What is a Learning Classifier System?

John H. Holland[1], Lashon B. Booker[2], Marco Colombetti[3], Marco Dorigo[4], David E. Goldberg[5], Stephanie Forrest[6], Rick L. Riolo[7], Robert E. Smith[8], Pier Luca Lanzi[3], Wolfgang Stolzmann[9], and Stewart W. Wilson[5,10]

[1] University of Michigan, USA
[2] The MITRE Corporation, USA
[3] Politecnico di Milano, Italy
[4] IRIDIA, Université Libre de Bruxelles, Belgium
[5] University of Illinois at Urbana-Champaign, USA
[6] Santa Fe Institute, USA
[7] Center for Study of Complex Systems
University of Michigan, USA
[8] The Intelligent Computer Systems Centre
The University of The West of England, UK
[9] University of Wuerzburg, Germany
[10] Prediction Dynamics, USA

Abstract. We asked "What is a Learning Classifier System" to some of the best-known researchers in the field. These are their answers.

1 John H. Holland

Classifier systems are intended as a framework that uses genetic algorithms to study learning in condition/action, rule-based systems. They simplify the "broadcast language" introduced in [26] by (i) eliminating the ability of rules to generate other rules, and (ii) by simplifying the specification of conditions and actions. They were introduced in [27] and were later revised to the current "standard" form in [28]. [31] gives a comprehensive description of this "standard" form, with examples. There are, however, many significant variants (e.g., Booker [9, this volume], Riolo [59], Smith [67, this volume], Wilson [83]).

In defining classifier systems I adopted the common view that the state of the environment is conveyed to the system via a set of detectors (e.g., rods and cones in a retina). The outputs of the detectors are treated as standardized packets of information, messages. Messages are used for internal processing as well, and some messages, by directing the system's effectors (e.g., its muscles), determine the system's actions upon its environment. There is a further environmental interaction that is critical to the learning process: the environment must, in certain situations, provide the system with some measure of its performance. Here, as earlier [26], I will use the term payoff as the general term for this measure.

1.1 Basic Questions

Classifier systems address three basic problems in machine learning:

Parallelism and coordination. A system is unlikely to have enough single, monolithic rules to handle situations like "a red Saab by the side of the road with a flat tire", but such a situation is easily handled by simultaneously activating rules for the building blocks of the situation: "car", "roadside", "flat tire", and the like. In short, a rule-based system can handle a broad range of novel situations if it can act on them with combinations of "building block" rules. Then combinatorics works for the system instead of against it. Moreover, because appropriate building blocks appear frequently, in a wide range of situations, they are tested and confirmed at a high rate. The problem is to provide for the interaction and coordination of a large number of rules that are active simultaneously.

Credit assignment. Deciding which rules in a rule-based system are responsible for its successes, particularly when long sequences of "stage-setting" actions precede success, is an interesting and difficult problem. If the system is not continually monitored by a referee, solution of this problem is a sine qua non for learning. The problem is to credit an early action, which may look poor (as in the sacrifice of a piece in chess), for making possible a later positive action (as in setting the stage for the capture of a major piece).

In realistic situations, exhaustive exploration of all possible action paths (as in dynamic programming and Q-learning) is not feasible. So the credit assignment scheme must be "local". When many rules are active simultaneously the situation is exacerbated. Only a few of the early-acting rules may set the stage, while other rules active at the same time may be ineffective or, even, obstructive. Samuel's early work [65], using prediction over extended action sequences, points the way, but few have exploited his insights.

Rule discovery. Rule discovery (or its equivalent) is the most recondite problem in machine learning. It was not even well-handled in Samuel's remarkable work. We know that rules receiving little credit ("low strength") should be replaced, but random generation of new rules can only work for the smallest problems. The key to effectiveness is use of past experience to generate plausible hypotheses (rules) about situations as yet poorly understood – hypotheses not obviously contradicted by past experience.

Behind these three basic problems is a deeper question: How does a system improve its performance in a perpetually novel environment where overt ratings of performance are only rarely available? Obviously such improvement is not possible unless the environment has repeating (sub-)patterns. A learning task of this kind is more easily described if we think of the system as playing a game of strategy, like checkers or chess. After a long sequence of actions (moves), the system receives some notification of a "win" or a "loss" and, perhaps, some indication of the size of the win or loss. There is little information about specific changes that would improve performance. Most learning situations for animals, including humans, have this characteristic – an extended sequence of actions is followed by some general indication of the level of performance.

1.2 How Classifier Systems Address These Questions

In classifier systems, parallelism and coordination are addressed by restricting rule action to the emission of messages. Tags in the messages allow flexible "addressing", providing coordination of activities. Detectors translate the current state of the environment into messages, and effectors translate selected messages into actions on that environment. The overall classifier system, then, can be viewed as a message processing system acting on the current list (set) of messages. Because messages only serve to activate rules, questions of "rule consistency" and "consistency maintenance" are avoided. More messages simply mean more active rules, and vice-versa.

Credit assignment is handled by setting up a market situation. In that market each rule acts as a go-between (broker, middleman) in chains leading from the current situation to (possible) favorable outcomes. At each instant, rules with satisfied conditions bid for the right to become active. If a rule becomes active, it pays its bid to the active predecessor(s) that sent messages satisfying its conditions (its "suppliers"). As an active rule, it then stands to profit from bids of subsequent bidders (its "consumers"). Rules active at the time of external payoff (reward, reinforcement) are apportioned the payoff as if it were income from "ultimate consumers".

Credit is accumulated by the rule as a strength (a kind of capital). Many of the variants of classifier systems offer alternative schemes for apportioning or accumulating credit. In general, whatever the apportionment scheme, stronger rules bid more, thereby being more likely to win the bidding process. That, in turn, makes those rules more likely to influence the system's behavior.

Rule discovery exploits the genetic algorithm's ability to discover and recombine building blocks. Classifier systems have "building blocks" at two levels: the parts (schemata) from which the condition and action parts of individual rules are constructed, and the rules themselves, as components of the overall system. The genetic algorithm works on this "ecology" at both levels. Because it is designed to work on populations (sets of rules), it is well suited to the task. Indeed classifier systems were designed with just this objective in mind.

Rule strength enters at both levels, being treated as a fitness (likelihood of being the parent of new rules) by the genetic algorithm. Variants offer different ways of translating strength into fitness, contingent on the particular variant used for credit assignment.

Each of the mechanisms used by the classfer system has been designed to enable the system to continue to adapt to its environment, while using its extant capabilities to respond instant-by-instant to that environment. In so doing the system is constantly trying to balance exploration (acquisition of new information and capabilities) with exploitation (the efficient use of information and capabilities already available).

1.3 Implementation

The computational basis for classifier systems is provided by a set of condition-action rules, called classifiers. A typical (single condition) rule has the form:

```
IF there is (a message from the detectors indicating)
            an object left of center in the field of vision,
THEN (by issuing a message to the effectors)
     cause the eyes to look left.
```

More generally, the condition part of the rule "looks for" certain kinds of messages; when the rule's conditions are satisfied, the action part specifies a message to be sent. Messages both pass information from the environment and provide communication between classifiers. Because many rules can be active simultaneously, many messages may be present at any given instant. From a computational point of view, it is convenient to think of the messages as collected in a list.

A computer-based definition of these rules requires a proper language for representing classifiers. It is convenient to think of the messages as bit strings (though representation via any alphabet or set of functions is equally possible). We can, in addition, think of each bit position as representing the value of a predicate (1 for true, 0 for false) though, again, this is only a convenience and not necessary for more complex representations. Classifier conditions then define equivalence classes (hyperplanes) over the message space using the *don't care* symbol '#'. More recently, it has proved useful to introduce a symmetry that allows messages to act over equivalence classes of conditions, using the 'fits all' symbol '?'. Other variants have been studied (e.g., [86, this volume], [1]).

If messages are bit strings of length m over the alphabet 1,0, then conditions are strings of length m over the alphabet 1,0,# and actions are strings of length m over the alphabet 1,0,?. The genetic algorithm mates these strings to produce new competing rules.

1.4 Future research.

In recent years there has been a focus on classifier systems as performance systems or evolutionary incarnations of reinforcement systems (e.g., Lanzi [47], Wilson [87, this volume]). This has been fruitful but, I think, falls short of the potential of classifier systems. Smith [67, this volume] makes the point that classifier systems, in their ability to innovate, offer broader vistas than reinforcement learning.

In my opinion the single most important area for investigation vis-a-vis classifier systems is the formation of default hierarchies, in both time ("bridging classifiers" that stay active over extended periods of time) and space (hierarchies of defaults and exceptions). Having been unable to convince either colleagues or students of the importance of such an investigation, I intend to pursue it myself next summer (year 2000).

