Machine Learning and Inductive Logic Programming for Multi-agent Systems

Dimitar Kazakov and Daniel Kudenko*

Department of Computer Science
University of York, York YO10 5DD
United Kingdom
{lastname}@cs.york.ac.uk

Abstract. Learning is a crucial ability of intelligent agents. Rather than presenting a complete literature review, we focus in this paper on important issues surrounding the application of machine learning (ML) techniques to agents and multi-agent systems (MAS). In this discussion we move from disembodied ML over single-agent learning to full multi-agent learning. In the second part of the paper we focus on the application of Inductive Logic Programming, a knowledge-based ML technique, to MAS, and present an implemented framework in which multi-agent learning experiments can be carried out.

1 Introduction

In recent years, multi-agent systems (MAS) have received increasing attention in the artificial intelligence community. Research in multi-agent systems involves the investigation of autonomous, rational and flexible behavior of entities such as software programs or robots, and their interaction and coordination in such diverse areas as robotics [16], information retrieval and management [17], and simulation [9].

When designing agent systems, it is often infeasible to foresee all the potential situations an agent may encounter and specify an agent behavior optimally in advance. In order to overcome these design problems, agents have to learn from and adapt to their environment.

The task of optimally designing agents is even more complex when nature is not the only source of uncertainty, and the agent is situated in an environment that contains other agents with potentially different capabilities, goals, and beliefs. Multi-Agent Learning, i.e., the ability of the agents to learn how to cooperate and compete, becomes crucial in such domains.

In this paper we present an overview of important issues concerning the application of Machine Learning to Agents and Multi-Agent Systems that hopefully will stimulate further thoughts. While we point to selected relevant work in our discussions, it is not our goal to provide an exhaustive literature review. For a broader review of past research we refer the reader to [37].

* In alphabetical order

M. Luck et al. (Eds.): ACAI 2001, LNAI 2086, pp. 246–270, 2001.

In the first part of the paper we discuss ML and MAS issues from a general viewpoint. In the second part we focus on Inductive Logic Programming (ILP), a logic-based learning technique that is able to exploit domain knowledge.

The paper is organized as follows: We begin with a general overview of the state-of-the-art in research in Machine Learning (ML) which has focused mainly on learning as an independent reasoning process and discuss the integration of these methods in a complete and situated agent system. In Section 4 we highlight several issues that are important for ML in a Multi-Agent setting, such as agent heterogeneity, awareness of other agents, communication, and distribution of learning tasks. Following this, we present a number of issues arising from the application of ILP to Agents and Multi-Agent systems and an example of such an application. We finish with a summary and an outlook on interesting open research questions within the area of Multi-Agent learning.

2 Principles of Machine Learning

In this section we briefly introduce the research area of machine learning (ML), and present some classifications of ML algorithms.

In his recent book [23], T. Mitchell defines machine learning as

> the study of computer programs that improve through experience their performance at a certain class of tasks, as measured by a given performance measure.

In more detail,

> the learning system aims at determining a description of a given concept from a set of concept examples provided by the teacher and from the background knowledge [21].

Here, 'concept' is usually interpreted as a subset of objects or observations defined over a larger set (*universe*), or, alternatively, as a boolean-valued function defined over this larger set [23]. Background knowledge is a set of concepts, i.e., statements which are known to be true for the given universe.

If the *target concept*, i.e. the concept to be learned, has a finite number of members, all of which are supplied to the learning algorithm, then the learning task can be reduced to the memorising of all examples of the concept. In this case, learning can aim at the more concise representation of the concept. However, much more often only some of the examples, called *training examples*, are used for learning. Learning from an incomplete set of examples is called *inductive learning*. Inductive learning is based on the following assumption, known as *inductive learning hypothesis:*

> Any hypothesis found to approximate the target [concept] well over a sufficiently large set of training examples will also approximate the target [concept] well over other unobserved examples [23].

The concept definition generated by inductive learning is evaluated on a set of *test examples*. None of the training examples should belong to the test set.

There are two general categories of ML algorithms: *black-box* methods, and *white-box* or *knowledge-oriented* ones. Black-box methods create a concept representation for their use, but its interpretation by the user is unclear or impossible. In this way, even if the algorithm is able, for instance, to correctly classify the unseen instances of the learned (or *target*) concept, it is not possible to *explain* its behavior. On the other hand, *knowledge-oriented* algorithms create symbolic knowledge structures that are comprehensible [21].

According to the type of examples provided, the learning algorithm may have to learn from (1) *positive* and *negative examples* of the same concept (birds and "non-birds"), (2) examples of several concepts (birds, flowers, and bees), or (3) *positive-only examples* of the same concept without any counter-examples (Donald Duck and Scrooge McDuck are birds). If in the first two cases the aim of learning is to become able to distinguish between different concepts (or the concept and its complement), in the last case the result is a description of the concept rather than discrimination from other ones. In other words, learning from only positive examples may result in a more compact representation of the target concept, or, in some statements about the target concept, which are true, but not necessarily sufficient to determine the class membership of unseen examples. Within a Bayesian framework, it has been proven that logic programs are learnable with arbitrarily low expected error from positive examples only [27]. The upper bound for expected error of a learner which maximises the Bayes' posterior probability when learning from positive examples only is within a small additive term of one which does the same from a mixture of positive and negative examples.

All three cases enumerated above belong to the *supervised learning* paradigm, where examples are annotated with their corresponding concept. Alternatively, within *unsupervised learning* the data is provided along with the definition of a language which has to be used to represent it. No additional information is supplied. Instead, there is a *language bias* specifying a certain order of preferences on the possible representations of the data. Then, among all possible descriptions of the input data, the one which has the highest score with regard to the bias is chosen.

In general, ML algorithms have to look for a somewhat more general description of the target concept than the mere list of training examples. The exact representation of the concept learned depends on the type of language used for its description. It also depends on the bias used for learning. For instance, the representation of the concept is usually expected to be more compact than the list of examples provided. In this context, a principle known as *Occam's razor* is often quoted. The statement of William of Occam (ca. 1285–1349) *"Non sunt multiplicanda entia praeter necessitatem"*—entities are not to be multiplied beyond necessity—came to be interpreted to mean that one should prefer the simplest hypothesis that fits the data [23].

Another principle, which is widely used as a language bias in ML, is *Minimal Description Length (MDL) principle*. For a given encoding, it recommends choosing the hypothesis that minimises the sum of the description length of the hypothesis and the description length of the data given the hypothesis [23].

An important issue in machine learning is also the way in which the algorithm handles misclassified examples, or noise, in the data. Even a single example, if incorrectly classified, can make the exact learning of a concept impossible and result in a description which is incomplete (i.e., does not cover all positive examples) or inconsistent (i.e., covers some of the negative ones).

