Regular Language Induction with Grammar-based Classifier System

Olgierd Unold
Wroclaw University of Technology
Poland

1. Introduction

In this chapter, we are interested in inducing a grammar that accepts a regular language (Hopcroft & Ullman, 1969) given a finite number of positive and negative examples drawn from that language. Learning regular languages is equivalent to the problem of learning Deterministic Finite Automata (DFA). Both problems have been extensively studied in the literature and it has been proved that learning DFA or regular languages is a hard task by a number of criteria (Pitt & Warmuth, 1993). Note, that induced DFA should not only be consistent with the training set, but also DFA should proper estimate membership function for unseen examples.

The approaches to learning DFA or equivalent regular languages (RL) base mainly on evolutionary algorithms (Dupont, 1996), (Luke et al., 1999), (Lucas & Reynolds, 2005), recurrent neural network (Giles et al., 1990), (Waltrous & Kuhn, 1992) or combination of these two methods (Angeline et al., 1994). While speaking about DFA/regular grammar induction, one cannot help mentioning one of the best known algorithm for learning DFA – EDSM (Cicchello & Kremer, 2002), which relies on heuristic compressing an initially large DFA down to a smaller one, while preserving perfect classification before and after each compression.

In this chapter we examine RL induction using Grammar-based Classifier System (GCS) – a new model of Learning Classifier System (LCS). GCS (Unold, 2005a), (Unold & Cielecki, 2005) represents the knowledge about solved problem in Chomsky Normal Form (CNF) productions. GCS was applied with success to natural language processing (Unold, 2007a), biological promoter regions (Unold, 2007b), and toy grammar (Unold, 2005b). In spite of intensive research into classifier systems in recent years (Lucas & Reynolds, 2005) there is still a slight number of attempts at inferring grammars using LCS. Although there are some approaches to handle with context-free grammar (Bianchi, 1996), (Cyre, 2002), (Unold, 2005a), there is no one work on inducing regular languages with LCS. This article describes GCS approach to the problem of inferring regular languages.

The generic architecture of learning classifier system is presented in the second paragraph. The third section contains description of GCS preceded by short introduction to context-free grammars. The fourth paragraph shows some selected experimental results in RL grammar induction. The chapter is concluded with a summary.

2. Learning Classifier System

A Learning Classifier System, introduced by Holland (1976), learns by interacting with an environment from which it receives feedback in the form of numerical reward. Learning is achieved by trying to maximize the amount of the reward received. There are many models of LCS and many ways of defining what a Learning Classifier System is. All LCS models, more or less, comprise four main components (see Fig. 1): (i) a finite population of condition-action rules (classifiers), that represent the current knowledge of a system; (ii) the performance component, which governs the interaction with the environment; (iii) the reinforcement component, called credit assignment component), which distributes the reward received from the environment to the classifiers accountable for the rewards obtained; (iv) the discovery component responsible for discovering better rules and improving existing ones through a genetic algorithm (GA).