2 Lashon B. Booker

Speculations about future directions for classifier system research can benefit from a look at Holland's original research goals in formulating the classifier system framework [27, 31]. The early motivations for classifier system research were derived from the many difficulties that a learning system encounters in a complex environment. Realistic environments are both rich and continually varying. They force a learning system to deal with perpetually novel streams of information, and often impose continual requirements for action given sparse payoff or reinforcement. There are many examples of natural systems and processes that handle such challenges effectively: the central nervous system, co-adapted sets of genes, the immune system, economies and ecologies. Most artificial systems, on the other hand, tend to be brittle in the sense that substantial human intervention is often required in order to realign system responses to address changes in the environment. Holland identified a list of criteria for avoiding brittle behavior. These criteria summarize key aspects of what appears to lie behind the flexibility exhibited by natural systems: they implement processes that build and refine models of the environment. Moreover, these models typically involve massive numbers of elements adapting to the environment and to each other through many local nonlinear interactions. Classifier systems were proposed as an example of a rule-based system capable of learning and using multi-element models of this kind.

While initial research on classifier systems focused on many of the now-familiar computational mechanisms and algorithms, it is important to keep in mind that "The essence of classifier systems is a parallelism and standardization that permit both a 'building block' approach to the processing of information and the use of competition to resolve conflicts." [31, p. 611]. The "standard" computational procedures are significant because they illustrate how it is possible, in principle, to design and build a system that meets the criteria for avoiding brittleness. However, in my view, it is the design principles that provide the insights about what a classifier system "is". For that reason, I find it useful to think of the classifier system framework in the broadest sense as a general-purpose approach to learning and representation. This approach provides a way of looking at problems and structuring flexible systems to solve them, but it does not necessarily prescribe the detailed methods that are best suited for solving a particular problem class. Research in classifier systems typically focuses on computational methods, but too often advances in methodology are not leveraged to further our understanding of the broader issues that the classifier system framework was designed to address. I believe that the most important goal for classifier system research is to develop deeper insights about how our computational methods bring us closer to realizing the capabilities Holland envisioned in his original concept.

One way to achieve this goal is to develop more formal characterizations of classifier system methods and processes. There has been relatively little work along these lines since Holland's [30] early attempt to characterize what a regularity is in a perpetually novel environment, and what it means for a classifier system to exploit these regularities. An agenda of theoretical issues that need to be pursued was proposed more than a decade ago [10] and it remains relevant today. Research like this is needed so that classifier system design decisions can be grounded in clear, preferably formal statements of the technical issues being addressed. We also need need to be able to characterize the capabilities and limitations of classifier systems more carefully, including some expectations regarding the quality of the solutions the system is likely to learn.

Another way to broaden our understanding of classifier system design principles is to study them from a wider variety of perspectives. The classifier system research to date has focused almost exclusively on the learning and representation issues that are prominent in reinforcement learning problems. Recently there have been systems and methods proposed that emphasize other perspectives on learning and representation. These perspectives have been derived from studies of the immune system [64], agent-based economic models [18], and ecologies [9]. New computational methods will emerge from this research, and similar excursions should be encouraged. Hopefully, this will lead to new insights about how to build systems that confirm Holland's original intuitions about the potential of the classifier system framework.

3 Marco Colombetti and Marco Dorigo

During most of the nineties, we have been using learning classifier systems (LCSs) to develop simulated animats and real robots able to learn their behavior. The types of behavior we focussed on were quite zoomorphic (exploring an environment, searching for and chasing a prey, fleeing from a predator, and so on). However, we were more interested in investigating ways of using reinforcement to *shape* the behavior of an agent, rather than in studying how a system can spontaneously adapt to a natural environment. We viewed our work mostly as a contribution to *behavior engineering*, as we call the discipline concerned with the development of flexible autonomous robots. Most of the results of this work were finally collected in a book [17].

Throughout our research, we regarded LCSs as a promising model for reinforcement learning (RL). For this reason, we have been surprised to see that LCSs are often contrasted with RL, as if they were different approaches. In our opinion, it is important to distinguish between RL as a class of problems, on one side, and the learning techniques typically studied by the RL community, on the other side. As is well known, the fundamental problem of RL is defined in the following setting (Fig. 1). An *agent* interacts with an *environment* through *sensors* and *effectors*. At times, the agent receives from the environment a scalar signal, called *reinforcement* (r). The problem is to exploit the agent's experience (i.e., the history of its interactions with the environment) to develop a behavior policy that maximizes some functional of reinforcement over time.

RL problems can be studied from two different viewpoints. They can be regarded as abstract problems, and dealt with in a purely mathematical way. The main point here is to define learning algorithms of low computational complexity

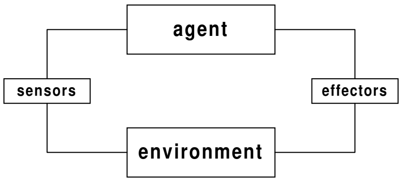

Fig. 1. The reinforcement learning setting.

that are guaranteed to converge to an optimal behavior policy. Alternatively, an RL researcher may take a more concrete standpoint, and focus on the agent as a software system. The main questions now become, H ow can we represent behavioral policies in a compact way? How can we efficiently implement RL algorithms on such representations? Should we trade policy optimality for higher computational efficiency, and how? Can we develop RL systems that scale up to problems of realistic size? These are the questions we had in mind when we started to apply RL techniques to robot shaping.

Since the beginning of our work, we felt that LCSs were fit for solving the RL problems we wanted to tackle. The features we found most appealing were the overall architecture of LCSs, and the use of a genetic algorithm (GA) in the discovery component. As far as architecture is concerned, we believe that the production rule paradigm, of which LCSs are an example, is particularly suitable for implementing adaptive agents. This paradigm allows for a very peculiar interaction of simple behavioral rules, in such a way that both cooperation and competition among several rules are exploited to generate opportunistic behavior. Production rules have been very successful within the classical approach of symbolic artificial intelligence, and we think that the reasons of such a success are still valid in a more low-level approach.

The main reason of our interest in LCSs, however, was due to its evolutionary component. As we have argued elsewhere [13], a major aspect of behavior engineering is the attempt to go beyond the limits of "rational design." By this we mean that the behavior of a real robot cannot be completely predefined. Learning, quite obviously, is a way of overcoming ignorance. However, ignorance is a problem not only for the learning agent, but also for the robot's designer. For an agent interacting with a physical environment, learning is going to be not only a matter of optimally tuning a set of parameters, but also of discovering structure. An optimal coupling of an agent with its environment is only possible if the dynamical structure of the environment is somehow mirrored by the structure of the agent's representation of its own behavior policy.

The evolutionary component of an LCS has the remarkable ability to produce a compact representation of a relevant subspace of all possible behavior rules. In principle, this gives LCSs the power to scale up to problems of realistic size. Unfortunately, in our work we found that the learning power of LCSs is still too weak to deal with the kinds of problems involved in concrete applications of autonomous robots. We tackled this difficulty in the following way. First, we gave our learning agents as much input information as possible, by adopting a step-by-step reinforcement schedule. Second, we endowed our agents with some "innate" structure, by predesigning a hierarchical architecture for behavior control. To do so, we exploited ALECSYS [16, 15], a transputer-based system that allows for the implementation of a network of interconnected LCSs. As a result, a substantial part of the final agent behavior is actually suggested by the designer, through the development of a suitable architecture and through a skillful definition of the reinforcement function. Notwithstanding this, we believe that the results are still of great interest for behavior engineering. In particular, the use of an RL approach allows the robot designer to concentrate on a high-level specification of the target behavior (through the definition of the reinforcement function), thus avoiding the risk of being caught up in a bundle of details.

Since we completed our work on robot shaping, much water has flown under the bridge of LCSs. The introduction of the XCS model by Wilson [83] appears to us to be a major achievement, for both theoretical and practical reasons. From a theoretical point of view, the main virtues of XCS are that it is a very neat model, and that it stresses the belonging of LCSs to the field of RL. We hope that, in the future, the RL community will be more inclined to regard LCSs as an interesting approach to the solution of RL problems. From the practical point of view, XCS has the advantage, at least to our eyes, that it gets rid of some aspects of the LCS model that, although very interesting in principle, have not proved to work effectively in practical applications. In particular, we want to mention the mechanism by which an LCS is expected to form rule chains that may be considered as simple behavior plans. In our experiments, we never happened to observe such a phenomenon. The current approach based on the use of memory registers [46, 52] seems to us to be a more promising way of overcoming the limitations of simple reactive behavior.

4 Stephanie Forrest

Classifier systems were originally proposed as a model of inductive processes in human-like cognitive systems [25]. The basic classifier system architecture incorporates ideas from artificial intelligence, machine learning, cognitive psychology, economics, evolution, and computer design into one framework. The system has the following components: parallel forward-chaining rules, a reinforcement learning algorithm (the bucket brigade), a set of "genetic operators" for evolving the rule set, a simple interface with the external environment, and a theory of model construction based on homomorphic maps.

