Relational Representations in Reinforcement Learning: Review and Open Problems

Martijn van Otterlo OTTERLO@CS.UTWENTE.NL
TKI - Department of Computer Science, P.O. 217, 7500 AE, Enschede

Abstract

This paper is about representation in RL. We discuss some of the concepts in representation and generalization in reinforcement learning and argue for higher-order representations, instead of the commonly used propositional representations. The paper contains a small review of current reinforcement learning systems using higher-order representations, followed by a brief discussion. The paper ends with research directions and open problems.

keywords: reinforcement learning, generalization, relational representations, review

1. Introduction

There is general agreement nowadays that intelligent agents should be *adaptive*, i.e. capable of learning. For learning to work, agents should be able to make the proper generalizations to reuse learned knowledge to apply it to new situations similar to encountered ones. Generalization though, or *inductive learning*, can be very difficult based on few examples.

To generalize based on examples one needs some form of *bias* for representing and organizing the possible hypotheses in a *hypothesis space* and some form of operators to go through the hypothesis space to find a suitable hypothesis that supports the experienced evidence. The set of modelling assumptions, representational language and hypothesis operators can be termed *inductive bias* (Mitchell, 1997). It guides the search for hypotheses consistent with the evidence and it restricts the number and structure of possible hypotheses. An important bias is the *language bias* that determines how concepts in the domain can be represented. Very roughly one has two choices; propositional and attribute-value (AV) representations, or relational (first-order) representations.

The AV approach has dominated the field of *Reinforcement Learning* (RL) (Sutton & Barto, 1998) ever since. Standard RL methods use an atomic, AV or propositional representation for the current state of the learner, and standard *Machine Learning* (ML) (Mitchell, 1997) techniques are used for generalizing experience. Popular approaches are *neural networks* and *decision trees* which are naturally based on propositional or AV representations.

Although this type of representation suffices for many applications, we argue that there is a need for relational representations. Relational representations reason over objects with properties and there are many possible relations between objects or types of objects. To give a simple example, imagine that we want to express a basic fact about a state such as 'there is a book on the table…', we can represent this by means of all the states where there is a book on the table. In this case we have to enumerate all these states and learn about each of them. In a propositional representation, we can express the fact by means of $\varphi(b_1) \vee \ldots \vee \varphi(b_n)$. In a first-order language we could express this state (or actually the set of states) by just the simple formula $\exists x \varphi(x)$. This last representation does not change if the number of domain objects is increased.

Recently, there is some interest in abstraction on different levels than generalization in statistical terms. Abstraction over time (Sutton et al., 1999) or primitive actions (Dietterich, 2000) are useful ways to abstract from specific sub-actions or time. Currently there is also interest in using higher-order representation *languages* in RL ((Kaelbling et al., 2001), (Dzeroski et al., 2001)). The use of subsets of the language of *first-order logic* (FOL) is desirable for its representational power, but it creates additional problems concerning tractability and learnability.

In this paper we would like to discuss some concepts and issues which underly using higher order representation languages in RL. We are specifically interested in RL for agents interacting with both humans and other agents in multi-modal, virtual worlds (Heylen et al., 2001). For these types of agents it will be necessary to represent and reason about the world in terms of cognitive notions like *beliefs*, *desires*, *intentions* and *emotions*. Also, the use of *natural language* communication is a central notion, both by means of speech and written text. Agents dealing with this rich information context should be able to represent their subjective view on the world in terms of *objects* and *relations* between these objects. Also, they should have beliefs about them and have ways of expressing their beliefs and experience in terms of looks (i.e. facial animation), language expressions and rational behavior. Thus, it would be highly desirable to learn on this same level of representation, i.e. in terms of *objects* (Kaelbling et al., 2001).

The outline of this paper is as follows. In section 2 we discuss briefly some concepts of RL. In section 3 we discuss generalization and representational issues in RL. Section 4 then gives a review of the main existing approaches in using higher-order representations in RL. Some preliminary comparisons and discussion of the approaches is given in section 5. This section also mentions directions for further research.

