Institute of Informatics
University of Fribourg (Switzerland)

EMuds: Adaptation in Text-Based Virtual Worlds

THESIS

submitted to the Faculty of Science of the University of Fribourg (Switzerland) in conformity with the requirements for the degree of
Doctor scientiarum informaticarum

submitted by

Antony ROBERT

of

Les Ponts-de-Martel, le Locle et la Chaux-du-Milieux (NE)

Thesis No. 1272
Printed at the University of Fribourg
February 2000

Accepted by the Faculty of Science of the University of Fribourg (Switzerland) following the proposal of:

- Dr. Michèle COURANT, University of Fribourg, Thesis Director;
- Prof. Marco TOMASSINI, University of Lausanne, Switzerland, Examinator;
- Dr. Dave CLIFF, HP Labs, Bristol, United Kingdom, Examinator;
- Prof. Rolf INGOLD, University of Fribourg, President of the Examination Committee.

Fribourg, the 17 December 1999

The Thesis Director:	The Dean:
Dr. Michèle COURANT	Prof. Béat HIRSBRUNNER

Acknowledgments

I wish to thank all who have helped me during the preparation of this PhD. More specifically, I would like to thank my parents who supported me when I really needed it. I would like to thank my examinators, Marco Tomassini and Dave Cliff, for reading and commenting on my work. I would then like to thank Michèle Courant and my research group for providing me with an environment in which I was able to lead my research very freely.

During the years I spent in Fribourg, a number of other people have also brought their more personal contribution to this work. I would like to mention Alban Soupper who has been a great help, first as a student, then as a collegue and always as a friend. Lukas Theiler and Simon Schubiger for always being ready to lend a hand or have some fun. Fabrice Chantemargue for sharing his energy in work. Patricio Lerena and Frederic Bapst for being symbols of “scientific curiosity”. Rolf Ingold for always giving me good counsel in many matters. And last but not least our secretary team, in chronological order: Monique, Sylvie, Marie-Claire and Yvette, who were always open for a good laugh and a healthy bit of criticism.

Résumé

Cette thèse introduit une approche méthodologique à la conception d'agents artificiels. En une tentative de résoudre le problème de l'enracinement symbolique en intelligence artificielle, une nouvelle perspective théorique sur la relation agent-environnement est explorée et un jeu d'expériences sont menées pour motiver cette méthodologie. Suite à l'idée selon laquelle l'enracinement d'un agent dans son environnement est dépendant, intuitivement, de l'adaptation de ses mécanismes de représentation internes à la structure de l'environnement, l'hypothèse discutée est que l'enracinement n'est possible que si un agent est capable d'appliquer un principe de compression d'information à sa représentation de l'environnement. Ceci mène à la suggestion que les algorithmes de contrôle d'agents soient classifiés selon les ressources nécessaires à ces algorithmes pour résoudre une série de problèmes types.

Pour soutenir cette hypothèse, une plateforme d'expérimentation appelée EMud a été concue et implémentée. Le modèle d'environnements supporté par cette plateforme à été élaboré pour que des expériences graduellement plus complexes puissent être effectuées, en particulier pour que le modèle puisse être utilisé dans le cadre de simulations de systèmes distribués dans lesquels des agents artificiels opèrent. Le type d'algorithme choisi pour tester les idées théoriques introduites dans la première partie de la thèse est le système classificateur XCS, dont une version à été implémentée. Au cours de la présentation formelle de ce système, une démonstration de l'équivalence d'une forme simplifiée de ce système avec l'algorithme de Q-Learning est faite. Une nouvelle illustration du rôle de la notion de précision dans ce système est aussi introduite.

Des expériences sur le problème du multiplexeur sont ensuite menées pour exhiber les propriétés de compression d'information or de généralisation du XCS. En utilisant une approche sur différentes représentations du multiplexeur, la capacité à généraliser du système XCS et son influence sur la résolution du problème est étudiée. Par la suite, une seconde expérience dans laquelle l'information nécessaire à la résolution du problème est répartie temporellement est menée. Les résultats dans cette situation montrent que le système XCS est incapable de généraliser efficacement. Il est conclu que par une classification des algorithmes basée sur l'efficacité avec laquelle ceux-ci peuvent représenter l'information nécessaire à la résolution de problèmes types, il devrait être possible de mieux comprendre comment concevoir des agents artificiels pour des problèmes spécifiques.

Abstract

This thesis introduces a methodological approach to artificial agent design. In an attempt to solve the symbol grounding problem of artificial intelligence, a new theoretical perspective on the agent-environment relation is explored and experimentation is led to motivate this methodology. From the original idea that in order to ground an agent in an environment, the agent's representation mechanisms must intuitively be close to the structure of the environment, the hypothesis that grounding is possible only when an agent is able to apply information compression to its internal representation of the environment is discussed. This leads to the suggestion that a classification of control algorithms be made, based on the resources an algorithm requires to efficiently solve various categories of problems.

In order to support this hypothesis, an experiment environment platform called the EMud is designed and implemented. The environment model is designed to support sufficiently general structures, so that gradually more complex environments can be designed within their specification framework, in particular to be able to use the model as a simulation of distributed systems where artificial agents operate. As a sample algorithm used to advocate the theoretical ideas, an XCS classifier system is implemented. A formal presentation of this system is made and the equivalence of Q-Learning with a simplified version of XCS is demonstrated. A reformulated illustration of the role of accuracy in the system is also presented.

Experimentation with multiplexer problems is then performed to exhibit the information compression or generalization properties of the XCS system. By using an approach where experiments on the representation of the multiplexer problem are performed, the generalization characteristics of the XCS system and their influence in solving the problem are studied. A further experiment in an EMud environment where temporal information is essential to solving the problem goes to show that the XCS system is unable to generalize in this situation, thus breaking down the efficiency of the system. It is concluded that by making a classification of algorithms based on the efficiency with which algorithms can represent various types of problem space in order to solve these problems, one should be able to better understand how artificial agents should be designed for specific problems.

Contents

Chapter 1

Introduction

Over the course of these few years spent in the field of artificial intelligence (AI), many topics have interested me, indeed, one can hardly focus AI research on algorithms only, without seeing the links and implications these might have in biology, psychology, philosophy of mind or cognitive science in general.

This particular status of artificial intelligence as an experimental platform for theories about intelligence has several positive and negative aspects. On the positive side, one can mention that there are very few, if any, other fields of research where natural science meets human science so vitally and where researchers have as much interest in understanding fellows from other disciplines. This gives a taste for variety and arouses curiosity for other domains in a sometimes shy scientific environment. Further, problems linked with mind, intelligence, or cognition also encourage a feeling of excitement about the importance of discoveries that could be made in the field. And with the slow maturation of complexity theories, system theories and cognitive theories, the (mistaken?) feeling that science is near a revolution of the type described by Thomas S. Kuhn [32] feeds the excitement. On the negative side, the very richness of the field makes it impossible to acquire a perfect overall understanding of the disciplines involved. Due to this, research has to be led with the constant attention not to be pulled into areas of personal incompetence or general ignorance, for example, as was elegantly put by Brooks [6], “the temptation to introduce artificial intelligence research with a definition of intelligence incurs the danger of deep philosophical regress without recovery”. I have attempted to avoid this trap, even though much of the theoretical work that I will introduce in the early chapters of this thesis has been heavily influenced by my philosophical readings.

To continue with an aspect that I have found difficult to manage over the course of this work, I have to mention the lack of structure in the overall domain. I think this is due to the youth of the field, since no effort to classify and formalize the main problems that must be solved has yet succeeded. Comparing this to mathematics, I have the impression that an endeavor comparable to that of Hilbert or the Bourbaki group in the early twentieth century still needs to be made (of course, mathematics have a history of two thousand years behind them and if computer science was able to structure itself in a comparable way after only sixty years of existence it would be an impressive result). I am hoping that through some of the ideas presented here this problem will be more easily addressed.

1.1 Research Context

In the parallelism and artificial intelligence (PAI) research group at the University of Fribourg, our interest is focussed on both the technical questions regarding parallel and distributed processing and the implementation of problem solving solutions in such computers or networks of computers through the use of artificial intelligence techniques.

This thesis is part of the AXE project, funded by the Swiss national fund for scientific research, in which aspects of massively parallel processing are studied. The overall aim of the AXE project is to provide a foundation and design methodology for evolutive parallel and distributed systems composed of artificial agents. In this global problematic, various aspects have been studied and are still under study, such as on the level of network operating systems (the WOS project [52, 45]), programming languages for distributed systems (the STL coordination language [31]), agent communication in distributed systems ([10] or the Saga project [51]) or agent population interactions with sexual and asexual reproduction mechanisms (Biomachines [35]). The direction in which this thesis is inscribed is in the autonomy and coordination part of the project, where the problem of interactions between agents in a complex environment is addressed.

The ultimate goal of this research is to understand how agents situated in an environment made of an open network of computers can compete, collaborate or simply maintain themselves in order to accomplish their tasks. In order to tackle this question, I have taken the path of understanding how artificial agents relate to their environment. It is in this perspective that I have developed a model of text-based virtual worlds, that have been implemented as EMuds. The EMud model is based on the idea that in order to understand the processes that occur in a computer network, an agent has to be able to adapt to unpredictable symbolic information generated by an environment in which it is embedded. EMuds provide such a framework by describing an environment as a virtual physical dimension in which agents exist and where virtual physical laws can affect the agents. The size of the environments that can be built in an EMud, along with the possibility of integrating multiple interacting agents in these environments make them an ideal platform for experimentation. In view of further extensions of research towards human-machine interfaces, EMuds also allow human agents to coexist and interact with the artificial agents in an EMud.

1.2 Symbol Grounding

When considering agents "living" in a dynamic environment, artificial intelligence is faced with problems of representation and perception of the environment. Various techniques have been developed to solve these problems, but for the time being, these methods are unable to fully integrate agents in a complex environment such as complex dynamic artificial environments or, a fortiori, the real world.

The problem encountered in artificial intelligence that is descriptive for this overall situation is called the symbol grounding problem. The symbol grounding problem is a problem that initially appears when one studies the relation between the real world and a system that uses some form of model or representation of reality to make decisions about a real world problem. Typically, when an expert system uses an internal representation of symptoms, illnesses and diagnoses to help a practician pronounce a patient's diagnosis or when a mobile robot uses an internal representation of the room it is in to decide how to move about, the representation and the world are two distinct entities. Since this situation appears to be a limiting factor for further development of artificial intelligence techniques, the question is how do these entities relate to each other? Can the symbols manipulated by the computer systems acquire an intrinsic meaning (related to the problem) in the system as opposed to an external meaning bestowed by a human observer of the system.

Now, if the symbol grounding problem is analyzed, there are several new problems in different areas that appear. First of all, why does one require symbols to have an intrinsic meaning? The answer is that if such symbols have no meaning for the system, new situations that were unplanned in the internal representation will be dealt with randomly by the system. A problem for computer science is then to find the best "random" choice and this seems to require some form of meaning to be included in the symbols. Secondly, what is meaning actually? Is there an internal and an external meaning, is there meaning without intent and then, what is intentionality? All questions leading to deep philosophical problems. Thirdly, can complete or more accurate representations of the world be used to avoid the problem of meaning? An approach that is often used, but leads to the problem of the observer that is very acute in physics or the problem of closed/open systems that has been studied extensively, with pessimistic results for this approach: there is no closed physical system. What then is the minimal size (minimal level of detail) of an accurate description for an open system? Fourthly, assuming there was a way of bestowing meaning to symbols in a representation, what meaning should the symbols acquire? Should some form of computer psychology be elaborated?

As is evident from these few remarks, any endeavor in which a computer system must make decisions about the real world is faced with far reaching questions that can either be addressed or ignored. In some situations, ignorance works well and in some others, not at all. The frontier is not clear and what is easy to solve for a human being is often intractable to the current state of computer science. We should be reassured that the reverse is also true, I for one have problems with solving 10^6 multiplications without mistake, but it seems natural for some reason. In this work, I have taken the option to acknowledge these difficulties and attempt to provide a (demystifying) scientific interpretation of meaning for symbol systems. For this purpose, I have sought to maintain a scientific approach to the problem studied by introducing ideas and using them as working hypothesis, although a philosophical perspective to these ideas could also be investigated. I have intentionally used generous reduction in the presentation of the philosophy underlying my personal position in the domain while hopefully giving a satisfying synthesis of the relevant ideas in the field that will situate the work and encourage further reading. On the other hand, since I intend to support the idea that reduction is an important part of understanding, the approach is coherent with the goal.

1.3 Document Presentation

For the purpose of the presentation, the document chapters are organized as a gradation from the more theoretical work to the practical implementations used to illustrate these theoretical ideas.

Theoretical Aspects

Chapter two introduces the philosophical framework in which the ideas of the third chapter are developed. It begins by explaining the general trends in theories of mind that philosophers have developped to this day and focusses on Functionalism, which is the theory of mind at least partially adopted by most AI researchers. The relevance of Functionalism to Computer Science is then introduced and the concept of abstract machine as artificial mind is presented. To conclude this chapter, the importance of the embodiment of a system in an environment is noted and enaction theory presented.

The goal pursued in chapter three is to hypothesize a model of understanding based on the process of information compression, and to use it as a classification scheme for problem solving algorithms. The chapter begins by a statement of the symbol grounding problem and an overview of previous attempts to solve this problem. It is here suggested that the current formulation of the symbol grounding problem based on an overgeneral sense of meaning makes it unsolvable in its present form. With a definition of meaning based on information compression, the problem becomes tractable. Grounding an agent is then a matter of degrees, where grounding is directly related to the efficiency of an agent's ability to represent environment structures in its own internal representation language. It is argued that the ability of an agent to accurately represent a complex environment with limited ressources through information compression is characteristic of the use of an appropriately expressive internal representation language for the problem faced by the agent. The mesure of algorithmic information is then suggested as an appropriate candidate for mesuring this degree of grounding of an agent. The key idea is to use algorithmic information not as a universal mesure of information content, but as a value calculated for each type of problem and each type of control algorithm so as to be able to compare algorithms by the complexity with which they represent problem environments.

With chapter four, the terminology of artificial intelligence used in the following chapters is clarified or introduced. The notions covered by agents, behaviors and autonomy are first introduced to give a foundation of agent theory. Concepts used when dealing with multiple agents sharing a same environment are then presented with multi-agent systems, interactions and emergence. The second part of the chapter gives an overview of the current practice in artificial intelligence. The subjet matter presented here is used as the basic theory upon which the next chapters are built.

EMud Environment Model

In chapter five, the type of experimentation platform used to support the hypothesis made in chapter three is described. Three types of problems that must be possible to adress with this platform are first described: simple symbolic problems such as multiplexer or Santa Fe Trail experiments, complex symbolic experiments such as the simulation of distributed computing problems such as load balancing or communication management and finally experiments where unpredictability plays a fundamental role, typically such as when human agents interact intensively with a system. Text-based virtual reality games are then introduced from a historical perspective and the possibilities of interaction in such systems are illustrated. Finally, it is shown how this type of gaming environments support the possibility of approaching the gradually more difficult artificial intelligence problems presented at the beginning of the chapter.

Chapter six describes the EMud environment implemented following the ideas introduced in the preceding chapter and presents its possible uses.

In the chapter, the structural components forming an EMud environment are described, followed by an explanation of how the dynamics of these environments are handled. Since the strength of this experimentation model is that it allows the definition of new environments for each experiment, a description of the environment specification procedure is made.

Practical Study and Conclusion

Chapter seven gives a formal description of the agent control system used to illustrate the methodological ideas suggested in chapter three. The algorithm studied is the XCS classifier system introduced by S. Wilson. The chapter proceeds by an introduction to these systems through their two main components: reinforcement learning and genetic algorithms. The architecture of the system is then described and an analysis of its functioning is made. In this analysis, an original result on the equivalence between the reinforcement learning algorithm of Q-Learning and a simplified version of XCS is demonstrated. The role of accuracy in the generalization mechanism of the system is then explained through a novel illustration. To conclude the chapter, a classical experiment is led on the system implemented in Fribourg, confirming the results previously observed for this experiment with other implementations of the XCS.

Chapter eight develops a new experimental approach used to highlight fundamental information compression characteristics of an algorithm, in order to support the thesis that the information compression / generalization properties of an algorithm determine its efficiency in dealing with an environment. The first set of experiments uses permutations on the representation of the multiplexer problem to show that the system obtains optimal results in solving the problem as long as the generalization mechanism is able to sucessfully store all the necessary information about the problem within the available ressources of the algorithm. So that even with a large search space, with generalization can produce a mapping of the problem in the limited representation space of the algorithm. As soon as this generalization cannot occur due to incompatible problem representation, the algorithm performance breaks down. It is then shown with a problem where temporal information is required for its resolution that the generalization mechanism is unable to extract the needed information for an optimal solution to the problem and that in this case, random permutations on the static representation of the problem have no noteworthy effect on the efficiency of the algorithm. The conclusion of this chapter is that as was suggested in chapter three, a classification of algorithms based on their compression of information characteristics be made. This can be done by using a similar line of thought as in the calculation of algorithmic information, by determining the total amount of ressources needed by various algorithms to express an optimal solution to various chosen benchmark experiments.

The conclusion in chapter nine reviews the question that were adressed in the thesis, starting by practical aspects of the design of an EMud experiment environment and its implementation. A critical review of the missing components of the implementation follows. An overview of the theoretical problems that were adressed in the work and their tentative answer is then presented, while proposing a future course of study for the actual implementation of the ideas introduced over the course of the first eight chapters.

Chapter 2

From Philosophy to Computers

2.1 Philosophy of Mind

Philosophy of mind is concerned with all the deep questions regarding the human mind, from freedom of will, personal identity or the existence of the soul to the nature of behavior, emotions or consciousness. With the advent of computing machines, the philosophy of mind has extended to the question of what minds are in general, and of whether human-made artifacts can possess minds, an interrogation that has renewed the interest in the mind-body problem. The mind-body problem, which is the question of how the physical world relates to the mental (experienced) world, now even appears to be the central question relating all problems concerning the mind. Although the roots of this discipline can be found with the Greek philosophers such as Plato and Aristotle, I start my introduction with the modern day answers that have been given to the mind-body problem.

2.1.1 Dualism, Mentalism and Materialism

Descartes was the first modern day philosopher to answer this problem with his theory of dualism [42, 14], where he supports the idea that the physical world, including plants, animals and the human body, is governed purely by mechanical/physical laws, whereas human beings "contain" some non-material mental substance. Although these two elements of nature are distinct and disjoint, they interact through the human body: sensations create thoughts, and thoughts drive actions such as bodily motion.

This view of the world has encountered many different problems over the years, essentially over the nature of the interaction between such different objects as matter and thought stuff and has spurred two different trends in philosophy that avoid this question. On one hand the mentalist approach that holds that all of nature is purely mental experience or thought stuff and on the other hand, the materialist (physicalist) approach, more typical of occidental thinking, that holds that all of nature is made of physical matter and that somehow, the mind emerges as a feature of the complex interactions in the physical substance that forms the human body.

2.1.2 To Functionalism

The objective of providing a theory of the mind has lead philosophers through various stages in materialism, starting with the behaviorist theory [57] that is grounded on an epistemological position: that the only way to observe mind properties is through the behaviors exhibited by the subjects that experience them. The behaviorist theory has thereby also founded the psychology paradigm of behaviorism, but has now lost some of its attractiveness since it fails to explain the mind in situations where there is the need of more powerful observation tools than the strict study of behavioral patterns. The typical thought experiments that have been used to contradict behaviorism being those of super-actors: how is one to explain the mental state of "pain" when someone could act as if he felt no pain when he is indeed in pain or how can one distinguish real pain from the pain that an actor simulates perfectly.

The identity theory [1] that followed reached into the human brain to identify physical brain events with mental states. In this theory, mental states or thoughts were explained in terms of neurophysiological states in the brain and the theory introduced the notion that the meaning of words for example must be stated in terms of brain configurations. Although this theory remains the basis of today's functionalism, it encountered problems in that it presupposes that mental items or types of items can be characterized by neurophysiological states once for all subjects. This precludes the possibility that different creatures from humans have minds and even that humans beings are able to function with different brain processes. To solve this problem, functionalism [47] introduces the idea that it is not a specific physical component that characterizes a mental state, but the causal role a physical component has in the organism that characterizes mental item types such as "pain" or "pleasure". This characterization differentiates token-token identity, one individual being's current pain state identified with his current neurophysiological state, to type-type identity, identifying pain with the more abstract functional role that some component may play in an organism.

The functionalist theory of mind is thus focussed on three levels of description [38]:

- the physiological description of an entity at a given time, corresponding to a physical state token;

- the functional description of this state, relative to the entity's organism;
- the eventual mental description of such a state if it instantiates a mental item type.

2.1.3 Computers

With functionalism, the philosophy of mind introduces a theory that is clearly meant to accommodate other types of minds than the human mind, it generally demystifies the idea of minds and states that Mind is simply brain stuff in (inter)action. The levels of description of the theory provide a framework that leads from the physical world to the mental world in three steps, independently of the realization of an entity in the physical world. On the first level, the emphasis is given to a physical description of an entity, that can potentially have any form and is independent of external interpretation. On the second level, the functional description of the entity's current state that depends only on this entity's internal organization. On the third level, the qualitative description of the current state instantiated by the entity as viewed by an observer in reference to himself. Of these three levels, the first two are independent of the observer and the third implies that an observer might not be able to ascribe a mental description to an entity if he is not capable of similar mental states (or at least understanding them), even though the entity might have such a mental description.

The obvious artificial candidate to validate this theory is the digital computer. If it is possible to show that a computer with an adequate design (hardware) and internal state (software) that can be described physically and functionally can also be shown to possess some form of mental description, then minds are purely emergent properties of some kinds of material constructs.

In practice, computer science has for first objective the resolution of problems with computers. The question of artificial minds is only raised by the fact that some problems seem to require elaborate cognitive faculties similar to those exhibited by humans if they are to be solved. In this context, if computer science needs to reproduce certain mental processes in order to solve some classes of problems that humans can solve, either, how brains "do" minds must be understood and imitated on computers or another method that is functionally equivalent but works on computers must be found. The current trend is to attempt the synthesis of some form of intelligence in the form of computational models that include processes functionally equivalent to those occurring in the brain. I will show later that computers are instances of physical symbol systems, but the essential comment that must be made here is that the underlying assumption of the field of artificial intelligence is the physical symbol system hypothesis

[44] (PSSH), which states that a physical symbol system has the necessary and sufficient means for general intelligent action. That is, the PSSH (or an equivalent) is generally accepted as a working hypothesis in the field and not as a problem to be answered. Of course, if by practicing AI, strong evidence for the correctness of this hypothesis can be exhibited, this will be considered as encouraging results for the discipline.

2.2 Symbol Systems and Artificial Agents

Some additional assumptions are now deemed necessary for building intelligent machines and I will introduce them shortly with a few key points at the end of this section, but before that, it is necessary to speak of the abstract machines we consider when speaking about computers in general.