Classifier systems represented an advance in our thinking about cognitive modeling in several ways. Representations in classifier systems were (and are still) fine-grained—the execution of a single rule represented a much smaller reasoning step that those commonly found in symbolic production systems [56], and information was encoded as simple bit strings, rather than complex data structures, as in semantic networks or conceptual dependency theory. One hypothesis about classifier systems was that symbolic-level reasoning would arise spontaneously from fortuitous interactions and combinations of individual rules. Such symbols would be discovered and manipulated naturally by the system, fluidly and implicitly, without the need for explicitly named locations in memory such as those found in the symbolic reasoning systems mentioned earlier. Many rules could be active simultaneously, thus allowing the system to consider multiple, and even conflicting, hypotheses up until the time that an output decision was made. But because the message list had finite size, there was also the possibility of competition among rules, allowing the system to focus. By contrast, most other reasoning systems of the day required that the system be maintained in a logically consistent state at all times. Classifier systems incorporated two important forms of learning—the bucket brigade to assign credit (reward) among among classifiers, and various discovery operators for generating variants and recombinations of existing successful rules. These learning mechanisms, combined with the 1,0,# vocabulary of rules, allow the system to discover and represent knowledge in terms of equivalence classes. This is important in environments with large numbers of states, where methods such as Q-learning [79] are problematic because of the huge number of possible state/action pairs. Finally, and in my view, most importantly, classifier systems were constructed as the realization of a formal theory about how intelligent systems construct internal models of their environment and use those models to enhance their existence. This theory, based on homomorphic maps and an extension, known as *quasi-morphisms*, remains as one of the central contributions of classifier systems.

With a few notable exceptions, for example [60], the past twenty years of research on classifier systems has focused on engineering the architecture so that it exhibits plausible computational behavior and can solve interesting real-world problems. Although great progress has been made on this front, there has been relatively little emphasis on the cognitive modeling issues that classifier systems were intended to address. Of these, perhaps the most central outstanding question is what symbols are and how they are discovered, maintained, and used effectively. It is commonly believed that higher animals, such as humans, rely heavily on the use of symbols to represent and manipulate models of the environment, for language, and other abstract cognitive tasks. However, we still have little understanding of how symbols are stored and processed in the brain. These and other questions related to classifier systems as a model of cognitive activity are an important direction for future research.

Ideas about situated intelligence, such as those described in [23, 12], have changed our views about the nature of intelligent artifacts. Natural systems exhibiting intelligent behavior are now understood as having co-evolved with their environments, rather than as isolated designs. The interactions between these intelligences and their environments are central to their success, even in the case of brains where the majority of neural activity is internal to the system (e.g., when we are asleep and dreaming). Brains are structurally organized around important sources of sensory input, such as the visual system, and are continuously exposed to environmental signals, even during embryonic development, where spontaneous retinal activity allows topographic mapping of the retina onto the developing visual cortex. Situated intelligent artifacts are perhaps more complex to think about, because they cannot be neatly separated from their environments, but they can in some cases use their environments in ways that simplify their computations.

Thus, a second important area of future research is to rethink our assumptions about classifier design from the perspective of situated intelligence. Classifier systems were originally intended as a generic architecture for cognitive systems, one that could be placed in a wide variety of different environments, and this has biased our ideas about how to deploy classifier systems. For example, most classifier systems have narrow-bandwidth interactions with their environments, and it might be appropriate to think about designing them from a more perceptual perspective, creating systems that are "awash" in environmental messages, as for example, in ref. [64] where a system resembling a classifier system is situated in a live local area network, exposed to a constant flow of TCP/IP packets.

One missing component in classifier systems is the ability to aggregate sets of rules into encapsulated components, analogous to subroutines in conventional programming languages. Classifier systems are "flat" in the sense that all rules have the same status, and groupings of rules into subroutines (corresponding to subassemblies in Hebbian models) are intended to occur automatically, without an explicit reinforcing mechanism. Although it may be technically possible to design rule sets that have this property, through the use of tags and bridging classifiers, it is highly unlikely that robust logically isolated components will be discovered and sustained through the learning operations of the classifier system. Rather than trying to improve our learning algorithms or to devise cleverer representations, I believe that an explicit mechanism is needed, analogous to Koza's automatic function definition for genetic programming [43]. Such a mechanism would recognize effective clusters of classifiers, aggregate them into a single unit such that the individual rules were not directly accessible from outside the cluster, define a limited interface to the rest of the system, and protect them from the ongoing pressures of mutation and crossover.

The original insights which inspired the design of classifier systems remain compelling, and they address important and unresolved issues in our understanding of intelligent systems. Few would argue that the exact mechanisms employed by classifier systems are those used in human brains, but the classifier system serves as an illustration of a set of design principles that are central in the design, and our understanding of the design, of many intelligent systems, including the brain. I believe that it is important for researchers to focus more on the basic principles exhibited by classifier systems and less on the specific implementation that was introduced nearly twenty years ago.

As an example, Steven Hofmeyr recently developed a model immune system which resembles the spirit of classifier systems, but implements few of the architectural details in the same way [24]. Detectors in this system represent generic immune cells, combining properties of T-cells, B-cells, and antibodies, and correspond to the condition parts of classifiers. However, instead of using a match rule based on the $1, 0, \sharp$ alphabet traditionally used in classifier systems, Hofmeyr adopted one called *r-contiguous bits* [58] based only on bit strings. To extend the immune detector into a full-fledged condition/action classifier rule, a few bits can be concatenated to each detector to specify a response (analogous to different antibody *isotypes*). Instead of associating a strength with each rule, each detector in the model immune system has a life cycle, consisting of immature, naive, activated, memory, and death states. These states reflect how successful the detector is and how long it will live. The mechanisms that control how detectors move through their life cycle correspond to the role of the bucket brigade (credit assignment) in classifier systems. However, learning in the model immune system is simpler than classifier systems in the sense that credit is assigned directly from the environment to the detectors, and strength is not passed among immune cells. Internal feedbacks and self-regulation are modeled through a primitive form of a cytokine system (signalling molecules secreted by immune cells, which diffuse locally and either stimulate or downregulate other immune cells). In place of the message list, the system is intended to live in a computer network environment, constantly exposed to new signals. Bidding for messages in classifier systems is analogous to immune cells competing to bind to foreign datapaths. Although the mapping between this model immune system and classifier systems is not $1-1$, it captures many of the important properties of classifier systems and provides an example of how the spirit of classifier systems might be realized in new ways.

5 David Goldberg

Some Reflections on Learning Classifier Systems. I appreciate the editors' invitation to contribute to this important volume marking what must be called a *renaissance of learning classifier systems* (LCSs). Although I have kept my finger in the LCS pie through occasional papers on LCS subjects, the main body of my work shifted following my 1983 dissertation applying genetic algorithms (GAs) and LCSs to gas pipeline control; in recent years, I have largely focused my efforts to develop (1) an effective methodology for the design of GAs, (2) an integrated design theory for selectorecombinative GA design, (3) competent genetic algorithms—GAs that solve hard problems, quickly, reliably, and accurately, and (4) efficiency enhancement technologies (EETs) for faster solutions.

In this short essay, I'd like to give some personal reflections on why I shifted my work away from learning classifier systems, what I learned from those efforts, and why I am renewing a more active program in LCS research. I start by reflecting on my involvement in classifier systems back in the eighties.

Some Ancient LCS History. When I first proposed my dissertation topic in 1980, I was under the impression that these things called classifier systems were well understood and that my dissertation would be a relatively simple exercise in application. After all, as I sat through John Holland's lectures at the University of Michigan, it all sounded so plausible: roving bands of rules fighting, mating, competing, dying, giving birth, and so forth. The first indication that there might be some work to do was when I went to look for some computer code to build on and found that there wasn't any available to me. The second indication of difficulty came when I actually built a classifer system on my little Apple II computer in Pascal, turned it on, and realized that productive system performance was completely drowned out by the action of myriad parasitic rules.

Thereafter, I realized that I couldn't shotgun my way to success. I knew that I needed to engineer the system well using bounding analyses, careful experiments, and intuition, and I proceeded to construct what I believe was the first classifier system to exhibit a default hierarchy. True, by today's standards, the accomplishment was modest, but I did succeed in creating a system with rules and messages, apportionment-of-credit, and genetic learning. The university granted me my PhD, and I moved on to the University of Alabama, wondering what I should do next.

A Quest for Competent GAs and 3 Lessons. Some pointed questioning in my dissertation defense helped me realize that EVERY aspect of classifier system design at the time was built on intellectual quicksand. Elsewhere, I have called classifier systems a *glorious quagmire*, and I believe this description at that time was apt. Genetic algorithms had bounding schema theory, but competent GAs had not yet been designed. The bucket brigade had the right idea, but the theorems of reinforcement learning were not yet available to us, and no one knew what rule syntax was necessary to solve the problems at hand. This situation seemed untenable to me. After all, in physical system design, engineers are used to having the laws of physics to guide system design. Here, the design environment was shifty, to say the least.

My immediate reaction to that environment was to look at the heart of genetics-based machine learning (GBML), the genetic algorithm, and see if I could tame GA design somewhat. My reasoning was that with effective GAs in hand, I would be able to return one day to classifier system design without the core rule induction method as a big question mark. This is not the forum for a detailed discussion of these explorations in competent GAs; various papers and a forthcoming monograph will better serve this function for the reader. Instead, I would like to briefly discuss three fairly high-level lessons of this journey, lessons that I believe carry over to the problem of competent LCS design.