2. Reinforcement Learning

Reinforcement Learning (Sutton & Barto, 1998) (RL) is a powerful technique for learning in domains where there is no *instructive* feedback (like in supervised learning) but only *evaluative* feedback. Learning agents are trained by rewarding and scaffolding, expressed in terms of real numbers (*rewards*).

In short, an agent perceives a state s_t, decides on some action a_t, makes a transition from s_t to s_{t+1} and receives the reward r_t. The task of the agent is to maximize the total reward it gets while doing actions. Agents have to learn a policy which maps states into actions. Usually in RL this is done by learning *value functions*. A *state value* (V) function returns the expected total reward when the agent starts acting from that state. A *state-action value* (Q) function returns the expected total reward when the agent takes some action in the state and continues to follow its policy after that. State values are useful when a model of the environment is available. Many iterative algorithms for learning value functions exist (Sutton & Barto, 1998).

3. Representation and Generalization

Many interesting, if not all, problems have a large or even infinite state space. In that case, storing all the V- or Q-values is not feasible. Even if this would be possible, there still remains a problem because all these states would have to be visited many times to approximate their value which is not feasible with many states. Thus, some form of *generalization* is needed for generalizing experience over multiple state-action pairs. In the literature many methods have been used for *Value Function Approximation (VFA)*. Popular methods include neural networks, e.g (Grossmann, 2001; Bazen et al., 2001), and decision and regression trees, (Mitchell, 1997; Sutton & Barto, 1998).

For RL, learning a value function can be very difficult. Just like in a supervised learning task, the correct features have to be induced by the learning algorithm but because the value function appears to be unstable, the learning algorithm must be capable of tracking this moving target. Next to this problem, VFA in general cannot be proved to converge to the optimal value function. For the table case, many RL algorithms can be proved to converge in the limit.

The large majority of generalization mechanisms in RL is based on AV representations and statistical phenomena and geometrical distances are used to group states together. Typical AV learners carve up the input space by means of geometrical structures, hyperplanes, and group data points on the basis of distances in the input space. An intuitive name for these learning mechanisms is *fence-and-fill-learner* (Thornton, 2000). These mechanisms put *fences* in the input space, and *fill* the resulting parts virtually with similarly labeled data points. The underlying assumption of all the methods is that there are regularities shared by data points that are in the same area in the input space. The problem, however is that in many problems the regularity must be found in the *relation between attributes of the data point itself.* The most simple example is the *n*-parity mapping. To classify this data set with a nearest neighbor method, one has to create as many areas of *similarly labeled* data points, as there are data points. The regularity between data points is not helpful at all. To solve the problem more efficiently, one has to look for regularities in the data point itself, i.e. *relations* between attributes.

The problems that are solvable through exploitation of observable statistical effects in the input data (e.g. probabilities) can be termed *type*-1 and problems only solvable efficiently by exploiting relations between attributes *type*-2 (Clark & Thornton, 1997). Type-2 problems can be learned by AV learners only at the cost of using many resources. For example, although the value function for *Tic-Tac-Toe* can be stored by means of a typical AV learner such as a multi-layer perceptron, it is clear that the problem is of type-2. A game state can be judged by looking at only a few relations between board items. The n-parity task is an extreme example of a type-2 problem.

To deal with intrinsically relational (type-2) problems, Thornton (2000) mentions two possible methods for AV learners. The first is called *virtual supercharging* which essentially is *sparse coding* (Sutton, 1996). The input dimensionality of the input is severely enlarged, which makes it easier to find clusters in the data based on geometrical distances. The second method is called *supercharging* and allows the learning algorithm to create additional *fences* while learning. A more general term is *constructive algorithms* (Grossmann, 2001). Some of these methods have been specifically designed for reinforcement learning problems (Grossmann, 2001; Chapman & Kaelbling, 1991; Utgoff & Precup, 1998) containing special methods to deal with *concept drift*.

To deal more directly with relational (type-2) problems, we need to consider relational learning algorithms (Mitchell, 1997), generally known as *Inductive Logic Programming* (Dzeroski & Lavrac, 2001a). In relational learning, the hypothesis space contains logical formulae. A *more-general-than* relation between hypotheses as a partial order over this space. Examples can be stated in a relational language and learning consists of searching the hypothesis space for proper predicate definitions. The search process can be guided by providing *bias* in the form of *background knowledge* or search restrictions (*mode declarations*).