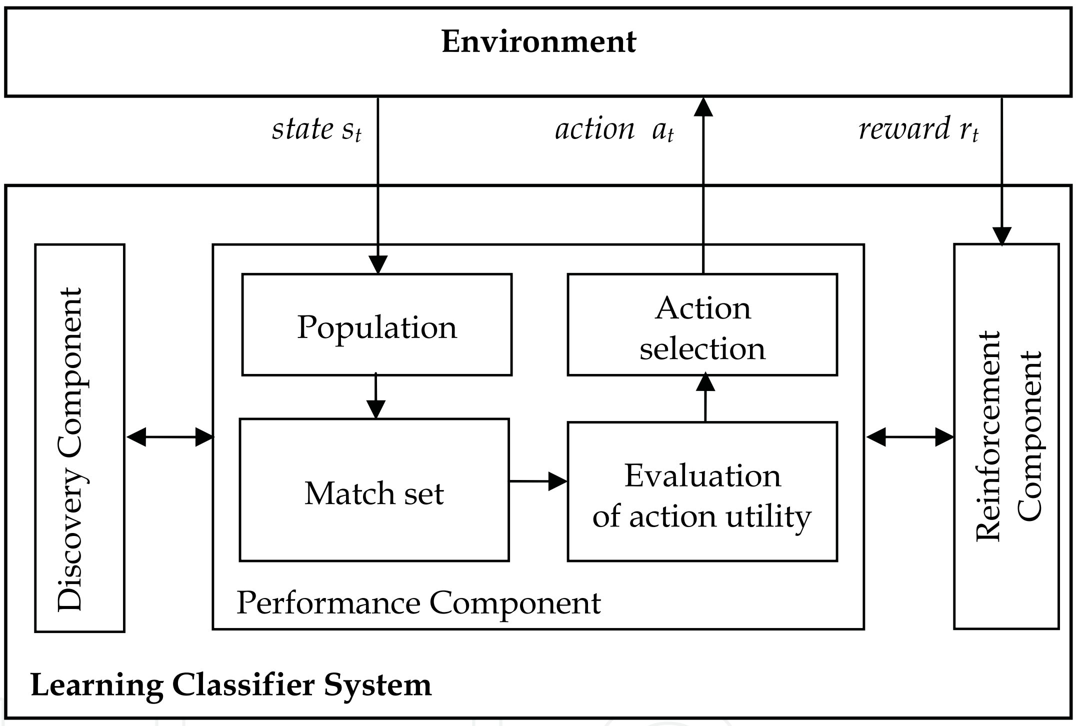

Fig. 1. The generic architecture of Learning Classifier System (Holmes et al., 2002)

Classifiers have two associated measures: the prediction and the fitness. Prediction estimates the classifier utility in terms of the amount of reward that the system will receive if the classifier is used. Fitness estimates the quality of the information about the problem that the classifier conveys, and it is exploited by the discovery component to guided evolution. A high fitness means that the classifier conveys good information about the problem and therefore it should be reproduced more trough the genetic algorithm. A low fitness means that the classifier conveys little or no good information about the problem and therefore should reproduce less.
On each discrete time step t, the LCS receives as input the current state of the environment s_t and builds a match set containing the classifiers in the population, whose condition matches the current state. Then, the system evaluates the utility of the actions appearing in the match

set; an action at is selected from those in the match set according to a certain criterion, and sent to the environment to be performed. Depending on the current state s_t and on the consequences of action at, the system eventually receives a reward r_t. The reinforcement component distributes a reward r_t among the classifiers accountable of the incoming rewards. This can be either implemented with an algorithm specifically designed for the Learning Classifier Systems (e.g. bucket brigade algorithm (Holland, 1986)) or with an algorithm inspired by traditional reinforcement learning methods (e.g., the modification of Q-learning (Wilson, 1995)). On a regular basis, the discovery component (genetic algorithm) randomly selects, with the probability proportional to their fitness, two classifiers from the population. It applies crossover and mutation generating two new classifiers.

The environment defines the target task. For instance, in autonomous robotics the environment corresponds roughly to the robot's physical surroundings and the goal of learning is to learn a certain behaviour (Katagami & Yamada, 2000). In classification problems, the environment trains a set of pre-classified examples; each example is described by a vector of attributes and a class label; the goal of learning is to evolve rules that can be used to classify previously unseen examples with high accuracy (Holmes et al., 2002) (Unold & Dabrowski, 2003). In computational economics, the environment represents a market and the goal of learning is to make profits (Judd & Tesfatsion, 2005).

For many years, the research on LCS was done on Holland's classifier system. All implementations shared more or less the same features which can be summarized as follows: (i) some form of a bucket brigade algorithm was used to distribute the rewards, (ii) evolution was triggered by the strength parameters of classifiers, (iii) the internal message list was used to keep track of past input (Lanzi & Riolo, 2000).