More dichotomies in the taxonomy of ML algorithms are introduced by the way in which they use examples for learning. *Batch* algorithms need to process all learning examples at once and, if more examples are available, the whole learning phase has to be repeated from scratch. *Incremental algorithms*, on the other hand, are able to consider additional examples as they come. An *eager* learning algorithm generates a hypothesis for the target predicate in advance, whereas a *lazy* learner postpones that decision until it is presented with a test example. Broadly speaking, the distinction between *eager* and *lazy* learning is related to the ability of *lazy* learners to generate local approximations of the target hypothesis w.r.t. the test example, whereas *eager* learners have to commit to a single hypothesis at training time [23].

Traditional ML separates learning from the acquisition of data as two independent processes. There is some recent work that brings these processes closer. *Active learning* [40] extends a standard supervised learner requiring expensive annotated data with a module browsing a much larger unannotated dataset and selecting for annotation the examples that are considered the most beneficial to the learning process. *Closed Loop Machine Learning* (CLML) [4] couples ML with a robot carrying out experiments. In this way the learning algorithm can suggest a hypothesis and verify it experimentally — if the hypothesis is rejected, the collected data can give rise to a new round of the same cycle.

3 Machine Learning for (Single) Agents

Most of the past ML research has been focused on non-agent-based, or "disembodied", learning algorithms, i.e., without taking into account that the learning algorithm may be embedded in an agent that is situated in an environment. An agent consists of many different interacting modules (e.g., vision, planning, etc.) and the learning module is just one of them. Therefore, learning has to support and be supported by a wide range of modules.

In this section we discuss the issues arising from integrating ML algorithms in an agent system, focusing on the following questions:

- What are the learning targets and the respective training data for agents?
- What are the applicable learning techniques?
- How can the ML system be integrated into the agent architecture?

3.1 Learning Targets and Training Data

The simplest model of an agent describes it as an entity that receives sensory input from the environment and based on this input and some internal reasoning acts in the environment. Given this model, we can define the learning target of an agent simply as a decision procedure for choosing actions. We deliberately omit the notion of "choosing *optimal* actions" in this definition, because the question of what "optimal" means depends very much on the source of the training data and the bias of the learner. We distinguish three types of data sources:

External Teacher: An external teacher provides examples of actions (or action sequences) with the corresponding classification indicating their optimality or appropriateness. This model is equivalent to *fully supervised learning*.

Environmental Feedback: While the agent acts, it receives a feedback from the environment indicating the benefit of the action(s). The feedback is usually defined in terms of the utility of the current state that the agent finds itself in. This training model corresponds to *reinforcement learning*.
It should be noted that not necessarily all states will result in feedback. This means that once some environmental feedback is received it has to be propagated to all actions that potentially contributed to it. But which actions have contributed to the current state and to what extent? Certainly, actions that contributed strongly should receive more recognition (or blame). This *credit assignment problem* is still mostly unsolved. A common technique to distribute rewards (or punishments) amongst actions is to reward (or punish) more recent actions higher using a discount factor. See Section 3.2 for more details.

Internal Agent Bias: While the agent is exploring the environment with its actions, it looks out for "useful" patterns and "interesting" properties of the environment that enable the agent to generate concepts describing the environment. Usefulness and interestingness are purely based on the agent's internal bias (e.g., Minimum Description Length, Occam's razor), and no explicit feedback is given to the agent. It is assumed that the discovered concepts will help the agent to perform future specific goals efficiently and effectively. This learning model is usually denoted by the term *unsupervised learning*.

We will focus in this paper on the first two learning models. There has been relatively little work on unsupervised learning — for more information see for example [8]. In Section 5 we will present ways how to apply a supervised learning method (namely Inductive Logic Programming) in an agent context.

In the next section we discuss reinforcement learning techniques and specifically focus on the most widely used agent learning algorithm: *Q Learning*.

3.2 Reinforcement Learning

As noted above, in reinforcement learning (RL) an agent adapts his decision process based on environmental feedback resulting from its choice of actions. In

this section we briefly describe a widely used RL technique, Q learning [43], and discuss the use of other ML algorithms in a RL context. For a more complete treatment of reinforcement learning and in particular Q learning we refer the reader to [23,12].

Q Learning In Q learning, an agent's decision procedure is specified by a *policy* π that maps states into actions. The environmental feedback is defined by a *reward function* R that maps states into numerical rewards. The goal of Q learning is to compute an optimal policy π^* that maximizes the reward that an agent receives.

Let s_t be the state that the agent reaches at time point t when using policy π to guide its actions (s_0 is the initial state). The reward for using policy π in state s_t is denoted by $V^\pi(s_t)$. This reward can be measured in several ways:

Discounted Cumulative Reward: This measures the cumulative rewards that an agent will get when starting in a state s_t and following a given policy π. Future rewards are discounted according to a discount parameter $0 \leq \gamma \leq 1$. The smaller γ is, the more the agent prefers short-term over long-term rewards.

$$V^\pi(s_t) = \sum_{i=0}^{\infty} \gamma^i R(s_{t+i})$$

Finite Horizon Reward: This is a modification of the discounted cumulative reward which takes only a finite number h of future states into account. The non-discounted version is:

$$V^\pi(s_t) = \sum_{i=0}^{h} R(s_{t+i})$$

Average Reward: This function measures the average reward that an agent receives for using policy π:

$$V^\pi(s_t) = \lim_{h \to \infty} \frac{1}{h} \sum_{i=0}^{h} R(s_{t+i})$$

While all above functions may have their uses, Q learning is based on discounted cumulative reward. Note that the function V defines ways in which actions receive a portion of future rewards. In the case of discounted cumulative reward, actions receive a proportion of the reward of future actions, where the size of the proportion is defined by the discount factor γ.

The optimal policy π^* can now be formally defined as:

$$\pi^* = argmax_\pi V^\pi(s)$$

The optimal policy is learned by associating an action with the direct rewards received in the resulting state and adding an estimate of the maximal

Table 1. Q Learning Algorithm

```
PROCEDURE Q-LEARN (g: discount factor) {
   Q(s,a) = 0 for all state-action pairs (s,a);
   s = initial state;
   Repeat forever {
      Select action a and execute it;
      s' = new state;
      E = maximum of Q(s',a') over all actions a';
      Q(s,a) = R(s') + g * E;
      s = s';
   }
}
```

future reward that can be achieved. The estimate is based on the agent's past experience. Given a state s, choosing an action a will result in the following *estimated* discounted cumulative reward Q_e:

$$Q_e(s,a) = R(s') + \gamma \max_{a'} Q_e(s',a')$$

where s' is the state resulting from executing action a in state s.

Given the correct Q values, the optimal policy can be computed as follows:

$$\pi^* = argmax_a Q(s,a)$$

An agent updates its estimates of the Q function while executing actions and observing the rewards of the resulting states. Table 1 shows the update algorithm (based on [23]).

The Q learning algorithm has been proven to converge towards the correct values (and thus learn the optimal policy) under the condition that the reward values (i.e., the image of R) have an upper bound. This convergence can take many of iterations. In fact, in order to guarantee optimal results all possible state-action pairs have to be visited infinitely often. Obviously, this may lead to problems in domains with continuous actions.