Besides the intrinsic properties of the data which can be a reason to use a relational learning representation for learning, there are a number of other reasons for opting for higher-order representations in RL:

• Almost all interesting problems are simply too large; at least some form of abstraction must be used. This can be done by means of AV representations as well, but FOL is more natural and compact

• Lookup tables, AV or propositional representations are not able to represent the structural aspects of states and actions. In *Tic-Tac-Toe* a board has spatial structure and special board positions, like that of *almost-wining* or *making-a-fork*, can be expressed as a relation between some parts of the board. If these structural aspects are useful, a representation that can represent these explicitly is favorable.

• The use of relational representations enables *knowledge transfer*. Learned knowledge can be reused for related tasks of a higher complexity, or as part of a larger task, because the operational knowledge is represented explicitly and can be manipulated as so.

• Relational representations allow for a more general and more intuitive way of specifying and using knowledge. A common knowledge representation in terms of FO knowledge is more suitable for doing this. Also, standard agent architectures and theories use subsets or extensions of FOL. (Modal) logics are common in agent modelling and design. (Wooldridge, 2002)

• As some argue (Kaelbling et al., 2001), a representation should enable representing and reasoning about *objects*. We argue that this is even more evident if we want to connect to existing agent architectures and programming languages. We have to be able to represent objects and relations in our language if we want to connect to higher-order *cognitive* notions like *beliefs*, *desires*, *intentions* and *emotions*.

• A last argument is more technical in nature. It is not necessary per se to use a representation language that supports objects and relations. We could *propositionalize* the domain by representing a state using a finite number of propositions. Three problems with this occur (van Laer & de Raedt, 2001): 1) one has to fix the number of objects, 2) one has to order the objects in the representation and 3) every possible relation should be present as a proposition in the representation, which can make it very large, even for small domains.

4. A Review of Relational RL

[Deictic Representations] From the perspective of being able to generalize, it seems likely to use relational representations. But, as was mentioned in section 2 much more experience exists with statistical learning with propositional representations. Finney et al. (2002b; 2002a) recently studied intermediate representations: *deictic representations* (DR). DRs are used in ordinary language, like *that-book-over-there* or *the-corridor-to-my-left* and have a semantics relative to the speaker (Kaelbling et al., 2001). DRs do not have the difficulties associated with relational representations, but do have some capability of generalizing over objects. What will happen with *the-thing-in-my-hand* if I drop it, is the same for all objects that I can have in my hand: it falls. The main advantage of DRs is that they avoid the arbitrary naming of objects.

The method uses so-called *markers* which can be placed upon objects and moved around. The study uses two types of DRs that differ in the amount of information the agent receives about the markers. Agents using these DR have a normal action set for acting in their environment, but additionally they have actions for moving the markers around like *move-focus(direction)* and *marker-to-focus(marker)*.

In DRs there is a substantial degree of *partial observability*: in exchange for focusing on only a few things, the agent loses the ability to see the rest. Propositional representations yield large observation spaces but full observability, while DRs yield small observation spaces but partial observability. Because in DRs it is no longer possible to observe the whole state, some history is included.

To experimentally study the possible advantages of DRs in RL, experiments are conducted in the blocks world, comparing the two types of DRs with a full-propositional representation. The task was to pick up some green block. Two types of propositional learning algorithms are used. A multi-layer perceptron (MLP) representation (one for each action) is trained with SARSA(λ) and also the G-algorithm (Chapman & Kaelbling, 1991), an incremental tree learner, is used. The experimental results show that when using MLPs the full-propositional representation outperforms the