During the last years new models of Holland's system have been developed. Among others, two models seem particularly worth mentioning. The XCS classifier system (Wilson, 1995) uses Q-learning to distribute the reward to classifiers, instead of bucket brigade algorithm; the genetic algorithm acts in environmental niches instead of on the whole population; and most importantly, the fitness of classifiers is based in the accuracy of classifier predictions, instead of the prediction itself. Stolzmann's ACS (Stolzmann, 2000) differs greatly from other LCS models in that ACS learns not only how to perform a certain task, but also an internal model of the dynamics of the task. In ACS classifiers are not simple condition-action rules but they are extended by an effect part, which is used to anticipate the environmental state.

3. Grammar-based Classifier System

GCS (Unold, 2005a) (Unold & Cielecki, 2005) operates similarly to the classic LCS but differs from them in (i) representation of classifiers population, (ii) scheme of classifiers' matching to the environmental state, (iii) methods of exploring new classifiers (see Fig. 2).

The population of classifiers has a form of a context-free grammar rule set in a Chomsky Normal Form (population P in Fig. 2). Actually, this is not a limitation, because every CFG can be transformed into equivalent CNF. Chomsky Normal Form allows only for production rules, in the form of $A \rightarrow a$ or $A \rightarrow BC$, where A, B, C are the non-terminal symbols and a is a terminal symbol. The first rule is an instance of *terminal rewriting rule*. Terminal rules are not affected by the GA, and are generated automatically as the system meets an unknown (new) terminal symbol. The left hand side of the rule plays a role of the classifier's

action while the right hand side - a classifier's condition. The system evolves only one grammar according to the so-called Michigan approach. In this approach, each individual classifier – or grammar rule in GCS – is subject of the genetic algorithm's operations. All classifiers (rules) form a population of evolving individuals. In each cycle a fitness calculating algorithm evaluates a value (an adaptation) of each classifier and a discovery component operates only on single classifiers.
The automatic learning CFG is realized with grammar induction from the set of sentences. According to this technique, the system learns using a training set that in this case consists of sentences both syntactically correct and incorrect. Grammar which accepts correct sentences and rejects incorrect ones is able to classify sentences unseen so far from a test set. Cocke-Younger-Kasami (CYK) parser, which operates in Θ(n3) time (Younger, 1967), is used to parse sentences from the corpus.

