammars such as context-sensitive grammars or type-1. This is a real challenge for GI community since there remain many obstacles in inferring DFAs, let alone context-sensitive grammars. Supplementary human-supplied expert codification is needed in order to account for this kind of inference.

3.1.2 Steps for using GI in control systems

To develop a grammatical description and a GI algorithm for controlled dynamical systems three steps are required [18].

3.1.2.1 Quantification of the variables

Quantification refers to the creation of alphabets for the output (controlled) variable y and the control variable U. The objective is to generate the control U in order to maintain the output y within some prescribed values. A terminal alphabet V_T is associated to the output variable y and the non-terminal alphabet V_N to the control variable U. The feedback control law generates the required value of the input U so as to keep the output y within a specified range. For so doing, a quantification of the variables is made, in a discrete way, dividing the variables range into equal intervals and associating each interval to a symbol in the alphabet.

3.2.2.2 Production rules

p-type productions are defined by the human expert to be some substitution rules of a given form. This human-supplied codification is necessary. A p-type production codes the evolution of the output variable, depending on its p past values and on the value of the control variable U. There is, therefore, a functional relationship between the dynamics of the

system and the *p*-type productions. Note that *p*-type productions as described here have nothing to do with Proportional-control or *P*-control action.

3.2.2.3 Learning

A learning algorithm is necessary to extract the productions from the experimental data. To obtain a sample of the language, a sequence of control signals is applied to the system in such a way that the output variable *y* takes values in a sufficiently wide region. The signal evolution is then quantified as described above, and a learning procedure is followed.

3.2.3 Comparing GI-controlled systems with other methods

A useful methodological comparison can be made between grammatical methods and other methods such as:

(i) Observer-based methods of control [13].

(ii) Soft computing, *e.g.* fuzzy control [6].

3.3 Three Levels for Solution

3.3.1 Extending GI to graph grammars

Level 1
Methodology 13
/ / Extend known techniques used in GI to graph grammars / /

1 Make state of the art of GI in the following order:

1.1 State of the art in regular GI.
1.2. Study CFGs inference.
1.3. Concentrate study on structural methods such as:
- Tree grammars,
- Graph grammars.

2. Study graph grammars and their algebra.

3. Investigate the use of inference in graph grammars.

Figure 1
Methodology for extending known techniques used in GI to graph grammars

3.3.2 Formal languages for systems control

The main issue here is to consider how formal languages can help in developing novel techniques in system control.

Level 2
Methodology 23
/ / Formal languages for systems control / /

1. Study current methods for system control based on formal languages.

2. Consider control methods based on (string) grammar inference.

2.1 Apply *ILSGInf* [11] in control drives.
2.2 Extend *ILSGInf* application to robot control.

Figure 2
Formal languages for systems control

3.3.3 Study of graph grammars for self-assembly

The main issue here is to address the issue of self-organizing systems and robotics self-assembly using graph grammars which provide a compact representation of subgraph transitions arising from local interactions. Although graph grammars have been successfully used to control robot systems [16], they best describe changing networks, not, for example, low-level robot motion. That is why the methodology provides another level of design dealing with graph grammar inference.

Level 3
Methodology 33
/ / Robotics self-assembly and graph grammars / /

1. Study available methods using graph grammars in self-assembly.

2. Apply graph grammars inference [5] to robotics self-assembly.

Figure 3
Robotics self-assembly and graph grammars

3.4 Summing Up

Networked robot systems require methods that control both the network dynamics and each

individual robot behaviors. We need therefore to make an account of both aspects of the problem.

Methodology 4
/ / Report specific research issues / /

1. GI-based control *vs*. other control methods, namely*:*

1.1 Observer-based methods [13].
1.2 Soft computing-based control methods.

2. Emphasise importance of inference in graph grammars.

2.1 Consider individual behavior
2.2 Consider networked behavior

3. Make recommendations and a feasibility study of the proposed method.

Figure 5
Specific research issues

4 Results

Since we are at the beginning of the work, results mainly concern the applicability of GI to machine drives as a simpler application of the novel GI-based control methodology.

4.1 Dynamical systems and GI

GI is used as an algorithm by which a grammar is inferred from a set of sample words produced by the dynamical system considered as the linguistic source. Therefore in order to apply GI procedures, a dynamical system must be considered as linguistic source capable of generating a specific language. The set of productions encodes the dynamics of the system that generates the language. Any word that can be derived from the start symbol S by a sequence of productions of the grammar is said to be in the language generated by the dynamical system. For example, a grammar of the form:

$G = (V_N, V_T, P, S,)$
$V_N = \{A, B, C, S\}$
$V_T = \{a, b\}$
$P = \{ A \rightarrow a, B \rightarrow b, S \rightarrow AB, C \rightarrow AS, S \rightarrow CB \}$

generates the language of the form:

$$L = \{ w = a^n b^n, n>=1\}$$

with words consisting of any non-zero number of a symbols followed by any non-zero number of b symbols. The GI procedure applied to this example is detailed in Section 4 below. We show how *ILSGInf* automatically generates the grammar from a few examples.