Specifically, my explorations in competent GA design have taught me the importance of three things:

1. a proper methodology of invention and design,
2. the right kind of design theory, and
3. the right kind of test functions.

I briefly discuss each somewhat further.

Elsewhere I have borrowed from my colleague Gary Bradshaw and used the Wright brothers and their method of invention and design to help designers of complex *conceptual machines* such as genetic algorithms and classifier systems to understand how to build conceptual machines that work. Too often in the domain of the conceptual, mathematical rigor is a siren song that wrecks the design voyage on the shoals of proof. It is important in designing any complex system to (1) decompose the design problem, (2) use effective, economic design theory to organize experimentation and guide design efforts, and (3) integrate the pieces of the design with dimensional reasoning. This method of invention is commonplace in the design of *material machines* such as airplanes, automobiles, and toasters, but it is not so frequently adopted in algorithmic circles. Design is design is design, and the rules of design don't change just because we are designing a genetic algorithm or a classifier system instead of an airplane or an automobile.

A key ingredient of this methodological prescription is economic design theory. By "economic" I mean that the models need to have *low costs in use*. In aerodynamics, designers use "little models" such as lift coefficients and drag coefficients to quickly and cheaply understand the magnitude of lift and drag forces an aircraft is likely to encounter and how those forces will scale as the aircraft is made larger and smaller. Genetic algorithmists and classifier system designers need to find and use appropriate "little" models to understand the various facets of GA and LCS design and how the "forces" acting on our systems scale as the systems change "size." Too much of our theory literature is devoted to elegant equations that tell us little about the design of our systems. The Wright brothers eschewed the Navier-Stokes equation in favor of lift and drag coefficients, much to the betterment of modern aviation. Perhaps we should do likewise.

One aspect of design theory that comes up in all contexts is the use of *boundedly difficult* test functions. In the domain of GAs, I have used the term *deception* to describe a key facet of GA problem difficulty. Some researchers are confused about the goals of such work, but the main idea is to imagine a system's problem from hell and use boundedly hellish problems to test the procedure. In general, we know the problem from hell is too difficult to solve quickly, but we should not give up on designing procedures that scale nicely on problems of lesser or bounded difficulty. Competent GAs being designed today have this nice scaling property and similar continuing concern for problem difficulty in LCS work should pay dividends.

A Return to LCSs. Recently, I have returned to a more active role in LCS research through collaborations with a number of LCS researchers, and I have been pleased (1) by the amount of fun that I'm having, (2) by the amount of progress that has been made in the field, (3) that my old LCS knowledge isn't completely useless, and (4) that the lessons of my competent GA journey appear to be quite applicable to the project of competent LCSs. I suppose I shouldn't be surprised by having fun with LCSs. The design of general-purpose learning devices such as LCSs is an engaging business, pure and simple. And I suppose that I shouldn't be surprised by the progress in the field. For a time it almost looked as if LCSs might die off, but stalwarts led largely by Stewart Wilson have kept the fire burning with important ties to the reinforcement learning literature and a number of novel approaches.

It does strike me that much of the progress has been made on the apportionment-of-credit/reinforcement-learning side of the ledger. The nature of the genetic algorithms in use appears not to have been much affected by the GA/ES/GP innovations of the last decade, and I hope that this an area where I might be able to make a contribution. At the very least, the competent GAs that have been developed over the last decade should be adapted to LCS usage and this should benefit the search for appropriate rules in difficult problems. Finally, it seems to me that the very complexity of the LCS design task deserves a systematic design methodology, a tractable design theory, and a systematic means of integrating and coordinating the function of different subsystems. My struggle with the design of competent GAs was greatly aided by discovering and using these things, and I believe that our continued struggle toward competent learning classifier systems will be similarly rewarded by adopting an analogous approach.

6 Rick L. Riolo

There are many answers to the question "What is a classifier system," all of them reflecting different views of the field and different uses for classifier systems. For me, the most interesting way to view a Holland/Burks (Michigan-style) classifier system [34] is as a way to *model* adaptive systems [33]. To that end, such a classifier system should have most or all of these general characteristics:

- A message board for communication.
- Rule-based representation of knowledge.
- A competition for rules to become active, biased by inputs, past performance, and predictions of expected outcomes.
- Parallel firing of rules, with consistency and coordination of activity arising endogenously, as an emergent property established and maintained by the dynamics of the bidding processing. Explicit conflict resolution is strictly enforced only at the effector interface.
- Credit allocation is done by temporal difference (TD) methods of some type, e.g., the traditional bucket-brigade algorithm, some kind of profit-sharing scheme, Q-learning algorithms, and so on. Note that there might be multile kinds of credit being allocated at the same time, e.g, traditional strength representing expected payoff based on past performance, some measure of rule payoff accuracy or consistency [83] or some measure of a rules ability to predict future state [59, 69].
- Rule discovery is done by heuristics appropriate to the system being modeled. Examples include triggered coupling to capture asynchronic causal connections [29, 63], surprise-triggered prediction [35], or traditional genetic algorithms with with mutation and recombination.

When it doesn't conflict with other modeling goals, a classifier system should also have one other feature which I think was in Holland's original view, namely it should use a *simple* rule syntax and semantics. One reason to use simple rules (and a simple architecture and mechanisms in general) is to retain the possibility of taking advantage of parallel hardware [62, 75]. But in addition, the use of simple rules (and mechanisms) makes it easier to build mathematical models which might be analytically tractable [30, 80]. However, despite these arguments for simplicity, both for some modeling goals and for classifier systems being designed to do a particular task, it may be more productive to allow more complex rules, both in terms of matching capabilities and processing power.

Over the past ten years there has been much emphasis on using classifier systems as just that, i.e., systems to solve classification problems of one kind or another. In that context, much progress has been made in improving classifier system performance on a wide variety of problems [49]. However, when using classifier system for modeling purposes, the goal is not just to get the best performance for the least computational effort. Instead, a more important criteria is how well the classifier system exhibits the relevant behavior of the system being modeled, using mechanisms plausible in that context. Thus studies which most interest me are those that either: (A) explicitly have as a goal modeling some cognitive or other adaptive system, as in [36, 8, 35, 19, 38, 2, 70, 14], or (B) explore the fundamental dynamical properties of classifier systems with particular architectures and mechanisms, with an an eye toward understanding what kinds of models could be built with such systems [81, 22, 61, 69].

Finally, I must admit that the kinds of systems that are most interesting to me also are the most difficult to study and understand. There is no question that in the past five to ten years much has been learned by studying simpler systems like ZCS [82], and that what we have learned will serve as a good base for future research. However, for classifier systems to reach the full potential originally outlined for them by Holland, I think we must move toward research on more complex systems. In particular, these include classifier systems which allow multiple rules to fire and post messages in parallel, which have the potential to link rule activation over time by way of tagging mechanisms, and which are set in environments that only give "payoff" occassionally and are rich enough to require extensive generalization. These are, then, environments which will require classifier systems to construct and use general, multi-step models if they are to perform well. Thus the recent work on multi-step environments and non-markov environments [51], and anticipatory cfsystems [71] are, to me, signs that the field is advancing very well.

Of course as we again move toward running more complex classifier systems, we should expect difficult issues and problems to arise which are not generally seen in simpler systems. In particular, I expect the biggest challenges to be a result of the fact that such classifier systems will have a rich "ecology" of rules, consisting of intricate interactions and complicated dependencies between rules which are involved in both competitive and cooperative relations with each other. For instance, we shall again be faced with the emergence of various kinds of parasites and free riders, which are ubiquitous in natural ecologies and other similarly complex adaptive systems. Since natural ecologies don't have an externally imposed task or performance metric, parasites are just another part of the glory of nature. On the other hand, complex adaptive systems that do have an external performance metric (e.g., individual metazoans must survive to reproduce) have developed a host of complex mechanisms to manage problems that always arise in multi-level organizations of competing entities which also must cooperate to perform best [11, 7, 55]. Both for those who are taking an engineering approach to classifier systems, i.e. who want to design and use them for particular exogenously determined tasks, and for those who want to use classifier systems to model complex adaptive systems, these and other unanticipated side effects will provide great challenges [39].

7 Robert E. Smith

To begin to consider learning classifier systems (LCSs), one must first consider what an LCS is.

This is not as clear as one might imagine. A typical description of a LCS will include rules, usually taken from the common {1,0,#} syntax, that are acting as population members in a genetic algorithm. The typical description will also include some form of match-and-act, conflict resolution, and credit assignment mechanisms, that facilitate the rules interacting to influence some "environment". There are typically two variations of LCS, the "Michigan approach", where each rule is a separate individual in the GA population, and the "Pitt approach", where each GA population member is a complete set of rules for the given application environment. Comments in this section will begin with an assumption of the "Michigan approach" LCS, but it is important to note bridges between these two approaches are an important area for further investigation. Although the elements discussed above are typical to LCS descriptions, they may not define the LCS approach. In particular, specific implementation details of LCS syntax, and the system components that facilitate rule interactions with the environment, may obscure the essence of the LCS approach. Fundamentally, the LCS is defined by a population of entities that

- act individually, responding to, and taking actions on, an external environment,
- while evolving as population members, under the action of evolutionary computation.