2.2.1 Abstract Machines

The notion of Physical Symbol System dates back to the origins of computing, with the work of A. Turing or J. Von Neumann. It is linked with the introduction of Turing machines that are used to define the concept of algorithm by simple operations on symbols. The first formalization of such systems by Newell and Simon [44] was intended to provide a general category of systems that are capable of intelligent behavior.

Turing Machines

To formally define algorithms, Turing reduced operations of the kind that are intuitively used as steps in an algorithm to simple mechanical operations in an idealized machine. In his theory, the machines he considers consist in a read / write head scanning the surface of a tape divided into cells (see figure 2.1). The tape cells can hold symbols from a predefined finite alphabet and extends to infinity at least in one direction. The head can move along the tape in both directions (to the “right” or to the “left”), reads symbols in the tape cells and can inscribe new symbols in these cells. At any given time, the machine is in a state chosen from a finite set of states and depending on this state and the symbol read on the tape, it can decide to write a new symbol in the current cell and eventually move to the left or the right on the tape.

Clearly, any Turing machine with its set of states, symbol alphabet and current internal state instantiates an algorithm. The Church-Turing thesis, on the other hand, states that any algorithm a human being would consider can be represented as a Turing machine, thus giving the equivalence between Turing machines and our intuitive understanding of algorithms. To this day, this thesis holds and is generally accepted by the scientific community as the right definition of an algorithm.

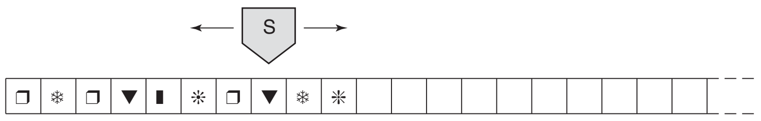

Figure 2.1: A Turing machine.

Symbol Systems

Since the definition of Turing machines, it has been shown that many formal systems are equivalent to Turing machines and thus define the same notion of algorithm. Moreover, the modern day computers built following the model of von Neumann are instanciations of the formal Turing machines, but where some restrictions have been introduced, such as a finite length tape. The theory applying to Turing machines can generally be extended to other equivalent formal or physically realized systems and Newell and Simon have called all these systems *symbol systems*. They then postulate that physical symbol systems are a category of systems that are capable of intelligent behavior in their "Physical Symbol System Hypothesis" [44]. Their work then goes on to try and convince us that this hypothesis is true and even that human beings are physical symbol systems. Clarifying the work of Newell and Simon, Harnad [22] gives the following definition for a symbol system.

A symbol system is:

- a set of arbitrary symbols called *physical tokens* manipulated on the basis of *explicit rules*, as well as strings of tokens built on the atomic tokens

- where *manipulation* is based solely on the "shape" of tokens (i.e. is syntactic)

- manipulation consists in *combining via rules* the atomic or composite symbol tokens

- the syntax can be *systematically* assigned a meaning, i.e. it is *semantically interpretable*

This description gives an intuitive view of what symbol systems can do, although it must be said that there is nothing more here than what was already in Turing machines. In fact, the foundational work in artificial intelligence led by Newell and Simon on symbol systems has even earned these systems the status of a "symbolic model of mind" [18] for psychology, where symbol strings capture mental phenomena such as thoughts and beliefs. But it suffices to say that such systems make explicit what computers can do and if we are to work with computers, a symbol system provides a good abstraction of these machines.

2.2.2 The System and its Environment

The symbol system definition typically omits a vital component in the study of artificial intelligence. It has long been assumed that a system can be considered as a closed system, but this can never hold in practice and I believe it is important to be aware that the whole underlying assumptions made about the systems under study are based on a conceptual shift of perspective that originates in the nineteen-thirties with Ludwig Von Bertalanffy's General Systems Theory (GST) [65] and the Cybernetics movement begun at the Macy conferences from 1946 to 1956 [66]. The two fundamental assertions that should be retained from this systemic enterprise are adequately illustrated by the two following quotes:

> When we try to pick-up anything by itself, we find it is attached to everything in the universe. *John Muir*
>
> The science of observed systems cannot be divorced from the science of observing systems. *Heinz Von Foerster*

That is, contrary to the common methodology often used in the physical sciences: 1) all systems must be considered open, there is no such thing as a closed system, 2) since all systems are open, an observer is always part of the system he is observing and plays a role that affects the objects under his observation. This role of the environment in the development of an entity is particularly important to artificial intelligence.

It should be noted that although the popularity of GST and Cybernetics has steeply fallen from the original enthusiasm they had generated (fallen in disrepute even). These theories are the seeds that allowed models such as Maturana and Varela's theory of *Autopoiesis and Cognition* [39] or the general field of Cognitive Science to grow [17], and these now play a dominant role in many areas of modern research, necessarily including artificial intelligence, but also philosophy of mind, psychology, economics or management.

2.2.3 Enaction theory

Maturana and Varela present their biological theory of cognition in [39]. To introduce their ideas, many concepts and terms are used, but essentially they emphasize the fact that living organisms are structures able to sustain their unity in an environment (via *autopoïesis* or self production) and situated in a physical space. These organisms are instances of a category of structures defined by their *organization*, with organization used in the sense of internal relations and dynamics. It is by the maintenance of this organization that an organism preserves its *autonomy* with respect to the rest of the environment.

When an organism evolves in an environment, the actual changes that it experiences are controlled by its structure as a result of the coupling between the environment and the organism's structure through senses. In relating an entity to an environment or other entities, *domains* of interactions can be defined such as the domain of relations (set of relations in which an entity can be *observed*). In this context, Maturana and Varela state that "Living systems are cognitive systems, and living as a process is a process of cognition". The necessary conditions for a system to be cognitive are thus that the system exists within a physical space and is structurally coupled to it (*embodiment*) and the cognition itself is the behavior of an entity engaging in interaction with the environment. Much prominence is given to the fact that the environment does not transform a living organism, but that the structure of the organism controls the changes it can undergo when subject to environmental perturbations. The notion of representation is also refuted by the theory, as an organism does not engage in the construction of a model of the world, but is presented with perturbations of its structure through its sensors, an experience that it can then actively rebuild as needed. Note however, that I will continue using the term representation even for such situations.

The influence this theory has over the field of computer science is in the methodology as I will show in the chapter on agent systems. It is with the concept of *experiential enaction* [63] that the relation between the mind and the body is explained. In the enactive view, the only relation between the mind and the body is to be found in the nature of the experience that is acquired by engaging in worldly activity with this body and mind. Therefore, to build artificial minds one requires bodies that exist and experience the world. And so, a new requisite for artificial intelligence can be postulated by stating that intelligence is contingent on being embodied in the world.

While this assumption appears essential to me, the restrictive view often taken that the world in which an agent must be embodied can only be the real world seems unnecessary. I like to assume that any type of world can be host to problems that may be solved by entities existing within them. That environments require some high level of complexity and/or unpredictability for cognitive level processes to evolve in the entities that inhabit them is not excluded, but even in simpler spaces, there are many interesting questions that remain to be answered.

Chapter 3

Symbolic or Functional Grounding

As I presented in my introduction to this thesis, the symbol grounding problem is the problem encountered when trying to relate the symbols in a physical symbol system to objects in the system's environment. S. Harnad [22] presents this as the inability of current agents to acquire an intrinsic meaning for the features they encounter in their environment, opposing this intrinsic meaning to the extrinsic meaning a human assigns to the components of the system he is observing or building. His formulation of the symbol grounding problem, focussed on meaning, is in fact inspired from the attack, focussed on intentionality, led by Searle [53] against computer intelligence and more precisely understanding in his well known Chinese room argument. With this argument, Searle intends to demonstrate that digital computers lack the causal power to do anything but produce a formally predefined next stage in a programmed calculation. He then concludes by saying that: *formal symbol manipulations have no intentionality, they are meaningless; they aren't even symbol manipulations, since the symbols don't symbolize anything. In the linguistic jargon, they have only syntax but no semantics.* This part of the doubts of Searle regarding artificial intelligence appear to be relevant since the production of meaning in symbol systems seems out of reach for the current approaches to AI. I would suggest that actually, the determinism that Searle reproaches to computers is not a true limitation. The fact that it appears as such is related to the idea that programs can always be simply interpreted by an observer. When this is not the case, formally predefined next stages in a calculation can induce very interesting (and unpredictable) behaviors in a program. I will now suggest a view of meaning independent of the (anthropologically flavored) intentionality of Searle which could allow further investigation into artificial intelligence.

3.1 Solving Grounding

The search for a solution to the symbol grounding problem has followed one main direction, namely that grounded systems need a tight relation with their environment and that this relation must be established by the sensory-motor equipment of an agent. A lot of research has thus gone into the study of perception in situated symbol systems, both from a top-down and from a bottom-up perspective, as described by Ziemke in [72].

To this end, the cognitivist approach embodied by classical AI attempts to devise high level input systems responsible for mapping perception into internal representations in a way that is coherent with this representation. Its recent methodology developed in the context of the symbol grounding problem generally consists in the extraction of perceptual features or invariants that can be related to atomic components of the internal representation. These components are then used to form the first “grounded” layer in the agent’s representation of the world and from which the whole world model is constructed. In the attempt to ground representations of these components to categories of objects in the world, the cognitivist approach also often distinguishes between a symbolic and subsymbolic level of representation. Perceptions are then analyzed on a subsymbolic level, typically implemented with a connectionist method, to bring forth perceptual categorization at the symbolic level. Ziemke [72] classifies the main proposals for symbol grounding in this framework into Harnad’s proposal of iconic representations [22] and Regier’s perceptually grounded semantics [48] to which I would add the work of L. Barsalou on perceptual symbol systems [3].

The new AI (bottom-up) approach to symbol grounding, on the other hand, is not only focussed on the agent’s perception, but on its active qualities. Following Maturana and Varela’s autopoietic theory [39], agents are characterized by their embodied situatedeness, emphasizing the fact that an agent experiences the world through its active participation in it, called experiential enaction. The enactive cognitive science that follows from this theory [63] gives the theoretical framework for bottom-up grounding. In fact, since bottom-up methodology boasts the absence of representation [7] and the internal structure of the agent built here is grown from perception, these agents should be immediately grounded in their environments. In practice, new AI is behavior based in that agents are usually equipped with some elementary behaviors that are designed by the scientist and then these behaviors are allowed to interact and combine to form more complex behavioral patterns. It is these behaviors and their combination mechanisms that need now to be grounded and this proves to be of comparable difficulty to the initial symbol / representation grounding. One approach that has been used to this aim is the evolution of all behaviors from perceptions by using a dynamically generated (evolved) connectionist architecture to control an agent [33, 11]. I believe this method can generate grounded agents because the internal functionality of such systems is evolved in conjunction with the problem domain and this is the fundamental point that I want to make in the following sections, unfortunately, the approach suffers from the size of the space of solutions that must be explored by the evolution mechanism.

3.2 Locating Meaning

The three components partaking in the grounding problem are the environment, the agent and the sensory-motor equipment linking an agent to its environment. In the various AI techniques described previously, the environment has played the role of problem domain with the assumption that in this domain, objects or object categories have an intrinsic meaning. Since the human ability of reasoning appears to be based on the acquisition of this meaning in the environment, attempts to transfer this meaning in a similar way to symbol systems have recently been focussed on the agent-environment interface. The various designs of agent control mechanisms that are supposed to accommodate this meaning have of course remained the main active area of the field of artificial intelligence, but generally with a distinct emphasis, boosted by new AI and the statement of the symbol grounding problem, on the action / perception properties of agents, since it is their only access to the world.

When considering a human agent, the assumption made is that there exists some form of meaning that he can acquire to relate objects and events in the environment either independently or in relation to himself. This meaning, whether intrinsically preexistent in the world or co-constructed by his embodied situatedeness in the world then serves him to reason about the environment and solve problems within it. If we are to build artificial agents that can acquire a similar form of meaning in order to make decisions, I believe the fundamental sense in which meaning must be understood is, following a functionalist approach, as the causal role of elements in their environment. In this sense, meaning becomes an intrinsic property of these elements in the environment, or globally, in the agent-environment system.

In the AI techniques described previously, it appears that attempts to ground agents in their environment are focussed on the means of transferring external meaning into an agent's internal mechanisms. The emphasis is put on the internal mechanism an agent is fitted with to accommodate this meaning and, more recently, on the transduction apparatus interfacing an agent with his environment. I believe this approach is too intent on "hooking" environment elements to agent internal symbols. Even in the case of enaction approaches, eventually excluding the previously cited

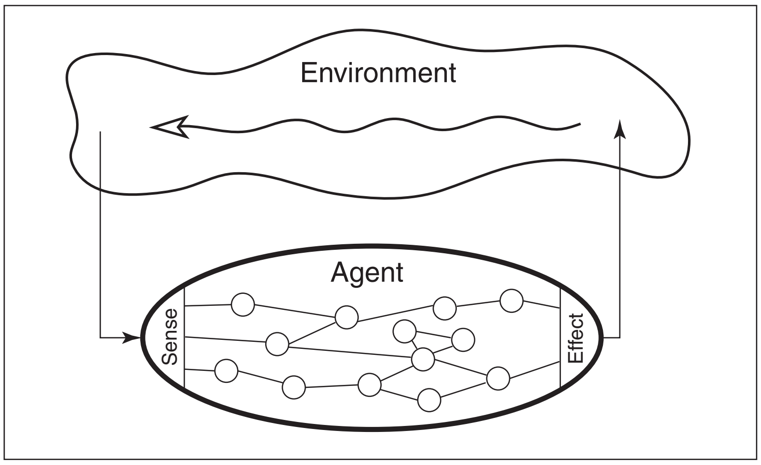

Figure 3.1: The functional structure in an agent.

evolutionary connectionist techniques, the intent lies in connecting sensory information to internal processes, while these processes are partially predefined and their dynamics not well understood. These methods give the impression that what is attempted is a token-token identification between world tokens and agent tokens. But within the functionalist theory of mind, this form of identification has been shown not to hold the essence of mental characterizations. What should be deemed most important in an agent is the functional role of its internal components and since I have considered meaning in terms of the causal role of elements in the environment, meaning in this sense also preexists in an agent, expressed in terms of the functional role of its internal components. From this perspective, agents own an intrinsic meaning for the symbols of their internal representation, which is given as the functional role those symbols play for their own internal dynamics. Thus a situated agent is always grounded in its environment through its interaction with it, but the current techniques of implementing agents fail to provide these agents with the proper internal structure for a coherent functional integration of these agents in their environment. In other words, the intrinsic meaning an agent possesses cannot be correctly related with that of the environment and fails to reach the higher levels of integration that are visible in living creatures.

If one looks at the current architecture of an agent 3.1, one usually sees some sort of functional structure linking components together and conveying information from sensors to effectors. Without adaptation, this structure is fixed and the dynamics of the agent will remain identical, whatever happens in the environment. With learning, the component links are usually malleable, being strengthened or weakened under the effect of external events, but the potential dynamics remain limited to what the implementer has imposed as his semantics for learning. Using evolutionary techniques, the components can then also be rearranged so as to adapt to the environment what functionality was given to the agent. The general trend of these methods is clearly to allow an increasingly functional grounding of an agent in its environment, but the missing element of the study is an analysis of the types of functionalities that can be provided to an agent in its components. These components are usually chosen because they express a function that is deemed important for the problem the agent will be faced with, but the question of what is important for the agent itself is rarely, if ever, investigated. In an somewhat excessive manner of speaking, where are the qualia[1] components of these agents? And without delving into abstract considerations such as those presented by Nagel in his article "What is it like to be a bat?" [43], the question of what it is like to be a digital agent surely requires of us a better understanding, without conceit for science, of the properties of the languages used to implement agent architectures.

3.3 Languages, Information and Dynamics

The definition of algorithms given by Turing has brought the abstract conception of machine or computer to science and served to build the current computers. With the study of universal Turing machines, the understanding of what machines can and cannot compute has also been understood and used, but the relation between program and data has always been problematic and while one universal Turing machine can theoretically compute what any other can, the information (under the form of symbol strings) that has to be provided to different universal machines for solving the same problem is also different. In fact, while it is at the heart of all of today's concerns, the simple notion of information has no unique definition in computer science.

At the center of this problem lies the difficult program versus data relation that has recently been studied in the field of information theory with various definitions of the information content of an object, an overview of which can be found in [50]. The most appropriate definition, that I will use here, is called the algorithmic information content or Kolmogorov-Chaitin complexity of an object. But it is worthy to note that information problems are not restricted to computer science, but were originally adressed by Shannon in the context of telecommunication [54] and also have implications that reach into biology [2] or physics [5, 36].

[1]Qualias have been introduced by some philosophers to describe the perceptive quality of sensations that cannot be communicated to people lacking the appropriate senses, examples of qualias are the *redness* of red objects or the *softness* of things that can be touched. The problem of qualia appears when for example one attempts to describe red to a blind person: the person can never completely understand this property of objects. A lot of discussion has centered on the actual existence of qualias, see for example [26, 13]

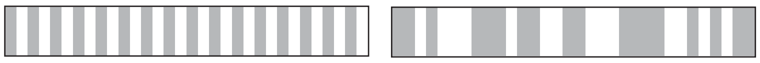

Figure 3.2: Two patterns to be described.

3.3.1 Algorithmic Information

The definition of algorithmic information takes into account both structures and dynamics by using descriptions and description languages. When we need to measure the complexity of an object, a description of the object is considered, intuitively, when this description can be short, the complexity of the object is to be considered to be small and when this description is necessarily long, the object is complex. The typical example of this is when one is presented with two pictures as in 3.2. The first consists in an alternating distribution of gray and white bars and the second in a sequence gray and white bars whose positions must each be specified independently. The first figure can be easily described by giving the description of a gray and white bar and specifying its repetition, for the other illustration, each start and end position must be included in the description.

Algorithmic information theory [8, 9] formalizes this understanding by measuring the information content of binary string descriptions (any string description can be written as a binary string) and applying the language of algorithms (Turing machines) to these strings. The algorithmic information of a description is the shortest description of an object that can be made using algorithms and strings. Formally, let s be a string that describes an object. There exists a family of Turing machines $\{T_i\}$ that when applied to strings $\{u_i\}$ will generate the string s, i.e. $T_i(u_i) = s$ for each i. Since the machines T_i can be written as strings t_i. One can form the strings $d_i = t_i \oplus u_i$ by concatenation and these strings are descriptions of the original object that are equivalent to s since they can be used to reconstruct s. The algorithmic information of the object is the length of the shortest string in the set $\{d_i\}$, written $K(s)$. Note that in the concatenation operation, a constant length delimiter must be used to distinguish algorithm part and string part. For a complete easy formal presentation of this theory, see [56].

In this definition, one uses Turing machines (programs) operating on input strings (data) to measure the information content of the objects under consideration. In practice, other types of description languages may be used to define equivalent information measures and it has been shown that for any given description language Δ, there exists a fixed constant k_{Δ} such that for any string s, $K(s) \leq K_{\Delta}(s) + k_{\Delta}$. This indicates that whatever the description language, the information of objects is of the same order as their algorithmic information and is interpreted as an optimality property for this information measure.

3.3.2 Expressivity

The interesting point in this theory is the optimality interpretation given to the previous result. I think that this is characteristic of the difference that exists between theory and practice, for when two measures differ by "at most" a constant term, in theory the measures are equivalent, but in practice such a constant term may make a big difference. Consider the following situation: who hasn't smiled when he heard the anecdote of Von Neumann almost firing his assistants for wasting time in developing a higher level language than assembly on the IAS computer. The importance of expressivity in programming languages today is evident, and most courses on programming emphasize the choice of the right language for the right problem. The fact that any program can be written in assembly language is theoretically interesting, but in practice useless when one must implement a specific "complex" program on a computer.

Admitting this intuition implies that the constant provided by the algorithmic information theorem is actually large enough in some cases that in practice the choice of language does have a role when implementing distinct problem solving algorithms. And, returning to the main discussion, in particular when an artificial agent architecture is implemented. Additionally, this theory gives a hint at what could be the functional features that one is looking for when an agent must be designed for a specific environment.

3.4 Understanding as Information Compression

With the intrinsic meaning in an agent defined as the functional roles of its various components (ie. the potential for self en environment transformation they define), the environment elements that will most firmly be grounded in an agent should be those that are closest to the agent's internal representation dynamics. The measure of this proximity can easily be seen as the algorithmic information of the description of the external events or elements in the agent's internal representational language. In this language, the objects in the environment of an agent and the dynamics in the environment can be described by using symbols and their functional role. For each environment object or property, there exists a large number of different equivalent descriptions in the agent language, each owning a distinct meaning and the search for the most efficient description in terms of algorithmic information can be seen as the process of understanding for an agent.

If this hypothesis is accepted, a new approach to implementing agents is to search for a representational language that gives the shortest possible descriptions for the agent's environment perceptions. If this approach is pushed further, one might even attempt to provide agents with mechanisms for the compression of perceived information, so as to allow the agent to form sub-architectures that are specialized languages that deal with certain perceptions and act as information compression components in the agent that could parallel the process of understanding. Furthermore, a study of the dynamics that algorithms are naturally adapted to express can be made with experimentation on test cases. Of course, since the description languages instantiated by various agent algorithms have distinct properties and none can be universally efficient, the goal must be to find algorithms that can be applied to specific problem domains encountered in practice. This last sentence can be summed up by the statement that generalization is impossible in general, but that the problems encountered in practice can be generalized over.

3.4.1 Classification

A first step that I consider essential would be to make a classification of the known algorithms with respect to their informational properties. By classifying the known algorithms, an easier introduction to the very wide number of different methods used in the field could be accessed by young researchers. By providing categories of algorithms, the problem domains addressed by artificial intelligence would also be clarified. To that aim, I propose a benchmarking methodology, based on a collection of test case experiments used to evaluate the characteristic features of algorithms. Such benchmarking experiments can be devised based on the tests already used by many of the schools of AI, such as “Santa Fe Trail” experiments used in artificial life or obstacle avoidance problems used in mobile robotics. I give an example of this type of experimentation in the application part of this work by studying the use of classifier systems both in the representation perspective and the dynamic perspective provided by EMud environments.

3.4.2 Theoretical Approaches

In further refinements, the theoretical approach of formalizing the expressivity properties of algorithms seen as Turing machines could be pursued. Here, difficulties arise mainly from the lack of mathematical tools suited to practical expressivity notions and the criteria for determining what problem domains are relevant to artificial intelligence problems when working on abstract machines. A first course of action that I have used is to apply non-standard analysis techniques to Turing machines. By using non-standard analysis, one can make an additional distinction in element sets, dividing them into the class of standard elements and non-standard elements. Intuitively, this distinction separates elements that can be constructed in practice from elements that have a theoretical existence, but cannot be produced. When the set of Turing machines is considered, the standard Turing machines are those that can be realized as computers and the non-standard ones are those that can serve in proofs for example, but could never be constructed in practice. Using such a distinction allows some basic problems to be understood, such as the fact that the famous halting problem does not exist for standard Turing machines, but some non-trivial results still need to be demonstrated in this theory.