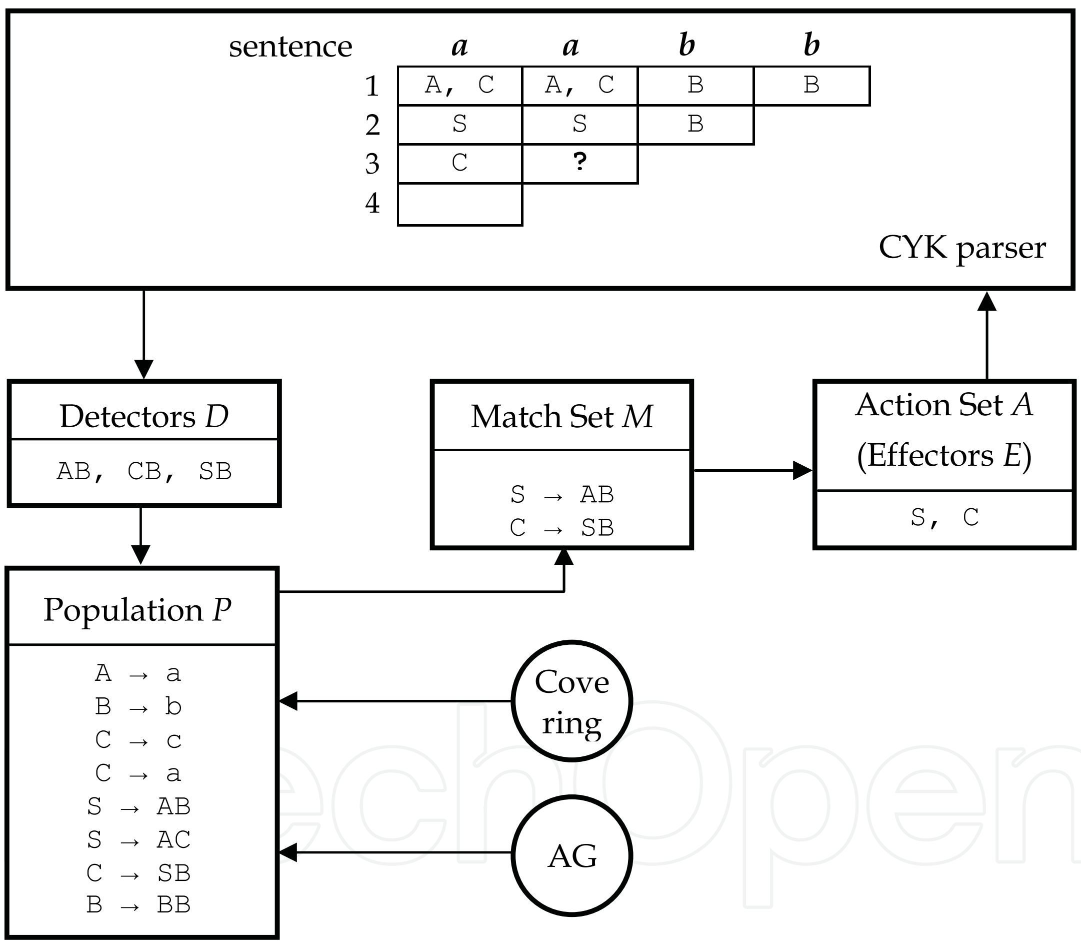

Fig. 2. The environment of Grammar-based Classifier System

The environment of a classifier system is substituted by an array of CYK parser (CYK parser module in Fig. 2). The classifier system matches the rules according to the current environmental state (state of parsing, Matching Set *M* in Fig. 2) and generates an action (or set of actions in GCS, effectors *E* in Fig. 2), pushing the parsing process toward the complete derivation of the analyzed sentence.

The discovery component in GCS is extended in comparison with a standard LCS. In some cases, a "covering" procedure may occur, adding some useful rules to the system. It adds productions that allow continuing of parsing in the current state of the system. This feature utilizes for instance the fact that accepting 2-length sentences requires separate, designated rule in grammar in CNF.
Apart from a "covering", a genetic algorithm also explores the space, searching for new, better rules. The first GCS implementation used a simple rule fitness calculation algorithm which appreciated the ones commonly used in correct recognitions. Later implementations introduced the "fertility" technique, which made the rule fitness dependant on the amount of the descendant rules (in the sentence derivation tree) (Unold, 2005b). In both techniques classifiers used in parsing positive examples gain highest fitness values, unused classifiers are placed in the middle, while the classifiers that parse negative examples gain lowest possible fitness values.
GCS uses a mutation of GA that chooses two parents in each cycle to produce two offspring. The selection step uses the roulette wheel selection. After selection a classical crossover or mutation can occur. The offspring that are created replace existing classifiers based on their similarity using crowding technique, which preserves diversity in the population and extends preservation of the dependencies between rules by replacing classifiers by the similar ones.

4. Regular Language Induction

4.1 Experimental testbed

The datasets most commonly used in DFA learning is Tomita sets (Tomita, 1982). The definition of Tomita languages is as follows:

L1: *a*,*

L2: *(ab)*,*

L3: *(b | aa)*(a* | (abb(bb | a)*))*
any sentence without an odd number of consecutive *a*'s after an odd number of consecutive *b*'s,

L4: *a*((b | bb)aa*)*(b | bb | a*)*
any sentence over the alphabet *a,b* without more than two consecutive *a*'s,

L5: *((aa | bb)*((ba | ab)(bb | aa)*(ba | ab)(bb | aa)*)*(aa | bb)**
any sentence with an even number of *a*'s and an even number of *b*'s,

L6: *((b(ba)*(a | bb)) | (a(ab)*(b | aa)))**
any sentence such that the number of *a*'s differs from the number of *b*'s by 0 modulo 3,