This definition is independent of syntax or implementation details. Given this definition, the most significant complexity in the LCS approach is the need for co-evolution of entities.

Co-evolution as an approach to solving complex problems is the key thrust of the LCS approach. Co-evolution is at the cutting-edge of evolutionary computation research. In general, it involves a complex balancing act between the competitive pressures of evolution, and the cooperative interactions needed to positively effect the environment. Co-evolution brings in some of the most complex issues of evolutionary systems, including emergence the maintenance of steady-state diversity, emergence of species, symbiosis, parasitism, and others.

Because these issues involve evaluating the utility of functional interdependent entities, the issue of credit assignment (and related issues of conflict resolution) must be carefully considered. Fortunately, recent advances in machine learning (specifically in reinforcement learning) have substantially clarified these issues. While the LCS credit assignment and conflict resolution schemes of the past relied on tenuous analogies as their bases, well established reinforcement learning schemes now provide a firmer foundation upon which to build LCS advances.

Within the field of reinforcement learning, there are substantial questions to which the co-evolutionary approach of the LCS may provide answers. Chief amongst these is the need for schemes that automatically form generalizations. Another issue of importance is the formation of internal memory processing to cope with non-Markovian environments. Both of these issues depend on complex interactions of entities (like rules and associated messages), for which the co-evolutionary approach of the LCS may be well suited.

Although reinforcement learning provides a framework in which LCSs are likely to advance, it is important that this framework not become a limitation. In particular, reinforcement learning usually focuses on well-defined concepts of Bellman optimality. While these formalisms provide a well-defined basis for evaluating and developing LCS technology, they may not encompass all the ways in which LCSs may be used.

So-called "artificial intelligence" and "machine learning" technologies have often responded to an antiquated vision of computation, where computers provide faster, more accurate solutions than humans. In this vision, concepts of well-defined optimality are the goals of AI. However, these goals are not those we would usually assign to humans. Humans are usually asked simply find solutions that are "better" than those found in the past, in some sense that has no clear mathematical definition. Moreover, there is often a real world premium on human solutions that are simply innovative or creative. Such concepts are unlikely to receive mathematical definition.

Given the ubiquity of networked computers, more modern visions of computation are emerging, where computers may be expected to provide innovation and creativity, like their human counterparts. When one observes that evolution is directed at ongoing adaptation and new ways of exploiting available resources, the co-evolutionary approach of the LCS may be of particular value within this new vision of computation.

In summary, important areas for immediate, formal investigation of LCSs include:

- Formal consideration of co-evolution,
- Integration of LCSs within the framework of reinforcement learning (including generalization and application to non-Markovian tasks).

As these investigations advance, application of LCSs should continue to be explored, both within realms of formally defined problems, and in systems that required automated innovation, novelty, and creativity.

8 The Editors

8.1 Pier Luca Lanzi

Since spring 1997, I have been teaching learning classifier systems as a part of a course on Knowledge Engineering and Expert Systems for Master students at the Politecnico di Milano. Teaching requires *neat definitions* to improve the students' understanding of new concepts and *clear motivations* to ground abstract models to real world problems. Thus teaching learning classifier systems necessitates an answer to two basic, though important, questions: *what* is a learning classifier system? And *why* do we use learning classifier systems?

As this chapter witnesses, there are many ways of answering the former question. Among the possible ones, my favourite answer is:

> Learning classifier systems are a Machine Learning *paradigm* introduced by John Holland in 1978. In learning classifier systems an agent learns to perform a certain task by *interacting* with a partially unknown environment from which the agent receives feedback in the form of numerical *reward*. The incoming reward is exploited to guide the *evolution* of the agent's *behavior* which, in learning classifier systems, is represented by a set of rules, the *classifiers*. In particular, *temporal difference learning* is used to estimate the goodness of classifiers in terms of future reward; *genetic algorithms* are used to favour the reproduction and recombination of better classifiers.

I like the term "*paradigm*" because it stresses the fact that classifier systems *do not* specify an algorithm but they characterize an *entire class* of algorithms. I use "*interacting*" and "*reward*" to point out that learning classifier systems tackle *also* reinforcement learning problems and therefore should be considered a reinforcement learning technique. The terms "*evolution*," "*genetic algorithms*," and "*behavior*" highlight the fact that, in classifier systems, learning is viewed as a process of ongoing *adaptation* to a partially unknown environment, *not* as an optimization problem as in most reinforcement learning. Finally, the term "*temporal difference learning*" states that the methods used to distribute the incoming reward to classifiers are analogous to (and sometimes the same as) those techniques employed in "traditional" reinforcement learning.

Given that there is a general agreement on what is a classifier system yet a question remains: *why do we use learning classifier systems?*

This question can be answered according to two different perspectives. First, we can look at learning classifier systems as reinforcement learning techniques. In such a case to answer the question above we should present a set of applications in which learning classifier systems prove to be better than "traditional" reinforcement learning techniques. Alternatively, we can observe that learning classifier systems are more general than those traditional reinforcement learning techniques that are inspired by methods of Dynamic Programming (e.g., Watkins' Q-learning [79]). Those techniques in fact usually make a number of assumptions on the environment (e.g., the environment must be a Markov Decision Process) and on the agent's goal (e.g., the agent's goal must be formally defined as a maximization problem) that learning classifier systems do not require. For instance, the goal of a learning classifier system might be very general (e.g., to *survive*) and not expressible in terms of an optimization problem. From this viewpoint, to answer the question "Why do we use learning classifier systems?" we should present a number of problems that classifier systems can solve but other reinforcement learning techniques cannot.

Whether we look at classifier systems as a reinforcement learning technique or as a more general framework it is interesting to note that in the literature only a few people have presented experimental results to support the use of learning classifier systems in place of other techniques.

Consider for example the area of reinforcement learning applications. It is general recognized that a sure advantage of learning classifier systems is their *generalization* capability. By means of don't care symbols (#) classifier systems can develop a compact representation of the concepts learned, whereas the complexity of traditional reinforcement learning technique grows exponentially in the problem size. At this point, the reader may argue that since generalization supports the use of learning classifier systems, then there are many papers that discuss the generalization capabilities of classifier systems. But, if we look at the works published in these twenty years, we note that most of the papers concerning generalization focus on the representational capabilities of classifier systems rather than on the degree of generalization that classifier systems can develop. Some authors (e.g., Riolo [61]) showed that classifier systems can develop interesting generalizations. But until 1996 no author provided extensive results to support the hypothesis that classifier systems could tackle reinforcement learning problems better than tabular techniques *because of their generalization capabilities.*

In 1996 Wilson [84] (see also [85]) presented a set of initial results showing that the solutions developed by his XCS classifier systems can be *significantly* more compact than that required by tabular techniques. Wilson's work was later extended by Kovacs [40] who hypothesized that Wilson's XCS develops the most general and accurate representation of the concept learned by the agent; Kovacs supported his *optimality* hypothesis experimentally.

I think that when looking at Wilson's results most people focused more on the novel classifier system architecture that Wilson proposed (XCS) rather than on the raw results that his XCS was able to achieve in terms of generalization capabilities. For instance, in this volume two papers, Booker [9] and Kovacs [41], discuss what is an adequate definition of classifiers fitness, while generalization is just partially discussed by Kovacs [41] and by Shaun and Barry [66].

On the other hand if we give a "crude" look at Wilson's results [85] we find that XCS was able to evolve a solution made of nearly 30 classifiers for a reinforcement learning problem which would require a Q-table equivalent to more or less 550 classifiers. Although these results are limited to a set of grid environments, they represent the first *direct* and *crude* comparison between classifier systems and tabular reinforcement learning. For this reason, I believe that these results, as well as those in [40], should be consider fundamental to classifier system research since they contribute (with respect to the reinforcement learning framework) to give an answer to a very important question, i.e., why do we use learning classifier systems?

For this reason, I think that the generalization capabilities of classifier systems should be extensively investigated in the next years so to provide a more solid basis and stronger motivations to the research in this area.

Reinforcement learning represents only *one* possible way of looking at classifier systems. As I previously observed, reinforcement learning make strong assumptions on the environment and, *most important*, on the goal of the learning process. In contrast, learning classifier systems do not make particular assumptions on the environment and on the agent's goal which is generally defined in terms of *adaptation to the environment*. But even of we change our viewpoint on classifier systems the question "Why do we use learning classifier systems?" is still important.