Chapter 4

Artificial Intelligence and Agent Systems

A lot of work in the field of artificial intelligence has recently been devoted to multi-agent systems. Since the AXE project of which this Ph.D. is part has been focussed largely on the concepts of autonomy and coordination, I will present in this chapter the position our group has taken towards these issues.

4.1 Autonomous Agents

4.1.1 Agents and autonomous agents

The term agent is commonly used to describe a software or hardware component *of an open system* that can be considered and studied in relation to this system. This implies some separation from the system in question, so that the agent can be considered as a distinct entity within the system, dividing the system into an agent part versus environment part (the agent part being a symbol system). The term agent is usually used to emphasize the fact that a component in a system has specific properties that have an interest of their own. It is important to be aware that this definition is not formal in that sense that any object in an object-oriented programming language or any function in an imperative language "could" be called an agent, but one relies on the fact that intuitively the word agent is used only for components that have the look and feel of behavioral entities. Also, there has already been much abuse of the term in precisely this sense.

Usually, some component in a system is thought of as an agent when its interaction with this system can be interpreted as operated through sensory-motor equipment (software or hardware). An artificial life entity and a robot are the two typical examples of respectively software and hardware agents. The artificial life entity is precisely studied for its seemingly independent behavior in a programmed environment and robots clearly interact with the real world through sensors and effectors. The fact that an agent is positioned in an environment with which it has direct contact through its sensors is called *situatedness* in agent theory and emphasizes the fact that the agent is placed in the context of its problem domain and not in a representation of this problem. A term that is closely related to situatedness and characterizes the fact that the agent receives direct feedback from its own actions in the environment is embodiment. Embodiment highlights the structural coupling between the agent and its environment, implying that the agent experiences changes in the environment and will have its behavioral principles changed by the environment's evolution.

Agent terminology is often used in conjunction with autonomy [19] as an agent is deemed interesting for artificial intelligence only if it owns some important feature of natural living systems. Under this criterion, the most highly regarded attribute of living things is their ability to insure their own survival in the world through constant adaptation of behavior. The term autonomy is used to describe the status of an agent that is self-sufficient in some way and sometimes autonomous agent is implicit in the term agent. An extreme example of this is when Sloman defines agents as "behaving systems with something like motives", that is, to him, agents are things that have some form of free will [58] and are thus necessarily autonomous. We will see that autonomy can have a range of meanings in the next paragraph.

4.1.2 Behavior and Autonomy

Behavior describes the pattern of actions that an agent expresses when confronted with its environment. For an autonomous agent, the behavior is the means by which it achieves its objectives and which make it autonomous. An agent may express a wide range of behaviors and these will give an indication of its degree of autonomy. On the lowest level, I have described an agent as being capable of interacting with its environment. For an agent to be considered autonomous, it is commonly admitted that the agent must at least be automatic, that is, *the agent must be able to operate in its environment, sense it and impact it in ways that are beneficial to the task it must accomplish* [59]. From there, an agent is considered as operationally autonomous when it is independent of human intervention, the agent can accomplish its goals on its own (in "normal" situations) [46]. The highest level of autonomy is reached when the agent is behaviorally autonomous, when it is additionally able to generate new behaviors originating in its own past experience of interactions with the world [59]. Thus, a behaviorally autonomous agent would be able to store past experience and interpret it in order to adapt its behavior to recurring or new situations, thus aiming at an optimization of its behavior in view of its goal.

Of course this description of autonomy is by no means exhaustive and could be further refined, but the steps of automaticity, operational autonomy and behavioral autonomy match the various types of agents that are currently used or studied. Actually, automaticity is what is achieved in many engineering solutions for assembly line robots. A very good example of operational autonomy is found in the Mars Pathfinder Rover robotic agent [60]. The rover is a robotic agent equipped with various sensors that allow it to monitor power consumption, obstacle proximity, wheel position, etc. Moving it around on the ground of Mars is achieved in a two step process: a human operator views the Mars site around the rover from a 3D reconstruction of what the robot has captured with its stereo imaging system and he designates a “safe” path from the current robot position to the goal position by defining a sequence of waypoints, the robot then proceeds to the goal without any further human interaction. In this process, the agent is told where it has to go and which different intermediate positions lead to that objective, thus the agent is not even automatic in this planning procedure; then when the operator has finished this description, the robot enters an operationally autonomous state where it deals with all the real-time problems it encounters while transiting from one point to another on the surface of Mars, such as bumper-rock contact and then avoidance.

In general, operational autonomy is what artificial intelligence can efficiently achieve and behavioral autonomy is what is sought to be achieved (and sometimes done in a limited manner). The category of behaviorally autonomous agents is in fact open and contains human agents at its higher levels. It is implicitly admitted here that the faculty of cognition is a property of systems possessing the highest levels of behavioral autonomy that can be achieved in agents.

4.1.3 Multi-agent Systems

A multi-agent system (MAS) is a system where multiple agents coexist in a common environment. In comparison with our previously defined agent system where a single agent is coupled to an environment through sensors and effectors, in a MAS, a family of agents is considered and each agent of the family is coupled with the environment that includes its fellow agents. On the other hand, since the family of agents can itself be considered as a single agent with all of the individually available perceptual equipment (or effectors) brought together as a single sensor (resp. effector), multi-agent systems can give rise to a recursive definition of an agent system [37] or can be viewed as a refinement to agent systems (see figure 4.1).

The central idea of multi-agent systems is that agents within a system may work together (cooperate) or against each other (compete), but as a whole bring forth a collective behavior and it is this collective behavior that will be the focus of research. This idea is simply advocated by Marvin Minsky as *agents are the members of a population that together produce a behav-*

For an agent system, let S be the system, A the agent and E the environment. We have $S = A \cup E$, and this view is centered on the agent A. When many agents can be distinguished in the system, we have many individual perceptions for the same system, that is: $S = A_1 \cup E_1 = ... = A_n \cup E_n$. For each agent A_i, the environment is made of the other agents $A_j (j \neq i)$ and the rest of the environment $E' = S \setminus (\bigcup_{i=1}^{n} A_i)$. A multi-agent system considers the agents A_i $(i = 1, ..., n)$ as a single agent A' with respect to the environment, so that we have $S = A' \cup E' = (\bigcup_{i=1}^{n} A_i) \cup E'$, but focuses both on the interactions between A and E and between the A_i themselves.

Figure 4.1: Formal description of a multi-agent system.

ing system with motives in [40]. What is attempted here is the application of a reduction principle on the individual behaviors in the sense of the question "what elemental behaviors can be brought together in interaction for some interesting meta-level behavior to be generated?" and the inspiration of this approach results from the observation of natural systems that rely on some collective behavior to successfully achieve autonomy. Typical often cited examples of such systems are anthills, beehives or termite colonies, but could also be every multicellular lifeform.

Up until now, I have held a view that was directed at interaction as information exchange between an agent and an environment. Within multi-agent systems, interaction between the various agents composing the system is fundamental. This shift of interest is grounded on the basic assumption that designing elementary behaviors for agents and hierarchically composing them is easier than hard coding intelligent behavior into one entity. Another interesting possibility that is often explored is that collective behavior will not be trivially derived from the singular behaviors and might in fact be more than what the individual behaviors could accomplish, or as in the established formula: the whole is greater than the sum of the parts, which brings us naturally to our next subsection.

4.1.4 Interaction and Emergence

Interaction is the history of transformations that have been effected by an agent on its environment and by the environment on the agent trough its sensors. Interaction is produced when a situated, embodied and behavioral agent "lives" in an environment. An agent is thus said to be in interaction with the environment when it is actively transforming and being transformed by its surroundings.

When the environment is composed of multiple agents, this definition can be extended to agent interactions as in the following: interactions are the behavioral patterns that form between agents coexisting within an environment. Knowing that agents may transform their surrounding environment by exercising their manipulative abilities, they are able to modify the information perceived by other agents in the same environment and thereby change their future actions (an agent's behavior depends on its perceptions). Since this ability to modify the behavior of another agent works both ways, interaction appears as a ping-pong process of behavior transformation between two agents, an effect dubbed specularity by J.-P. Dupuy in the context of human interaction [16]. When more than two agents coexist in an environment, interaction between all the agents quickly becomes difficult to apprehend (the number of interaction relations between agents grows on the order of the square of the number of agents involved).

A designer or observer of a MAS may call a behavior emergent when this behavior cannot easily be deduced from the individual properties of the agents in the system, or when these properties are not readily accessible. Emergence arises from the interactions of the agents in the system in the same way as the chemical properties called viscosity or fluidity "emerge" from the physical properties of the molecules composing the liquid. Once again, it is vital to emphasize the role of the observer in this definition, the notion of emergence is an epistemological one: a property of a collective emerges from the individual properties when the most adequate tools used to study the individuals are not the same as those needed to study the collective. Clearly, recalling my previous example, if quantum mechanics were a perfect model of reality and we had access to perfect tools, viscosity would be most adequately studied through quantum effects and elementary particles.

4.2 Agent Methodology

Artificial intelligence has tried to solve the problem of intelligence by attempting to synthesize it in artifacts (more specifically, computers). Over the years, its methods have evolved from reasoning over models of problem domains to emergence of behaviors in situated artifacts. I will here motivate this evolution by going through the two major steps that led the field to its current state and describe what can be done to build behavioral agents.

4.2.1 Top-down Approach

The classical artificial intelligence perspective on intelligent behavior is that of reasoning. That is, an intelligent agent trying to solve a problem is expected to try and summarize its problem, make an abstraction of it (in the form of symbols and expressions combining the symbols) and then apply deduction and inference to extract a solution from the symbolic representation of the problem, again in the form of an expression. It is then supposed to convert the answer back into the problem domain from the abstract representation it has of this domain.

This approach reflects the methodology of the natural sciences as described by Herbert Simon in [55]: *The central task of a natural science is to make the wonderful commonplace: to show that complexity, correctly viewed, is only a mask for simplicity; to find pattern in apparent chaos.* Here, intelligence is equated to the ability of reasoning upon a (reduced) model of reality. The hypothesis is then that a physical symbol system, a device that can hold and manipulate symbols in a representation, is sufficiently general to be able to produce intelligent action. This framework is called top-down because its starting assumptions are about high level cognitive capacities such as modeling, planning and reasoning. It is from these assumptions that classical artificial intelligence expects to generate intelligence that will deal even with the down to earth problems that a living being encounters every day.

4.2.2 Bottom-up Approach

New AI on the other hand considers intelligence as emergent from a systems ability to deal with many simple problems that appear in its interaction with an environment. For this bottom-up approach, the high level cognitive abilities such as forming internal representations of the environment and reasoning on these representations is a feature that an agent might develop if faced with an environment of sufficient complexity, but is by no means preexisting in an agent. The aim of new AI is to build agents that can deal with their environment by reacting to changes in this environment in a way that will ensure their continued integrity, usually this is accomplished by providing the agent with a set of basic abilities and bringing it to arrange the use of these abilities in search of an efficient, self-sustaining behavior. An agent may then be called intelligent if it is able to deal with problems that seem to require intelligence, but there is no preliminary assumption about what the internal processes of intelligence in an agent are or should be.

4.2.3 Adaptation

The earliest methods used in artificial intelligence had the intention of programming behavior in an agent by attempting to specify all the possible situations the agent could encounter and providing it with means to cope with such situations. Unfortunately, such techniques have proven not to be scalable since the programmer is responsible for thinking of every possible situation the agent might encounter and the number of such situations follows an explosively growing curve as the complexity of the environment grows (this has been called the *frame problem* for classical artificial intelligence). To counter this problem, the available solutions are either to use generalization in problem recognition and thus give an agent some default behavior to use when an unforeseen situation occurs, an unsatisfactory solution in many cases, or to allow the agent to build new solutions for unforeseen situations through some adaptive algorithm.

I will use here the term adaptation for any internal mechanism that allows an agent to modify its own behavior-generation processes depending on the experience it acquires from the environment. Actually, almost all artificial intelligence research is now devoted to understanding adaptation. This is due to the transfer of effort from the concept of intelligence as an explicitly programmed feature in an agent, to the concept of intelligence as an emergent property of agent interaction with the environment. Since we are not able to write a definition of intelligence, we cannot make a model of intelligence and implement it in an algorithm, rather, we expect to produce the appearance of properties akin to intelligence in a system where interaction between components occurs.

The fundamental difference between adaptive agents and non-adaptive agents is that for an adaptive agent, different environmental conditions will produce different agents after some time, whereas a non-adaptive agent will still behave in the same way after any period of time it is left in the environment. It should be noted that most often, typically for agents that are used to solve real problems, the agent is adapted in some preliminary phase, but is then used as an non-adaptive agent in the state that was reached through the adaptation phase. The reason for this is that an analysis of the agent behavior can then be made on the adapted agent before it is used on the real problem environment where failure could prove dangerous or costly to the user.

I identify only two basic techniques used today to induce adaptation in agents at this time, these are learning and evolution which I will now describe.

4.2.4 Learning

Learning is an internal (ontogenetic) adaptation mechanism, whereby an agent sees its internal behavior generating algorithm modified by the effect of external environmental conditions perceived through its sensors. There are many specific learning techniques that can be used to this goal but essentially the process is the same for all of these. We can assume that a learning agent has a two layer internal algorithm, the first layer is the behavior generating algorithm component and the second acts as a critic of

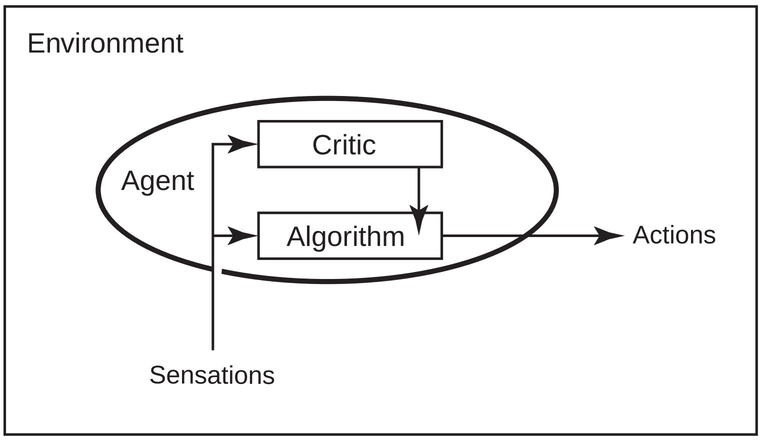

Figure 4.2: A learning agent.

the first layer. Whenever the first layer generates an action based on its perceptions or internal state, the critic attempts to evaluate the quality of the action with respect to the agent's predefined objective. Based on this evaluation, the critic may change the behavior generating algorithm layer. To pursue its goal, the critic in a learning agent may have access to more information than the agent algorithm in some cases. For example, supervised learning techniques use feedback from an external observer of the system to provide the critic with an evaluation of the agent's performance.

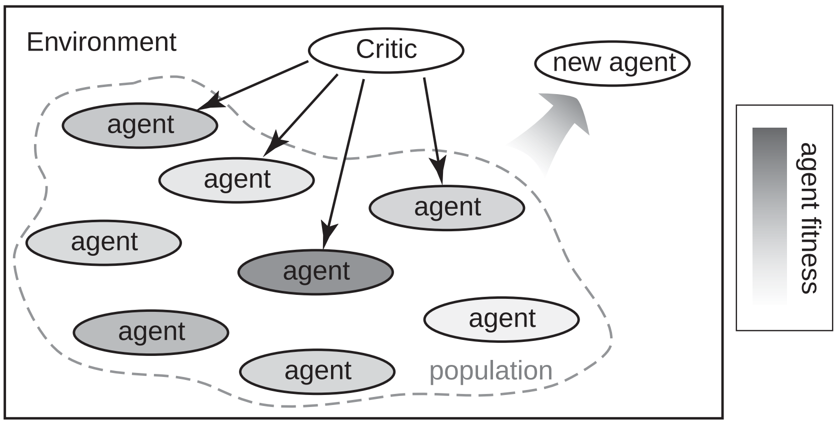

Figure 4.3: Evolving a population of agents.

4.2.5 Evolution

Evolution is an external (phylogenetic) adaptation mechanism operating on a population of agents, whereby successive generations of agents are modified according to the relative success of previous agents operating in the environment. Evolutionary techniques are inspired by the Darwinist theory of natural selection, where environmental pressure is used to ensure the survival of fittest. In an evolutionary algorithm, a population of agents is considered and an external critic belonging to the environment evaluates the relative quality of individuals in the population of agents by assigning to each of them a fitness value. New individuals are then occasionally created by combining the properties of multiple individuals in the population. The selection of which individuals will contribute to the creation of the new individuals is biased towards high fitness individuals. In the illustration of this process 4.3, I show the critic as an agent that is external to the evolving population. In fact, this is a representational abstraction: the critic may very well be a distributed entity that is partially incorporated in the evolving population through competition mechanisms as is the case for evolution in the real world.

Chapter 5

Games and Text-based Virtual Reality

When the AXE project was formulated, the main idea was that the problems of global networking can be solved by using populations of agents distributed in the network. This vision is based on the fact that agents living in a virtual world made of interconnected computers should naturally be able to solve questions of distributed problem solving, network maintenance or load balancing by adapting them to perceive the properties of their current location and learning how to move in the network, possibly picking up data from one location and transferring it to another location. With this goal in mind, an experimental platform that could support these ideas was deemed necessary and it is this platform that was designed in the EMud environment model. In order to think about distributed networks of computers, a game metaphor was the first idea that occurred to me, inspired by my role-playing background. It is this gaming environment concept that I introduce in this chapter by showing the evolution of computer games from their beginning to the current Mud games.

5.1 The Origins of Computer Games

The evolution of the use of computers for gaming purposes has basically gone through three stages. At first, computers were used as a replacement for a human partner or opponent in a classical game that was until then played without computers. This is typical of the strategy genre of games, which are usually of the board game type, but transposed to computers.

In the next stage, games were developed specifically for computers and these cannot easily be played without a computer base. Two genres represent this evolution: action games, where the player must match his dexterity against a series of challenges generated by the computer, and the adventure kind of games where a player wanders around in a virtual setting described by the computer (textually or visually) and issues short command sentences to try to solve enigmas. Early representatives of these game families are typically chess or Othello (Reversi) for strategy games, Pacman or space invaders for action games and Advent or Zork (Dungeon) for adventure games.

Although these categories can be further divided and are slightly mingled in more recent games such as wargames, simulators or computer role playing games (CRPG) of which we will speak later, the last transition in computer game concept was introduced in the late seventies with the use of networking for online gaming. Technically, online games are not a new genre, but are rather an extension from all other genres to human-human interaction through the computer media.

Although online games have been available since the seventies, it is only during the last few years that their success has been rising sharply. Until recently, there were no commercial games of that type, which can be explained by the limited popular impact of such a product in a world without widespread Internet access. But today, it seems that the "multiplayer" feature will be inescapable for any game in the near future. Indeed, a large part of the success of games such as the shooting game called Quake is that they can be played cooperatively or competitively with other players over the Internet. Maybe one could consider this as one of the failures of artificial intelligence techniques used to generate ally / enemy behaviors for games where the rules are more numerous than in classical board games, but in any case, the challenge of comparing one's skill to that of other human player will always remain.

5.2 Muds

5.2.1 From Fantasy Literature to Advent and Computer Role Playing Games

The publication of "The Lord of the Rings" in 1937 [62] marks the beginning of fantasy literature by using typical mythological and epic elements in a modern fiction novel. A large community of fans gathered around this book and its intricate description of the fantasy world in which the story takes place. Much literature has since then been published with a similar setting, but the importance of the genre is also due to its propensity to produce the attractive imaginary worlds that attract players of role playing and adventure games.

In 1974, Gary Gygax releases his game system called Dungeons & Dragons (D&D), Basic Set [21], which is the first role-playing game (RPG) to be published. In D&D and role-playing games in general, players design a character based on the game system's rules and writes this description on a character sheet. A referee (the game master or GM) invents a story with descriptions of puzzles, opponents, treasures all fitted in a world setting that he has imagined. The players and the GM then meet around a table to "live" through the scenario created by the game master, who acts as an omniscient and omnipotent entity ruling out the results of the behaviors of the characters in his world. Role-playing games now come in a wide variety of settings, from fantasy and modern times to science fiction worlds, but D&D was distinctly based on Tolkien's world of elves, dragons and other mythological creatures.

Actually, this type of games is quite similar to role-playing experiments in psychology [41] except that players do not act out their chosen behavior, but explain it to the referee who then describes the results. (Which is probably a good thing since role playing games often involve quite a lot of hacking and slashing at monsters with swords!)

It is in this context that in 1976, Will Crowthers designed Advent (the six character name restriction, that was then common place on computers, for adventure), a computer refereed fantasy game inspired both by fantasy literature and his own (real) caving experience. The game was subsequently greatly expanded and released by Don Woods and was the first adventure game [25]. In Advent, the player explores a text-based description of a cave system, populated with creatures and objects, where he must solve puzzles to reach deeper into the cave system and score more points. Shortly after the appearance of Advent, another game called Zork [34] (later called Dungeon for its Fortran version) was written along the same line by Tim Anderson, Marc Blank, Bruce Daniels and Dave Lebling in a language called MDL.

5.2.2 Playing Adventure Games

In a text-based adventure game like Advent, the player acts as if he was a person in an imaginary world, this person is usually called his character in the game. He is given textual descriptions of what that person can see around the place he is standing in the adventure world such as:

> You are standing at the end of a road before a small brick building. Around you is a forest. A small stream flows out of the building and down a gully.

Which is the first screen of the Advent game. The player may then issue various commands that his character will attempt to execute. A command usually consists in an action verb followed by the object of that action. After mucking about a little, the player might try to enter the described building:

> \> enter building
>
> You are inside a building, a well house for a large spring.

```
There are some keys on the ground here.
There is a shiny brass lamp nearby.
There is food here.
There is a bottle of water here.
```

And pick up the objects he sees:

```
> get everything

Sorry, I don’t know the word “EVERYTHING”.

> get lamp

OK

> get keys

OK
```

When commands are not understood, the game will inform the player and ignore the attempts. Otherwise, various humorous or gruesome things can happen to a character playing Dungeon, the player might encounter the following situation:

```
You are in a dark and damp cellar with a narrow passageway
leading east, and a crawlway to the south. To the west is the
bottom of a steep metal ramp which is unclimbable.
The door crashes shut, and you hear someone barring it.
Your sword is glowing with a faint blue glow.

> inventory

You are carrying:
A sword.
A lamp.
A newspaper.
A leaflet.

> east

You are in a small room with passages off in all directions. Blood-
stains and deep scratches (perhaps made by an axe) mar the
walls. A nasty-looking troll, brandishing a bloody axe, blocks
all passages out of the room. Your sword has begun to glow
very brightly.

> hit troll with sword

The fatal blow strikes the troll square in the heart: he dies. Al-
most as soon as the troll breathes his last, a cloud of sinister
black smoke envelops him, and when the fog lifts, the carcass
has disappeared.
Your sword is no longer glowing.
```

Which is the happy outcome, the other would signal an end game...