L7: *b*a *b*a*.*

By the way, it is worth mentioning that the L3 language given in (Luke et al., 1999) comprises improper, i.e. not according to the definition, two sentences *baaabbaaba* oraz *aabaaabbaab*. The same work gives incorrect definition of L5 language, permitting sentences which contain odd number of symbols a and b.
Grammatical (grammar) inference methods that employ DFAs as models can be divided into two broad classes: passive and active learning methods (Bongard & Lipson, 2005). In passive methods, a set of training data is known before learning. In active learning

approaches, the algorithm has some influence over which training data is labeled by the target DFA for model construction.
Passive methods, and to this class belongs GCS, usually make some assumption about the training data. In (Pitt, 1989), (Porat & Feldman, 1991), (Dupont, 1996), (Lang et al., 1998) a learning data was selected at random from sample data, in (Pao & Carr, 1978), (Parekh & Honavar, 1996) a learning data consisted of a structurally complete set, (Oncina & Garcià, 1992) assume a characteristic sample; and (Angluin, 1981) assumes a live complete set. Luke et al. (1999) and Lucas & Reynolds (2005) used equal amounts of positive and negative training examples when inferring the Tomita languages, so a learning set was balanced as in (Tomita, 1982), (Angeline, 1997), (Waltrous & Kuhn, 1992). In passive methods once the sample data has been generated and labeled, learning is then conducted.
In this chapter Grammar-based Classifier System, a method which employs evolutionary computation for search, will be compared against the evolutionary method proposed by Lucas & Reynolds (2005), and Luke et al. (1999). (Lucas & Reynolds, 2005), as well as (Luke et al., 1999) present one of the best-known results in the area of DFA/regular language induction. All of compared evolutionary methods will assume the same training and test sets. Some comparisons will be made also to EDSM method (Cicchello & Kremer, 2002), the current most powerful passive approach to DFAs inference.
Table 1 shows the details of applied data sets: number of all learning examples |U|, number of positive learning examples |U+|, number of negative learning examples |U–|, number of all test examples |T|, number of positive test examples |T+|, and number of negative test examples |T–|. Note, that test sets are not balanced, and contain much more negative sentences than positive once.

Lang.	$\|U\|$	$\|U^+\|$	$\|U^-\|$	$\|T\|$	$\|T^+\|$	$\|T^-\|$
L1	16	8	8	65 534	15	65 519
L2	15	5	10	65 534	7	65 527
L3	24	12	12	65 534	9447	56 087
L4	19	10	9	65 534	23 247	42 287
L5	21	9	12	65 534	10 922	54 612
L6	21	9	12	65 534	21 844	43 690
L7	20	12	8	65 534	2515	63 019

Table 1. Learning and test data sets.

4.2 Experiments
A comparison set of experiments with GCS was performed on the above Tomita corpora. Fifty independent experiments were performed, evolution on each training corpus ran for 5,000 generations, with the following genetic parameters: number of non-terminal symbols 19, number of terminal symbols 7, crossover probability 0.2, mutation probability 0.8, population consisted of maximal 40 classifiers where 30 of them were created randomly in the first generation, crowding factor 18, crowding size 3.
In the first attempt GCS was compared to the approach presented in (Luke et al., 1999) (denoted by GP). GP applies gene regulation to evolve deterministic finite-state automata. In this approach genes are states in the automaton, and a gene-regulation-like mechanism determines state transitions. Each gene has Boolean value indicating whether or not it was an accepting state. The main results are summarized in Table 1. For each learning corpus,

the table shows the target language, and three sets of results. The first indicator nSuccess is the number of runs with success gained by GCS within 50 experiments and compared approach presented in (Luke et al., 1999). The second one nEvals indicates the average number of generations needed to reach the 100% fitness, and the last one nGen is the percentage of all unseen strings correctly classified.