One of the problems that has to be addressed is how to select actions in the main loop of the Q learning algorithm. If an agent always chooses a seemingly optimal action it might get stuck in a local optimum. Therefore it is better to choose random actions with a higher probability in the early training stages and a lower probability in the later stages when the Q value estimates have converged towards the true values.

Storing all Q values in a table may quickly lead to memory space problems, and therefore a function approximation of Q, e.g., a Neural Net [39], is often used instead. This approach may lead to a more complex update mechanism.

While Q learning is an efficient and highly suitable learning method for learning agents because of its coupling of exploration and learning, it has a number of disadvantages:

- The technique is only applicable to reactive agents (i.e., agents that base their action choice only on the current state).
- Defining a numerical reward function can be a non-trivial task for some application domains.
- As mentioned above, convergence may take a long time.
- The learning result is not transparent in the sense that no explanation is given for action preferences. Nevertheless, given a state, actions can be ranked, which may yield some limited insight into the performance of the agent.

Other ML Algorithms for Reinforcement Learning Even though Q learning is the most popular reinforcement learning technique, there is no reason why other learning algorithms such as decision tree learning or inductive logic programming (ILP) could not be used in an RL context.

Džeroski et al. [7] explore the combination of relational regression tree learning and Q learning. The result is a more expressive and more general Q function representation that may still be applicable even if the goals or the environment of the agent change.

In another approach, Dietterich and Flann [6] combine explanation-based learning (a form of learning that is focused on speeding up reasoning processes) with reinforcement learning.

In Section 5 we discuss ways of combining ILP with reinforcement learning.

3.3 Integrating Machine Learning into an Agent Architecture

As Section 2 showed, machine learning algorithms can be made proactive. CLML is probably the latest stage in the development of standard ML towards active exploration of the world. It is a very specific case of a learning agent, as learning is the sole goal of the embodied agent (robot). Denying ML that privileged status and making it just one of the ways of improving the agent's performance on other tasks brings in a whole new range of issues, which will be discussed here. In the following, a simulated system of agents will be assumed, as it is the more general case, of which embodied agents acting in the real world are just an instance.

Time Constraints on Learning Machine learning algorithms are traditionally judged by the predictive accuracy of the theories learned, while time complexity is only considered as a secondary factor, a price to pay, which, if low, adds to the attractiveness of the algorithm. Time complexity may make learning from too large a data set infeasible; however, the value of a theory learned in a minute and one learned in a day is exactly the same as long as they have the same accuracy. The situation changes when machine learning is to be used in a framework of agents. There are many other activities, such as perception, planning, and control of the effectors, which compete with learning for the resources of an agent. Here, the allocation policy chosen is, obviously, of great importance.

The time constraints imposed on learning would depend on the learning task. If eager learning is to provide a theory of past experience that is more compact and also has a better generalisation power, then all that is needed is enough time for the learning algorithm to run, ideally at a time when the agent is idle so that all computational resources can be used. If lazy learning is used along with training data (past observations) to provide an interpretation for (classification of) each test example (new observation) as they come, then the time available becomes very limited. The correct recognition of a danger has to be done before the normal functioning of the agent is compromised by that danger.

Another factor for the feasibility of machine learning in agents is whether the algorithm used has a clear halting condition or it is a representative of the so called *any-time learning* in which, in general, there is always a potential for improving the current hypothesis by running the algorithm longer. Systems with exhaustive search of the hypothesis space belong to the former class; those which only sample it (e.g., genetic algorithms) to the latter. If the agent is to operate under rigorous time constraints, e.g., in real time, worst-case time complexity analysis of the algorithm can be used to determine whether to start learning at all. For any-time learning, one has to be able to implement a decision procedure to halt the learning in order to meet hard deadlines or when its cost outweighs the benefits of improved accuracy.

In general, there is a big gap between the reaction times required in Real-Time Systems (RTS) and the time complexity of machine learning at present. The situation can be compared to the one in the area of Speech Recognition (SR) and Natural Language Processing (NLP). In both cases, one has to deal with tasks which are inherently interrelated and have the same goal. Indeed, understanding speech can be improved by the use of parsing or semantic analysis, and a learning agent can potentially outperform one that uses a hard-wired behavior. The very different hardware and application requirements (it is acceptable for several NLP tasks to be carried out off-line whereas speech recognition is essentially an on-line task) kept SR separate from the rest of NLP well into the mid-1990s, when off-the-shelf hardware and software platforms, and tailor-made applications could at last meet the requirements of both areas [41]. One can expect the same development at the intersection of machine learning and real-time systems as the latter search for greater efficiency and make a conceptual shift from dealing with hard time constraints to frameworks allowing for more flexibility, such as the imprecise model of computation and flexible real-time systems [31]. It is clear though that the process of mutual approaching of ML and RTS is at its beggining.

To make the integration of ML with agents easier, one could use simulations in which the time constraints on ML are gradually taken more and more into account. One could, for instance, discriminate between the following three cases:

- unlimited time for learning
- upper bound on time for learning
- learning in real time

Synchronisation between Agents' Actions When a simulated environment of learning agents is studied, one has to choose a method that will inform an agent about the current state of its world, prompt it to choose an action, and then carry out the action and change the environment accordingly, if the action is possible. Consider the following three methods:

One-step-at-a-time, simultaneous update of environment The agents are prompted to choose their moves one by one. After each of them has selected an action, the environment carries out all the actions simultaneously. The method has to be able to deal with conflicting actions.

One-step-at-a-time, immediate update Same as above, but each agent's action is carried out immediately upon generation. Here the order in which agents take turns will be important.

Agents as asynchronous processes Of the three, this is the best model of reality. However, its implementation requires more hardware resources, such as separate computers or processors, and more complex software implementation.

The choice of the type of time constraints and time synchronisation of the individual agents' actions is schematically represented in Table 2.

Table 2. Synchronisation × time constraints

	Unlimited time	Upper bound	Real time
1-move-per-round, batch update	Logic-based MAS for conflict simulations [18]		
1-move-per-round, immediate update	The York Multi-agent Environment [14]		
Asynchronous			Multi-agent Progol [25]

Learning and Recall Each time the sensory information about the world is updated, an agent that is able to learn has to choose whether to *recall* its existing model of the world to decide about its next action, and learn from its outcome immediately to update its current model of the world, or to postpone learning for later. If lazy learning is used, the distinction between learning and recall disappears, since a new theory with a limited coverage is built to classify the new example, and all that is carried forward is the new observation (but none of the theories). In the case of eager learning though, learning and recall are two separate processes, the coordination of which will be discussed next.