One possible answer to this question is provided by the many applications of classifier systems to the problem of modeling the emergence of complex behaviors in real systems. For instance, learning classifier systems can be used to model adaptive agents in artificial stock markets (e.g., Brian Arthur et al. [2], Lebaron et al. [53], Vriend [76–78], Marimon *et al.* [54]). On the other hand, it is not completely clear at the moment whether classifier systems are the *only* approach to tackle these types of applications. For instance a number of researchers are currently applying other techniques (e.g., reinforcement learning [57]) to model artificial stock markets.

Recently, Smith et al. [68] (see also [67, this volume]) presented an application of classifier systems which *undoubtedly* cannot be easily modeled with other learning paradigm and thus strongly supports classifier systems. In particular Smith et al. [68] developed a classifier systems that can *discover innovation* in terms of novel fighting aircraft maneuvering strategies. Their results, in my opinion, are fundamental to classifier system research because they prove that learning classifier systems can successfully tackle problems that are beyond the *plain* reinforcement learning framework; for this reason they also provide motivation to the use of classifier systems in very diverse and "*previously unthinkable*" application areas.

Since the early days of classifier systems, more than 400 papers on classifier systems have been published. Looking at the bibliography at the end of this book we note that there was a time in the mid 1990s when there was only a little research on classifier systems. I remember that in 1997 when I presented my first work on XCS [45] at ICGA97 [3], there were *only* three papers on learning classifier systems: mine, Holmes' [37], and Federman's [20]. Basically, we were not enough even to fill one session; at that time I thought that the area I was working in was actually quite small. In 1998, during GP98 [44], there were two sessions on learning classifier systems and more than ten papers. In 1999, at GECCO [4] there were four sessions on learning classifier systems and one workshop: more or less thirty papers were presented. But why are classifier systems experiencing this renaissance?

There are surely different interpretations of what happened in the last five years, of this reborn interest in learning classifier systems. My belief is that much of this is due to the effort of a number of people who traced novel research directions, suggesting new *motivations* so to provide answers to some important questions about classifier systems. These people renewed part of this area without giving up original Holland's principles and their unique flavor. And certainly, part of the current renaissance is also due to the people who followed and currently follow those directions.

8.2 Wolfgang Stolzmann

The field of learning classifier systems (LCS) is young. However, two approaches have been developed: the Michigan Approach and the Pitt Approach. I restrict myself to the Michigan approach because it is the classical approach. The foundations for LCS were laid by Holland [26] within the framework of his theoretical studies of genetic algorithms. The first LCS, called CS-1, was introduced by Holland and Reitman [36]. Since then many different types of LCS have been described in the literature. Thus, it is difficult to give a unified definition for an LCS. Therefore I would like to focus on the points that I consider most important.

A learning classifier system (LCS) is a machine learning system that learns a collection of simple production rules, called classifiers. Its knowledge is represented in a classifier list. An LCS can be regarded as a learning agent that acts in an arbitrary environment. In order to interact with the environment it has an input interface with detectors for sensory information from the environment and an output interface with effectors for motor actions. Each detector of the input interface contains information about one attribute of the environmental state and delivers values 0 and 1. Thus, all an LCS knows about the environmental state is represented in a bit-string called message. A message is the internal representation of an environmental state. Each component contains the information of one detector. Formally messages belong to the set {0,1}k (k is the number of detectors). A classifier is a condition/action-rule. If the condition-part of a classifier matches the current message, then the classifier can become active. The action-part of an active classifier interacts with the effectors and causes a motor action of the LCS. *Conditions* belong to the set $\{0,1,\#\}^k$. A condition is matched by any message that has 0's and 1's in exactly the same positions as the 0's and 1's in the condition. A '#' in a condition is called a "don't care"-symbol. It matches both a 0 and a 1. For example the condition 1## is matched by any message that starts with a 1; #00 is matched by 100 and 000 and the condition

010 is only matched by the message 010. A learning classifier system derives its name from its ability to learn to classify messages from the environment into general sets like $\{m — m$ starts with a $1\}$, $\{100 , 000\}$ and $\{010\}$.

The basic execution cycle of an LCS consists in an iteration of the following steps:

1. A message m is perceived by the input interface.
2. m is compared with the condition parts of all classifiers of the current classifier list. The matching classifiers form a match set.
3. One classifier out of the match set is selected by roulette-wheel selection [21] or another kind of competition.
4. The selected classifier becomes active. Its action part interacts with the output interface and causes a motor action of the LCS.
5. Reinforcement learning is applied (see below).
6. Rule discovery is applied to produce new classifiers while replacing other, low- strength classifiers (see below).

As mentioned above I focus on the points that I consider most important. I ignored the fact that classifiers can produce new messages so that the LCS has to deal with a message list, I ignored that a classifier can contain more than one condition and I ignored the possibility that more than one classifier can become active during one execution cycle. If there is more than one active classifier, then the LCS has to deal with inconsistent information for the output interface. There are two levels of learning in an LCS: A first level of learning called credit assignment, consists of reinforcement learning on the classifiers. A classifier is reinforced (i.e. its rule strength is modified) in dependence on environmental reward called payoff (Bucket Brigade Algorithm, Profit Sharing Plan, Q-Learning, ...). A second level that is usually independent of the first one consists of rule discovery in order to generate new classifiers. For that an LCS typically uses genetic algorithms. Up to now Stewart W. Wilson's XCS is the most important kind of LCS (cf. "State of XCS Classifier System Research", this volume). Tim Kovacs's LCS bibliography [42], contains 33 entries out of 479 that deal with XCS. XCS were applied in various environments like the (n+2^n)-Multiplexer (e.g. [83]) Woods-environments (e.g. [83, 48]), and Finite-State-Worlds (e.g. [5, 6]. The results show that an XCS is capable of forming complete reward-maps, i.e. $X \times A \Rightarrow P$ maps where X is the set of states, A is the set of actions and P is the set of expected payoffs. "It does not learn what input sensation will follow a given action. That is, it does not learn an $X \times A \Rightarrow Y$ map, where Y is the following sensation" (Wilson [83, p. 173]). Or in other words it does not learn an internal world model. Holland [32], Riolo [59], and Stolzmann [72, this volume] show how internal world models can be learned in LCS. Very simple internal world models are used in reinforcement learning especially in Dyna-Q (e.g. Sutton [73]; Sutton & Barto [74, p. 233]) Future work will have to show how LCS can be used to learn internal world models with a minimum number of classifiers and how these internal world models can be used in reinforcement learning. Furthermore, in my opinion future LCS research should include:

– real world applications,
– a definition of standard test environments like the (n+2n)-Multiplexer, Woods- environments and Finite-State-Worlds in order to facilitate the comparison of different approaches,
– applications in non-deterministic environments, especially in non-Markov environments (cf. Lanzi & Wilson [52, in press]),
– and LCS work should include comparisons to relevant reinforcement learning work.

8.3 Stewart W. Wilson

A classifier system is a learning system based on Darwinian principles. The system consists of two parts: a collection or "population" of condition-action rules called classifiers; and an algorithm for utilizing, evaluating, and improving the rules. Holland's great insight was to see that a learning system might be based on a Darwinian process applied to a rule base. To me, his idea is intriguing and inspiring because it represents a plausible computational picture of our mental world. Moreover, it is a proposal that seems rich enough to encompass, eventually, very difficult questions like multi-level autonomous adaptation.

Besides animals' more-or-less well-learned and reflexive condition-action responses, we are constantly making predictions, often relative to rewards, about the consequences of behavior, and choosing accordingly. But there is a "fog of uncertainty": we are not quite sure what will result in the present situation, not even sure *if* the present situation is what we think it is. But we go with what we have, try things, and attempt to register the outcomes. In short, everything we know is hypothetical, whether perceptual or cognitive. We hope to find better hypotheses, ones that more accurately define the antecedent condition and therefore predict better, and also cover the largest domain and reduce our mental effort. New hypotheses come largely from pieces of what works already—where else?—but the process is mostly unclear and mysterious. It is often invisible. A new hypothesis arrives suddenly; it is not consciously cranked out. There may be mental work, but then the key steps just happen.

To all this would seem to correspond, in Holland's scheme, the population of classifiers viewed as a set of competing hypotheses, each in some state of relative confirmation, subject to modification and replacement by a better hypothesis born suddenly from chance recombination and mutation of existing above-average material. Some classifiers generalize over sensory inputs, providing a rudimentary imitation of early-stage perception and an example of symbol grounding—the classifiers' messages being the symbols. Other classifiers respond to internal messages by posting further messages, affording computational completeness but more importantly, room to erect any so-called cognitive structure. Still other classifiers cause actions and thus complete the loop with the environment, whose response keeps the system viable and fuels the Darwinian calculation. The principles of variation and selection being ubiquitous elsewhere in life, it is natural to look seriously for them inside the head, and for this Holland has provided a splendid computational framework that merits every effort to understand and develop its implications.