4.2 Using *ILSGInf*

4.2.1 Initial grammar generation

ILSGInf starts by generating an initial grammar of the form :

$G_0 = (V_{N0}, V_{T0}, P_0, S)$ where:

V_{N0}= {A / A non-terminal of derivative tree}
V_{T0} = {a / a is a symbol of input character string}
P_0 = { R / R rule of the form $A \rightarrow BC$;or $A \rightarrow a$}
with A, B, C non-terminals in derivation tree.
S = initial symbol.

Figure 6 below describes the algorithm for generating the initial grammar, from which the inference starts.

Algorithm for initial grammar generation

/* Algorithm for the construction of initial grammar
$G_0 = (V_{N0}, V_{T0}, P_0, S)$ */

```
Begin
string[i]  /* table containing the string example */

n          /* length of initial global string */

Initial_symbol:="S"     /*creation of initial symbol,
                          by convention "S"*/

i:=1, k:=1                  /*indices*/

/* Associate to each terminal one non-terminal */
  /* create the set of  initial rules as follows */

for i=1 to n do
    if string[i] is not yet associated with a non-
       terminal
    then  create_the_rule  non-terminal(k)  →
            string[i]


 k:=k+1

    endif

endfor
```

```
if n<= 2                    /* Derivation from S* /
  then  create_rule S → <non-terminal(1) <non-
          terminal(2)

       else  /*Construction of derivation tree from
              bottom to top */

       create_the_rule non-terminal(k) → <non-
        terminal(1) <non-terminal(2)

       i:=3; k:=k+1

   while i<n do
    create_the_rule non-terminal(k) → <non-
     terminal(k-1) <non-terminal(i)

k:=k+1; i:=i+2

    endwhile

 /* For string to be recognized, it must derive
     from root * /
    create_rule S → <non-terminal(k-1) <non-
   terminal(i)

endif

end
```

Figure 6
Algorithm for initial grammar generation

4.2.2 Generating lists and sub-lists for a Context-Free Language (CFL)

Using the same example as above, we give some results obtained by *ILSGInf* in the identification of a context-free language (CFL) of the form:

$$L = \{ w = a^n b^n, n>=1\}$$

A possible grammar for L is:

$G = (V_N, V_T, P, S,)$
$V_N = \{A, B, C, S\}$
$V_T = \{a, b\}$
$P = \{ A \rightarrow a, B \rightarrow b, S \rightarrow AB, C \rightarrow AS, S \rightarrow CB \}$

(1) Filling the fact base

This grammar is stored as facts as follows:

```
          /* Application 1 */

          /*  Fact Base for CFL
        L = { w = a^n b^n, n>=1}  */

    /* Production rules stored as facts */

FACT  RULE A              // Fact1 //
FACT  RULE B  b           // Fact2 //
FACT  RULE S   A  B       // Fact3 //
FACT  RULE S   C   B      // Fact4 //
FACT  RULE C   A   S      // Fact5 //

FACT  initial_symbol S  // Fact6 //


 /* Sentence to be parsed and its length */

FACT  string  aaabbb    // Fact7 //
FACT  length  6         // Fact8 //
```

Figure 7 Fact base for the CFL
$L = \{ w = (a^n b^n, n>=1\}$

Each production rule in the grammar is stored as a fact (Fact1,2,3,4,5,6). The sentence to be parsed (Fact7) and its length (Fact8) are also introduced in the fact base.

(2) Inference Cycles

Like any inference process in knowledge-based systems, it is composed of three main steps, namely detection, conflict resolution and termination.