Lazy learning in its simplest forms, such as case-based reasoning, only requires to search through the already seen cases and find the closest match for the new one that is to be analysed (classified). Although suitable indexing, such

as hash tables, can be used to reduce look-up times, the applicability of lazy learning is ultimately bounded by its limited generalisation power, and the fact that its time complexity increases with the number of examples. Eager learning can help to obtain theories with coverage far beyond the one guaranteed by lazy learning. For instance, Explanation-Based Learning (EBL) is potentially able to learn a theory from a single example. On the other hand, there are tasks where lazy learning guarantees the best results [5,28]. However, if recall times are an issue, eager learning can be used to compress training examples (previous observations) into a more concise theory in the hope that this will also result in faster recall.

Different ways of combining learning with recall can be summarised in the following clasification:

Parallel learning and recall at all times Learning and recall are possible within the same course of actions.

Learning and recall as separate modes (one thing at a time) The agent has two separate modes. In the first, it is active and uses its existing knowledge to choose its actions, but cannot update it; observations are collected for later processing. In the second mode, the only activity performed by the agent is learning, when it processes the examples accumulated in the active phase to incorporate them into its model of the world. The examples can then be disposed of, if learning is incremental, or they can be kept if the agent uses batch learning that starts from scratch each time it is invoked. In the context of agent learning, incremental learning seems better suited as it has lesser computational and memory requirements.

Cheap on-the-fly learning, off-line computationally expensive learning The amount of raw information that an agent collects through its sensors can be considerable, which may lead to infeasible memory requirements if separate modes for learning and recall are used. In those cases, an agent could employ some limited on-the-fly learning to reduce the strain on memory. For instance, one could reduce the dimensionality of the object language, introducing a new set of attributes if necessary, cluster examples and replace each cluster with a single representative, or ignore "uninteresting" examples altogether. Then, off-line learning can be applied at a later time as in the previous case. Here one is tempted to recall the case of human learning—altough one's knowledge is apparently updated in periods of awakeness, the process continues during sleep, and the latter phase is crucial in the acquisition of knowledge for long-term use.

4 Machine Learning for Multi-agent Systems

As one moves from the single-agent setting to an environment where many agents are acting and potentially interfering with one another, acting optimally and consequently learning how to act optimally becomes a highly complex task. When applying learning techniques to multi-agent systems there are many issues

to be taken into account. In the following sections we will discuss a few questions that arise in this context and point to relevant literature:

- What impact does awareness of other agents and their behavior have?
- What is the importance of communication and how does it influence the learning process?
- How does learning influence team heterogeneity?
- How does distributed learning differ from other multi-agent learning approaches?

4.1 Awareness of Other Agents

A multi-agent system is generally defined as a collection of agents that observe and act in the same environment. It is important to stress that this does not imply *social awareness*, i.e., awareness of other agents in the environment and knowledge about their behavior.

While it may seem useful to have an increased social awareness, it is sometimes not necessary in order to achieve near-optimal performance. Steels [38] describes an application domain where the task for a group of robots is to collect rock samples and return them to a central storage facility. A near optimal solution defines a small set of simple reactive rules for the robots that do not require explicit social interactions[1].

Vidal and Durfee [42] define levels of social awareness as follows:

0-level agents: have no knowledge about other agents or their actions, and observe them only as changes in the environment.

1-level agents: recognize that there are other agents around, but have no knowledge about their behavior. These agents model other agents as 0-level agents.

2-level agents: have some knowledge about the behavior of other agents and their past observations. The model of the other agents does not include any influence of their knowledge about the agent himself, i.e. a 2-level agent models other agents at most as 1-level agents.

These definitions can be continued to higher levels *ad infinitum*, where agent A knows that agent B knows that A knows that B knows, etc.

It should be noted that 0-level agents are able to learn implicitly about other agents, as long as they are able to observe the results of the actions of other agents in the environment. Even though 0-level agents have no explicit social awareness, they will incorporate other agents behavior in their learned

[1] Admittedly there is a limited amount of implicit social interaction involved in that robots drop rock samples that are used by other robots in their decision process. Nevertheless, none of the robots needs knowledge about the actions of other robots and treats rock samples in the same way, whether they have been dropped by other robots or have been there to begin with.

hypothesis as the behavior of the environment. Therefore, multi-agent learning, in principle, already starts with 0-level agents.

Mundhe and Sen studied the performance of Q learning agents up to level two in two-agent single-stage games. Interestingly, their results show that two 1-level agents display the slowest and least effective learning, much worse than two 0-level agents. In general, the results show that a diversity of agents is beneficial and that myopic agents outperform non-myopic agents. While these results are interesting, they are restricted to a specific appliation domain, and there is need for further research into these issues in order to be able to define more general principles.

4.2 Communication and Learning

In the eyes of a linguist, communication and learning are inherently related, since the same language concepts are used in both processes. A difference is usually made between *knowledge* and *skills*: the first can be clearly formulated and conveyed by the means of language; the second can only be acquired through personal experience, and the theory learned (if there is any) cannot be put in words. Using ML terminology, one could say that learning "knowledge" as defined in this context corresponds to white-box learning, and learning skills to black-box learning. It should be noted that even the exchange of parts of Neural Networks (that are usually associated with black-box learning) would require the agents to "understand" what is transmitted and thus become an exchange of knowledge rather than skills.

It is also a well-known fact that human experts are reluctant to formulate general rules they base their decisions on. One the other hand, they find it easier to motivate their decision if a particular case is studied. A quotation of the architect Frank Lloyd Wright will help to make the point: "*an expert is someone who does not think anymore—he knows*". The above descriptions match the definitions of various forms of lazy learning.

If learning is used in a single agent, the only reason to choose a learning method is the quality of its results. However, the situation changes with the move from single-agent learning to societies of learning agents which can communicate with each other. For an agent, the cost of asking information (knowledge) from another, experienced agent is usually much lower than the cost involved in acquiring the information on its own, either by exploring the environment or by purely observing the actions of other agents.

However, the information should be expressed in a formalism (language) shared by both agents. In other words, only the results of white-box learning can be shared between agents. Consider also the case when the communication channel has a (very) limited bandwidth. Although lazy learning may still be the best for the individual, some form of eager learning must be employed if the agents are to benefit from their communication abilities, as "downloading" another agent's exaustive list of personal experiences may be infeasible.

4.3 Team Heterogeneity

In a team of agents that is solving a task with combined forces, it often seems useful for agents to specialize in different subtasks and thus share the effort in a more efficient way.

One way to achieve this kind of heterogeneity in the multi-agent system is to equip agents with different sensor and effector capabilities or behaviors and thus pre-define the roles that they are going to play in the team effort. While this method may lead to good results [32], it has a number of drawbacks. Firstly, it is not always obvious how to specify an optimal (or even useful) distribution of sensors and effectors or behaviors. Secondly, it may be quite expensive (in terms of hardware or in terms of development time) to design a system of heterogeneous agents.

An alternative is to use a team of learning agents that are homogeneous to begin with, but with time and experience will diversify and specialize. Balch [2] studied the conditions under which a team of agents based on reinforcement learning will converge towards heterogeneity. In his research, he distinguishes between three types of reward functions:

Local performance-based reinforcement: Each agent receives rewards individually when he personally achieves the task.