DRs in terms of total reward per trial, but the opposite is true when using the G-algorithm. However, when using the G-algorithm, none of the representations reaches the performance level of the MLP approach, unless the trees are allowed to grow very large. Although the authors' first idea was that this could be explained by the longer action sequence that is needed in DRs, further experimentation shows that the actual cause is the exploration process. In a DR the location of the focus is crucial. The larger the domain is, the more locations a marker can be placed upon and this creates a large exploration space. This can be solved to some degree by adapting the action set to the very specifics of the task being solved but this degrades the flexibility of the DR approach. It seems that an action set that includes the ability for the agent to control its own attentional focus inherently increases the difficulty of the exploration problem by allowing it to easily spend a lot of time exploring a useless part of the state space.
One of the conclusions by the authors is that the experiments and the analysis show that although DRs have been shown to be useful to some extent, in the end none of the approaches for converting an inherently relational problem into a propositional one seems like it can be succesful in the long run. Naive propositionalization grows in size and is very redundant. DR has the problem that it results in inherently long early trials because of the exploration involving the markers. Furthermore the partial observability of DRs involves some extra problems. According to the authors the experiments show that it is almost inevitable to move towards relational representations.

[Relational Reinforcement Learning] *Relational Reinforcement Learning* (RRL) (Dzeroski et al., 2001) is a combination of RL and *Inductive Logic Programming* (Dzeroski & Lavrac, 2001a). It combines a standard RL algorithm (Q-learning) with a relational regression algorithm, TILDE-RT (Blockeel et al., 1998). TILDE-RT can be seen as a first-order extension of the C4.5 decision tree algorithm (Mitchell, 1997).
TILDE-RT is used as a representational tool to store the Q-function in a first-order logic decision tree, which is called a *Q-tree.* RRL collects experience in the form of state-action pairs with corresponding Q-values. During an episode, actions are taken according to the current policy, according to the current Q-tree. All newly encountered state-action pairs are stored, while the values of already encountered pairs are updated according to the Q-learning algorithm.
After each episode TILDE-RT is used to induce a first-order decision tree from the example set. The resulting tree contains nodes that are basically Prolog-queries. Different nodes in the tree can share variables. To find a Q-value one has to construct a Prolog KB with the tree, all the facts in the state, the action and the goal. Then, running the query ?-qvalue(Q) will return the desired value. An example of (part of) a Q-tree for *Tic-Tac-Toe* might be (left part):

```
action(move(Sq))?           |  qvalue(10) :-
                            |    winX(Sq), !.
winX(Sq)?                   |  qvalue(6) :-
 +-- yes: [10]              |    winO(Sq), !.
 +-- no:  winO(Sq)?         |  qvalue(3).
            +-- yes: [6]    |
            +-- no:  [3]    |
```

The decision trees can be transformed deterministically into a normal Prolog program (right part), which can be used as a Q-function. The tests in the nodes depend on the declarative bias and the tests in the nodes higher in the tree. Individuals are not referred to in the tree itself directly, but only through the variables of the goal.
Note that this decision tree represents the tree containing very few partitions. It is likely that an induced tree is much larger and uses other predicates, simply because different experience was acquired or the more complex predicates *winO* and *winX* are not present as background knowledge.
The P-RRL algorithm is an extension of the RRL algorithm that abstracts aways from Q-values and represents just the policy. In P-RRL after each episode first a Q-tree is induced, after which a so-called P-tree is induced which represents the optimality of state-action pairs. So essentially, whereas a Q-tree maps state-action pairs to Q-values, so does a P-tree map state-action pairs to *optimal* or *non-optimal.*
As was experimentally verified in the blocks world, RRL is capable of doing RL with a relational representation and it is able to transfer learned knowledge to related tasks of higher complexity (Dzeroski et al., 2001).
Although RRL is an effective tool for relational RL, it suffers from a number of problems. First, after each episode, a new Q-tree is induced from the examples, which is clearly not efficient. Second, the set of examples is constantly growing; all state-action pairs have to be memorized. Third, updating values for already experienced state-action pairs requires searching through the whole example set. Fourth, generalizing is not done in an efficient way. When updating a value for one state-action pair, the values of all pairs in that leaf should be updated, but in standard RRL only those pairs are updated that are experienced exactly again.
To solve these problems, an incremental algorithm was

proposed (Driessens et al., 2001). This algorithm is a first-order extension of the constructive, propositional tree building algorithm, the G-algorithm (Chapman & Kaelbling, 1991). The new algorithm, called TG, *incrementally* builds the same kind of trees as TILDE-RT. Additionally, in the trees, each leaf contains some statistics concerning a.o. Q-values and number of positive matches of examples. A node is split when it has seen enough examples and some statistical test on the node's statistics becomes significant with high confidence.