> You are in a small room with passages off in all directions. Bloodstains and deep scratches (perhaps made by an axe) mar the walls. A nasty-looking troll, brandishing a bloody axe, blocks all passages out of the room.
> The flat of the troll's axe hits you delicately on the head, knocking you out.
> Conquering his fears, the troll puts you to death.
> It appears that the last blow was too much for you. I'm afraid that you are dead.
> Do you wish me to try to patch you?
>
> > no
>
> What? You don't trust me? Why, only last week I patched a running RSX system and it survived for over thirty seconds.
> Oh, well. Your score is 35 [total of 585 points], in 14 moves.
> This gives you the rank of Amateur Adventurer.

The aim of these games is to collect the objects that are lying around the various places in the imaginary world, finding those that serve to solve riddles allowing further progression and bringing home the treasure objects that increase the character's final score. Part of the fun is in discovering how to solve the riddles or avoiding the evil monsters that lurk in the darkness and part comes from exploring the world that the designer has written. In the examples that I have given it is not immediately apparent, but the syntax of these games is very restricted and a good deal of patience is also necessary to find the right formulation for an action that the character must execute but "doesn't understand".

5.2.3 The MUD and Muds

Inspired by Advent, Zork / Dungeon and in a lesser way by Hack (another adventure game), Roy Trubshaw at Essex University decided to write a multi-player adventure game in spring 1979. After writing a first basic assembler version where people can move around rooms and chat together, he rewrites it completely in a higher level language (BCPL). With Richard Bartle now doing much of the game play oriented programming, they finished in 1980 what is now believed to be the first MUD [4]. The name MUD is the acronym for Multi User Dungeon, where Dungeon comes from the Fortran name of Zork that inspired the authors of MUD. Nowadays, MUD is also sometimes taken to mean Multi User Dimensions since it is a more accurate reflection of the variety of settings (dimensions) in which one can play Muds.

The two ideas that brought R. Trubshaw to write the MUD were his interest in writing a database definition language and the making of a multiplayer adventure game. It is the latter aspect that became most important over time and when he left Essex University, R. Bartle took over the project and added most of its game oriented features, that is, puzzles to solve, atmosphere, etc.

Popularized by students throughout the UK, MUD quickly spreads to Norway, Sweden, Australia and the USA. Since then, there have been many programs written either over the original MUD code or strongly inspired by it. These kernels form the basis of today's Muds, examples of these are typically LPmud or DikuMUD, and most current running Muds use these as a code base.

5.2.4 Playing Muds

The player of a Mud is faced with the same environment as that of an adventure game player. He receives textual descriptions of his immediate surroundings, including the objects the he may pick up or the enemies he can fight. There are also various simple riddles that he may solve in order to progress in the game and the commands that can be executed take the same form as in an adventure game. Where differences appear is in the characters that are played and the scoring system. A Mud character can choose among various professions and has a more detailed description. His skills can vary depending on the choices the player makes while playing and may have access to commands that not all other players are allowed to use, for example, thief-like characters might be allowed to *steal* objects from creatures and wizard-like characters may have access to magic spell *cast*ing. A multiplayer game character is in general very customizable, allowing players to decide how they look like, what their name is and what clothes they are wearing.

When exploring the world in a multiplayer game, the player's character encounters creatures in the same way as in an adventure game, but might also come across other characters. He can then engage in conversation by issuing the *say* command, followed by a sentence the he wants the other player to hear (see, in fact), or even make his character act in various visual ways by using the *emote* command to show the other player his emotions:

> > east
>
> You enter the central courtyard in Tintagel castle. Big double pane doors lead to the round table room in the north. To the south and west, secondary entrances lead inside the castle. In the middle of the courtyard stands large podium where King Arthur can address his people.

```
A barrel of water is here
Some hay is lying here
A horse is standing here
You see Vania

> look Vania

You remember having seen Vania at the court once, she must
be a minor lord from the country. She is a mature woman of
forty years who clearly has seen rough times. Her clothing is
correct, but obviously designed for riding, and while her stance
is proud, she could easily pass for a commoner.

She is carrying:
a leather backpack
a sword
a suit of leather armor
black riding boots

> say Hello Vania

You say “Hello Vania”
Vania nods

> emote bow Vania

You bow before Vania
```

Depending on the Mud, mutual aggression may be possible or not, but it is generally considered bad taste and most Muds have a board of internal regulations to explain the behaviors that are allowed, tolerated or forbidden. These rules are enforced by special characters that are played by the maintainers of the Mud or simply people that have been playing for a long time and are deemed responsible. A lot of the reputation of various Muds that can be played on the Internet comes from the general atmosphere that is maintained for the players.

The scoring system in a Mud differs considerably from that of other games and is the same as in role playing games. In a Mud, characters are given an experience point score. When creating a new character, the score starts at zero and it is by fighting creatures (killing them actually) or solving quests that the character can gather experience. These points are used to calculate the level of advancement of the character in his profession and by accumulating them, the character can increase this level. With higher levels, character gain proficiencies in various areas dependent on their profession and the choices the player makes. All in all, more advanced characters become more powerful, allowing players to compare their characters according to what each can do or which monsters they are able to beat.

5.3 MUCK, MUSH and MOO

While early Muds emphasized competition, riddle solving, character development and can be called adventure Muds, so-called social Muds have since then been designed. These Muds emphasize creativity, programming skills and communication in the sense that their worlds are not created with populations of dangerous creatures, quests or generally hostile environments, but exist for the players to modify and enlarge by themselves. Characters on social Muds are allowed to build new places and new objects to interact with through the use of a programming language. Their aim is to design places of their own, that other characters can visit, admire and / or criticize. On such a Mud, the time not spent building is spent on communicating with other players or exploring their creations. The first decisively socially oriented Mud, called TinyMUD, was written by Jim Aspnes in 1989. It allowed players to create new object, places, etc. by spending pennies (Mud cash) acquired on the Mud. All characters had equal powers and the idea was that TinyMUD had to be a place where non-competitive people were happy to be. Following TinyMUD came TinyMUCK (the programming language used for creations involved less MUCKing about than on TinyMUD), TinyMUSH and many others who inherited the MUCK or MUSH nickname. Following the same basic concepts, MOOs were written to implement socially oriented Muds where building is done through the use of and object oriented programming language.

5.4 Text-Based Virtual Reality

Today, virtual reality is everywhere and the text-based virtual worlds of early games appear like the poor fathers of their fully animated, three dimensional, sound enhanced children that can be bought in every superstore. But, taking away the gore, every element that is truly entertaining was already in these first games: incredibly large worlds to explore, the challenge of tricky quests to solve, rewards for the (play) efforts involved and mainly places to live out one's imagination. These are even elements that now are often overlooked by the entertainment industry in favor of the thrill of new sensory experiences, even though their comeback can be felt is some recent games. This additional sensory experience that appears so important nowadays is a trend that appears in all computer-human interactions in the attempt to interface man and machine. But if we consider artificial agents (not interacting with humans, that is), this is probably the element that can most safely be omitted from the environments designed for them. Instead, an environment customized for an agent would more appropriately be implemented as a discrete symbolic world, since this agent will have a discrete symbolic internal mechanism.

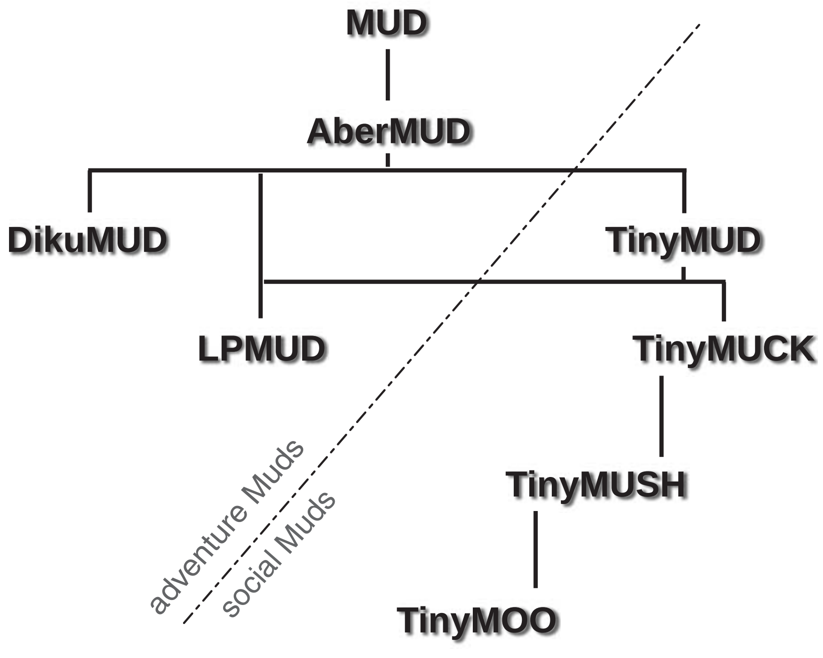

Figure 5.1: A historical hierarchy of Muds [29].

In an attempt to ground symbolic agents in an environment, I believe that our first attempts should be concerned with environments that are intuitively close to the representation mechanisms these agents possess, making them feel at home in a sense. Once our methods enable such agents to solve complex problems in these environments, the next step will be to bring them to real world problems that anyhow must be converted to a discrete description for the agent perceptions. The idea that only real world problems can bring agents to integrate high level behavioral patterns, appears misguided to me. The real difficulty lies in an agent handling situations of a structural complexity level larger than its internal representational structure, whereby the agent must discover or be provided with adequate generalization / compression capacities in order to solve the problems he is faced with. I have chosen to use the EMud environments introduced in the next chapter as experiment environments for this reason. An EMud environment is a virtual text-based world designed from the principles of Muds. In this world, places and objects are of a symbolic nature, the transformations in this world are rule-based and so, the world can be directly experienced by an agent placed in it. Additionally, since an EMud is a Mud, human players can control an agent in the virtual world, allowing interaction with artificial agents. For these reasons, both complexity and the basic unpredictability of human actions can be introduced in the environment to provide a sufficiently challenging problem space for agents.

Chapter 6

EMuds, Aims and Methods

6.1 Aims

The objective of the model presented in this chapter is to provide an experimental platform for multi-agent systems that can be used as an abstraction for a network of computers. The concepts used to define this model are strongly inspired from the Internet games called Muds that were introduced in the previous chapter. Designing the system was done by separating structural elements such as space topologies and contents from the dynamics of the environments considered.

The resulting model gives an abstract description of virtual universes that can easily be implemented on computers [49]. These universes can either be seen as purely virtual structures or as a representation of real world problem domains. When used as an experimental testbed, an EMud environment will be implemented as a representation of a problem that must be solved by agents living in it. To the agents, on the other hand, the world acts as a purely imaginary structure in which they can perceive local properties and change them, and where they must adapt their behavior.

6.2 EMud Environment Model

An EMud is an application that manages environments called Worlds. The worlds define environmental structures and associated dynamics and thus provide experiment testbeds. I will define here the model used to describe a world.

Worlds can be described by separating structure and dynamics, structure consisting in the virtual physical components of the world and dynamics in the virtual physical laws that the structure obeys.

6.2.1 Structure

There are three main types of components in the world structure. These components globally define the world's topology and are called parts. The various parts, with the most important features in their implementation scheme, have been outlined on figure 6.1 at the end of this subsection.

Elements

Elements are the physical objects of the virtual world. The objects that have a substance and thus can be manipulated by other elements possessing adequate detector / effector (in a general sense) equipment. In comparison to the real world, elements are the car you drive, the glass you drink from, your cat and even yourself or other human beings. As seen from the preceding examples, elements encompass both the intuitively active and passive objects of the environment. Elements that can "contain" other elements are called containers and are a specialisation of elements.

Spaces

Spaces are the places where parts dwell. They serve as a mean to regroup subworlds into zones and they provide a location for elements to exist in and sense their surroundings, as well as being the medium in which interaction can occur. Spaces come in two varieties: zones that allow to regroup other spaces into areas and loci which may hold elements. Within a locus, positions can also be defined when specification is necessary. A room is a locus in a zone and an inventory is a locus within a container. An important feature to note is the recursive nature of spaces: spaces may exist within spaces or elements. A typical real world example of this is the universe space containing our galaxy space or a room space containing crates that have internal space for contents. The default top-level space is a zone called the void that contains all other spaces.

Exits

Exits are the paths from locus to locus. The exits define the basic physical neighborhood relation over spaces. Although the virtual physical laws of an environment may define other neighborhood relations, the exits form the basic topology for the set of structural components. It is essential for any experiment to have at least this topological feature, even though some active elements might not be perceptually equipped to be aware of it.

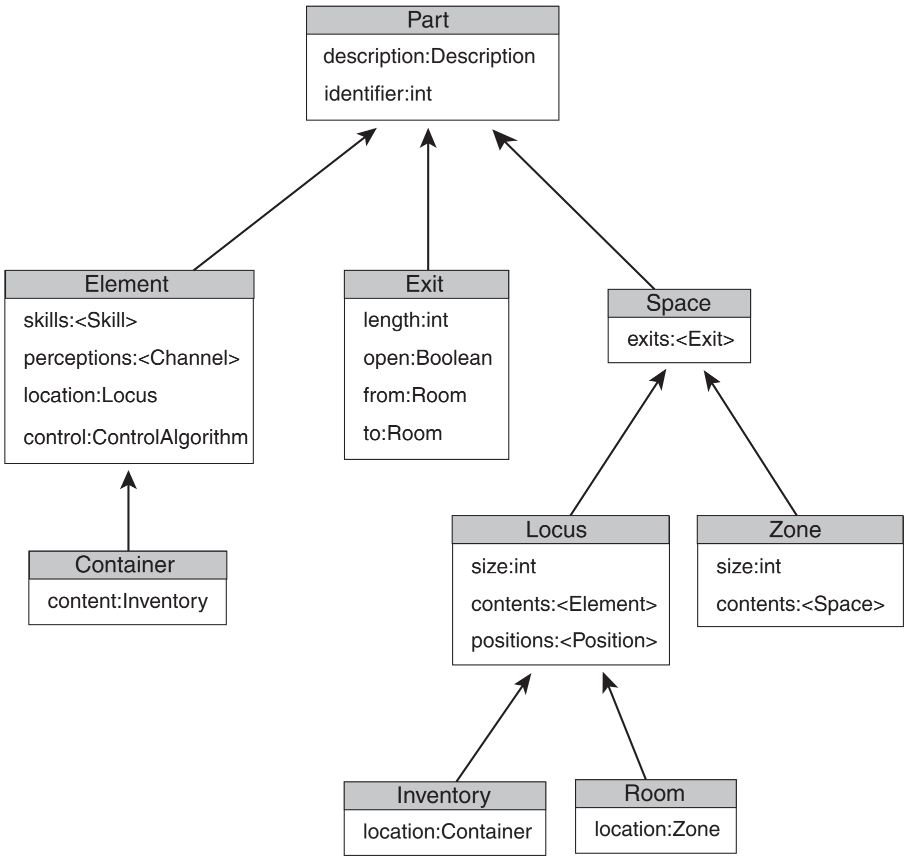

Figure 6.1: Structural components of and EMud.

6.2.2 Dynamics

Perceptions

Elements within a world are given a set of perceptions (a sometimes empty set). These perceptions consist in information channels where events of a specific physical type or nature are recorded. The various perceptible events are propagated through the world's structure following rules of propagation defined by the actions that can cause them, these events are only communicated to the elements if they possess adequate perceptual channels, even if their position is "within range". Thus, shouting at a table (inactive, unperceptive virtual table of course) has no effect on the table element. Generally speaking, if an element is not perceptive to a specific type of physical event, that type of event can have no effect on the element's internal mechanisms (unless the event destroys the element, which then has no further effect on the element...). Of course, the existence of translator elements (comparable to radio receptors, magnifying glasses, etc.) is not excluded by this rule, but otherwise, such unperceived events can be considered as an epiphenomenon to the element in question.

Actions

Actions are the transformations that can be effected on the world's structure. They are considered as components of the world in the sense that elements may produce them and they are then stored until execution time. Typical actions are moving, picking up elements, emitting sound, etc. The events in the world that such actions generate may be delayed or have durable effect, depending on the action. To allow a variety of effects (such as temporary effects), actions may generate other actions. A temporary effect is obtained by an action that modifies the environment and then generates a new action (with appropriate delay) to set it back to its initial state. The set of all possible actions, with their internal applicability rules, defines the physical laws governing the virtual world. This can be understood by the fact that any transformation of the world must result from the application of an action. A component of the world that may generate actions is said to possess skills, that is, a skill is the basic description of how an action is to be generated. The set of skills of such a component is called its proficiency.

Control Algorithms

Control algorithms are the means by which elements may make decisions to commit actions in the world. As it was explained in the element section, elements can be either active or passive. By default, an element is passive, but it can be equipped with a control algorithm that allows it to interact

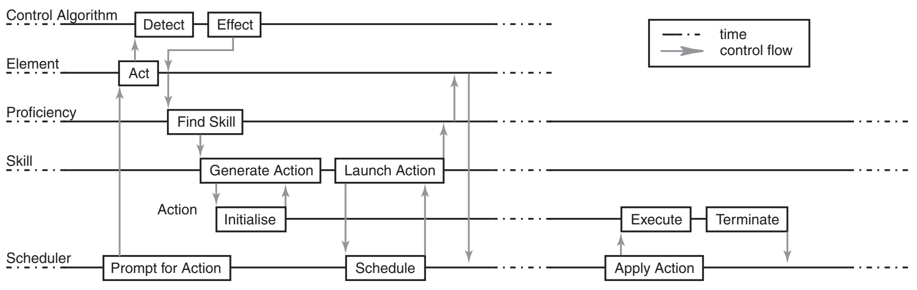

Figure 6.2: Scheduling of an element's actions.

with the world. Such an algorithm has access to both the perceptions and the internal state of the element and from this information, may generate actions to effect the environment. Any type of control algorithm may be fitted to an element and in fact, a human player connected to an element is considered as a control algorithm (whatever the connection method). A typical experiment world could contain elements driven by neural nets, genetic algorithms, expert systems and remote human players connected via telnet ports.

Time

Time in the EMud is a global, discrete ordering of the occurrence of actions in the world. Since actions are executed sequentially, the order of their execution is well defined and can be divided into time slots. Every action has a positive integer execution delay that allows it to be scheduled in world. The temporal evolution of the environment is the sequence of environmental states taken before every time step. In this context, the scheduler is the entity that coordinates all actions in the world, it is through the scheduler that elements are prompted for action and in the scheduler that element actions are stored before the time of their execution (see figure 6.2 for an illustration of this mechanism).

6.2.3 The World

Together, the previous components form the EMud world, where elements "live" in the structure formed by spaces connected through exits. In this model, agents are implemented as elements with a control algorithm. These agents, following our perception / action agent model (see the section on autonomous agents), acquire information about the world through perception channels and use actions as effectors to modify this world.

6.3 EMud environment definition

Environment definition is made by creating an appropriate set of structural definition files and dynamics definition files that are then loaded into the application at startup. A simple grammar based on name-value couples is used for writing the files and each couple represents a parameter of the environment in a hierarchical manner. The hierarchical decomposition is done through the use of complex values for certain parameter names. Typically, an element is described as an *ELEMENT* name with complex value consisting in the various parameters defining an element. Within this complex value, a parameter named *LOCATION* would have a simple value that is the integer identifier of the element's location in the EMud environment. Examples of complex values are those associated with part descriptions, element attributes or element proficiencies. Examples of simple values are those associated with description names, exit lengths or space sizes (see figure 6.1).

An environment will usually consist in many such definition files, the minimal environment having one each of the default space, element and exit files. When creating a more complex environment, the configuration files may be structured into any number of environment areas, with space, element and exit files for each area, along with one area description file. Since the world is structured into sub-spaces thanks to the use of the space parts, building an extension to an environment is done by adding a new space called a zone. The new rooms to be added are then situated in this zone, delimiting the newly built area. Elements and exits added in rooms of this zone are also marked with their current zone identifier. For each zone, a separate set of definition files is used. When the EMud is running, the current state of the mud can be saved into these environment definition files and when the mud shuts down, this operation is automatically launched. Thus continuity of the world can be established by starting up the mud from a previously saved state and a study of the world's evolution can access successive states of the world in these various files.

Control algorithm states are also stored in a similar fashion as environment description files, but the current implementation of the model does not yet provide a way of defining new control algorithms for the EMud without touching the code of the application. The EMud actions are also only modifiable at this time by transforming the EMud application.

In an EMud, all parts are uniquely distinguished by a part identifier that specifies their zone membership , their type identifier (whether the part is a room, an exit, a player or another part type), and their personal identity within these subsets of parts. This constraint of unicity limits the maximal size of an EMud world to 254 zones, each containing a maximum of 2^{20} parts of each type (about a million). Obviously, this limit is not too restrictive and I believe the processing power of current computers is more the limiting factor at the moment. I have currently implemented seven zones for demonstration and experimental purposes, altogether containing about 300 parts among which twenty are active and controlled by elementary control algorithms. In this configuration, no delays at runtime are perceptible, but the use of complex control algorithms, such as an XCS classifier system with a population of classifiers over 1000, does slow down the environment slightly. It should be noted that, since its complete rewriting in a project in collaboration with A. Soupper, the application has become very stable, running for several weeks without interruption.

6.4 Conclusions for the Model

The EMud environment model provides support for multi-agent system experimentation. Special attention has been devoted to allow adaptive agents to evolve in incrementally complex environments. To fulfill this goal, it is possible to build environments in steps of increasing complexity along the three distinct directions described below; from pure simulation, towards real-world modelling.

Rich Structures

The model uses high-level descriptions of environments through the use of three basic building blocks: elements, spaces and exits. This allows virtual environments such as those found in the typical artificial life experiments to be built. For example the Santa Fe trail or the Woods experiments can be reproduced by using sets of interconnected rooms containing pheromone or food elements. On the other hand, abstractions of the real-world can be achieved with the same building blocks, for example by mapping a building layout on the rooms / elements / exits framework, identifying offices to rooms, doors and stairways to exits, interior equipment to elements. Since the number of such components can be very large, a lot of detail can be used for these descriptions.

Complex Dynamics

As the elements of an EMud can be given active properties and the actions available to elements can be customized, the laws of evolution of an environment can be closely tailored to the desired simulation. Elements reacting to specific environmental conditions can be used and even hidden in the environment, so as to simulate atmospheric conditions for example.