Global performance-based reinforcement: All agents receive a reward when one of the team members achieves the task.

Local shaped reinforcement: Each agent receives rewards continuously while he gets closer to accomplishing the task.

Balch applied the different reward functions to multi-agent reinforcement learning in the domains of multi-robot foraging, robotic soccer, and formation movements. Using *social entropy* [3] as a diversity measure, he observed the following results:

- Globally reinforced agents converge towards a heterogeneous team, while local reinforcement leads to homogeneous agents.
- Heterogeneity is not always desirable: in multi-robot foraging a locally reinforced and therefore homogeneous team outperforms a globally reinforced and therefore heterogeneous team. On the other hand, in the robotic soccer domain the results are opposite, i.e., heterogeneity yields the better performance.

Clearly, these results show that team heterogeneity is an important factor in multi-agent performance and different learning methods can yield different levels of heterogeneity. It is an interesting open research problem to gain a clearer and more detailed understanding of the relationship between agent diversity and performance and learning in general.

4.4 Distributed Learning

While much of the multi-agent learning research is concerned with the question on how to learn local hypotheses that enable agents to collaborate, compete, and/or communicate more effectively, distributed learning is concerned with applying multi-agent techniques to learn a global hypothesis given local and distributed learning agents.[2]

As an example for distributed learning, consider a robotic soccer domain, where in the absence of a global view the computation of an opposing team's strategy has to be based on local observations of the individual players. It would be infeasible to simply send all these observations to a super-agent that learns from them, and therefore the agents need to first generalize from their observations and then share the results in order to compute the global strategy of the opposing team. We are currently working on algorithms to achieve this task.

Weiss [45] distinguishes three types of multi-agent learning:

Multiplied Learning: Each agent learns independently of the other agents. While there may be interactions concerning the exchange of training data or outputs, no interference in the learning process itself takes place.

Divided Learning: The learning task is distributed amongst a team of agents. The division takes place on a functional level, i.e., agents take different roles in a team and learn them separately.

Interacting Learning: Agents interact during learning and cooperate in generating a hypothesis beyond the pure exchange of data. Weiss describes this as a "cooperated, negotiated search for the solution of a learning task".

Most multi-agent learning systems fall into the first two categories. The third category describes "true" distributed learning, and very little research exists in this area.

Provost and Hennessy [34] describe a method for parallel learning where learning agents receive different subsets of the training data, learn a local hypothesis, and generalize from this a global hypothesis. An agent A checks the validity of a learned rule by sending it to another agent B in order to compute the accuracy on B's dataset. The most valid rules form the final hypothesis.

Other approaches related to distributed learning include Weiss' ACE and AGE algorithms [44] where a group of Q learning agents jointly decide on the actions via a bidding scheme and then jointly receive reinforcement for the executed actions.

5 Inductive Logic Programming and Agents

Inductive Logic Programming (ILP) is a branch of machine learning built on three pillars: *Logic Programming* (LP) as the representation formalism for both

[2] Of course, the distributed learning process may still lead to more effective coordination, but the learning metaphor is different.

training examples and theories learned, *background knowledge* in the form of predicates, which extend the concept language, *i.e.*, the one used to express the target concept, and, finally, *induction* as a method of learning.

5.1 Logic Programming as Object and Concept Language

In ILP, both the *object language* used to represent examples of the target concept, and the *concept language* used to represent the concepts (theories) learned are based on first order logic or, more precisely, on logic programming. That means that all facts and concepts are represented as predicates of first order logic. In most ILP systems, there is one target predicate only. Also, the training examples are most often represented as ground facts, *i.e.*, the object language is restricted to propositional logic in the very same way as in propositional learners such as ID3. In other words, the training data can be represented as a single table, the name and columns of which correspond to the name and attributes of the target predicate (see Table 3).

What makes ILP different from the propositional learners is the concept language used. For that purpose, ILP uses relations expressed as Horn clauses. For instance, the concept of equality of two arguments can be expressed by the clause `equal(X,X)`; to say that Argument 1 is greater than Argument 2 one writes `greater(X,Y):- X > Y`. In propositional logic, one cannot use variables to define relations between target concept arguments, so the above examples have to be covered by explicit enumeration of all pairs of `X` and `Y`, a task which is tedious if the range of the two arguments is finite, and impossible otherwise.

ILP systems may require the use of *modes* and *types*. *Modes* serve to distinguish between the input arguments of the predicate, *i.e.*, those which will be instantiated in a query, and the output arguments which will be instantiated as a result of the query. The former case is usually indicated by a plus '+', and the latter by a minus '–'. Systems such as Progol allow the user to indicate that in the clauses of the target predicate to be learned an argument will be instantiated to a constant by marking that argument with the hash symbol '#'.

Types specify the range of values for each argument, and are usually defined by user-defined unary predicates; some systems provide built-in types, such as `nat`, `real`, and `any` in Progol. It is important whether these predicates can only be queried with an instantiated argument (*is* 5 *a natural number?*) or are generative, allowing for the sampling of the type range (*give me a (random) natural*

Table 3. Training data representation

Good bargain cars				ILP representation
model	**mileage**	**price**	**y/n**	`:- modeh(1,+model,+mileage,+price)?`
BMW Z3	50,000	£5000	+	`gbc(bmw_z3,50000,5000).`
Audi V8	30,000	£4000	+	`gbc(audi_v8,30000,4000).`
Fiat Uno	90,000	£3000	–	`:- gbc(fiat_uno,90000,3000).`

number). This importance is related to the type of training examples available. If both positive and negative examples of the target concept are provided, the types are only used to mark which variables in the Horn clause can be unified, restricting in this way the target concept search space. On the other hand, generative type definitions are required for positive-only learning in the ILP system Progol [27]. The reason for that requirement becomes clear after a closer look at the mechanism underlying that type of learning.

There are several situations when positive-only learning would be the most natural setting: when a child learns to speak, one could hardly imagine the parents providing examples of ill-formed sentences; another example is the case of an animal or agent learning the definition of "non-lethal action". If the search for the target concept is only guided by the accuracy as tested on the training examples, the trivial definition classifying any entry as a positive example would cover all positive examples, and no negative ones (since none are provided). The (quite ingenious!) solution provided to that problem in Progol is to generate *random examples* of the target predicate by sampling the type range of each attribute, and then treat these examples as negative. This approach proves successful when combined with a learning bias based on a trade-off between the theory size (the shorter, the better) and its coverage (the more positive and the fewer random examples covered, the better).

5.2 The Importance of Background Knowledge

In the examples of `equal/2` and `greater/2` as predicates that ILP can learn, only very simple mathematical relations such as identity and "greater than" have been used. In fact, the concept language of propositional learners can easily be extended to include these relations. The truly unique feature of ILP is that one can define background knowlegde — a set of relations (predicates) that can be used in the definition of the target concept. The use of certain background predicates may be a necessary condition for finding the target concept. On the other hand, using redundant or irrelevant background predicates slows the learning down.