TG is much faster than the original RRL, but a problem is how to set the *minimal sample size* that determines when to split a node. A larger value means a smaller representation, but also slower convergence.

Recently, RRL was used for learning the computer game *Digger* (Driessens & Blockeel, 2001). In this game the player controls a 'digger' that can dig tunnels, has to evoid predators and collect valuable things. Learning the whole task is very difficult but there are two natural subtasks that can be distinguished. One is to avoid monsters or shoot them and the other is to collect gold. Usually in hierarchical RL (e.g. (Dietterich, 2000)) a hierarchical task decomposition consists of a *sequence* of subtasks, whereas in digger the two subtasks have to be performed in parallel.

Learning Digger with RRL is essentially a two-step process. First, for both subtasks a Q-tree is induced by learning on the isolated subtask. Learning to play the complete game is done again by means of RRL, where the Q-trees for both subtasks are given as background knowledge. In this way learning can make use of knowledge about both subtasks. RRL can add, subtract, compare and negate Q-values and in this way is able to learn even actions that are both suboptimal for each subtask but optimal for the whole task. While learning the whole task, RRL can use other features of the game, but as expected, prefers using the knowledge from the Q-trees.

[Symbolic Dynamic Programming] The *Situation Calculus* (SC) (Reiter, 2001) is a system for specifying and implementing dynamical systems, based on FOL. The logic used is based on three typical constructs in the language, *actions*, *situations* and *fluents*.

Actions are first-order terms consisting of an action symbol and some arguments. Examples of actions are for example *open(door)* and *goTo(x,y)*.

A *situation* is a first-order term denoting a sequence of actions involving the operator *do*. The SC uses one special situation S_0 which is the *initial situation*, an empty sequence of actions. For example, the situation term $do(action_3, do(action_2, do(action_1, S_0)))$ actually denotes the sequence of actions: $[action_1, action_2, action_3]$

The third ingredient, the *fluents*, are relations or function symbols of which the extensions can vary in different situations. So, *relational* and *functional fluents* are situation-dependent and have therefor as last argument a situation term. The fluents can be viewed as the variables in the system. For example *open(door,s)* is true if the door denoted by *door* is open in situation s, and *president(UnitedStates,s)* = *Bush* if s denotes January 2002.

Formalizing a domain consists further of specifying the so-called *action precondition axioms* (apa) and *successor state axioms* (ssa). The former is of the form $Poss(A(\vec{x}, s)) \equiv \Pi_A(\vec{x}, s)$ where $\Pi_A(\vec{x}, s)$ gives the preconditions for action A. The latter is of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$ which characterizes *completely* the truth values of the fluent F in the next situation $do(a, s)$.

Regression of a formula ϕ *through an action a* is a formula ψ that holds prior to a being performed iff ψ holds after a. Successor state axioms naturally support regression because of the way they are defined. The outcome of the regression of $F(\vec{x}, do(a, s))$ is given by $\Phi_F(\vec{x}, a, s)$ itself.

The first *Symbolic Dynamic Programming* (SDP) algorithm was only recently developed (Boutilier et al., 2001). This method solves a first-order MDP specified in the SC and produces a *logical description* of the optimal value function and policy.

The language of the SC is extended with stochastic actions that are mapped onto a number of normal SC actions, following some probability distribution. For example, we can have a stochastic action a and two primitive actions *aFail* and *aSucc*, with corresponding probabilities 0.1 and 0.9. In this way it is represented that in 90% of all cases, action a succeeds. This can be expressed by a case statement:

$$pCase(a) = [aSucc, 0.9, aFail, 0.1]$$

In the formalization of the domain, these probabilities have to be given, and for each primitive action, action effect and successor state axioms have to be specified. The algorithm itself uses the fact that, in general, from a V-function, one can induce a Q-function if one knows the model of the environment , and vice versa. In short, the algorithm starts with an initial partition of the state space which is determined by the goal states and their values. Then subsequentially a number of new partitions is created by using logical *regression* on actions. The algorithm ends with a symbolic description of the value function.