(3) Parsing Final Result

Figure 8 describes the final result using aaabbb as training example.

sub-list 0	*sub-list 1*
I_0	**I_1**
S →•CB, 0	**A → a •, 0**
S →•AB, 0	**S →A•B, 0**
C →•AS, 0	**C →A•S, 0**
A → • a, 0	**B → • b,1**
	S → •AB,1

	S → •CB,1 **A → •a,1** **C → •AS,1**
sub-list 2	*sub-list 3*
I_2 **A → a•,1** **S → A•B,1** **C → A•S,1** **B → •b,2** **S → •AB,2** **S → •CB,2** **A → •a,2** **C → •AS,2**	**I_3** **A → a •,2** **S →A •B,2** **C → A•S,2** **B → •b,3** **S → •AB,3** **S → •CB,3** **A → •a,3** **C → •AS,3**
sub-list 4	*sub-list 5*
I_4 **B → b •,3** **S →AB •,2** **C → AS•,1** **S →C•B,1** **B → •b,4**	**I_5** **B → b •,4** **S→ CB •,1** **C → AS •,0** **S → C •B,0** **B → •b,5**
sub-list 6	
I_6 **B → b •,5** ***S→ CB •,0***	

Figure 8 Progressive construction of sub-lists for CFL $L = \{ w = (a^n b^n, n>=1\}$ with positive example aaabbb

Discussions and decisions
Decision: The introduced sentence aaabbb is accepted because in sub-list 6, we find the item
S →CB •, 0
6.2.2 Counter example
Let's consider the same language L as above but with a counter example of the form aabbb.

(1) Fact Base

```
/ * Application 2 * /

/ * Fact Base with Counter Example * /

/ * Production rules stored as facts * /

FACT  RULE A  a              / / Fact1 / /
FACT  RULE B  b              / / Fact2 / /
FACT  RULE S   A  B          / / Fact3 / /
FACT  RULE S   C   B         / / Fact4 / /
FACT  RULE C   A   S         / / Fact5 / /

FACT  initial_symbol S       / / Fact6 / /


/ * Sentence to be parsed and its length * /

FACT  string   aabbb         / / Fact7 / /
FACT  length                 / / Fact8 / /
```

Figure 9 Fact base with counter example for CFL $L = \{ w = (a^n b^n, n>=1\}$

(2) Inference cycles
As above

(3) Parsing Final Result
Figure 10 describes the final result using aabbb as counter example.

sub-list 0	***sub-list 1***
I_0 **S →•CB, 0** **S →•AB, 0** **C →•AS, 0** **A → • a, 0**	**I_1** **A → a •, 0** **S →A•B, 0** **C →A•S, 0** **B → • b,1** **S → •AB,1** **S → •CB,1** **A → •a,1** **C → •AS,1**
sub-list 2	***sub-list 3***
I_2 **A → a•,1**	**I_3** **B → b•,2**

S → A•B,1 C → A•S,1 B → •b,2 S → •AB,2 S → •CB,2 A → •a,2 C → •AS,2	S →AB •,1 C → AS•,0 S → C•B,0 B → •b,3
sub-list 4	*sub-list 5*
I_4 B → b •,3 S →CB•,0	I_5 empty

Figure 10 Progressive construction of sub-lists for CFL $L = \{ w = (a^n b^n, n>=1\}$ with counterexample aaabbb

Discussions and decisions

Decision: The introduced sentence aabbb is NOT accepted because sub-list 5 is empty. Note that although *ILSGINf* has successfully classified the example as negative, it does not use this information for improving the grammar. This latter is only improved by positive examples only.

5 Conclusion

The present research work paves the way towards an objective evaluation and a thorough study of the effectiveness and usefulness of grammatical inference in the control of robots for self-assembly purposes. It represents an early contribution as far as graph grammars inference integration is concerned. A unification of the diversified works dealing with robotic self-assembly while concentrating on graph grammars as an alternative control method is made possible. This is done using a three-level methodology for self-assembly robotic control starting with string grammatical inference and ultimately leading to inference in graph grammars. However, the results report only a tiny aspect of the overall issue, since these describe only the case of context-free language (CFL) inference as part of the control of machine drives. Much work is still required on both sides, *i.e.* robotics and formal languages, for the development of fully-integrated systems that scale up to real-life applications.

References

[1] de La Higuera, C. "A bibliographical study of grammatical inference". *Pattern Recognition*, 38:1332-1348, 2005.

[2] Cicchello, O., S. C. Kremer, "Inducing grammars from sparse data sets: A survey of algorithms and results," *J. Mach. Learn. Res.*, 4:603–632, 2003.

[3] Cohen, D. I. A. "*Introduction to Computer Theory"*. John Wiley & Sons, Inc.1991.

[4] Ehrig, E., "Introduction to the algebraic theory of graph grammars," In *Graph-Grammars and Their Application to Computer Science and Biology*, by V. Claus, H. Ehrig, and G. Rozenberg, (Eds.), Springer-Verlag, pp. 1–69, 1979.

[5] Flasinski, M. "Inference of parsable graph grammars for syntactic pattern recognition", *Fundamenta Informaticae*, IOS Press, 80:379–413, 2007.