For instance, one can use the predicate `imported_model/1` to search for a potential correlation between that feature and the price of a car; in case of large import taxes that predicate may prove quite useful. This is still an example where all information used in learning can be represented in a single table by adding an extra attribute to it. The target concept, in its simplest case (`imported_car=no`) is also propositional. To show the advantage of using ILP, imagine that the product of the car mileage and price was a good indicator of its being a good bargain. One could then define a background predicate `prod(Miles,Price,Threshold) :- Miles*Price < Threshold` and use it with modes `:- modeb(1,prod(+Miles,+Price,#Threshold))?` to learn the following theory from the data in Table 3:

```
gbc(Model,Miles,Price):-
      prod(Miles,Price,250000001).
```

One can also allow the target concept to be recursive when appropriate or define constraints such as "no car is a good bargain if one can buy the same model and mileage for less". The latter can be expressed in Progol as:

```
prune(gbc(Model,Miles,Price),Body) :-
      Body, gbc(Model,Miles,Price2), Price2 < Price.
```

5.3 Inductive Learning

Rather than proving clauses starting from some general theory, as other LP systems, such as Explanation-Based Learning, do [22], ILP systems try to find a general theory that can explain the given clauses. While resolution is generally used for automated deduction, there are many ways to perform induction. Some of them can be generally called *inverse resolution* [29], since they can be derived from applying the resolution step in the reverse way. There are also other induction techniques, such as least general generalisation (lgg) [33].

5.4 ILP Learning for Agents

The description so far shows ILP as symbolic, knowledge-oriented learning which can explore complex hypotheses and represent them in a concise way. Here the discussion of ILP continues with a number of implementation-specific issues, and their relevance to the use of ILP in agents.

Learning Pure Logic Programs *vs.* Decision Lists ILP algorithms can be divided into those that learn theories expressed as pure logic programs (FOIL [35], Golem [30], and Progol [26] fall into that group), and others which include extra-logical predicates. A concept language including the predicate *cut* (!) allows for the learning of *first order decision lists*, in which each clause (but the last) ends in a cut. In pure logic programs, the clause order is irrelevant. However, when a decision list is queried, the list is searched in a top-down fashion, and only the first applicable clause is used. The intuitive notions of rule and exception are easily represented with decision lists: if the exception preceeds the rule, the latter does not have explicitely to mention that it does not cover the former. A decision list example follows in Table 4. The example models the behavior of a cat, which is afraid of all dogs but the ones that belong to its owner, and has no fear of other animals.

Theories expressed as decision lists are often simpler then their corresponding pure LP representation. Decision lists can be useful to integrate previously learned theories with newly acquired training examples. If the examples are misclassified by the existing theory, they can be added as exceptions at the top of the decision list to ensure their correct classification. Later on, learning can be used to replace these exceptions with rules, if possible. FOIDL [24] and CLOG [15] are two of the first order decision list learners. It is also worth mentioning that CLOG, unlike Progol, is an incremental learner.

Table 4. Example of decision lists

```
%action(Cat,ObservedAnimal,Action)
action(Cat,Animal,stay) :-
   dog(Animal),
   owner(Owner,Animal),
   owner(Owner,Cat),!.

action(Cat,Animal,run) :-
   dog(Animal),!.

action(Cat,Animal,stay).
```

Eager ILP *vs.* Analogical Prediction The relative merits of eager and lazy learning have already been discussed. ILP belongs to the eager learning paradigm, with the sole exception of Analogical Prediction (AP) [28]. AP constructs a separate theory (first-order Horn clause) to classify each test example; at the same time, that theory provides an explanation of the decision made, which is uncommon for lazy learning. AP would be suitable in the context of a rational learner that has to provide motivation for its actions. However, the time complexity of the method can present a serious obstacle to its larger use.

Single-predicate *vs.* Multi-predicate Learning When learning is judged in terms of predictive accuracy, the use of additional training examples can only be beneficial, if there is no noise in the data. However, in agent learning one should also be concerned about the time complexity of the theory learned, *i.e.*, how long it takes to query that theory. Since ILP learns logic programs, which can be recursive or allow for a lot of backtracking, the complexity of these programs can easily become an issue. The only experimental results known to the authors involve bottom-up ILP based on *lgg* [13], p. 117. In those experiments, the curve representing the relationship between the number of training examples, and the average time needed to prove a goal or reject it within the theory learned, is bell-shaped rather than linear. To avoid learning theories of high time complexity, an agent could content itself with using a simple, low-coverage theory until there are enough additional training examples to produce a theory the complexity of which is beyond the critical point.

Existing Work ILP has been applied to a number of tasks related to agents. For instance, Džeroski et al. have combined the ILP system TILDE-RT with reinforcement learning [7] for the task of learning actions in a simple world of blocks. The Q-function is learned in the form of a logical regression tree, *i.e.*, a binary tree in which each non-leaf node corresponds to a condition expressed as a logic clause, and is split into two branches labelled 'yes' and 'no', whereas leaf nodes are labelled with Q-values.

Reid and Ryan have used ILP to improve planning in hierarchical reinforcement learning , in which 'optimality is traded for speed hoping to find good solutions quickly by breaking a monolithic reinforcement learning task into smaller ones and combining their solutions" [36]. Rather than learning a Q-tree, here ILP is used to learn descriptions for subsets of the state space.

Matsui et al. have proposed to develop an ILP agent that avoids actions which will provably fail to achieve the goal [20]. The suggested application area is the one of RoboCup, a robot football tournament with world-wide popularity.

Another ILP application in the same domain provides the agent with a way of analyzing its behavior and summarizing (in a decision tree) the rules that are implicitely present. Declarative specifications of the software implementing the agent's behavior are obtained, and compared with the intentions of the agent's designers to help the debugging process [11].

A Progol-based algorithm for learning action theories of a single agent (mobile robot) is described by Lorenzo and Otero [19].

Alonso and Kudenko [1,18] investigate the application of ILP and EBL to complex multi-agent domains such as conflict simulations. The learning is based on reinforcement and recomputation of hypotheses, once they become out-of-date.

5.5 The York Multi-agent Environment

Some of the ideas discussed so far can be studied on a system that is being developed at the University of York [14]. The system is intended to be a general purpose Java-based platform for the simulation of learning and natural selection in a community of agents. The system draws a parallel with the world of nature. It provides tools for the specification of a two-dimensional environment with a range of features, such as different types of terrain, food and water resources. The user can edit to a large extent the specification of each of a number of *species* of agents—what they eat and fear, what types of sensors they possess, in which terrains they can hide and walk.

Single Agent Learning Each of the agents can be specified to have a Progol "mind", *i.e.*, it can communicate with a dedicated Progol process (see figure 1). All observations collected by the agent's sensors are sent to that process as (ground) logic clauses; Progol on its turn can send back to the agent directions about its behavior. This provides a framework for single agent learning in a multi-agent environment. Learning is not limited to ILP, as the implementation of Progol includes a full-scale Prolog interpreter, which can be used for the implementation of various learning techniques. Also, both learning and recall can be carried out in an integrated way, as the newly learned clauses can be added to the Progol database. A new, multi-agent Progol is being developed at present, which, when available, will allow the agents to query each other's databases [25].