Given a value function V for states, the classical expression for Q^V is given by:

$$Q^V(A(\vec{x}, s) = R(s) + \gamma \cdot \{\sum_{t \in S} \Pr(s, A(\vec{x}, t)) \cdot V(t)\}$$

The FO definition is given by means of regression:

$$Q^V(A(\vec{x},s) = rCase(s) \oplus$$
$$\gamma \cdot [\oplus_j \{pCase(n_j(\vec{x}),s) \otimes Regr(vCase(do(n_j(\vec{x}),s)))\}]$$

The important notion which derives from this formula is that by treating stochastic action as *deterministic nature's choices* it is allowed to use regression directly to derive *properties of the pre-action state that determine the value-relevant properties of the post-action state.* So, from a logical description of V we can derive a logical description of Q. If the formulas in the case statement for V form a *partition* then so does the case statement for Q. From the new Q-partitioning, we can derive a new partitioning for V by taking the best actions. So, we can start for example with the following partitioning according to the goal states (partition, value):

$$case[xWon(s), 10; oWon(s), 0; draw(s), 5]$$

Now, take the following shortcuts:

$$W^x \equiv \exists q.winForX(q,s)$$
$$W^o \equiv \exists q.winForO(q,s)$$

Then a resulting partition could be:

$$case[W^x, 10; \neg W^x \wedge W^o, 6; \neg(W^x \vee W^o), 3]$$

which resembles the Q-tree from the previous section. The SDP algorithm is capable of generating a logical description of the value function for a FO-MDP without explicit state enumeration. To scale up better methods for logical simplification are needed for the method relies heavily on the simplification of partition descriptions.

[**Learning Background Knowledge**] In the previous two sections we have seen complete algorithms for using relational representations in RL domains. The background knowledge used by RRL and SDP can be specified *a priori*, but some of this knowledge can be learned as well. Karimi and Hamilton (2000) use a modification of the C4.5 decision tree algorithm to induce rules like *class(A1,X1,0) :- A1=2, X1=1* which specifies that the next value of variable X1 is 0 if the action A1 is 2 and the value of X1 was 1. Lorenzo and Otero (2000) use a modified version of Progol (an ILP algorithm) to learn action theories for a Situation Calculus domain. Action effect learning in a robotic soccer domain by means of ILP was considered by Matsui et al. (2000).

5. Conclusion

From the previous section we have seen efforts to move beyond propositional representations in RL. Deictic representations are an intermediate step and make smart use of the available knowledge about RL with propositional representations, but are shown to be too limited compared to relational representations. For relational representations we discussed a model-based method (SDP) and a model-free method (RRL). Both methods can be considered as the *core* of the field of relational RL today.

Both RRL and SDP partition the state space. SDP explicitly handles a case statement containing the formulae that make up the partition, while RRL delivers a Q-tree which implicitly represents a partition. The partition made by RRL is built up constructively with the only operation being to split a part of the partition into two parts. The problem with this is that unnecessary splits made in the beginning of learning cannot be made undone. SDP on the other hand, manipulates the partitions by means of logical conjunction and simplification. It combines partitions according to for example stochastic action decompositions and value partitions to create a new partition of the state space. With this, SDP can also merge parts of the state space and unnecessary splits do no occur. This is mainly due to the model.

Both DR and RRL use the G-algorithm, although RRL uses a FO extension of it. The combination of exploration problems with this algorithm generates massive growth of the decision tree. This is due because of unnecessary splitting of nodes. Many leafs have to be recreated. It is possible that the same problem occurs in TG as well. In general the decision when to split a leaf is very difficult and once a split is being made it cannot be reversed.

Both RRL and SDP use a relational language, but their representational power differs. RRL uses a restricted set of FOL containing *Horn Clauses.* SDP uses a many-sorted first-order language including quantification over variables. The representational power of SDP dominates that of RRL, but the latter does not require a model. Both methods can make use of background knowledge in the form of predicate definitions. SDP additionally requires successor state and action effect axioms.