Lang.	*nSuccess*		*nEvals*		*nGen*	
	GP	**GCS**	**GP**	**GCS**	**GP**	**GCS**
L1	31/50	**50/50**	30	**2**	88.4	**100**
L2	7/50	**50/50**	1010	**2**	84.0	**100**
L3	1/50	1/50	12 450	**666**	66.3	**100**
L4	3/50	**24/50**	7870	**2455**	65.3	**100**
L5	0/50	**50/50**	13 670	**201**	68.7	**92.4**
L6	47/50	**49/50**	2580	**1471**	95.9	**96.9**
L7	1/50	**11/50**	11 320	**2902**	67.7	**92.0**

Table 2. Comparison of GCS with GP approach (Luke et al., 1999).

For compared methods induction of L3 language appeared to be hard task. Both in GP and in GCS only the one run over 50 successfully finished. But GP found the solution in 12450 iterations, whereas GCS in only 666 steps. For the same language GCS correctly classified all of the unseen examples, while GP achieved 66%. As to an indicator nGen, GP was not able correctly classified unseen strings for any language from the tested corpora, while GCS induced a grammar fully general to the language in 4 cases. It is interesting to compare the results of induction for L5 language. GP approach could not find the proper grammar (DFA) for any run, while GCS found the solution in all runs, on average in 201 steps. While learning L1 and L2 languages, GP found the proper grammars not in all runs, whereas for GCS this task appeared to be trivial (100% nGen, 50/50 nSuccess, and nEvals 2 steps).
Table 3 shows the cost of induction (an indicator nEvlas) for the methods Plain, Smart, and nSmart taken from (Lucas & Reynolds, 2005), GP approach, and GCS.

Lang.	**Plain**	**Smart**	**nSmart**	**GP**	**GCS**
L1	107	25	15	30	**2**
L2	186	37	40	1010	**2**
L3	1809	237	833	12 450	666
L4	1453	177	654	7870	2455
L5	1059	195	734	13 670	201
L6	734	93	82	2580	1471
L7	1243	188	1377	11 320	2902

Table 3. Cost of induction (nEvals) for different evolutionary methods.

Lucas and Reynolds (Lucas & Reynolds, 2005) used different method to evolving DFA. In contrary to (Luke et al., 1999), only transition matrix was evolved, supported by a simple deterministic procedure to optimally assign state labels. This approach is based on evolutionary strategy (1+1). Three versions of induction algorithm were prepared: an approach in which both the transition matrix and the state label vector evolve (Plain), so-called Smart method evolving only the transition matrix and the number of the states was fixed and equal to 10, and finally nSmart method in which the number of the DFA states is

equal to the size of minimal automata. Recall that both GP and GCS belong to the so-called variable size methods, whereas Plain, Smart, and nSmart approaches represent the fixed-size structure methods. In general, the second group of methods gains better results.
GCS obtained the best results for the L1 and L2 languages among comparable methods. The result 201 steps for L5 is comparable with the best result of 195 reached by nSmart. Although GCS reached similar result for language L3 as the best method (666 for GCS, and 237 for Smart), it is hard to compare for this language these methods, because of low value of nSuccess for GCS – only one run over 50 finished with success (see table 2). For the languages L4, L6, and L7 fixed-size structured methods achieved better results than variable-size methods.

Lang.	Smart	nSmart	EDSM	GP	GCS
L1	81.8	100	52.4	88.4	100
L2	88.8	95.5	91.8	84	**100**
L3	71.8	90.8	86.1	66.3	**100**
L4	61.1	100	100	65.3	100
L5	65.9	100	100	68.7	92.4
L6	61.9	100	100	95.9	96.9
L7	62.6	82.9	71.9	67.7	**92**

Table 4. Percentage of all unseen strings correctly classified (nGen) for different methods.

Table 4 shows the percentage of all unseen strings correctly classified (an indicator nGen) for the methods Smart, nSmart, EDSM, GP, and GCS. Recall that the EDSM, as a heuristic and non-evolutionary method, was single-time executed during learning phase. Model GCS achieved the best results from all tested approaches for L1, L2, L3, and L7 languages. For the language L4 the same 100% accuracy was obtained by proposed method, nSmart, and EDSM. For the L5 and L6 languages GCS obtained the second result, higher than 90%.