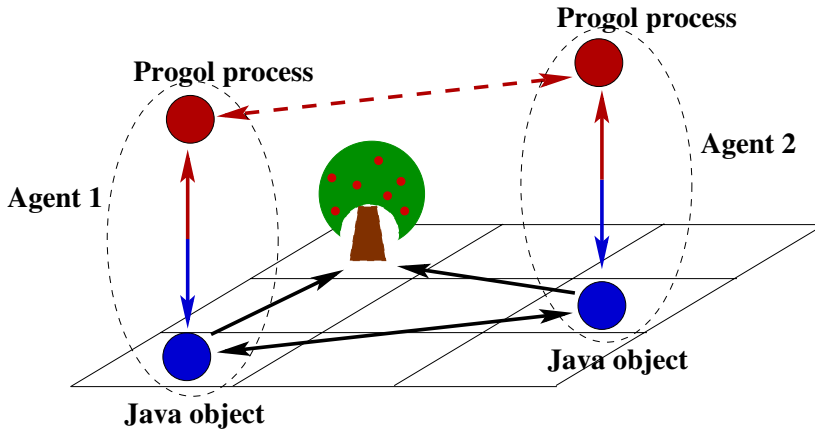

Fig. 1. Agents with a Java body and Progol mind

Synchronisation and Time Constraints To simplify the debugging and allow for easier analysis of the agents' behavior, the coordination between agents is implemented in the "One-step-at-a-time simultaneous update of environment" fashion, rather than having agents implemented as separate threads. Also, the system can easily be modified to wait indefinitely until Progol responds with an answer (selects the next action) or to allocate each agent a limited amount of time to make its move or be ignored until the next round.

Default Behavior The agents have a default, hard-coded behavior, which can be used as a baseline reference in the evaluation of learning agents or as a fallback plan if the adaptive behavior modified by learning cannot decide on an action within the required time limit. The behavior is based on the notion of "drives", which represent the intensity of each of the agent's basic needs. There are four such drives in the current implementation: hunger, thirst, fear, and sex drive. At each step, the drives are evaluated, and an action is taken to reduce the one with the highest intensity. A snapshot of an ongoing experiment is shown in figure 2. The drives of the agent marked with a black dot are shown in the left-hand side of the screen. In this case, it is a herbivor surrounded by two predators. As a result, the prevailing drive is fear, and the next action of the agent will be chosen accordingly—the agent will attempt to run away.

Natural Selection It has been mentioned that the agents have a "sex drive". Indeed, the definition of each agent includes an array of integers, which can be used in combination with the built-in genetic operators crossover and mutation to implement some of the agents' parameters as inherited features. Since reproduction involves a fixed cost, namely a contribution to the initial energy level of

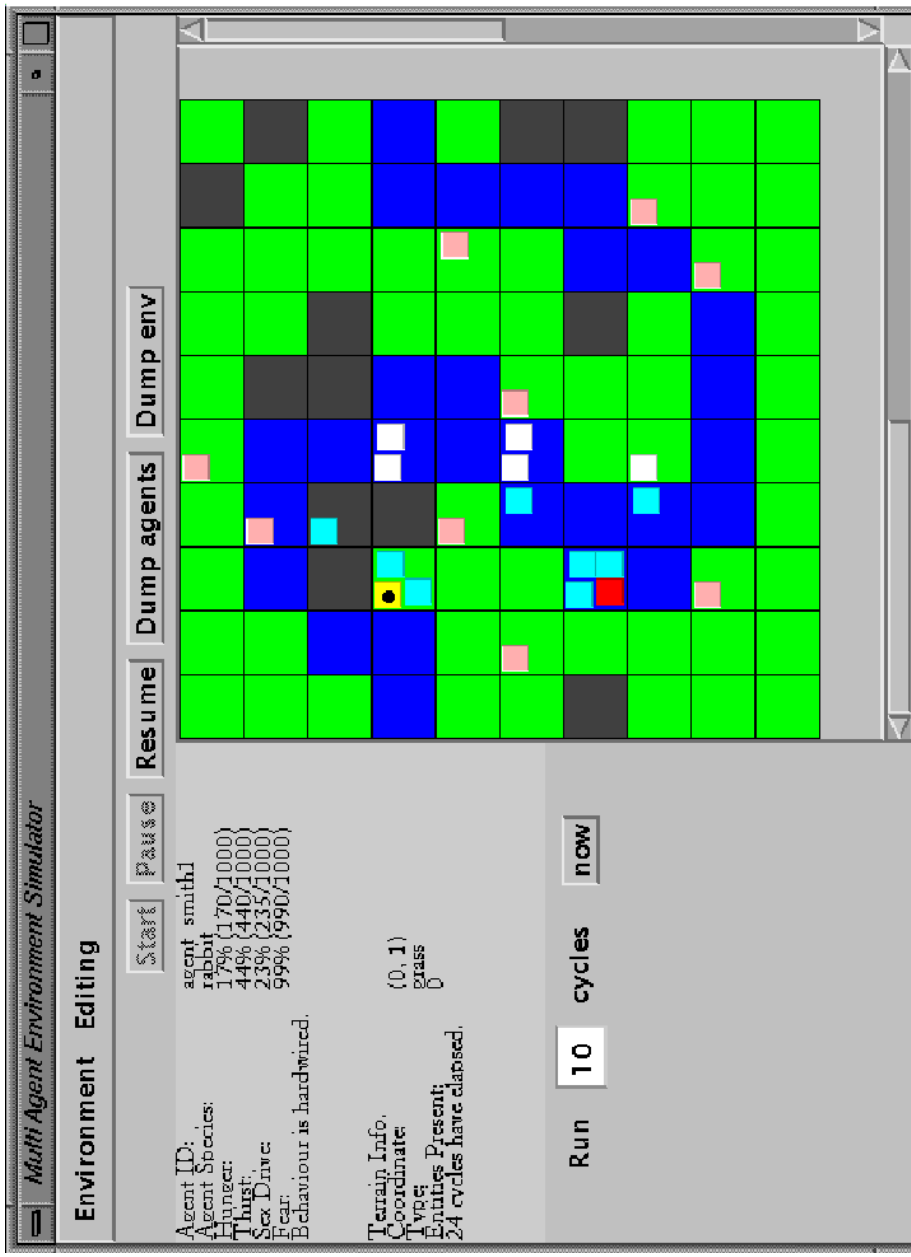

Fig. 2. A snapshot of the York multi-agent environment

the offspring, natural selection will put evolutionary pressure to select the sets of parameters which improve the agent's performance.

6 Conclusion and Outlook

We presented an overview of important issues in the application of machine learning algorithms to multi-agent systems (and vice versa), starting with a description of disembodies ML algorithms, moving to single agent learning, and finally multi-agent learning (MAL). Furthermore, we discussed the application of ILP, a logic-based learning technique to MAS.