The framework is many-faceted, and it's not surprising that besides successes, research over the past twenty years has identified certain tensions among the framework's elements. Two seem most important. One tension arises because the classifiers are in Darwinian competition, but a certain amount of cooperation among classifiers is necessary. For example, to reach external payoff the system may depend on a chain of classifiers acting sequentially through time: the chain members cooperate in the sense that each member depends on the activation of its predecessors and on the ability of its successors to push on to payoff. But the competition may disturb this. Selection of one member of the chain reduces the survival chances of the other members, so that the chain may break causing all members to lose out. This competitive/cooperative tension occurs in several guises, and the consequence is often instability leading to reduced performance.

A key part of the solution was the early proposal—made mainly for other reasons—to restrict the action of the genetic algorithm to the match sets, i.e., it was proposed to replace the panmictic (population-wide) GA with a GA operating just in the niches defined by each match set. This lessens the competition between sequential classifiers, since sequential classifiers occur in distinct match sets. The cost in discovery capability seems to be minor, since classifiers in separate match sets are basically solving different problems and crosses between them are not particularly fruitful. However, the niche GA introduces a new hazard: overgeneralization. A more general classifier will tend to show up in more match sets. With a niche GA, it will have more reproductive opportunities so that generality becomes an advantage in itself, and increasingly general—eventually overgeneral—classifiers will tend to multiply in the population.

The second major tension is between performance and generality. Consider an individual classifier. Its current prediction ("strength" in the older terminology) is an average of payoffs received in the situations (niches) in which it matches and its action is taken. Some situations may have large payoffs, others quite small ones: all are averaged in the prediction, which masks any underlying variance. However, if the variance is great, there can be a negative effect on system performance. In low-payoff situations one classifier's action may be chosen over a better action if the first classifier's prediction is misleadingly high due to averaging in of payoffs received in higher-payoff niches. The result will be degradation of performance in some degree. Moreover, the offending classifier, and the degradation, will persist since selection under the GA is based on the prediction. Thus the framework's potential for generalizing classifier conditions—essential for formation of percepts and concepts—is at odds with performance.

The first step toward solution was to ask: What would happen if fitness for the GA were based not on the prediction, but on some measure of the prediction's accuracy? Then the offending high-variance classifier above would not receive a high fitness, and would be eliminated from the system. On the other hand, a classifier with low prediction variance would survive and multiply; desirably, such a classifier would *not* cause the kind of performance error cited earlier. However, this solution is not complete since basing fitness on prediction accuracy does not in itself address the framework's need to find classifiers that are *both* general and accurate. Fitness based on accuracy would tend to favor very specific classifiers, since greater accuracy usually goes with greater specificity.

I would argue that these two tensions, competition vs. cooperation and performance vs. generality, were at the root of much of the field's earlier difficulties. Interestingly, the full solutions to each of them are complementary. I said that the first step in solving competition/cooperation was the niche GA. The second step turns out to be fitness based on accuracy, because that stops the niche GA's tendency toward overgeneralization. I said the first step in solving performance/generality was fitness based on accuracy. The second step turns out to be a niche GA, since its generalization pressure is just what is needed to push toward classifiers that are maximally general as well as accurate. Thus fitness based on accuracy and a niche GA *in combination* appear to overcome the principal problems of classifier systems' first twenty years, opening the way to a strong advance in the period just beginning.

There are many promising directions for classifier system research. Some are suggested at the end of my "State" paper in this volume. But an important one not listed there would take a new look at an old subject—the full Holland framework itself. In recent years there has been something of a retreat from research on the full system with its message list and other aspects. Progress has come from paring the system down and putting a simpler beast under the microscope to see more clearly the function of various mechanisms. The principal result, in my opinion, has been understanding the two tensions discussed above, and their solution. Now it is time to go back to the full framework and see what happens if in it fitness is indeed based on accuracy and the GA occurs in the niches. As far as I know, that has not been done. The tests occurred and were successful in the simpler systems. It seems quite possible, and worth the effort required, to try the changes in the full system, testing on a problem in which the message list's capacity to represent internal state is essential for high performance.

References

1. Manu Ahluwalia and Larry Bull. A Genetic Programming-based Classifier System. In Banzhaf et al. [4], pages 11–18.
2. W. Brian Arthur, John H. Holland, Blake LeBaron, Richard Palmer, and Paul Talyer. Asset Pricing Under Endogenous Expectations in an Artificial Stock Market. Technical report, Santa Fe Institute, 1996. This is the original version of LeBaron1999a.
3. Thomas Bäck, editor. *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97)*. Morgan Kaufmann: San Francisco CA, 1997.
4. Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*. Morgan Kaufmann: San Francisco, CA, 1999.

5. Alwyn Barry. Aliasing in XCS and the Consecutive State Problem: 1 – Effects. In Banzhaf et al. [4], pages 19–26.
6. Alwyn Barry. Aliasing in XCS and the Consecutive State Problem: 2 – Solutions. In Banzhaf et al. [4], pages 27–34.
7. John Tyler Bonner. *The Evolution of Complexity.* Princeton University Press, Princeton, New Jersey, 1988.
8. Lashon B. Booker. *Intelligent Behavior as an Adaptation to the Task Environment.* PhD thesis, The University of Michigan, 1982.
9. Lashon B. Booker. Do We Really Need to Estimate Rule Utilities in Classifier Systems? In Lanzi et al. [50], pages 125–142. (this volume).
10. Lashon B. Booker, David E. Goldberg, and John H. Holland. Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 40:235–282, 1989.
11. Leo W. Buss. *The Evolution of Individuality.* Princeton University Press, Princeton, New Jersey, 1987.
12. H. J. Chiel and R. D. Beer. The brain has a body: Adaptive behavior emerges from interactions of nervous system, body and environment. *Trends in Neurosciences*, 20:553–557, 1997.
13. Marco Colombetti and Marco Dorigo. Evolutionary Computation in Behavior Engineering. In *Evolutionary Computation: Theory and Applications*, chapter 2, pages 37–80. World Scientific Publishing Co.: Singapore, 1999. Also Tech. Report. TR/IRIDIA/1996-1, IRIDIA, Université Libre de Bruxelles.
14. Michael Sean Davis. *A Computational Model of Affect Theory: Simulations of Reducer/Augmenter and Learned Helplessness Phenomena.* PhD thesis, Department of Psychology, University of Michigan, 2000.
15. Marco Dorigo. Alecsys and the AutonoMouse: Learning to Control a Real Robot by Distributed Classifier Systems. *Machine Learning*, 19:209–240, 1995.
16. Marco Dorigo and Marco Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 2:321–370, 1994. ftp://iridia.ulb.ac.be/pub/dorigo/journals/IJ.05-AIJ94.ps.gz.
17. Marco Dorigo and Marco Colombetti. *Robot Shaping: An Experiment in Behavior Engineering.* MIT Press/Bradford Books, 1998.
18. E.B. Baum. Toward a model of intelligence as an economy of agents. *Machine Learning*, 35:155–185, 1999.
19. J. Doyne Farmer, N. H. Packard, and A. S. Perelson. The Immune System, Adaptation & Learning. *Physica D*, 22:187–204, 1986.
20. Francine Federman and Susan Fife Dorchak. Information Theory and NEXTPITCH: A Learning Classifier System. In Bäck [3], pages 442–449.
21. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, Mass., 1989.
22. David E. Goldberg. Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding. *Machine Learning*, 5:407–425, 1990. (Also TCGA tech report 88002, U. of Alabama).
23. H. Hendriks-Jansen. *Catching Ourselves in the Act.* MIT Press, Cambridge, MA, 1996.
24. S. Hofmeyr and S. Forrest. Immunity by design: An artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1289–1296, San Francisco, CA, 1999. Morgan-Kaufmann.
25. J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, 1986.
26. John H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975. Republished by the MIT press, 1992.

27. John H. Holland. Adaptation. In R. Rosen and F. M. Snell, editors, *Progress in theoretical biology*. New York: Plenum, 1976.
28. John H. Holland. Adaptive algorithms for discovering and using general patterns in growing knowledge bases. *International Journal of Policy Analysis and Information Systems*, 4(3):245–268, 1980.
29. John H. Holland. Escaping brittleness. In *Proceedings Second International Workshop on Machine Learning*, pages 92–95, 1983.
30. John H. Holland. A Mathematical Framework for Studying Learning in Classifier Systems. *Physica D*, 22:307–317, 1986.
31. John H. Holland. Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In Mitchell, Michalski, and Carbonell, editors, *Machine learning, an artificial intelligence approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann, 1986.
32. John H. Holland. Concerning the Emergence of Tag-Mediated Lookahead in Classifier Systems. *Special issue of Physica D (Vol. 42)*, 42:188–201, 1989.
33. John H. Holland. *Hidden Order: How Adaptation Builds Complexity*. Addison-Wesley, Reading, MA, 1995.
34. John H. Holland and Arthur W. Burks. Adaptive Computing System Capable of Learning and Discovery. Patent 4697242 United States 29 Sept., 1987.
35. John H. Holland, Keith J. Holyoak, Richard E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, Cambridge, 1986.
36. John H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems*. New York: Academic Press, 1978. Reprinted in: Evolutionary Computation. The Fossil Record. David B. Fogel (Ed.) IEEE Press, 1998. ISBN: 0-7803-3481-7.
37. John H. Holmes. Discovering Risk of Disease with a Learning Classifier System. In Bäck [3]. http://cceb.med.upenn.edu/holmes/icga97.ps.gz.
38. Keith J. Holyoak, K. Koh, and Richard E. Nisbett. A Theory of Conditioning: Inductive Learning within Rule-Based Default Hierarchies. *Psych. Review*, 96:315–340, 1990.
39. Kevin Kelly. *Out of Control*. Addison-Wesley, Reading, MA, 1994.
40. Tim Kovacs. Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996. Also tech. report CSR-96-17 and CSRP-96-17 ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1996/CSRP-96-17.ps.gz.
41. Tim Kovacs. Strength or Accuracy? Fitness calculation in learning classifier systems. In Lanzi et al. [50], pages 143–160. (this volume).
42. Tim Kovacs and Pier Luca Lanzi. A Learning Classifier Systems Bibliography. In Lanzi et al. [50], pages 323–350. (this volume).
43. John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. The MIT Press, Cambridge, MA, 1994.
44. John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors. *Genetic Programming 1998: Proceedings of the Third Annual Conference*. Morgan Kaufmann: San Francisco, CA, 1998.
45. Pier Luca Lanzi. A Study of the Generalization Capabilities of XCS. In Bäck [3], pages 418–425. http://ftp.elet.polimi.it/people/lanzi/icga97.ps.gz.