Basic Unpredictibility

With the possibility of linking elements of the EMud environments with real-world objects (most often human actors), the basic unpredictibility of the real-world can be introduced into environments and used to modify them dynamically. This allows to build environments that might bridge the gap between simulation and real world experimentation.

Chapter 7

Learning Classifier Systems

Originally described by Holland in [24], learning classifier systems (LCS) are learning systems, which exploit Darwinian processes of natural selection in order to explore a problem space. As such, LCS are among the few AI techniques that integrate both an internal adaptation process (reinforcement learning) and an external adaptation process (genetic algorithms) in a single decision-making algorithm. While Goldberg used LCS in his book about genetic algorithms [20] as an application of the theory presented, learning classifier systems had remained relatively unexplored until Wilson successively simplified and improved some aspects of the original systems, dubbing the successors ZCS and XCS. The main transformations are explained in detail in [69, 70, 30] and a summary of the state of research in XCS systems can be found in [71]. The resulting system, XCS, which I will describe in this chapter exhibits some very good learning and generalization abilities in the tested environments and has the advantage of being well adapted to symbolic discrete environment problems.

7.1 Classifier Systems

A classifier system (CS) is a rule-based system for decision making, classifier systems are a special kind of production system [12]. Each rule maps a problem state into a solution or an intermediate new state, where the system can be applied again. This sequence of deductions leads to the systems answer to the problem. Rules in classifier systems are called “classifiers” and are of the form *if <condition> then <action>*. When applied to a problem, a CS is usually presented with a message representing the current situation, the classifier system then attempts to match this message with one or more classifier conditions. If some conditions match, then an action among those advocated by the matched classifiers is selected and applied. The basic system is thus based on a set of classifiers, a matching process and an action selection process.

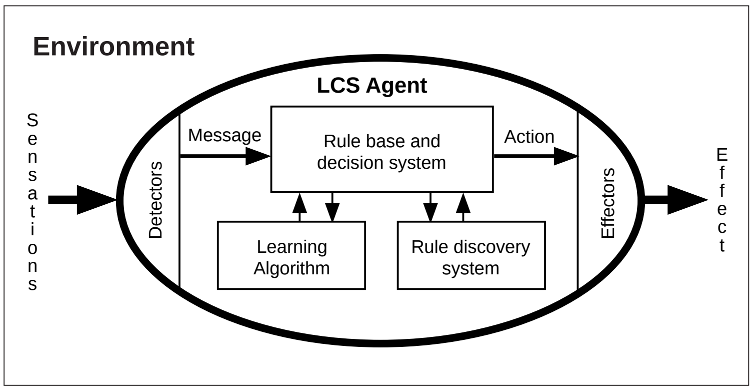

Figure 7.1: Learning classifier system architecture.

The learning classifier systems add adaptation to the basic CS through two components. The first is a reinforcement learning algorithm similar to Q-Learning [27] that operates on the action selection process and that I introduce in section 7.4.3. The second is a rule discovery system implemented as a genetic algorithm [23, 20] that operates on the classifiers as a population to generate diversity in the classifier set, allowing exploration of the problem space. This component is introduced in section 7.4.4. The overall architecture of an LCS agent is illustrated in figure 7.1.

Within an agent system context, the classifier system is the agent's control algorithm with the problem space being the environment and messages the perceived current environment conditions. A learning classifier system provides the agent with an adaptive mechanism to deal with varying environment situations and learn better action patterns through experience. Depending on the type of environment, detectors and effectors have to be customized for the agent to convert perceptions into messages and actions into effector operations. I will present the basics of reinforcement learning and genetic algorithms in the next two sections, before giving an analysis of the XCS classifier system and its operation principles.

7.2 Reinforcement Learning

Reinforcement learning (RL) is a form of learning that is well suited for unsupervised learning situations and is guided by the following principle [61]:

> If an action taken by a learning system is followed by a satisfactory state of affairs, then the tendency of the system to produce that particular action is strengthened or reinforced. Otherwise, the tendency of the system to produce that action is weakened.

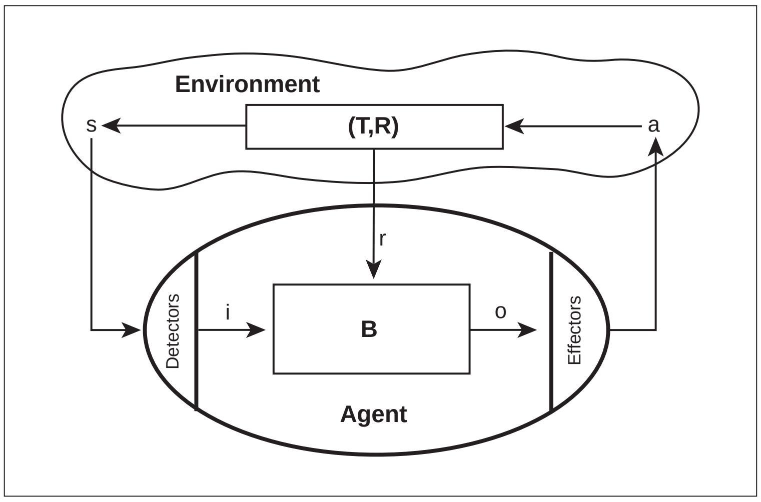

Figure 7.2: The reinforcement learning model.

In reinforcement learning (associative reinforcement learning to be precise), a system attempts to learn a mapping from inputs to outputs, guided by a reinforcement signal that represents satisfaction in the sense of the preceding principle. From the agent perspective, an agent, connected to an environment by sensors and effectors, perceives states in the environment and converts them to inputs describing such currently perceived states. The agent then produces an output that is effected as an action in the environment. From the transformation operated in the environment, the agent then receives as reward a scalar reinforcement signal. The behavior of the agent is aimed at maximizing rewards received as reinforcement. The reinforcement learning model (see figure 7.2) consists in:

- environment states $s \in S$;
- possible actions $a \in A$;
- reinforcement signals $r \in \mathbb{R}$ described by a reward function R from $S \times A$ into $\Pi(\mathbb{R})$, a probability distribution of reward values;
- a state transition function $T : S \times A \longrightarrow \Pi(S)$ for the environment that maps state-action pairs to probability distributions over the state space;
- agent input signals $i \in I$ and output signals $o \in O$.

The purpose of reinforcement learning theory is to find an “optimal” policy B for the mapping $B : I \rightarrow O$. Optimality being defined in relation to the reinforcement signals r received by the agent over time. Depending on the context, many different algorithms can be applied in order to solve reinforcement learning problems, and actually, even evolutionary techniques such as genetic algorithms could be considered as reinforcement learning within the above framework. Traditionally, though, reinforcement learning algorithms are based on statistical techniques and dynamic programming used to estimate and improve behavior-generating algorithms.

7.2.1 RL Methodology

The main aspects that influence the choice of method are the following:

- Optimality criterion: defining what is an optimal behavior depends on making the choice of an optimality criterion and is the first step to finding a solution to a reinforcement learning problem. There are basically three models of optimality. 1) *Finite horizon*, where the agent has a predefined finite life span and has to maximize reinforcement received during this time. 2) *Receding horizon*, where the agent's life time expectation is not known and it is asked of the agent that at each time step, he maximizes the total expected reward for the next h steps. 3) *Infinite horizon*, where the agent takes all expected future rewards into account for the maximization. This can be achieved for example by using discounted sums of expected rewards.

- Environment stability: actions in the environment may or may not influence future states of the environment, depending on this factor, actions may change the future expected rewards and this should be taken into account by the behavior. A similar case happens with delayed rewards, in some problems, reinforcement cannot be given immediately following an agent's action, it is only when certain specific situations occur in the environment that the agent receives reinforcement.

- Perceptive limits: when the agent perceives the environment, a descriptive input signal i is built by the sensors observing the current environment state s. This input signal may not be a complete or accurate description of the current state of the environment and thus the agent must be able to generalize its knowledge in the input-output mapping so that the reinforcement predictions may remain accurate even under uncertainty.

Given the problem specifics, Kaelbling et al. [27] introduce formally justified methods of reinforcement learning and ad-hoc methods. Unfortunately, most demonstrated techniques suffer from scaling problems when applied to the more complicated situations (in particular when the environment is not stable), also, algorithms used for calculation in these techniques can be relatively inefficient. The most commonly used methods in the field of Artificial Intelligence are thus the ad-hoc methods, that are more easily implemented to deal with generalization problems and delayed rewards.

7.2.2 A Mathematical Formulation of Optimality in RL

From the reinforcement learning model previously described and a discounted infinite horizon optimality model, the problem of reinforcement learning can be formulated mathematically in a Markovian decision process perspective as follows [27].

The optimal value of a state s is the maximum over all action selection policies ψ of the expected discounted sum of rewards over all stochastic transitions (with γ the discount factor and r_t the reward at time t):

$$V^*(s) \overset{df}{=} \max_{\psi}[E(\sum_{t=0}^{\infty} \gamma^t r_t)]$$

It can be shown that this value is unique and is a solution of the equation:

$$V^*(s) = \max_{a \in A}[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')]$$

where $T(s,a,s')$ is the probability of making a transition from state s to state s' through action a. From the previous definition and the formula for V^*, an optimal decision policy $\psi^* : S \longrightarrow A$ is a policy that finds an action a such that $R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s') = V^*(s)$ in every given state s. The objective of a reinforcement learning algorithm is thus to find a policy ψ that minimizes the value $\sum_{s \in S} |\psi^*(s) - \psi(s)|$.

Finding an exact solution for ψ^* and V^* is possible by using dynamic programming methods, when T and R are known, the problem faced by reinforcement learning methods is to find a solution when this knowledge is not directly available, but must be sought in the environment through trial and error.

7.2.3 Q-Learning

I present here a well-known algorithm developed by Watkins [67, 68], that can be proved to converge to an optimal solution under certain conditions. This algorithm forms the basis of the reinforcement learning algorithm used in the XCS classifier system. In Q-Learning, a state value is defined for a given policy ψ:

$$V^{\psi}(s) = R(s,\psi(s)) + \gamma \sum_{s' \in S} T(s,\psi(s),s')V^{\psi}(s')]$$

and then a Q value is defined for this policy and every pair (s, a):

$$Q^{\psi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^{\psi}(s')$$

indicating the expected discounted reward for executing action a at state s and pursuing the same policy thereafter. For this definition, if ψ^* is an optimal policy, then $V^*(s) = \max_{a \in A}[Q^*(s,a)]$, where $\max_{a \in A}[Q^*(s,a)] \overset{df}{=} \max_{a \in A}[Q^{\psi^*}(s,a)]$ and:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a' \in A}[Q^*(s',a')]$$

If this Q^* value can be learned, then an optimal policy can be built from it by finding one action that maximizes $Q^*(s,a)$ for each state s.

The Q-Learning algorithm estimates this optimal Q value by building a table of randomly initialized Q values for all state-action pairs and updating these values with a Widrow-Hoff delta learning rule. The delta rule adjusts a parameter x towards an estimate of its target value y by replacing x with $x + \beta(y - x)$, $(0 < \beta \leq 1)$, which is simply written $x \xleftarrow{\beta} y$. The value β being the learning rate. It is clear that when y is stationary, this forms a sequence of x values that converge to y. In the algorithm, the delta rule is expressed as:

$$Q(s,a) \xleftarrow{\beta} r + \gamma \max_{a' \in A}[Q(s',a')]$$

With r the actual reinforcement received for executing action a in state s. Each time a table state-action position is chosen to act, this rule is applied to the position to update its value. Over time, the table positions are thus alternatively updated and it can be shown that if all positions are updated regularly enough as time goes on and the learning rate is appropriately adapted at each step, this tabular representation of $Q(s,a)$ will tend to $Q^*(s,a)$.

7.3 Genetic Algorithms

The second component of a learning classifier system is a rule discovery system that is implemented as a genetic algorithm (GA). A genetic algorithm is a search and optimization algorithm based on the Darwinian principles of natural selection observed in nature, but applied to a set of digital genotypes. These digital genotypes are then interpreted as the parameters that can be operated upon for solving the optimization problem at hand. Traditionally, a genetic algorithm is used as a phylogenetic adaptation mechanism that operates on a population of potential solutions to a problem and is based on a measure of the quality (called fitness) of each individual's solution. Individuals are identified by an artificial genome that describes their solution to the problem. The search for an optimal solution works by randomly selecting individuals with a bias towards the fittest and propagating their genetic material in new individuals, possibly through genetic operators such as crossover and mutation. This process usually leads to an increasing average fitness in a population since fitter individuals are more likely to be selected for reproduction.

7.3.1 Implementation of a GA

In many genetic algorithms, an individual is represented by a bitstring (i.e. a string of zeroes and ones) in the set B_l of bitstrings of a specified length l, this string is both the genome of the individual and a coding of the solution that it provides to the problem under study. The algorithm operates on a population $P_t = \{s_i(t) \mid s_i(t) \in B_l,\ i \in \{1,..,n\}\}$ of such individuals at every time step t, the population having a predefined number n of individuals. A mapping $\phi : B_l \longrightarrow \mathbb{R}$ has to be provided that defines the fitness of individuals in order to provide a selection criterion for the algorithm. The search process then iteratively selects individuals with probability proportional to their fitness and generates new individuals from their genome, thereby forming new populations of individuals. There are a few different methods that can be used to produce population updates either as in the original genetic algorithm that was described as selecting individuals from a population P_t at time t and generating n new individuals to form the next population P_{t+1}, or as in the variant of steady-state populations used in the XCS where, at each step, a few (one or two) new individuals are created from the current population's fitter individuals and replace unfit individuals of the current population. The three genetic operators used to produce new individuals are:

- simple replication: the selected individual is duplicated;
- mutation: the various sites in a duplicated individual's code are swapped to the opposite bit with probability μ;
- crossover: two individuals are selected and one or more random positions in their genome are chosen randomly as crossover points. Two new individuals are formed by alternating pieces of genetic code from the two selected individuals, the lengths of these pieces being delimited by the crossover points chosen.

The sequence of populations generated by the algorithm explores the solution space of the problem by accumulating individuals in regions of high fitness value and converging to the (or one of the) global maximums of this fitness landscape.

The search procedure provided by a genetic algorithm is, in most cases, provably better than a random search in the solution space of a problem, although for a large search space the procedure can be slow. The convergence of the algorithm has been proved in the Schemata Theorem [20] by studying generalizations of bitstrings called schemata that represent families of individual bitstrings. Schemata are generalizations of bitstrings and are identical to the classifier conditions used by the XCS system that I introduce in the next section.

7.4 XCS Design

The XCS classifier system is an improvement on the original design of classifier systems that was presented by S. W. Wilson in his 1995 article *Classifier Fitness Based on Accuracy*; it promotes a different approach to the reinforcement learning / genetic algorithm relation in the system adaptation process that allows better generalization of knowledge stored in the form of classifiers. Since much of the relevant work in the field of LCS is now based on this system design, I introduce it here without presenting the original designs of classifier systems. The following subsections describe the elements used in a cycle of the classifier system life. The steps in a cycle are applied repeatedly until the system is stopped by a user and consist in:

1. detecting an environment message and forming a set of classifiers that match the message
2. forming action sets in the match set and selecting an action
3. applying the reinforcement algorithm to classifiers
4. applying the genetic algorithm to the classifier population

Along such a cycle, the learning / genetic algorithm is focussed on the evaluation and use of the three following parameters:

- *prediction* $\pi(\chi)$ of a classifier χ: the estimate of the reward gained by the system when this classifier is matched and its action is selected. This parameter is used by the action selection mechanism;
- predictive *error* $\varepsilon(\chi)$ of a classifier χ: an estimate of the difference between prediction value and actual reward obtained by the system when this classifier is matched and its action selected. This parameter is used to calculate the accuracy of the classifier;
- classifier accuracy, used to update *fitness* $\phi(\chi)$: a function of classifier error that attempts to estimate how good a classifier is at guessing the reward the system would receive if its action was selected.

This parameter is used by the genetic algorithm component to select classifiers and by the action selection mechanism to weigh prediction values of action sets.

7.4.1 Conditions, Messages and the Matching Process

When dealing with classifier systems, sensory stimuli are converted to messages which are conventionally formed of a bitstring of specified length. A population $\mathcal{C}$ of classifiers forms the system, each classifier χ consisting in a condition string and an action string (c_χ,a_χ). The condition string is of the same length as messages and holds three different types of elements: zeroes, ones or wildcard characters. I will therefore refer to these characters as trits and the strings involved as tritstrings. When the matching process is invoked, it uses the condition strings of the classifiers in the classifier population to test whether a match has occurred by comparing successive bits in the message with trits from the classifier. A position is matched in both strings when either the message string holds a one and the condition string a one or a wildcard character, or the message string holds a zero and the condition string a zero or a wildcard character. The wildcard character thereby acts as a don't care symbol that I will hereafter note #. A classifier matches a message if all positions of the classifier condition match corresponding positions in the message. For example, the message 0010110111 is matched by the condition 0##011#11# since every specified bit in the condition is the same as that of the message and we "don't care" about the other positions. A match set is created from the subset of classifiers that match the current message before proceeding to the action selection stage.

7.4.2 Action Selection

The action string in a classifier is a bitstring that codes the associated action identifier of that classifier and has a length depending on the number of possible actions the agent may execute. Once a match set has been formed, the system decides among the possible actions which one it will execute. To this end, a prediction value is stored by each classifier, giving an estimate of the reward the classifier usually receives when it is matched and its action is chosen, along with an estimate of the accuracy of this prediction (these parameters are calculated by the reinforcement learning mechanism which we will study in the next subsection). For each action $a \in A$, the set $\mathcal{A}(a)$ of classifiers in the match set that advocate this action is formed. A prediction array p is then formed by calculating a prediction value of each of these sets. The prediction value of such a set is the average of the predictions of the classifiers in the set, weighted by their accuracy:

$$p = \{p(a)\}_{a\in A} \text{ , with } p(a) = \frac{\sum_{\chi\in\mathcal{A}(a)} \pi(\chi)\phi(\chi)}{\sum_{\chi\in\mathcal{A}(a)} \phi(\chi)}$$

One of these action sets is selected based on its prediction and named the current action set $\mathcal{A}$. For example, using deterministic action selection, the action set with highest average prediction is selected. Another way to choose the action would be by using a biased random selection mechanism over the predictions of each set.

7.4.3 Reinforcement Learning Component

The reinforcement learning algorithm used in an XCS is applied to the prediction, error and fitness parameters defined previously. The updates of these parameters are usually applied in the following order: prediction error, prediction, fitness, but other orders also work.

Two types of problems are distinguished when calculating parameter updates, single step problems and multi step problems. Single step problems are problems where reward depends only on the current state-action pair and the transition function maps all pairs to the uniform probability distribution over the state space (i.e. the state of the next step does not depend on the current state and action). A multi step problem is the more general situation, where the state transition function is not constant and where the system must also learn it. Both situations are studied in the experimental chapter.

In a single step problem, the reinforcement is applied to all classifiers of the current action set, using a reinforcement value of $\rho_t = r_t$. In a multi step problem, the reinforcement is applied to the previous step's action set, using a discounted reinforcement value $\rho_t = r_{t-1} + \gamma \max_{a \in A}[p_t(a)]$, the t indicating to which time step the value belongs and $p_t(a)$ being the prediction value of a's action set at time t, as defined in the preceding subsection. For each classifier χ to update, the reinforcement rules are:

- $\pi(\chi) \xleftarrow{\beta} \rho$
- $\varepsilon(\chi) \xleftarrow{\beta} |\rho - \pi(\chi)|$
- $\phi(\chi) \xleftarrow{\beta} \kappa'(\chi)$, where κ' is the relative accuracy of the classifier calculated from its accuracy κ in the following way: if $\varepsilon(\chi)$ is larger than a threshold ε_0, then $\kappa(\chi) = \frac{1}{10}\,\alpha^{(\varepsilon(\chi)-\varepsilon_0)/\varepsilon_0}$, otherwise, $\kappa(\chi) = 1$. The relative accuracy of χ is then $\kappa'(\chi) = \kappa(\chi)/\sum_{\chi' \in \mathcal{A}} \kappa(\chi')$.

In these learning rules, β controls the learning rate, ε_0 is the limit from which a classifier is not considered accurate anymore and α with $0 < \alpha < 1$ is the accuracy value given to a classifier of error $2\varepsilon_0$.

In practice, in XCS, the technique of the "moyenne adaptive modifiée" (MAM) introduced by Venturini [64] is applied for the first $1/\beta$ action cycles of the system, to speed up the initial convergence of the system.

7.4.4 Genetic Algorithm Component

The XCS system is made of a population of classifiers $\mathcal{C}_t \subseteq \mathcal{C}$ that does not usually hold all $\chi \in \mathcal{C}$. Since the action selection mechanism is applied to this population $\mathcal{C}_t$, it is essential that this set hold relevant classifiers for all environment states encountered by the system. The genetic algorithm's role is to discover which classifier is relevant and must be part of the classifier population and which can be safely omitted from this population. The basis on which this is accomplished is described in the analysis of the XCS section.

As was mentioned earlier, the genetic algorithm operates on the classifier population $\mathcal{C}_t$. At every step, the genetic algorithm is applied to the population with a probability $1/\theta$. If it is applied, two individuals are selected in the current action set proportionally to their fitness $\phi(\chi)$ as calculated by the reinforcement learning component. These individuals are then either reproduced with a mutation factor of μ at each of their sites or, with probability ν, they are crossed over at one random position along their condition tritstring or action bitstring. The two new individuals are then inserted in the population and if this population is larger than its predefined maximum size, two unfit classifiers are deleted from the population.

7.5 An Analysis of the XCS

In this section, I give my mathematical understanding of an XCS classifier system and show that under certain restrictive assumptions, the XCS actually implements the Q-Learning algorithm. I will then discuss the full XCS implementation and the importance of its genetic accuracy based component for generalization purposes. But first I start by making a summary of the notations I will use, some of which were previously defined and used.