The area of MAL is very young and there is still plenty to investigate. Here are some examples of interesting future work:

Formal models of MAL: To date most developers are not able to predict the behavior of learning agents and depend purely on observing emergent patterns. Formal models of MAL that can be used to predict (or at least constrain) the behavior of learning agents would be very useful. For a first approach see [10].

More complex applications: Most MAL application domains are relatively simple. It would be interesting to see MAL research for more complex, real-world applications. Eventually, such applications would encourage researchers to look beyond Q learning.

We hope that this paper will generate interest in multi-agent learning and encourage new researchers to look into the open issues.

References

1. E. Alonso and D. Kudenko. Machine learning techniques for adaptive logic-based multi-agent systems. In *UKMAS '99*, Bristol, UK, 1999.
2. T. Balch. Behavioral diversity as multiagent cooperation. In *SPIE'99 Workshop on Multiagent Systems*, 1999.
3. T. Balch. Hierarchic social entropy: an information theoretic measure of robot team diversity. *Autonomous Robots*, July 2000.
4. C. H. Bryant and S. H. Muggleton. Closed loop machine learning. Technical Report YCS 330, University of York, Department of Computer Science, Heslington, York, YO10 5DD, UK., 2000.
5. W. Daelemans, A. V. D. Bosch, and J. Zavrel. Forgetting exceptions is harmful in language learning. *Machine Learning*, 34:11, 1999.
6. T. Dietterich and N. Flann. Explanation-based learning and reinforcement learning. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 176–184, 1995.
7. S. Džeroski, L. De Raedt, and H. Blockeel. Relational reinforcement learning. In D. Page, editor, *The Eighth International Conference ILP-98*, pages 11–22, Madison, Wisconsin, USA, 1998. Springer-Verlag.
8. K. Furukawa, editor. *The First International Conference on Discovery Science*, LNCS, Fukuoka, Japan, 1998. Springer-Verlag.

9. G. Gilbert and R. Conte (Eds.). *Artificial Societies: The Computer Simulation of Social Life*. UCL Press, London, 1995.
10. D. F. Gordon. Asimovian adaptive agents. *Journal of Artificial Intelligence Research*, 13:95–153, 2000.
11. N. Jacobs, K. Driessens, and L. De Raedt. Using ilp systems for verification and validation of multi-agent systems. In D. Page, editor, *The Eighth International Conference ILP-98*, pages 11–22, Madison, Wisconsin, USA, 1998. Springer-Verlag.
12. L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
13. D. Kazakov. *Natural Language Applications of Machine Learning*. PhD thesis, Czech Technical University, Prague, Czech Republic, January 2000.
14. D. Kazakov, L. Mallabone, and S. Routledge. Implementing natural selection and symbolic learning in a multi-agent environment. Manuscript.
15. D. Kazakov and S. Manandhar. Unsupervised learning of word segmentation rules with genetic algorithms and inductive logic programming. *Machine Learning*, 2001. Forthcomming.
16. H. Kitano, Y. Kuniyoshi, I. Noda, M. Asada, H. Matsubara, and E. Osawa. Robocup: A challenge problem for AI. *AI Magazine*, 18:73–85, 1997.
17. M. Klusch (Ed.). *Intelligent Information Agents*. Springer-Verlag, 1999.
18. D. Kudenko and E. Alonso. Logic-based multi-agent systems for conflict simulations. In *UKMAS '00*, Oxford, UK, 2000.
19. D. Lorenzo and R. P. Otero. Using an ilp algorithm to learn logic programs for reasoning about actions. In J. Cussens and A. Frisch, editors, *Work-in-Progress Reports of ILP-2000*, pages 163–179, London, UK, 2000.
20. T. Matsui, N. Inuzuka, and H. Seki. A proposal for inductive learning agent using first-order logic. In J. Cussens and A. Frisch, editors, *Work-in-Progress Reports of ILP-2000*, pages 180–193, London, UK, 2000.
21. R. S. Michalski, I. Bratko, and M. Kubat, editors. *Machine Learning and Data Mining - Methods and Applications*. Wiley, 1998.
22. T. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1:4–80, 1986.
23. T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
24. R. J. Mooney and M. E. Califf. Induction of first–order decision lists: Results on learning the past tense of English verbs. *Journal of Artificial Intelligence Research*, June 1995.
25. S. Muggleton. Personal communication.
26. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.
27. S. Muggleton. Learning from positive data. *Machine Learning*, 1998.
28. S. Muggleton and M. Bain. Analogical prediction. In *Proc. of the 9th International Workshop on Inductive Logic Programming (ILP-99)*, pages 234–244, Berlin, 1999. Springer-Verlag.
29. S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference of Machine Learning*, pages 339–352. Morgan Kaufmann, San Mateo, CA, 1988.
30. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, Tokyo, 1990. Ohmsha.
31. S. Natarajan. *Imprecise and Approximate Computations*. Kluwer, 1995.
32. L. Parker. *Heterogeneous Multi-Robot Cooperation*. PhD thesis, MIT Department of Electrical Engineering and Computer Science, 1994.

33. G. Plotkin. A note of inductive generalization. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, 1970.
34. F. Provost and D. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI '96)*, 1996.
35. J. Quinlan. Learning logical definitions from relations. *ML*, 5:239–266, 1990.
36. M. Reid and M. Ryan. Using ilp to improve planning in hierarchical reinforcement learning. In J. Cussens and A. Frisch, editors, *The Tenth International Conference ILP-2000*, pages 174–190, London, UK, 2000. Springer-Verlag.
37. S. Sen and G. Weiss. Learning in multi-agent systems. In G. Weiss, editor, *Multi-Agent Systems: A modern Approach to Distributed AI*, pages 259–298. MIT Press, Cambridge, MA, 1999.
38. L. Steels. Cooperation between distributed agents through self-organization. In Y. Demazeau and J.-P. Mueller, editors, *Decentralized AI — Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*, pages 175–196. Elsevier Science, Amsterdam, 1990.
39. G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.
40. C. A. Thompson, M. E. Califf, and R. J. Mooney. Active learning for natural language parsing and information extraction. In *Proc. of the Sixteenth International Machine Learning Conference*, Bled, Slovenia, 1999.
41. URL: http://verbmobil.dfki.de/.
42. J. Vidal and E. Durfee. Agents learning about agents: A framework and analysis. In *Working Notes of the AAAI-97 workshop on Multiagent Learning*, pages 71–76, 1997.
43. C. Watkins and P. Dayan. Q learning. *Machine Learning*, 8:279–292, 1992.
44. G. Weiss. Learning to coordinate actions in multiagent systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI 93)*, pages 311–316, 1993.
45. G. Weiss and P. Dillenbourg. What is 'multi' in multiagent learning? In P. Dillenbourg, editor, *Collaborative Learning. Cognitive and Computational Approaches*, pages 64–80. Pergamon Press, 1999.