5. Summary

Our experiments attempted to apply a Grammar-based Classifier System to evolutionary computation in evolving an inductive mechanism for the regular language set. Near-optimal and/or better than reported in the literature solutions were obtained. Moreover, performance of GCS was compared to the Evidence Driven State Merging algorithm, one of the most powerful known DFA learning algorithms. GCS with its ability of generalizations outperforms EDSM, as well as other significant evolutionary method.

6. References

Angeline, P. (1994). *Evolutionary Algorithms and Emergent Intelligence.* PhD Thesis. Computer Science Department, Ohio State University
Angeline, P. (1997). An alternative to indexed memory for evolving programs with explicit state representations, In: Koza, J.R. et al (eds.) *Proc. 2nd Conf. Genetic Programming (GP97)*, pp. 423–430, Morgan Kaufmann, San Francisco, CA

Angeline, P.; Saunders, G.M. & Pollack, J.P. (1994). An Evolutionary Algorithm that Constructs Recurrent Neural Networks, *IEEE Trans. Neural Networks*, vol. 5, no. 1, 54–65

Angluin, D. (1981). A note on the number of queries needed to identify regular languages, *Information and Control*, 51, 76–87

Bianchi, D. (1996). Learning Grammatical Rules from Examples Using a Credit Assignement Algorithm, In: *Proc. of The First Online Workshop on Soft Computing (WSC1)*, pp. 113–118, Nagoya

Bongard, J. & Lipson, H. (2005). Active Coevolutionary Learning of Deterministic Finite Automata, *J. of Machine Learning Research*, 6, 1651–1678

Cicchello, O. & Kremer, S.C. (2002). Beyond EDSM, In: *Proc. Int'l Colloquium Grammatical Inference*, vol. 2484, pp. 37–48

Cyre, W.R. (2002). Learning Grammars with a Modified Classifier System, In: *Proc. 2002 World Congress on Computational Intelligence*, pp. 1366–1371. Honolulu, Hawaii

Dupont, P. (1996). Incremental regular inference. In: Miclet, L. & de la Higuera, C. (eds.) *Proc. 3rd ICGI-96*, pp. 222–237, LNAI, vol. 1147

Dupont, P.; Miclet L. & Vidal, E. (1994). What Is the Search Space of the Regular Inference? In: Carrasco, R.C. & Oncina, J. (eds.) *Proc. Grammatical Inference and Applications: Second Int'l Colloquium (ICGI-94)*, pp. 25–37

Giles, C.; Sun, G.; Chen, H.; Lee Y. & Chen, D. (1990). Higher order Recurrent Neural Networks and Grammatical Inference, In: Touretzky, D. (ed.) *Advances in Neural Information Processing Systems 2*, pp. 380–387. San Mateo, Calif.: Morgan Kaufman

Holland, J. (1976). Adaptation. In: Rosen, R. & Snell, F.M. (eds.) *Progress in theoretical biology,* Plenum, New York

Holland, J. (1986). Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In: Michalski, R.S. et al. (eds.) *Machine Learning, an Artificial Intelligence Approach*, vol. II, 593–623. Morgan Kaufmann

Holmes, J.H.; Lanzi, P.L.; Stolzmann, W. & Wilson, S.W. (2002). Learning classifier systems: new models, successful applications, *Information Processing Letters* 82(1), 23–30

Hopcroft, J.E. & Ullman, J.D. (1969). *Formal Languages And Their Relation to Automata,* Reading, Mass.: Addison-Wesley

Judd, K.L. & Tesfatsion, L. (2005). Agent-Based Computational Economics, *Handbook of Computational Economics*, vol. 2, Elsevier, North-Holland

Katagami, D. & Yamada, S. (2000). Interactive Classifier System for Real Robot Learning. In: *IEEE International Workshop on Robot-Human Interaction ROMAN-2000*, pp. 258–263, Osaka, Japan

Lang, K.; Pearlmutter, B. & Price R. (1998). Results of the Abbadingo One DFA Learning Competition and a New Evidence Driven State Merging Algorithm. In: *Proc. Int. Colloquium on Grammatical Inference ICGA-98*, pp. 1–12, LNAI, vol. 1433, Springer, Berlin, Heidelberg