7.5.1 Notations

Mathematical notations:

- $\mathcal{P}(E)$ is the set of subsets of E
- $\Pi(E)$ is the set of probability distributions over E
- $|E|$ is the number of elements in the set E

Reinforcement learning notations:

- $*$ denotes optimality as in an optimal policy ψ^*
- S is the set of environment states
- A is the set of actions

- T is the state transition function $T : S \times A \longrightarrow \Pi(S)$, and $T(s, a, s')$ denotes the probability of the environment reaching state s' when action a is selected in state s.
- R is the reward function of the environment $R : S \times A \longrightarrow \mathbb{R}$ and r_t the actual reward received at a given time t
- ψ is a policy for choosing actions and ψ^* an optimal policy
- $V^{\psi}(s)$ is the state value of s for the given policy ψ
- $Q^{\psi}(s, a)$ is the Q value of state-action pair (s, a) for policy ψ
- γ, β are the discount factor and the learning rate used in a learning procedure

XCS notations:

- I and O are the input and output sets of the XCS, which I will for now consider as isomorphic to S and A respectively
- $\chi = (c_{\chi}, a_{\chi})$ is a classifier with condition and action
- $\mathcal{C}$ is the set of all possible classifiers, $\mathcal{C}_t \subseteq \mathcal{C}$ the population of classifiers available in the system at time t, $\mathcal{C}' \subset \mathcal{C}$ the set of specific classifiers in $\mathcal{C}$, that is, classifiers whose condition part have no wildcards
- $\pi : \mathcal{C} \longrightarrow \mathbb{R}$ is the prediction function over the set of classifiers
- $\varepsilon : \mathcal{C} \longrightarrow \mathbb{R}$ is the prediction error function over the set of classifiers
- $\phi : \mathcal{C} \longrightarrow \mathbb{R}$ is the fitness function over the set of classifiers
- $\mathcal{M}_t(s) \subseteq \mathcal{C}$ is the match set for state s at time t
- $\mathcal{A}_t(a) \subseteq \mathcal{M}_t(s)$ is the action set for action a at time t

7.5.2 Simplified XCS is Q-Learning

In [15], Dorigo and Bersini present a comparison between the original LCS of Holland and Goldberg; they show that by restricting the LCS to a subsystem called the very simple classifier system with a modification to the learning rule, their system functions in a way that is identical to a Q-Learning algorithm. I will show here that the XCS system can also be considered as a Q-Learning algorithm given some restrictions.

For the XCS to become a Q-Learning implementation, one restriction is necessary, although it is a major one, the removal of the genetic algorithm component of the system. One assumes (enforces) that the population of classifiers present in the system at every time-step consists in only and all the specific classifiers, that is $\mathcal{C}_t = \mathcal{C}', \forall t$, so that these classifiers form a table similar to that used in tabular Q-Learning. This implies that there is no genetic algorithm component and only the prediction values of classifiers need to be learned (accuracy is not needed since action sets hold only one classifier, as we will see). The XCS algorithm then runs in three steps: acquire the environment state s and form a match set $\mathcal{M}(s)$ for this state, evaluate the prediction value of the action sets in $\mathcal{M}(s)$ and select an action, obtain reward and reinforce the selected action set.

Since the classifier population consists in only the specific classifiers, the match set will hold $|A|$ classifiers, one for each action in A, and every action set will hold only one classifier, the classifier whose condition is exactly the current environment state. The prediction value of these action sets will thus be the prediction value $\pi(\chi)$ of their only classifier (accuracies simplify away in the weighted sum calculation) and action selection as well as reinforcement can be considered to operate on the classifiers individually. Note also that we have an isomorphism between the population of classifiers and the set of state-action pairs: $S \times A \cong \mathcal{C}'$, so that each classifier actually represents a state-action pair $(s, a) = (c_\chi, a_\chi)$.

Remembering that in Q-Learning, the Q value of an optimal policy is estimated by the learning rule:

$$Q(s,a) \xleftarrow{\beta} r + \gamma \max_{a' \in A}[Q(s', a')]$$

We find that in XCS, the learning rule for classifier prediction corresponds to the Q-Learning rule since it is precisely:

$$\begin{aligned} \pi_\chi \xleftarrow{\beta} \rho &= r_{t-1} + \gamma \max_{a' \in A}[p_t(a')] \\ &= r_{t-1} + \gamma \max_{\chi' \in \mathcal{M}(s')}[\pi(\chi')] \\ &= r_{t-1} + \gamma \max_{\chi' \in \mathcal{M}(s')}[\pi((c_{\chi'}, a_{\chi'}))] \end{aligned}$$

And so, the prediction value of classifiers learned in this form of XCS is the same as the values found in the table formed by the Q-Learning algorithm, $\pi^*(\chi) = Q^*(c_\chi, a_\chi)$. Following the hypotheses necessary for the Q-Learning convergence theorem, this simplified XCS thus learns an optimal policy as long as every classifier prediction is updated sufficiently frequently over time.

7.5.3 Generalization in XCS

As already mentioned for the Q-Learning algorithm, the main problem encountered in the simple XCS system is that maintaining a full population of specific classifiers is impractical for most problems due to the size of the $S \times A$ space. This is precisely the object addressed by the discovery component (genetic algorithm) of an accuracy based classifier system. To achieve a reduction of the size of the space needed to maintain classifier predictions, general classifiers are allowed in a classifier population. These general classifiers, holding wildcard symbols in their condition part, can be said to subsume a family of more specific classifiers. With subsumption defined by the relation $\succ$:

- $c \succ c' \overset{df}{\Leftrightarrow}$ positions in c' are matched by the corresponding positions in c, or both positions hold wildcards
- $\chi \succ \chi' \overset{df}{\Leftrightarrow} a_\chi = a_{\chi'}$ and $c_\chi \succ c_{\chi'}$

Since general classifiers represent a family of classifiers (in the same way as schemata represent families of individuals in a genetic algorithm), as long as all classifiers of the family have the same predictive value, the whole family can be replaced by just one (general) classifier. Regularities in the environment are thus compressed as the information of a single classifier.

To observe what happens to the action selection mechanism when generalization is used, it is necessary to see that for a general classifier χ, the prediction is the average expected prediction of the classifiers it subsumes:

$$\begin{aligned}
\pi(\chi) &= \sum_{\omega \in \Omega} \pi(\omega)/|\Omega|, \quad where\ \Omega = \{\omega \in \mathcal{C} | \chi \succ \omega\} \\
&= \sum_{\omega \in \Omega} \pi(\omega)/|\Omega|, \quad where\ \Omega = \{\omega \in \mathcal{C}' | \chi \succ \omega\}
\end{aligned}$$

For specific classifiers χ, the relation still holds since $\Omega = \{\chi\}$ and one can easily convince oneself of the second equality by listing the family of classifiers a general classifier subsumes and writing down their respective prediction values. Now, the action selection mechanism of the classifier system operates on action sets $\mathcal{A}(a)$ that will contain one or more classifiers, both general and specific, by evaluating the mean prediction value of classifiers in $\mathcal{A}(a)$, weighted by their accuracy and it is expected that the action finally chosen is that which would have been chosen in the simple XCS case. It could seem unnecessary to use the accuracy-based selection, thereby simplifying action selection, but by observing the following simple example, the importance of accuracy in generalization becomes clear.

Action Selection in a Sample Classifier without Accuracy

The immediate generalization of action set prediction value from the simple classifier example would be to take $p(a)$ as the mean of the prediction values of classifiers in the set $\mathcal{A}(a)$, that is, $p(a) = \sum_{\chi \in \mathcal{A}(a)} \pi(\chi)/|\mathcal{A}(a)|$. Unfortunately, this will usually not give satisfactory results because the

χ	0	1	χ	0	1	χ	0	1
00	1	0	#0	$\frac{1}{2}$	3	##	$\frac{1}{2}$	$\frac{3}{2}$
01	1	0	#1	$\frac{1}{2}$	0			
10	0	6	0#	1	0			
11	0	0	1#	0	3			

Figure 7.3: Generalization of π values, a prediction landscape.

regularities in the environment (reward landscape) will most often not be compatible with the generalization mechanism of the classifier system.

Suppose that the state space is $S = \{00, 01, 10, 11\}$ and the action space $A = \{0, 1\}$. The current state of the environment is detected as 00. If the current classifier population is made of all possible classifiers, match set and action sets will be given by:

$$\mathcal{M}(00) = \{(00,0), (\#0,0), (0\#,0), (\#\#,0), (00,1), (\#0,1), (0\#,1), (\#\#,1)\}$$

$$\mathcal{A}(0) = \{(00,0), (\#0,0), (0\#,0), (\#\#,0)\}$$
$$\mathcal{A}(1) = \{(00,1), (\#0,1), (0\#,1), (\#\#,1)\}$$

If the prediction landscape is as illustrated on figure 7.3, we can evaluate the prediction values of both action sets.

$$\begin{aligned}
p(0) &= \frac{1}{4}[\pi(00,0) + \pi(0\#,0) + \pi(\#0,0) + \pi(\#\#,0)] \\
&= \frac{1}{4}[\pi(00,0) + \frac{1}{2}(\pi(00,0) + \pi(01,0)) + \ldots] = \frac{3}{4} \\
p(1) &= \frac{1}{4}[\pi(00,1) + \pi(0\#,1) + \pi(\#0,1) + \pi(\#\#,1)] \\
&= \frac{1}{4}[\pi(00,1) + \frac{1}{2}(\pi(00,1) + \pi(01,1)) + \ldots] = \frac{9}{8}
\end{aligned}$$

And so, even with full knowledge of the predictive values of all classifiers, the selected action is not the most beneficial one. Clearly, from the prediction values given, the action that should be selected if we were relying on specific classifiers is the action 0, but here, using deterministic action selection, the selected action will be 1 because of the high prediction value of classifier $(10, 1)$ that is reflected in the prediction value of classifier $(\#0, 1)$ and enters the prediction value calculation of action set $\mathcal{A}(1)$. Therefore, with generalization comes the need of an accuracy criterion that allows the action selection mechanism to distinguish between accurate generalizations and inaccurate generalizations.

7.5.4 The Role of Accuracy in XCS

As shown in the above example, when a general classifier subsumes a family of classifiers that have contradicting prediction values, the action selection mechanism of the XCS might drift away from the mechanism applied in Q-Learning (which will intuitively lead to bad performance). The factor that influences this drift is that general classifiers will obtain various rewards in any non-trivial environment. Given that their prediction value is the expectation of reward, what becomes important here is the variance in rewards received, intuitively, by how much these rewards differ over the set of state-action pairs they match.

In the simple classifier system with only specialized classifiers, this variance will be zero for a single-step environment, where a state-action pair is always equally rewarded. This variance will remain small with delayed rewards as long as the discount factor used is small and the environment sufficiently regular. This remains true when considering general classifiers whose subsumed family of specialized classifiers has consistent predictions. In essence, there are "good" accurate general classifiers (marked by small predictive variance) and "bad" inaccurate general classifiers (characterized by a high predictive variance) and if the XCS system is to generalize efficiently, it has to be able to distinguish between these accurate and inaccurate classifiers. The role of the prediction error and fitness functions in the reinforcement learning component of the XCS is to learn this distinction and provide a criterion to both exclude some general classifiers from the population and minimize the effects of existing inaccurate classifiers on action selection. The actual search for accurate classifiers is handled by the genetic algorithm component which is applied to the classifier population.

Since the learning rule for the ε function updates $\varepsilon_t(\chi)$ with $\varepsilon_{t-1}(\chi) + \beta(|\rho_t - \pi_t(\chi)| - \varepsilon_{t-1}(\chi))$, $\varepsilon(\chi)$ is an estimate of the average difference in the prediction of χ and the rewards received when applying χ. ε thus has a similar role to that played by variance in statistics. If the GA was operating on a population of classifiers for which we had full information about prediction values and prediction errors, and fitness was taken as the inverse function of prediction error, the classifier population would tend to a population made of an ever greater proportion of accurate classifiers, due to the schemata theorem for genetic algorithms. In this more general situation, these values must simultaneously be learned by exploration in the environment and so, due to incomplete information, a fitness function must be estimated from the prediction error by the reinforcement learning component of the system, allowing an error tolerance to be introduced in the selection of "good" and "bad" classifiers.

Overall, the XCS system uses two cooperating algorithms to provide the action-selection mechanism with the best information acquired in the environment at the time a decision must be made. The RL component attempts to derive information about the utility of making a particular decision and the GA selects the classifiers that accurately describe the problem domain in which this decision process occurs. The most interesting result remaining to discover is now a convergence result for the joint RL and GA. It seems that although such a result is difficult to obtain, it is not impossible with the right constraining assumptions. Some typical assumptions I believe necessary would be based on: population size requirements, rate of application of the genetic algorithm, number of explorations by the reinforcement algorithm before the selection or deletion of a classifier by the GA. These parameters are all controllable in the classical XCS. There are also some problems that I have not discussed here that can have a great influence on the classifier system, such as the relation between environment states and representation of such states (input function) or the possible reliance of the environment state transition function on hidden parameters. These problems are typical of the current artificial intelligence algorithms and linked to the functional grounding problem that I introduced in the theoretical part of this thesis.

7.6 Test Case: The 6-multiplexer Problem

The multiplexer problems have often been used in conjunction with learning classifier systems to show their adaptation capabilities. These problems are considered interesting because the function that must be learned to solve the problem is irregular but does allow generalizations to be made in its standard representation and this representation is easily used as input to a classifier system. With my implementation of the XCS classifier system, I have been able to reproduce the results that were presented in [70] and [30] and I will give a short presentation of these results here. For a complete presentation, the two preceding documents can be referred to.

7.6.1 Multiplexer Problems

A multiplexer is a logical function m that maps a tuple of $n+2^n$ boolean values to a boolean result. If one considers the input string of boolean values as an ordered bit string, this function m maps signals from the set $\{0,1\}^{n+2^n}$ to the set $\{0,1\}$. It uses the first n bits in the bit string as an address in the remaining 2^n bits and returns the value at that address as a result, i.e. if the value of the first n bits considered as a number is x, the value of bit x in the second part of the bit string is returned. The figure 7.4 describes this mechanism.

Since the number of possible addresses depends on the n chosen, there are multiplexer problems for each $n \in \mathbb{N}$, that is, 3-multiplexers, 6-multiplexers, 11-multiplexers, etc. Results have been published on the 6, 11 and

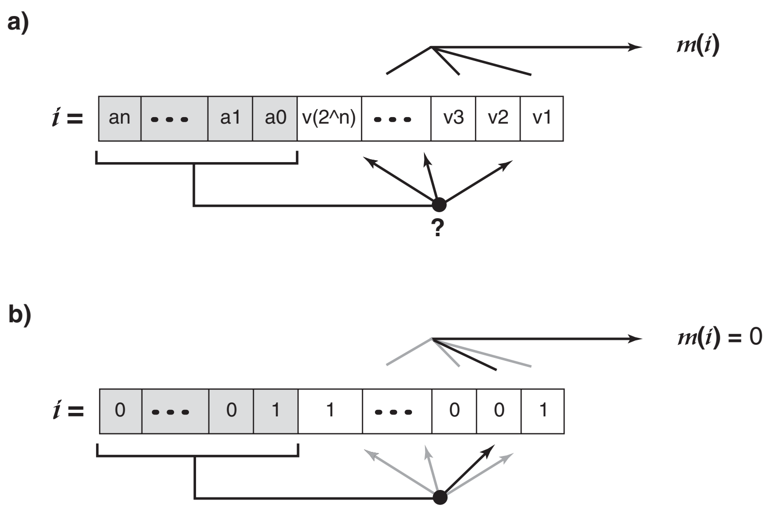

Figure 7.4: Multiplexers, a) general case, b) evaluation of the function.

20 multiplexer problems for the XCS system, but the tuning is usually done on the 6-multiplexer case.

7.6.2 The 6-multiplexer

A 6-multiplexer has an input string consisting in 2 address bits and four value bits. The m function maps each of the 64 possible input strings to a zero or one value. Given that in this representation, each input string has only 3 significant bits: the 2 address bits and the bit at that address, the function landscape has regularities that could be learned by a classifier system. For example, the strings 010010, 011010 or 011110 are all mapped on 1 because they have the same 01**1* pattern. Since the XCS generalizes by using conditions that are patterns of this form, it be able to adapt to the problem by generating classifiers with this type of general conditions.

One can see in table 7.5 that there are 8 (2^3) general patterns that sum up the multiplexer function. Optimally, the classifier system should thus

input	result	input	result
00***0	0	00***1	1
01**0*	0	01**1*	1
10*0**	0	10*1**	1
110***	0	111***	1

Figure 7.5: Regularities of the 6-multiplexer function.

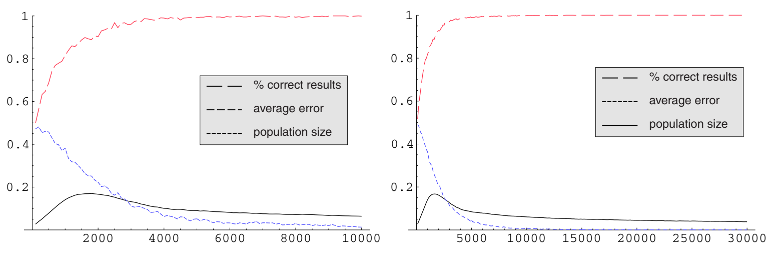

Figure 7.6: 6-multiplexer results over 100 experiments (10k and 30k steps).

explore the space of classifiers until it discovers these 8 general conditions and then limit its population to the classifiers that map these conditions to the correct actions. Unfortunately, these general patterns are not the only ones, since *0*1*1 or 1*00** are other equivalent ways of expressing regularities in the function graph and this will affect an XCS classifier system.

7.6.3 Applying XCS to the 6-multiplexer

In order to learn the 6-multiplexer problem, a reward function must be provided to the system for its decisions. The immediate choice, that is used here, is to reward the system with a value when it correctly maps an input to an output and provide no reward for incorrect answers. With this reward scheme, the classifier must discover classifiers that accurately predict the reward, and use these classifiers take decisions about the result of an input. Since in this problem, the regularities of the reward function are symmetric with respect to the two possible action choices (given an input that is rewarded when action 1 is chosen implies no reward for the choice of action 0 in the same situation), the accurate population of classifiers includes both correct general classifiers and their incorrect counterparts. The expected optimal population for the XCS will thus include at least 16 general classifiers.

Experimenting with the classifier system that I have implemented provides the learning curves illustrated on figure 7.6. In this illustration, the curves plotted represent the averaged results of one hundred different experiments. In each experiment, every decision step was alternated with an exploration step. On exploration, an input is used by the system to test its answer. Reward is distributed to the classifier for this answer. On a decision step (exploitation), the result given by the system is used for the plot data, but no reward is distributed and no reinforcement or discovery process takes place in the system. The dashed line plot on the figure represents the percentage of correct answers returned by the system in the last fifty decision steps. The dotted line represents the overall error in prediction over the last fifty decision steps and the continuous curve is the number of different types of classifiers existing in the population (the value is divided by one thousand for scaling purposes).

The results obtained here are equivalent to those presented in [70, 30]. One observes that the predictions of the system become almost perfect after 2000 exploration cycles (4000 steps), the error prediction simultaneously decreases, with a slight delay. The system is initialized without any classifiers at first and one sees that while the population has not reached its maximum number of classifiers (which happens around step 1200), the new classifiers that were generated by the genetic algorithm to fill in the population are very diverse. Maximal diversity is reached around step 1900 with about 180 different types of classifiers. This variety then decreases until it reaches the number of 40-60 different types in the process of elimination of inaccurate classifiers.

7.7 Conclusions

In order to experiment the theoretical ideas that I presented in the first part of this thesis, I have chosen the XCS algorithm. With this chapter, I have introduced the basic theory of this type of algorithm, including reinforcement learning and genetic algorithms. Based on this theory, I have explained the practical considerations necessary to implement the XCS system according to Wilson's design. In the next step, I have taken a personal stance towards XCS in order to give a better understanding the XCS adaptation principles. This understanding will then help in the next chapter when we come to studying the structural and dynamic properties of such systems. Three essential points are made here. Firstly, since the simple version of XCS is equivalent to Q-Learning, we can reasonably expect some of the good convergence properties of Q-Learning to be inherited by XCS systems. Secondly, the advantage of an XCS system with respect to Q-Learning is that it can generalise. A property that is equivalent to the ability of compressing information and allows an adaptive algorithm to acquire information for overall problem regularities. Thirdly, the accuracy parameter of the XCS system gives it the ability to distinguish appropriate / relevent generalisations of problem spaces from the inadequate / irrelevent ones.

A final experiment is led to reproduce the results of Wilson and others in the case of multiplexers, so as to show that the system I have implemented is identical to the previously implemented systems, and that results obtained here can be compared with other results obtained on XCS classifier systems.

Chapter 8

Experimentation

8.1 Multiplexer Representation Test

In the preceding chapter, I have shown how the XCS system learns the 6-multiplexer problem. I want to show here that this learning ability is a property that holds for this problem because of the regularities that are compatible with the XCS system's representation and generalization functionality (that I associate with information compression). In the general case of a problem that is not well understood and that must be solved, applying a particular algorithm will help to understand the functionality and limits of the algorithm, as well as the properties of the problem. I intend to show by a trivial example that the representation of a problem plays a fundamental role for adaptive behaviors and will relate this to the question of information compression.

8.1.1 The Multiplexer Description

I have described previously why the multiplexer is used as a test case for the XCS system. The multiplexer function is a boolean function, which is naturally suited for learning by the use of classifiers; its standard description is based on the use of an address set indexing a value set that allows generalization to occur on individual bits as patterns of zeroes and ones with wildcards. The term "description" that I use here should be understood as the symbolic representation of input signals. The internal symbols used by classifier systems as messages are strings of bits and the natural symbol associated with an input signal is the bit string of the signal itself, but other problems do not necessarily have such a "natural" mapping of inputs onto internal symbols.

For the multiplexer problems, the natural description is based on the use that is made of the function by humans, but actually, the description depends on perception of the problem by the system. And the actual problem to solve is of distinguishing inputs into two categories, those that are mapped to a one and those that are mapped to a zero. The regularities of the function are simply regularities of the description of this function. In the multiplexer case, we use a description that is known to be expressive simply because it is the one we use as humans to understand the problem and associate an additional meaning to the order of the bits and their signification. This proves to be a good description for a classifier system to learn.

In the general case of a misunderstood problem, there is no guarantee that the classifier system would obtain descriptions that are compatible with its internal adaptation mechanisms. Imagine, for example, that the multiplexer problem is described as a random permutation of the input signals and that the XCS must learn this new mapping. There will be no reason for it to be able to make good generalizations if these permutations do not offer some degree of compatible regularities (i.e. regularities over single bits in the description).

8.1.2 Regularities in the Description Space

By looking more attentively at the XCS classifier system applied to this problem, there being only two levels of reward implied by the input messages, the information content of an input message is only one bit of information, distinguishing the reward options available to the system when it chooses an action. The role of the algorithm is thus to extract this information and use it to make a decision about the amount of reward it has been programmed to choose. To this aim, the generalization mechanism of the XCS system finds a minimum of sixteen general classifiers that resume this decision policy, the classifiers shown on table 8.1. For each general condition, the classifiers that advocate actions zero or one are accurate. Obviously, this representation of the problem used is suboptimal, since the only information content of an input string is one bit, but it still is very close to an optimal representation.

description	action	description	action
00***0	0 or 1	00***1	0 or 1
01**0*	0 or 1	01**1*	0 or 1
10*0**	0 or 1	10*1**	0 or 1
110***	0 or 1	111***	0 or 1

Figure 8.1: Complete family of 16 accurate maximally general classifiers.