46. Pier Luca Lanzi. Adding Memory to XCS. In *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98)*. IEEE Press, 1998. http://ftp.elet.polimi.it/people/lanzi/icec98.ps.gz.
47. Pier Luca Lanzi. *Reinforcement Learning by Learning Classifier Systems*. PhD thesis, Politecnico di Milano, 1998.
48. Pier Luca Lanzi. An Analysis of Generalization in the XCS Classifier System. *Evolutionary Computation*, 7(2):125–149, 1999.
49. Pier Luca Lanzi and Rick L. Riolo. A Roadmap to the Last Decade of Learning Classifier System Research (from 1989 to 1999). In Lanzi et al. [50], pages 33–62. (this volume).
50. Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems: An Introduction to Contemporary Research*, volume 1813 of *LNAI*. Springer-Verlag, Berlin, 2000.
51. Pier Luca Lanzi and Stewart W. Wilson. Optimal classifier system performance in non-Markov environments. Technical Report 99.36, Dipartimento di Elettronica e Informazione - Politecnico di Milano, 1999. Also IlliGAL tech. report 99022, University of Illinois.
52. P.L. Lanzi and S. W. Wilson. Toward optimal classifier system performance in non-Markov environments. *Evolutionary Computation*, 2000. to appear.
53. Blake Lebaron, W. Brian Arthur, and R. Palmer. The Time Series Properties of an Artificial Stock Market. *Journal of Economic Dynamics and Control*, 1999.
54. Ramon Marimon, Ellen McGrattan, and Thomas J. Sargent. Money as a Medium of Exchange in an Economy with Artificially Intelligent Agents. *Journal of Economic Dynamics and Control*, 14:329–373, 1990. Also Tech. Report 89-004, Santa Fe Institute, 1989.
55. Richard E. Michod. *Darwinian Dynamics: Evolutionary Transitions in Fitness and Individuality*. Princeton University Press, Princeton, New Jersey, 1999.
56. Alan Newell and Herbert Simon. *Human Problem Solving*. Prentice Hall, Engelwood Cliffs, NJ.
57. E. Oliveira, J.M. Fonseca, and N. Jennings. Learning to be competitive in the Market. 1999. Proceedings of the AAAI Workshop on Negotiation, Orlando (FL).
58. J. K. Percus, O. Percus, and A. S. Perelson. Predicting the size of the antibody combining region from consideration of efficient self/non-self discrimination. *Proceedings of the National Academy of Science*, 90:1691–1695, 1993.
59. Rick L. Riolo. Lookahead Planning and Latent Learning in a Classifier System. pages 316–326. A Bradford Book. MIT Press, 1990.
60. Rick L. Riolo. Lookahead planning and latent learning in a classifier system. Ann Arbor, MI, 1991. In the Proceedings of the Simulation of Adaptive Behavior Conference, MIT Press, 1991.
61. Rick L. Riolo. Modeling Simple Human Category Learning with a Classifier System. pages 324–333. Morgan Kaufmann: San Francisco CA, July 1991.
62. George G. Robertson. Parallel Implementation of Genetic Algorithms in a Classifier System. In John J. Grefenstette, editor, *Proceedings of the 2nd International Conference on Genetic Algorithms (ICGA87)*, pages 140–147, Cambridge, MA, July 1987. Lawrence Erlbaum Associates. Also Tech. Report TR-159 RL87-5 Thinking Machines Corporation.
63. George G. Robertson and Rick L. Riolo. A Tale of Two Classifier Systems. *Machine Learning*, 3:139–159, 1988.
64. S. A Hofmeyr and S. Forrest. Architecture for an Artificial Immune System. Submitted to Evolutionary Computation. Available at `http://www.cs.unm.edu/ steveah/ecs.ps`, 1999.

65. Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journ. R & D*, 3:211–229, 1959. Reprinted in Feigenbaum, E., and Feldman, J. (eds.), *Computer and Thoughts*, pp. 71–105, New York: McGraw-Hill, 1963.
66. Shaun Saxon and Alwyn Barry. XCS and the Monk's Problems. In Lanzi et al. [50], pages 223–242. (this volume).
67. R. E. Smith, B. A. Dike, B. Ravichandran, A. El-Fallah, and R. K. Mehra. The Fighter Aircraft LCS: A Case of Different LCS Goals and Techniques. In Lanzi et al. [50], pages 285–302. (this volume).
68. Robert E. Smith, B. A. Dike, R. K. Mehra, B. Ravichandran, and A. El-Fallah. Classifier Systems in Combat: Two-sided Learning of Maneuvers for Advanced Fighter Aircraft. In *Computer Methods in Applied Mechanics and Engineering*. Elsevier, 1999.
69. Wolfgang Stolzmann. Learning Classifier Systems using the Cognitive Mechanism of Anticipatory Behavioral Control, detailed version. In *Proceedings of the First European Workshop on Cognitive Modelling*, pages 82–89. Berlin: TU, 1996. http://www.psychologie.uni-wuerzburg.de/stolzmann/.
70. Wolfgang Stolzmann. Two Applications of Anticipatory Classifier Systems (ACSs). In *Proceedings of the 2nd European Conference on Cognitive Science*, pages 68–73. Manchester, U.K., 1997. http://www.psychologie.uni-wuerzburg.de/stolzmann/.
71. Wolfgang Stolzmann. Anticipatory classifier systems. In *Proceedings of the Third Annual Genetic Programming Conference*, pages 658–664, San Francisco, CA, 1998. Morgan Kaufmann. http://www.psychologie.uni-wuerzburg.de/stolzmann/gp-98.ps.gz.
72. Wolfgang Stolzmann. An Introduction to Anticipatory Classifier Systems. In Lanzi et al. [50], pages 175–194. (this volume).
73. Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, Austin, TX, 1990. Morgan Kaufmann.
74. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning – An Introduction*. MIT Press, 1998.
75. Kirk Twardowski. Implementation of a Genetic Algorithm based Associative Classifier System (ACS). In *Proceedings International Conference on Tools for Artificial Intelligence*, 1990.
76. Nicolaas J. Vriend. On Two Types of GA-Learning. In S.H. Chen, editor, *Evolutionary Computation in Economics and Finance*. Springer, 1999. in press.
77. Nicolaas J. Vriend. The Difference Between Individual and Population Genetic Algorithms. In Banzhaf et al. [4], pages 812–812.
78. Nicolaas J. Vriend. An Illustration of the Essential Difference between Individual and Social Learning, and its Consequences for Computational Analyses. *Journal of Economic Dynamics and Control*, 24:1–19, 2000.
79. C.J.C.H. Watkins. Learning from delayed reward. PhD Thesis, Cambridge University, Cambridge, England, 1989.
80. Thomas H. Westerdale. An Approach to Credit Assignment in Classifier Systems. *Complexity*, 4(2), 1999.
81. Stewart W. Wilson. Adaptive "cortical" pattern recognition. pages 188–196. Lawrence Erlbaum Associates: Pittsburgh, PA, July 1985.
82. Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994. http://prediction-dynamics.com/.
83. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.

84. Stewart W. Wilson. Generalization in XCS. Unpublished contribution to the ICML '96 Workshop on Evolutionary Computing and Machine Learning. http://prediction-dynamics.com/, 1996.
85. Stewart W. Wilson. Generalization in the XCS classifier system. In Koza et al. [44], pages 665–674. http://prediction-dynamics.com/.
86. Stewart W. Wilson. Get Real! XCS with Continuous-Valued Inputs. In Lanzi et al. [50], pages 209–220. (this volume).
87. Stewart W. Wilson. State of XCS Classifier System Research. In Lanzi et al. [50], pages 63–82. (this volume).