Lanzi, P.L. & Riolo, R.L. (2000). A Roadmap to the Last Decade of Learning Classifier System Research, In: *LNAI*, vol. 1813, pp. 33–62. Springer Verlag

Lucas, S. & Reynolds, T.J. (2005). Learning Deterministic Finite Automata with a Smart State labeling Evolutionary Algorithm, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27 (7), 1–12

Luke, S.; Hamahashi, S. & Kitano, H. (1999). 'Genetic' Programming". In: Banzhaf, W. et al. (eds.) *Proc. Genetic and Evolutionary Computation Conf.*, pp. 1098–1105

Oncina, J. & Garcià, P. (1992). Inferring regular languages in polynomial update time, In: Perez, N. et al. (eds.) *Pattern recognition and image analysis*, pp. 49–61, Singapore, World Scientific

Pao, T. & Carr, J. (1978). A solution of the syntactic induction-inference problem for regular languages, *Computer Languages*, 3, 53–64

Parekh, R.G. & Honavar, V.G. (1996). An incremental interactive approach for regular grammar inference. In: *Proc. 3rd ICGI-96*, pp. 238–250, LNAI, vol. 1147, Springer, Berlin, Heidelberg

Pitt, L. (1989). Inductive inference, DFAs and computational complexity. In: *Proc. Int. Workshop on Analogical and Inductive Inference*, pp. 18–44, LNAI, vol. 397, Springer, London, UK

Pitt, L. & Warmuth, M. (1993). The Minimum Consistent DFA Problem Cannot Be Approximated within Any Polynomial, *J. ACM*, vol. 40, no. 1, 95–142

Porat, F. & Feldman, J. (1991). Learning automata from ordered examples, *Machine Learning*, 7, 109–138

Stolzmann, W. (2000). An Introduction to Anticipatory Classifier Systems, In: *LNAI*, vol. 1813, pp. 175–194, Springer-Verlag

Tomita, M. (1982). Dynamic construction of finite automata from examples using hill climbing, In: *Proc. 4th Annual Cognitive Science Conf.*, pp. 105–108, USA

Unold, O. (2005a). Context-free grammar induction with grammar-based classifier system, *Archives of Control Science*, vol. 15 (LI) 4, 681–690

Unold, O. (2005b). Playing a toy-grammar with GCS, In: Mira, J & Álvarez, J.R. (eds.) *IWINAC 2005,* pp. 300–309, LNCS, vol. 3562,. Springer Verlag

Unold, O. (2007a). Learning classifier system approach to natural language grammar induction, In: Shi, Y. et al. (eds.) *ICCS 2007*, pp. 1210–1213 Part II, LNCS, vol. 4488

Unold, O. (2007b). Grammar-based classifier system for recognition of promoter regions, In: Beliczynski B. et al. (eds.) *ICANNGA07*, pp. 798–805, Part I, LNCS, vol. 4431

Unold, O. & Cielecki, L. (2005). Grammar-based Classifier System, In: Hryniewicz, O. et al. (eds.) *Issues in Intelligent Systems: Paradigms,* EXIT Publishing House, Warsaw, 273–286

Unold, O. & Dabrowski, G. (2003) Use of learning classifier system for inferring natural language grammar. In: Abraham, A et al. (eds.) *Design and application of hybrid intelligent,* Amsterdam, IOS Press, 272–278

Waltrous, R. & Kuhn, G. (1992). Induction of finite state automata using second-order recurrent networks. In: Moody, J. et al. (eds.) *Advances in Neural Information Processing 4* , pp. 309–316, Morgan Kaufmann, San Francisco, CA

Wilson, S.W. (1995). Classifier Fitness Based on Accuracy, *Evolutionary Computation* 3 (2), 147–175

Younger, D. (1967). *Recognition and parsing of context-free languages in time n3,* University of Hawaii Technical Report, Department of Computer Science (1967)