New problems enter the RL process when we turn to relational representations. In RRL so-called *query packs* are used in order to make the process of query generation more efficient. In SDP special care is needed for the logical simplification module. Because the formulae making up a partition can become very complex syntactically, logical simplification is needed in order to keep them as small as possible.

For further research, first of all, research should be performed on the role of representation in RL. Much of the research in RL focuses on performance issues of *algorithms*, but more insight is needed in the interplay between representation, value functions and rational behavior. An interesting conceptual framework based on the psychological notion of *cognitive economy* was recently proposed by Finton (2002).

The need for higher-order languages and relational learning algorithms in RL is being operationalized by work in model-based RL (Boutilier et al., 2001) and model-free RL (Dzeroski et al., 2001). Even though these systems are now being developed, much has to be done to meet the demands in (Kaelbling et al., 2001). Fortunately much has been done in traditional RL already (Sutton & Barto, 1998), so maybe much of that can be upgraded towards higher-order representations (van Laer & de Raedt, 2001). Another two strategies are outlined in the systems that we discussed briefly. We can either use traditional RL algorithms with relational VFA as in RRL (Dzeroski et al., 2001) or we can work our way from existing specification languages or executable logics and extending these to deal with uncertainties and value functions as in SDP (Boutilier et al., 2001).

Upgrading traditional RL algorithms with new representational tools yields a large open space for further research. ILP methods were applied in RRL and in model learning (see section 3) and an upgraded version of the G-algorithm was used in the incremental extension of RRL, TG. Many more upgraded versions of representational tools that are traditionally propositional can be used as part of a relational RL system. For example, recent FO extensions of *neural networks* (Kijsirikul & Chongkasemwongse, 2001) and *Bayesian networks* (Getoor et al., 2001) can be used for RL. The direction of upgrading traditional learning mechanisms to the FO case in RL is part of a more general in Machine Learning (Dzeroski & Lavrac, 2001b). With the field of ILP being mature, learning algorithms should be capable of using knowledge representation schemes such as XML, ER-diagrams and relational databases, which are common practice in non-learning systems.

A second direction is to start from FO methods used for programming and specifying non-learning agents, and extend these methods with probability measures and value functions. This was done in the SDP algorithm (section 3) where the Situation Calculus was extended with means for specifying stochastic actions and value functions, after which a modified version of *value-iteration* was used with concepts from the SC itself, i.e. regression. In the agent literature (Wooldridge, 2002) one can find other executable logics like the SC framework by Reiter (Reiter, 2001) and possibly extend these for use in RL contexts. A problem is that although the semantics of extending standard logics with probability measures (Halpern, 1990) is relatively understood, finding efficient computational methods for these logics is still largely an open problem.

Our own work will investigate more in detail the semantics of relational representations in RL. We strive to incorporate RL into *Belief-Desires-Intentions* (BDI) reasoning agents (Rao & Georgeff, 1991; Wooldridge, 2002) for VR environments (Heylen et al., 2001).

References

Bazen, A., van Otterlo, M., Gerez, S., & Poel, M. (2001). A reinforcement learning agent for minutiae extraction from fingerprints. *Proceedings of the Belgium-Netherlands Artificial Intelligence Conference (BNAIC'01')* (pp. 329–336).

Blockeel, H., de Raedt, L., & Ramon, J. (1998). Top-down induction of clustering trees. *Proceedings of the 15th International Conference on Machine Learning* (pp. 55–63). Morgan Kaufmann.

Boutilier, C., Reiter, R., & Price, B. (2001). Symbolic dynamic programming for first-order MDP's. *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)* (pp. 690–697). San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)* (pp. 726–731). San Mateo, Ca.: Morgan Kaufmann.

Clark, A., & Thornton, C. (1997). Trading spaces: Computation, representation, and the limits of uninformed learning. *Behavioral and Brain Sciences*, *20*, 57–90.