To illustrate the generalization capability of the XCS system, I have used various permutations of the natural representation of the problem and applied the classifier system to these new problems. In order to give a quantitative measure of the distance between the original description and the permutation over this basic description, five types of permutations have been used. The permutations are of order 2 to 6, meaning that in permutations of order n, n interactions between bits of the original description can be introduced (interactions in a sense that can be related to the epistatic interactions as studied by Kaufmann in [28]). The permutations are constructed in the following way: n bits of the original representation are swapped, then a random permutation over the n-dimensional space of the selected bits is operated. As a result, the larger the n, the further the descriptions provided to the classifier system are from the original. When $n = 6$, any function of $\{0,1\}^6 \longrightarrow \{0,1\}$ can be reached. Additionally, the permutation leading to an optimal representation for the classifier system has been used as a reference for the learning curves of the system. This permutation can be obtained by mapping subspaces of the input space onto a permuted description as described on table 8.2. The inputs that match input patterns on

input	description	input	description
00***0	00***0	00***1	00***1
01**0*	01***0	01**1*	01***1
10*0**	10***0	10*1**	10***1
110***	11***0	111***	11***1

Figure 8.2: An optimal permutation of the input signal.

the table are mapped to the description patterns in the following column of the table. The unspecified bits can be mapped to any other unspecified bit position. For this permutation, all the information content of the input that must be used to obtain a reward or not is contained in the last bit of the description strings, so that the classifier system can build a complete and accurate family of classifiers with only four general classifiers in its population, those with conditions *****0 or *****1 and actions 0 or 1. Other descriptions are optimal in this sense, but due to the genetic search algorithm used to find new classifiers, classifiers with longer defining lengths are slightly suboptimal for crossover operations (defining length is the distance between two specified positions in a pattern, for example the pattern 01*00* has a defining length of 4 and the pattern *0100* a defining length of 3).

For the experiments, one must be aware of the fact that the space of all possible inputs to a function over $\{0,1\}^6$ has $2^6 = 64$ different elements. Learning to map all the inputs correctly for a reward function with just two levels requires twice that number of specific classifiers or 128, when no generalization is used (cf. tabular Q-Learning). For this problem, there are $3^6 * 2 = 1458$ different classifiers (including the general classifiers) available to the system and, depending on the regularities in the function to be learned, the minimal amount of classifiers needed to accurately learn the function range between 128 in the worst case and 4 in the best case. Therefore, whichever function the XCS must learn, if it is allowed to have a population of more than 128 classifiers, it could learn the function by using only specific classifiers. On the other hand, if less that 128 classifiers are allowed in its population, the function must have regularities that are compatible with the generalization mechanism of the classifiers system, if the system is to learn it correctly.

8.1.3 Result Plots

In the figures 8.3 to 8.5, I illustrate the learning curves obtained for each order of permutation, when the classifier population size allowed in the system is 400. The various curves show that with more interactions in between the bits of the problem description, the learning ability of the classifier system is reduced. In the optimal case where only one bit sums up all the information about the function, the classifier system quickly reaches 100% performance. The population diversity then slowly reduces down to 25 individuals as the unnecessary classifiers are eliminated by the genetic algorithm. The normal case is then only slightly better than the learning of functions with order two permutations from the normal description and we can see that each increased permutation level introduces a degradation of the learning ability of the system. Although this degradation is apparent, the system still exhibits a good learning ability for these problems and the prediction curves are still on a slight increase at the 10,000th step. On the population curves, one sees that in each case, the classifier system first generates a wide variety of classifiers, until the 400 individuals limit is reached at 1300 steps. These individuals are then evaluated and after the initial evaluation phase, the inaccurate classifiers start to be eliminated from the population and the overall population diversity gradually reduces to a minimal amount of distinct types of classifiers. Except for the optimal description case, the increase in population continues until about the 2000th step, where it reaches its peak (the overall population is now 400). In the optimal case, the elimination process is intense enough to compensate the increase in population very early in the experiment.

In the figures 8.6 to 8.8, I illustrate the learning curves obtained for each order of permutation, when the classifier population size allowed in the system is 100. The number of steps considered here are 15,000. The optimal problem description produces a function that can still be learned perfectly by the system after about 3000 steps, allowing the population to be compressed to about 12 individual types. In the other experiment, one can see that having only 100 individuals in the population is problematic, although learning still occurs.

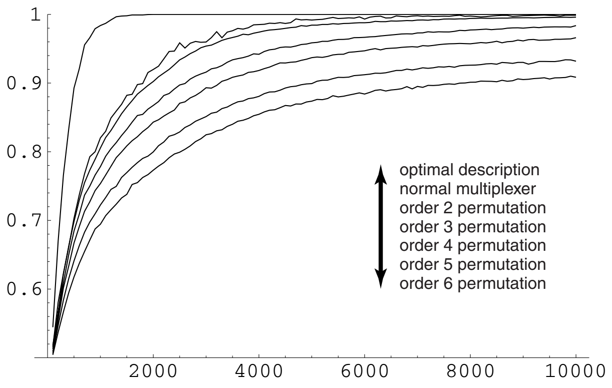

Figure 8.3: Prediction results of permutation experiments (pop. 400).

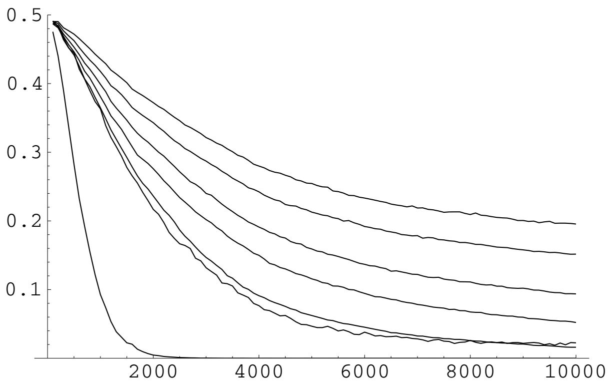

Figure 8.4: Error results of permutation experiments (pop. 400).

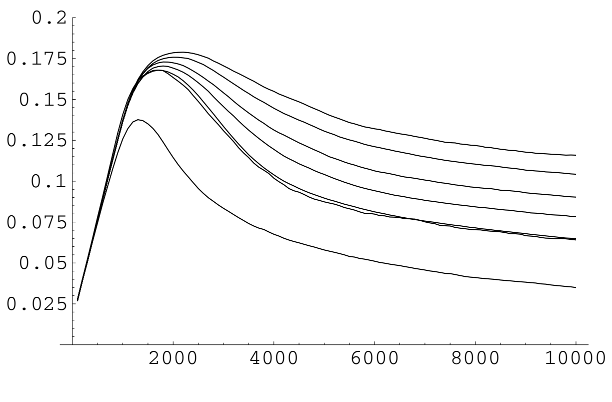

Figure 8.5: Population results of permutation experiments (pop. 400).

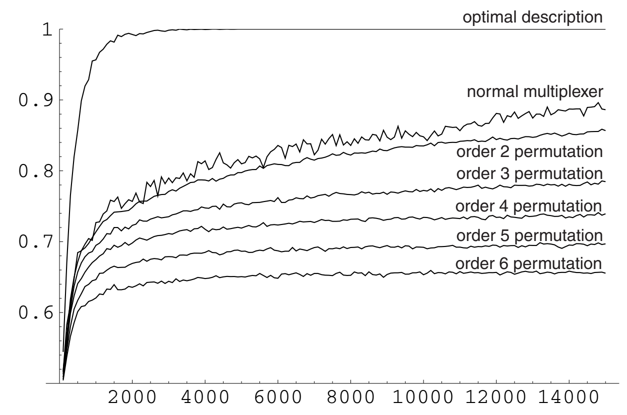

Figure 8.6: Prediction results of permutation experiments (pop. 100).

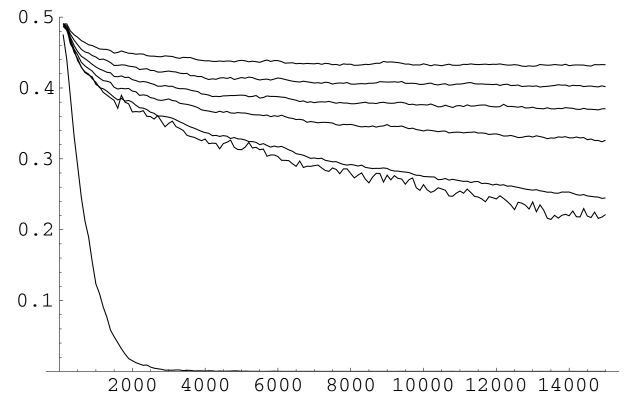

Figure 8.7: Error results of permutation experiments (pop. 100).

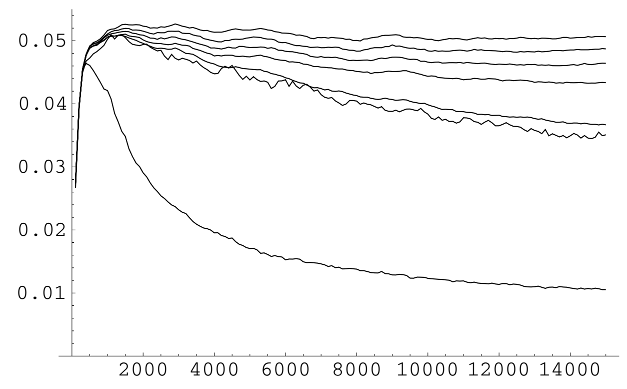

Figure 8.8: Population results of permutation experiments (pop. 100).

To explain this phenomenon, two reasons can be postulated. First of all, the classifier system cannot rely on a complete population of specific classifiers, this is a problem that will always occur in more complex experiments and must be coped with. The second reason is that although generalization would be possible, the system would need more working space to hold various classifiers as a genetic resource pool in addition to the accurate classifiers useful for action selection. We see for example that the number of different types of classifiers initially present in the system is around 50, that is, the genetic algorithm produces an average number of 50 different classifiers in a population of 100 before the accuracy of these classifiers starts becoming differentiated. In the different permutation experiments, this variety is barely sufficient to differentiate the accuracies of the classifiers in order for the system to start eliminating classifiers. With the normal multiplexer problem and order two permutation problems, this compression of the population can engage and will eventually produce satisfactory results for learning the function, but with permutations of a higher order, the population compression stabilizes too fast for satisfactory learning of the function. Still it is remarkable that even in the case of completely random functions (order 6 permutations), the system is capable of making correct decisions 60% of the time on average.

8.1.4 Information Content

An optimal permutation that leaves the input information that is relevant to the classifier system in the rightmost bit is shown in table 8.2. Each one of the value bits that is addressed by a particular configuration of the first two bits is simply swapped with the rightmost bit (whatever its value). In this situation, the XCS can learn only the position of this bit and generalize away all the other bits, a situation for which I have plotted the learning curves on figure 8.10. One notices here that the number of steps until the system correctly predicts the choices that must be made is much shorter since the number of general classifiers that accurately describe the problem is reduced to a minimum of four. From the step 1500 onwards, the classifier system always makes the correct decision to maximize its reward and prediction error vanishes completely from step 2000 on. The diversity of classifiers in the population also reduces and stabilizes to 20 different types of classifiers after 15 thousand steps.

The objection that might be raised by this example is that by finding an optimal description of the classifier inputs, one is in fact solving the problem for the system and that there is no point in further experimentation. This is precisely the point I am making, the compression of information in the inputs to provide the system with a description that is compatible with the functionality of the system is the missing feature of this system and this feature is also missing from most other adaptation algorithms that are used

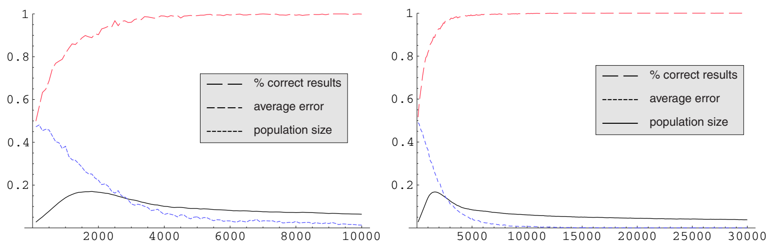

Figure 8.9: Results of the normal multiplexer: 10k and 30k steps.

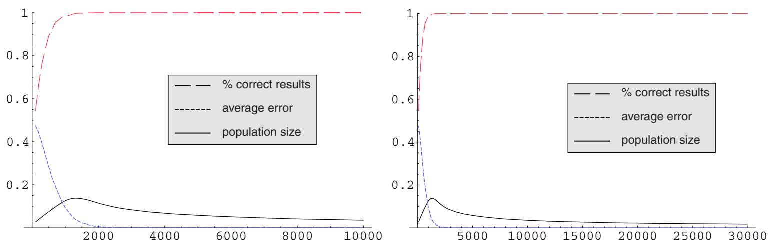

Figure 8.10: Results of the optimal description problem: 10k and 30k steps.

in AI. The regularities of the environment have to be written in the terms of an adaptation algorithm, here of the reward levels. In this problem, the environment that is the space of input strings in standard form has regularities that are not entirely detectable in terms of the classifier system. But to the system, there are only two important situations in the environment: those that provide a reward and those that don't. Usually, the complexity of an environment is much larger that what can be represented in a symbol system, but the situations that are important to the system can be expressed much more simply in the terms of that system. An algorithm that must deal with such environments should be able to extract this information itself in order to learn intelligent behaviors.

8.1.5 Experiment Parameters

The preceding results have been produced by generating a number of different test runs from random XCS system initializations. In the simple experiments, the standard description of the multiplexer problem and the optimal description of the multiplexer problem, 100 test runs were used to generate the averages plotted on the curves for correct results, error in prediction and populations. In the permutation experiments, 100 random permutations were generated for each order of permutation and in each case 20 runs of the XCS system were used to average the data, the overall averages are thus calculated from 2000 test runs.

The number of classifiers allowed in the system were respectively 400 and 50 for the two different sets of experiments. With four hundred individuals, the classifier system has sufficient resources to build a complete representation of the problem space with specific classifiers, although this will not usually happen since the classifier system is driven towards generalization by its internal mechanisms. With a population of 50 classifiers, the resources of the system are not sufficient to cover all the problem space and the system must be able to generalize in order to solve the problem.

The system parameters are the same as those given by Wilson in [70], with his more recent clarifications about implementation issues introduced: (available on the Internet at http://world.std.com/~sw/imp-notes.html)

- rewards distributed for correct or incorrect answers of the system are 1000 or 0 respectively
- the reinforcement parameters are: $\beta = 0.2$, $\gamma = 0.71$, $\varepsilon_0 = 10$, $\alpha = 0.1$, $\tau = 0.1$
- the genetic algorithm based parameters are: $\theta = 25$, $\mu = 0.04$, $\chi = 0.8$, $\delta = 0.1$, $P_{\#} = 0.33$
- initialization parameter for new classifiers were: strength $= 10$, error $= 0$, fitness $= 0.01$, experience $= 0$, covering experience $= 0$ and match set size estimation $= 0$

8.2 EMud Dynamic Test

Even though an XCS classifier system is seriously challenged when having to learn the dynamics in an environment, a simple experiment can be made, that shows the functional ability of the classifier system to adapt to an environment where the causal role of actions chosen is important. It was shown in the preceding experiment that the generalization capabilities of the XCS system depend largely on the coding of the information perceived, but that overall, the system is able to generalize in a satisfactory manner (at least when the information perceived is not too large). In this experiment, I want to show that assuming the perceptions of the environment can be generalized over, the system's ability to adapt to the dynamics of an environment is not influenced very much by the description of this environment.

8.2.1 The Switch Test

The experiment that I have used to show this is of learning a simple switch in the environment. Let me use the following informal terminology for the description of the problem, the system used in the experiment consists in:

- an agent: the (virtual) rat
- an agent: the cheesemaker
- a set of three connected rooms
- pieces of cheese
- rat "traps"

In this system, the rat is the XCS agent that must learn to find and eat pieces of cheese. The three rooms are aligned with a western room, a central room and an eastern room each connected to its neighbor(s) by one exit. The cheese maker is an automatic agent that starts in the western room which is then empty. If the rat appears, he drops a piece of cheese and a "trap", moves to the east, drops a "trap" and moves to the east again, waiting in this eastern room until the rat appears. If the rat enters the room, he drops a piece of cheese and moves to the westernmost room, collecting the "traps" on his way. The cycle then starts over. In the experiment, the "traps" have no role other that to act as markers in the environment.

The rat agent must find a decision policy among four actions: moving east, moving west, eating cheese, being inactive for a turn. The sensory information he is provided with to make decisions consist in a string of five true/false statements: is there an exit to the west, is there an exit to the east, is there a trap here, is there a cheese maker here and is there cheese here. His goal is to accurately and generally associate each of the perception signals to one action. The perceptions thus form a space of 32 different possibilities and the possible classifiers in the system number 128. This problem, although simple, requires the system to break out of the typical looping behavior it is prone to enter when in a dynamic environment, indeed, a classifier system has the tendency to enter two step oscillations in such environments, since usually, one classifier starts having maximal prediction value and accuracy in one room, and then when moving to the next, another classifier will take over to bring it back to the first room, creating a regular two step pattern.

Here, the system must adapt internally to an optimal chain of more that two classifier choices, since the optimal behavior is to eat cheese if some is available, move west when no trap is present and a west exit exists or move east when an east exit is present and a trap is present. This forces a set of a minimum of three general classifiers to be found and maintained in the system. An XCS classifier system has no memory (in the sense of internal memory states), the way this problem must be learned by the system is through the multi-step reinforcement procedure of the algorithm. This multi-step reinforcement algorithm works by rewarding the classifiers that

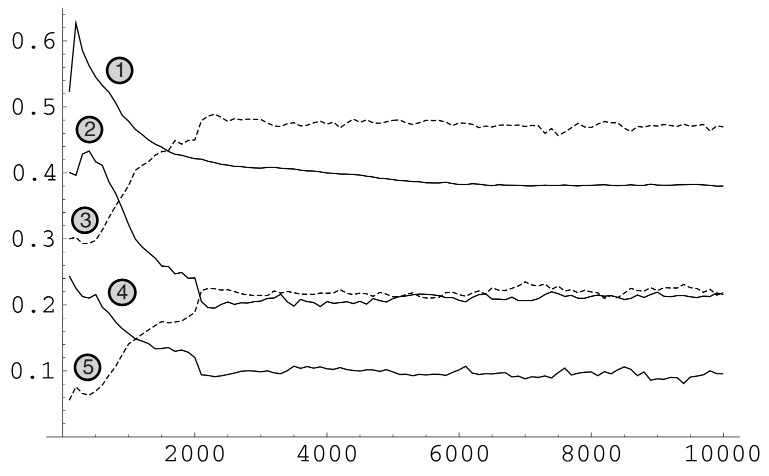

Figure 8.11: Results of the "rat" experiment (averages over 200 cases).

advocated an action on a preceding step with the immediate reward of that step and the discounted predictive value of the current situation. Here, reward was given for eating food and no other action was rewarded. The predictive value of moving or doing nothing is thus only derived from the probability of finding cheese and eating it on the next step, which leads to high levels of error value for the movement or inaction classifiers.

8.2.2 Result Plots

The results plotted here are those obtained by the situation described above, with the same parameter values as in the multiplexer problems. Since the goal of this experiment is not to find a perfect (but maybe unstable solution), no parameter adjustment was used, even though one can obtain almost perfect results by so doing. The result curves are plotted on figure 8.11, with the number of action cycles on the horizontal axis. Curve (1) is the normalized number of distinct types of classifiers in the XCS system, where a maximum population of 200 was allowed ($.5$ means $200 \times .5 = 100$ classifier types). Curves (2) to (5) are then respectively the number of movements executed, the number of pieces of cheese eaten, the failed attempts to act and the inactive steps of the rat over the last hundred action cycles. In order to initialize the system with an exploration phase, a function is used to probabilistically allocate exploration and exploitation for each step. The function used has decreasing values between zero and one in the first two thousand steps and then stabilizes so that an average of one action cycle in twenty is an exploration cycle. In order to satisfy these criteria, the (some-

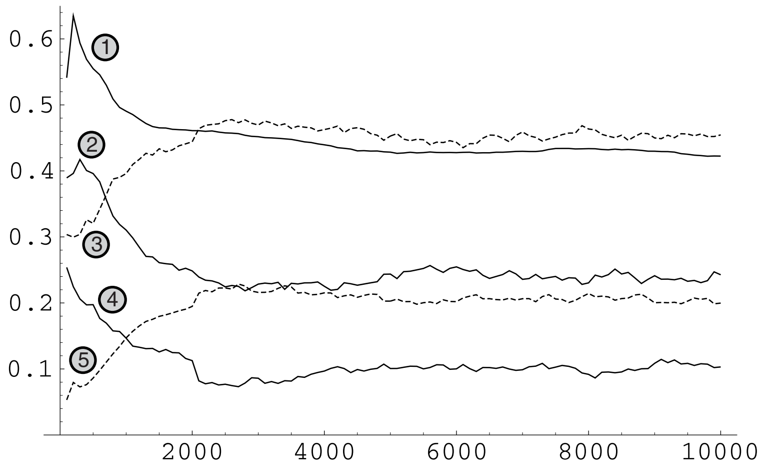

Figure 8.12: Results of the "rat" permutation experiments.

what arbitrary) function I have used is:

$$f(x) \overset{df}{=} \begin{cases} (\frac{x}{500} + \frac{9}{10})^{-1} & \text{if } 0 \leq x < 2000 \\ .05 & \text{otherwise} \end{cases}$$

As can be seen on the graph 8.11, the population diversity in classifier types exhibits a similar behavior as in the multiplexer experiments, sharply rising at first and then reducing as accurate general classifiers are progressively found for the problem. This population stabilizes at around 80 distinct varieties of classifiers which is quite a large amount, given the maximum of 128 possibilities and the fact that only 4 should suffice to sum up the solution to the problem. The curves numbered (3) to (5) then represent the total actions of each type that the system uses to solve the problem (move, eat, do nothing). Since each action is equally likely to occur at the beginning of the experiment, there are about double the amount of moves as eat or "noop" actions. The initial differences reside in the fact that choosing to be inactive never fails, whereas choosing to eat will fail every time there is no food in the current locus. The last curve (4) is the number of failed attempts to do something, such as moving east when there is no east exit or eating when there is no food. Curves (2) to (5) sum up to 1 at each stage.

As illustrated, the system is able to find an equilibrium between the number of moves at each stage and the number of pieces of cheese eaten, the fact that the moves stabilize to about double the amount of pieces of cheese eaten indicate that the algorithm never oscillates between two neighbor locations, moving to the extremities of the environment at each passage. The failed attempts can then be accounted for by the fact that the agent then pushes on, trying to move further until the failed attempts (there is no east exit to move through in the eastern room) bring the predictive value of moving in that direction lower than that of moving in the other direction. Overall, this behavior is a good attempt at modeling the switching function of the problem, without the expressive ability to have a better approximation of the problem solution. With some changes in the parameters of the algorithm, it should be possible to minimize the failed attempts phase of this solution, but I have not investigated the matter in detail. The point that I want to make is that we can see here the way in which the classifier system attempts to express the dynamics of the environment within its own limited expressive capability.