[6] Hagras, H. **"**Type-2 FLCs: A New Generation of Fuzzy Controllers"**,** *Computational Intel. Mag., IEEE***,** 2(1):30-43, Feb. 2007.

[7] Hamdi-Cherif, A. "The CASCIDA project - A computer-aided system control for interactive design and analysis" *Proc. of IEEE / IFAC Joint Symp. on CACSD (CASCD'94)*. pp. 247-251, Tucson, AZ, USA, 7-9 March 1994.

[8] Hamdi-Cherif, A. "The intellectual environment for a CACSD project". *Proc. of the IEEE Conf. on Control App. (CCA'94)*, pp.1365-1366, Glasgow, Scotland, 24-27 Aug. 1994

[9] Hamdi-Cherif, A. (1994-4). A unifying framework for computer-aided engineering (CAE) with reference to computer-aided control system design (CACSD). *Proc. Of the First Asian Control Conf. (ASCC'94)*, 1:45-48. Tokyo, Japan, 24-27 July 1994.

[10] Hamdi-Cherif, C., (*alias* C. Kara-Mohammed), A. Hamdi-Cherif, "Apprentissage inductif de grammaires: Le système GASRIA. [Inductive learning for grammars: The GASRIA System]". *Revue d'Intelligence Artificielle*, Hermes-Lavoisier Edition, Paris, France, 21(2):223-253, May-June 2007.

[11] Hamdi-Cherif, C., A. Hamdi-Cherif, "*ILSGInf* : Inductive learning system for grammatical inference". *WSEAS Trans. on Computers*, 6(7):991-996, 2007.

[12] Tanner, H.G., A. Kumar, "Formation stabilization of multiple agents using decentralized navigation functions," In *Robotics: Science and Systems I*, by S. Thrun, G. Sukhatme, S. Schaal, and O. Brock, (Eds.), MIT Press, pp. 49–56, 2005.

[13] Harmas, M.N., A. Merzouki, A. Hamdi-Cherif, "Observer based intelligent power system stabilizer". *Asian J. of IT*, 6 (10):1057-1063, 2007.

[14] Hoffmann B. "Hierarchical Graph Transformation". *Int. J. of Comp. and Syst. Sci*., 2000. pp. 98-113.

[15] Klavins, E., S. Burden, N. Napp, "Optimal rules for programmed stochastic self-assembly," In *Robotics: Science and Systems I*, by S. Thrun, G. Sukhatme, S. Schaal, and O. Brock, (Eds.), MIT Press, pp. 9–16, 2007.

[16] Klavins, E., R. Ghrist, D. Lipsky, "A grammatical approach to self-organizing robotic systems," *IEEE Trans. Automat. Contr*., 51(6):949–962, June 2006.

[17] Klavins, E., "Programmable self-assembly", *IEEE Control Syst. Mag.*, 27(4):43-56, Aug. 2007.

[18] Martins, J. F., J.A. Dente, A.J. Pires, and R. Vilela Mendes "Language Identification of Controlled Systems: Modeling, Control, and Anomaly Detection", *IEEE Trans. On Syst. Man and Cyb. – Part C: Appl. And Rev.* 31(2):234-242, 2001.

[19] Mitchell, T.M., "*Machine Learning"*, McGraw-Hill, New York, 1997.

[20] McNew, J. M., Klavins, E., M. Egerstedt "Solving coverage problems with embedded graph grammars". In *Hybrid systems: computation and control*, A. Bemporad, A. Bicchi, and G. Buttazzo, (Eds.), Springer-Verlag, pp. 413-427, 2007.

[21] Unold O.: Context-free grammar induction using evolutionary methods, *WSEAS Trans. on Circuits and Systems*, 3(2):632-637, 2003.

[22] Unold, O., Grammar-based classifier system: a universal tool for grammatical inference *WSEAS Trans. on Computers* 7(10):1584-1593, 2008.

[23] Rozenberg, G., (Ed.), "*Handbook of graph grammars and computing by graph transformation"*. Volume 1-3: Foundations, World Scientific, 1997.

[24] Winfree, E. "Algorithmic self-assembly of DNA: Theoretical motivations and 2-D assembly experiments," *J. Biomolecular Structure Dynamics*, 11(2):263–270, May 2000.

[25] White, P., V. Zykov, J. Bongard, H. Lipson, "Three dimensional stochastic reconfiguration of modular robots," In S. Thrun, G. Sukhatme, S. Schaal, O. Brock, (Eds.), "*Robotics: Science and Systems I"*, MIT Press, pp. 161–168, June 2005.