Dietterich, T. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Driessens, K., & Blockeel, H. (2001). Learning digger using hierarchical reinforcement learning for concurrent goals. *Proceedings of the Fifth European Workshop on Reinforcement Learning, Utrecht, The Netherlands (EWRL-5).*

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. *Proceedings of ECML - European Conference on Machine Learning* (pp. 97–108). Springer-Verlag.

Dzeroski, S., de Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*, 7–52.

Dzeroski, S., & Lavrac, N. (2001a). Introduction to inductive logic programming. In S. Dzeroski and N. Lavrac (Eds.), *Relational data mining*, 48–73. Springer-Verlag.

Dzeroski, S., & Lavrac, N. (2001b). *Relational data mining.* Berlin: Springer.

Finney, S., Gardiol, N., Kaelbling, L., & T.Oates (2002a). *Learning with deictic representation*Technical Report AI MEMO 2002-006). MIT AI Lab, Cambridge, Massachusetts.

Finney, S., Gardiol, N., Kaelbling, L., & T.Oates (2002b). The thing that we tried didn't work very well: Deictic representations in reinforcement learning. *Proceedings of the 18th International Conference on Uncertainty in Artificial Intelligence, Edmonton, 2002 (UAI-02).*

Finton, D. (2002). *Cognitive economy and the role of representation in On-line learning.* Doctoral dissertation, University of Wisconsin, Madison.

Getoor, L., Friedman, N., Koller, D., & Taskar, B. (2001). Learning probabilistic models of relational structure. *Proc. 18th International Conf. on Machine Learning (ICML'01)* (pp. 170–177). Morgan Kaufmann, San Francisco, CA.

Grossmann, A. (2001). *Continual learning for mobile robots.* Doctoral dissertation, School of Computer Science, University of Birmingham, UK.

Halpern, J. (1990). An analysis of first-order logics of probability. *Artificial Intelligence*, *46*, 311–350.

Heylen, D., Nijholt, A., & Poel, M. (2001). Embodied agents in virtual environments: the AVEIRO project. *Proceedings European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems, Tenerife, Spain.*

Kaelbling, L., Oates, T., Hernandez, N., & Finney, S. (2001). Learning in worlds with objects. *The AAAI Spring Symposium.*

Karimi, K., & Hamilton, H. (2000). Learning with C4.5 in a situation calculus domain. *The Twentieth SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence (ES2000).*

Kijsirikul, N., & Chongkasemwongse, S. S. K. (2001). Approximate match of rules using backpropagation neural networks. *Machine Learning*, *44*, 273–299.

Lorenzo, D., & Otero, R. (2000). Using an ILP algorithm to learn logic programs for reasoning about actions. *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming* (pp. 163–171).

Matsui, T., Inuzuka, N., & Seki, H. (2000). A proposal for inductive learning agent using first-order logic. *Work-in-progress reports of ILP-2000* (pp. 180–193). London, UK.

Mitchell, T. (1997). *Machine learning.* New York: McGraw-Hill.

Rao, A. S., & Georgeff, M. P. (1991). Modeling agents within a BDI-architecture. *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)* (pp. 473–484). Cambridge, MA, USA: Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Reiter, R. (2001). *Knowledge in action: Logical foundations for specifying and implementing dynamical systems.* Cambridge, Massachusetts: The MIT Press.

Sutton, R., & Barto, A. (1998). *Reinforcement learning: an introduction.* Cambridge: The MIT Press.

Sutton, R., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems* (pp. 1038–1044). The MIT Press.

Thornton, C. (2000). *Truth from trash: How learning makes sense.* Cambridge, Massachusetts: The MIT Press.

Utgoff, P., & Precup, D. (1998). Constructive function approximation. In H. Liu and H. Motoda (Eds.), *Feature extraction, construction and selection: A data mining perspective*, vol. 453 of *The Kluwer International Series in Engineering and Computer Science*, chapter 14. Kluwer Academic Publishers.

van Laer, W., & de Raedt, L. (2001). *How to upgrade propositional learners to first orde logic: A case study*, vol. 2049 of *Lecture Notes in Articial Intelligence*, 102–126. Springer-Verlag.

Wooldridge, M. (2002). *An introduction to MultiAgent systems.* West Sussex, England: John Wiley & Sons Ltd.