8.2.3 Information in the Problem Dynamics

In the multiplexer experiments, the full information about the problem could be stored in terms of stable classifier predictions, with accuracy indicating whether a classifier was a good description of an environment regularity or not. Here, the information that must be acquired by the system is not static and each classifier does not have a definitive predictive value. Instead, predictive values will change over time for some classifiers, depending on the current situation of the agent: the full information for the problem resides in multiple step evaluations. For an XCS to capture this, it must still store the information in the classifiers, but most of this information is now held by the prediction, error and accuracy values of these classifiers, since the predictions now have a continuous range of possible values that dynamically vary from situation to situation. The characterization of the system's functionality now depends much less on its ability to generalize over the input perceptions of the system, than on its ability to express the dynamics of the problem in the predictive values of the classifiers.

To highlight this, I have reproduced the same experiment, but with other perceptual descriptions of the environment. As in the multiplexer problems, I have coded the perceptual inputs from the environment by using random permutations of the input space. When this is done, the XCS system must generalize its representation of the world in terms of classifiers, but without the same regularities in the inputs. The data for these experiments (averaged over 200 sample random permutations) show that although the diversity of classifiers in the population is affected (it increases), the solution to the problem that is found by the algorithm gives the same results. The global results are shown on figure 8.12, while the comparative results are shown on figures 8.13 and 8.14. Clearly, the perception of the environment affects the classifier system in its operation, but this effect is limited to the representational ability of the system and not the way in which it can learn the dynamics of the environment.

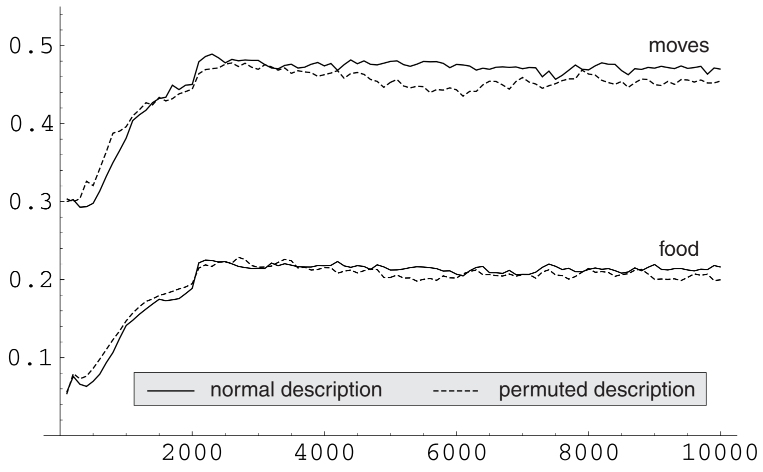

Figure 8.13: Comparative results, moves and eating.

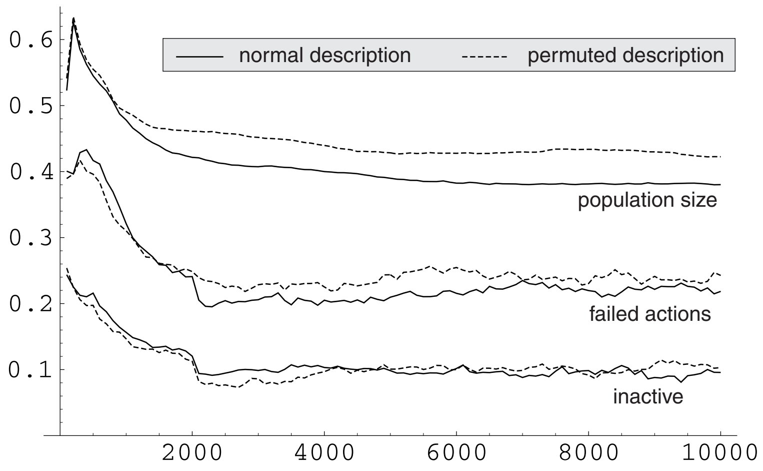

Figure 8.14: Comparative results, population, failed actions and “noops”.

8.3 Discussion

The two experiments described in this chapter show that information compression (in the sense of generalization for a classifier system) is an essential feature of an agent algorithm that must learn in rich environments. Every algorithm developed in the field of AI for such agents is capable of some form of generalization, be it implicit or explicit. What differs from system to system is the internal structure and functions in which this information compression is expressed. It is this language of representation that characterizes an algorithm. As we have seen in the preceding experiments, there is a range of problems that can naturally be addressed by a classifier system, and others that simply are not adapted to such systems. This expressivity factor is fundamental when choosing an algorithm to solve a problem and is already intuitively understood by most scientists, but has not been formalized as I find necessary: when attempting to solve real world problems, a connectionist approach is often chosen for its "continuity" properties and when a modern expert system is implemented a symbolic approach is preferred, but no precise characterization of the relevant properties of each system has been attempted. Since every problem has a distinct type of regularities that must be integrated by agents programmed to solve them, I would like to propose that it is now time for AI researchers to focus on a classification of algorithms, based on their generalization capabilities and internal dynamics.

Chapter 9

Conclusion

The work presented in this thesis originates in a global effort to understand and implement multi-agent systems in the context of open computer networks as a solution for distributed process optimization and management. In the current work, the element of this objective that is addressed is the modeling of a virtual environment that can be mapped over a network and allows agents to be embodied in the network. It is in the course of this realization that philosophical questions about the nature of agents and their relation to an environment were approached. The ensuing study of the symbol grounding problem has then served as a general context for the overall work. The results that can be drawn from the thesis are thus twofold: on the practical side with the realization of an environment model for distributed agent systems and on the conceptual side with an attempt at formulating an operational theory of meaning.

9.1 Practical Aspects

Using a computer game metaphor, the realization of a general environment model was designed and implemented. In this model, agents are situated in places of a virtual physical space that forms a virtual topology. The nodes of the topology may contain both active and passive elements corresponding to processes and data. Virtual physical laws govern the evolution of the environment, as agents move or in act on features of the environment structure.

9.1.1 Design

The environment model is designed as a collection of interconnected worlds in which agents act in the pursuit of their individual tasks. With the locus / exit notion, network topologies can be implemented with loci as computer nodes and exits as the network connections linking computers. In this artificial physical space, elements can be situated to represent data objects or processes. The agents in this type of environment are elements imbued with control algorithms that allow them to operate in the environment. Perception channels associated to agents let them detect the local environment features and sense changes in the environment.

Taking inspiration from games, the notion of human actors in the environment is also included in the model by allowing humans to take control of agents in the environment via remote connections. With this feature, human-computer interaction can also be studied in view of future work on human-agent interfacing.

In the future, taking advantage of the available features of the model, a further step that needs to be taken into account is the observation methods available for the study of dynamics in such an environment. In this direction, observation tools and a formalization of the observer situation should be added. Typically, degrees of visibility of elements in the environment were considered an important future extension in the preliminary phase of the project. This concept would allow observation of systems with minimal perturbation. In the prototype application, this had been implemented as two different skills, first, the possibility for an human to connect to a preexistent agent by a “snoop” command. In effect, snooping allowed a person to perceive the world from the perspective of the agent (entering the body of the agent and experiencing the same perceptions), but without allowing him to act. The second characteristic that had been implemented to that effect was that of allowing elements to be invisible to others. In the current model, this ability has been partially replaced by the notion of perception channels, but has not yet been implemented.

Another important missing feature is that of grouping or splitting elements into subcomponents. Initially, this was not considered fundamental, but with the work of my collegue P. Lerena on Bio-machines in the context of the AXE project [35] this has become an important notion. The study he is leading with Bio-machines is focussed on the evolution of populations of agents that are allowed to reproduce both sexually and asexually and combine into multicellular entities. With this last possibility, the combination notion becomes essential in EMuds if he is to use the environments for future experimentation.

9.1.2 Implementation

The implementation of the previous model takes the form of the EMud application, which is a simulation of the model on a single computer. In EMuds, the whole world in which agents may evolve is built by writing definition files for the environment state and population of elements. The dynamics of the environment are then simulated as if the environment was distributed over a network of computers. Human actors can also connect to the application with the telnet utility, as on a Mud, and interact with the agents evolving in the environment. Through the definition of skills, the virtual physical laws of the environment can be set up and creation or destruction of objects in the environment regulated. Some sample environments in which multiple agents coexist have been written and the current demonstration of the EMud runs on one of the computers at the Fribourg University, allowing anyone to connect and explore the current demonstration world.

In future work, the features of the model that must be introduced in the application are the generalized perception channels concept (only one perception channel now exists for all agents), interlinking of zones, many more skills and dynamic environment generation. Other aspects from which the application would benefit if they were implemented are an inline control algorithm scripting language, a character generation scheme (a gaming and agent specification feature), and a graphical interface for environment creation. If the model then proves to be relevant, a distributed implementation should be written.

9.2 Theoretical Aspects

The writing of this Ph.D. has brought me to realize that unlike many other scientific disciplines, artificial intelligence is faced with difficulties even in defining the problems it has to solve. Much work in the field is devoted to finding better algorithms for specific problems that are either invented for a particular algorithm or type of algorithm, or found in a real world environment. The reasons for this situation are multiple, but two reasons that stand out are, on the one hand, related to philosophical problems about the nature of intelligence and of the mind, on the other hand, with the scientific limits of understanding questions about complexity, information and algorithm dynamics.

9.2.1 Philosophical Problems

I begin this thesis with a short discussion about the philosophical perspective on artificial intelligence and introduce the symbol grounding problem, which is considered, at least by philosophers, to be the main obstacle to machine intelligence. In discussing this problem, I show that if intelligence is an emergent property of the functioning brain, one can reduce the symbol grounding problem to a problem of discovering equivalent functional properties in computer algorithms, which is an implicit assumption most AI researchers make.

Reducing the problem in this sense brings me to consider the concepts of meaning and understanding in terms of the causal role of components (symbols) in an algorithm and, respectively, compression of information within the description language that constitutes this algorithm. Although this view can be further discussed from a philosophical point of view, it has the advantage (when used as a working hypothesis) of giving a scientific base on which a theory can be built.

9.2.2 The Information Problem

Unfortunately, even reducing the philosophical question in such a way does not produce an immediate solution to the problems of artificial intelligence. Our knowledge about the nature of information is very limited. What is currently known is a good definition of an algorithm and from this definition, a measure of algorithmic information can be created. Even if this notion of algorithmic information allows the demonstration of very interesting mathematical results (typically about randomness), it has practical limitations related to the fact that it is built as a theoretical universal measure that attempts to capture a unique information content value in a description. I do not believe this is possible because the information content of a description must relate to an observer of this description. With algorithmic information, this dependence on an observer is evacuated by accepting that information is defined up to a finite constant value that depends on the language used for calculating the information content of a description, which is theoretically satisfactory. My objection is that artificial intelligence is not only a theoretical problem, but also essentially a practical problem, that is, finding implementations of algorithms that exhibit high level autonomous behaviors. Here, the agent that must extract the information from the environment is an observer and if, in his terms, the information content of what is perceived differs by a “finite constant” value from what another agent can extract from the environment, this might make a great difference.

9.2.3 Dealing with these Questions

I suggest two courses of action for future research in artificial intelligence, based on the preceding observations. The first method I feel would greatly improve research in the field is a unification of the many results that have already been published in the discipline. To do this, a categorization of the properties of all the algorithms that have been studied should be attempted by first designing a set of problems that are considered important on the path to machine intelligence. Based on this set of problems, some benchmarking experiments can be devised to evaluate the functional properties of each type of algorithm, allowing comparisons to be made between methods. Most potential test experiments have in fact already been invented since every school of AI has used some method of evaluating the results of their algorithms. The experimental part of the thesis provides an idea of how this classification may be initiated and gives a basic environment model in which benchmarking can be done for a wide variety of problems.

A theoretical approach to the information problem is to study the “practical” properties of Turing machines, since even the so-called universal Turing machines differ greatly in practice. The difficulties underlying this approach are twofold. First, the set of Turing machines is infinite and it might not be possible to give a categorization of these machines into general classes of machines, with some classes relevant to the problems found in AI and others not. Second, the mathematical tools available for this investigation are not well suited for distinguishing practical cases and theoretical cases. One attempt I put forward is by applying non-standard analysis to the theory of Turing machines. In this mathematical theory, the term standard allows a distinction to be made between standard objects (those that can be observed in practice) and non-standard objects (the theoretical objects). This permits a distinction to be made between standard and non-standard Turing machines. It remains to be seen whether fundamental properties of these two classes of Turing machines can be exhibited with this theory.

Bibliography

[1] D. M. Armstrong. The nature of mind. In Peter A. Morton, editor, *A Historical Introduction to the Philosophy of Mind*, pages 225–33. Broadview Press, Peterborough, Ontario, 1997.

[2] H. Atlan. *L'organisation biologique et la théorie de l'information*. Hermann, 1972.

[3] L. W. Barsalou. Perceptual symbol systems. *Behavioral and Brain Sciences*, (To appear).

[4] R. A. Bartle. Early MUD History. Internet News:rec.games.mud, origin confirmed by the author, November 1990.

[5] L. Brillouin. *Science and Information Theory*. Academic Press, New York, 1962.

[6] R.A. Brooks. Intelligence without Reason. In *Proceedings of IJCAI-91*, Sydney, Australia, 1991.

[7] R.A. Brooks. Intelligence without Representation. *Artificial Intelligence, Special Volume: Foundations of Artificial Intelligence*, 47(1-3), 1991.

[8] G. J. Chaitin. *Algorithmic Information Theory.* Cambridge University Press, 1987.

[9] G. J. Chaitin. *The limits of mathematics: A course on information theory & limits of formal reasoning.* Springer-Verlag, Singapore, 1997.

[10] F. Chantemargue, M. Courant, T. Dagaeff, and A. Robert. A pragmatic approach to conflict. In *Proceedings of the 13th Biennal European Conference on Artificial Intelligence (ECAI'98)*, Brighton, UK, 1998.

[11] D. Cliff, I. Harvey, and Ph. Husbands. *From Living Eyes to Seeing Machines*, chapter Artificial Evolution of Visual Control Systems for Robots. Oxford University Press, July 1997.

[12] R. Davis and J. King. An Overview of Production Systems. In E. W. Elcock and D. Mitchie, editors, *Machine Intelligence*, volume 8, pages 300–332. Wiley, New York, 1976.

[13] D. C. Dennett. Quining qualia. In Peter A. Morton, editor, *A Historical Introduction to the Philosophy of Mind*, pages 409–434. Broadview Press, Peterborough, Ontario, 1997.

[14] R. Descartes. Meditationes de prima philosophia, in quibus dei existentia, & animae humanse corpore distinctio, demonstrantur. Electronic Document (original Latin, French and English translations): http://philos.wright.edu/DesCartes/Meditations.html, 1641.

[15] M. Dorigo and H. Bersini. A comparison of q-learning and classifier systems. In *Proceedings of From Animals to Animats, the Third International Conference on Simulation of Adaptive Behavior (SAB94)*, Brighton, UK, 1994.

[16] J.-P. Dupuy. *Logique des phénomènes collectifs, Introduction aux Sciences Sociales*, chapter "Common Knowledge" et Sens Commun. Collection Ellipses, Ecole Polytechnique, Paris, 1992.

[17] J.-P. Dupuy. *Aux origines des sciences cognitives*. Editions la Découverte, Paris, 1994.

[18] J. A. Fodor. *The Language of Thought*. Crowell, New York, 1975.

[19] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag, 1996.

[20] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., Reading, MA, January 1989.

[21] G. Gygax. *Dungeons and Dragons, Basic Set*. TSR Hobbies Inc., 1974.

[22] S. Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

[23] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[24] J. H. Holland. Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine learning, an artificial intelligence approach.*, volume II. Morgan Kaufmann, Los Altos, California, 1986.

[25] The on-line hacker jargon file, v4.0.0, July 1996.

[26] F. Jackson. Epiphenomenal qualia. *Philosophical Quarterly*, 32:127–136, 1982.

[27] L. Kaelbling, M. Littman, and A. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

[28] S. Kauffman. *Origins of Order: Self-organization and selection in evolution*. Oxford University Press, New York, 1993.

[29] M. Keegan. The mud tree. http://userweb.cyburbia.net.au/ martin/mudtree.html, 1996.

[30] T. Kovacs. Evolving Optimal Populations with XCS Classifier Systems. Master's thesis, University of Birmigham, UK, 1996.

[31] Oliver Krone. *STL and Pt-PVM: Concepts and Tools for Coordination of Multi-threaded Applications*. PhD thesis, University of Fribourg, 1997.

[32] Th. S. Kuhn. *The Structure of Scientific Revolution*. University of Chicago Press, 3rd edition, 1996.

[33] D. Law and R. Miikkulainen. Grounding robotic control with genetic neural networks. Technical Report AI 94-223, Departement of Computer Science, University of Texas at Austin, Austin TX, 1994.

[34] P. David Lebling, Marc S. Blank, and Timothy A. Anderson. Zork: A Computerized Fantasy Simulation Game. *IEEE Computer*, 12(4):51–59, April 1979.

[35] P. Lerena. Bio-machines. In *Artificial Life*, volume V, Nara, Japan, 1996.

[36] S. Lloyd and H. Pagels. Complexity as thermodynamic depth. In *Annals of Physics 188*, pages 186–213. Academic Press, 1988.

[37] M. Ludwig and M. Courant. Modèle de communication dans un système multi-agents. In B. Hirsbrunner, M. Courant, and M. Aguilar, editors, *Parallelism and Artificial Intelligence*, volume 3 of *Series in Computer Science*, chapter 9. University of Fribourg, Fribourg, August 1994.

[38] W. G. Lycan. *The Blackwell Companion to Philosophy*, chapter The Philosophy of Mind. Blackwell Publishers Ltd., Cambridge, MA, 2nd edition, 1996.

[39] H. Maturana and F.J. Varela. *Autopoiesis and Cognition: the realization of the living*, volume 42 of *Boston Studies in the Philosophy of Science*. D. Reidel Publishing Co., Dordecht, Holland, 1980.

[40] Marvin Minsky. *Society of Mind*. Simon and Schuster, New York, 1985.

[41] J. L. Moreno. *Psychodrama.* Beacon Press, New York, 1946.

[42] P. A. Morton. *A Historical Introduction to the Philosophy of Mind.* Broadview Press, Peterborough, Ontario, 1997.

[43] T. Nagel. What is it like to be a bat? *Philosophical Review*, 83:435–450, 1974.

[44] A. Newell and H. A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, March 1976. 1975 ACM Turing Award Lecture.

[45] S. Schubiger O. Krone. WebRes: Towards a Web Operating System. In *Proceedings of Fachtagung Kommunikation in Verteilten Systemen, KIVS '99*, March 2–5 1999.

[46] R. Pfeifer. Building Fungus Eaters: Design Principles of Autonomous Agents. In Maes, Mataric, Meyer, Pollack, and Wilson, editors, *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, volume 4 From Animals to Animats, Cambridge, MA, 1996. MIT Press / Bradford Books.

[47] H. Putnam. The nature of mental states. In Peter A. Morton, editor, *A Historical Introduction to the Philosophy of Mind*, pages 320–27. Broadview Press, Peterborough, Ontario, 1997.

[48] T. Regier. *The Acquisition of Lexical Semantics for Spacial Terms: A Connectionnist Model of Perceptual Categorization.* PhD thesis, Departement of Computer Science, University of California at Berkeley, Berkeley CA, 1992.

[49] A. Robert, F. Chantemargue, and M. Courant. Emuds: Grounding agents in emud artificial worlds. In J.-C. Heudin, editor, *Proceedings of the 1st International Conference on Virtual Worlds*, Lecture Notes in Artificial Intelligence, pages 193–204, Paris, France, 1998. Springer.

[50] A. Robert and M. Courant. Du collectif au social, vers une distinction fonctionelle quantitative. In Ecole Nationale Supérieure des Télécommunications, editor, *Du Collectif au Social, Actes des Journées de Rochebrune*, pages 31–38, Paris, France, 1996.

[51] L. Schmutz. *Une morphologie d'agent pour la coordination émergente.* PhD thesis, University of Fribourg, 1998.

[52] S. Schubiger and O. Krone. Interactive Resource Sharing on the Web. In *Proceedings of Workshop on Distributed Computing on the WEB*, June 22-23 1998.

[53] J. R. Searle. Minds, brains, and programs. *The Behavioral and Brain Sciences*, 3:417–24, 1980.

[54] C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, IL, 2nd edition, 1963.

[55] H. A. Simon. *The Science of the Artificial*. MIT Press, Cambridge, MA, 3rd edition, 1996.

[56] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, Boston, MA, 1997.

[57] B. F. Skinner. Science and human behaviour. In Harvard University Press, editor, *Readings in the Philosophy of Psychology*, volume 1, pages 37–41. Ned Block, Cambridge, MA, 1980.

[58] A. Sloman. How to dispose of the free will issue. *AISB Quarterly*, 82:31–2, Winter 1992 / 3.

[59] L. Steels. When are robots intelligent autonomous agents? *Robotics and Autonomous systems*, 1995.

[60] H. W. Stone. Mars pathfinder microrover: A small, low-cost, low-power spacecraft. In *Proceedings of the 1996 AIAA Forum on Advanced Developments in Space Robotics*, Madison, WI, August 1996.

[61] R. S. Sutton, A. G. Barto, and R. J. Williams. Reinforcement learning is direct adaptive optimal control. In *Proceedings of the American Control Conference*, pages 2143–2146, Boston, MA, 1991.

[62] J. R. R. Tolkien. *The Lord of the Rings:The Return of the King/the Two Towers/the Fellowship of the Ring*. Houghton Mifflin Co, Boston, 1988.

[63] F. Varela, E. Thompson, and E. Rosch. *The Embodied Mind: Cognitive and Human Experience*. MIT Press, Cambridge MA, 1991.

[64] G. Venturini. *Apprentissage Adaptatif et Apprentissage Supervisé par Algorithme Génétique*. PhD thesis, Université de Paris-Sud, 1994.

[65] L. von Bertalanffy. *General System Theory: Essays on its Foundation and Development*. George Braziller, New York, 1968.

[66] H. von Foerster, M. Mead, and H. L. Teuber. Cybernetics - circular and causal feedback mechanisms in biological and social systems. In *Transactions of the 6th to 10th Macy Conferences*, New York, 1950-55. Josiah Macy Jr. Foundation.

[67] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.

[68] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[69] S. W. Wilson. ZCS: a Zeroth Level Classifier System. *Evolutionary Computation*, 2(1):1–18, 1994.

[70] S. W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[71] S. W. Wilson. State of XCS Classifier System Research. Technical report, Prediction Dynamics, Concord, MA 01742 USA, March 18, 1999.

[72] T. Ziemke. *Understanding Representation in Cognitive Science*, chapter Rethinking Grounding. Plenum Press, New York, 1999.