A Survey of Reinforcement Learning in Relational Domains

Martijn van Otterlo otterlo@cs.utwente.nl
Human Media Interaction (HMI)
Department of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract

Reinforcement learning has developed into a primary approach for learning control strategies for autonomous agents. However, most of the work has focused on the *algorithmic* aspect, i.e. various ways of computing value functions and policies. Usually the *representational* aspects were limited to the use of attribute-value or propositional languages to describe states, actions etc. A recent direction – under the general name of *relational reinforcement learning* – is concerned with upgrading the representation of reinforcement learning methods to the first-order case, being able to speak, reason and learn about *objects* and *relations* between objects. This survey aims at presenting an introduction to this new field, starting from the classical reinforcement learning framework. We will describe the main motivations and challenges, and give a comprehensive survey of methods that have been proposed in the literature. The aim is to give a complete survey of the available literature, of the underlying motivations and of the implications of the new methods for learning in large, relational and probabilistic environments.

1. Introduction

Acting under uncertainty plays a major role in the study of sequential decision making problems and it has been addressed in such diverse fields as decision-theoretic planning, operations research, reinforcement learning and economics. The goal of acting optimally in a rational manner is a core problem in many fields more or less related to artificial intelligence. A large portion of the problems involving optimal sequential decision making can be formalized as Markov decision processes (MDP) (Puterman, 1994). For environments formalized in terms of MDPs, many *dynamic programming* techniques have been developed to maximize the expected utility of an acting agent. Advantages of dynamic programming techniques are that they can handle uncertainty, i.e. uncertain outcomes of stochastic actions. Typical dynamic programming techniques require a transition model of the environment in order to compute optimal behavior policies. Even when such models are not available, online, sample-based techniques under the general name of *reinforcement learning* (Sutton and Barto, 1998) exist to compute (optimal) policies. However, the *representational* capabilities of traditional models for MDP are limited in that they require all possible states of the environment to be represented explicitly. Considering the fact that most realistic problems have enormous state spaces, this is not feasible, and without abstraction or approximation techniques, dynamic programming does not scale up well.

In order to cope with the infamous *curse of dimensionality* (Bellman, 1957), i.e. the fact that the number of possible environment states grows exponentially in the number of features that are important for optimal behavior, in recent years many methods have been

proposed to abstract or approximate parts of the problem. Some of them abstract from details that are not important, or at least not important in some stages of the problem. Some methods use approximations to strike a balance between computational efforts to compute a policy and the quality of that policy. Yet other methods try to structure the learning problem by identifying different stages or different abstraction levels to view the task. Another set of methods tries to scale up to different, more powerful representation languages in order to specify the problem and compute policies in a more abstract form. It is especially this direction that we are concerned with in this paper.

The current state of the art in representation of MDPs is based on propositional languages (Boutilier et al., 1999; Boutilier, 1999). However, usually representational languages used in much of the literature on *artificial intelligence* (AI) (Russel and Norvig, 2003) as well as *intelligent agents* (Wooldridge, 2002) are based on *first-order logic*. Furthermore, much of the planning literature assumes a first-order logic in which domains can be described in terms of *objects* and *relations*. The use of first-order logic enables powerful abstractions to be used in order to solve and reason over complex problem domains. This apparent discrepance between representations used in both contexts does not allow for much cross-fertilization between the fields.

Recently, there has been a rapidly growing interest in the use of first-order logical languages for modelling and solving MDPs. The idea is to upgrade the representation used in dynamic programming and reinforcement learning techniques to the first-order case such that the environment can be described more naturally in terms of *objects* and *relations* (see (Kaelbling et al., 2001; van Otterlo, 2002)). The starting point of this direction was given by Džeroski et al. (1998) who introduced the first method explicitly dealing with an MDP that was modelled in a relational language. With the combination of the most popular reinforcement learning algorithm Q-learning (Watkins and Dayan, 1992), the standard relational tree-learning algorithm TILDE (Blockeel and de Raedt, 1998) and one of the most intuitive relational domains, the Blocks World (Slaney and Thiébaux, 2001), a strong case was made for the viability of reinforcement learning in relational domains.

The use of relational representations in reinforcement learning contexts offers many advantages. An important advantage is *generalization* across objects and possibly *transfer* of learned knowledge to different tasks in similar environments. For example, a policy learned for a logistics domain with 10 boxes will often quite naturally generalize to a domain containing more boxes. Furthermore, the use of relational representations enable the use of *background* (or: prior) knowledge in a natural way. This has been known very long in the inductive logic programming (ILP) (Muggleton and Raedt, 1994) community, and it can be used in relational reinforcement learning as well. Partial policies can sometimes be provided in logical form, and complex background predicates – when available – can be used in inducing powerful abstractions over states and value functions.

Since the seminal work by Džeroski et al. (1998) an increasing number of systems has been proposed in the literature. Although these systems use a wide variety of logical languages and representational devices, all have the same goal of solving MDPs that are specified over a relational domain. The general idea is to upgrade MDPs to relational contexts by using (first-order) AI formalisms [1] or to extend logical (agent) action languages

1. The so-called Good-old-fashioned-Artificial-Intelligence, or GOFAI approaches.

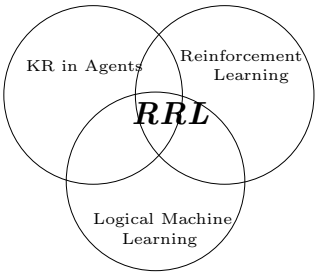


Figure 1: The field of Relational Reinforcement Learning (RRL) is mainly formed by the combination of i) the traditional field of *reinforcement learning* and *Markov decision processes*, ii) knowledge representation and action languages used in *planning*, *agents* and *artificial intelligence* and iii) the field of logical machine learning algorithms (such as *inductive logic programming* and its recent extension *probabilistic logic learning*).

with means to express probabilities and utilities. Many of the earlier attempts dealt with specific learning techniques that were upgraded to the relational case, or with specific domains that were shown to be modelled naturally using a first-order logical language. More recently the field is becoming more mature, and theory is developing – such as concerning error bounds and convergence issues – and the added benefits and complexity is studied of reinforcement learning in highly structured and huge domains. The growing interest in relational reinforcement learning is also supported by recent events such as the *Relational Reinforcement Learning* workshop[2] at *ICML'04*, the *Rich Representations for Reinforcement Learning* workshop[3] at *ICML'05* and the *Dagstuhl* seminar [4] on *Probabilistic Logic Learning* where relational reinforcement learning was one of the special focus areas.

The field of *relational reinforcement learning* has many connections with other fields such as probabilistic planning, inductive logic programming, probabilistic logic learning and knowledge representation (see also Figure 1). In this survey we focus explicitly on learning in large, relational, probabilistic domains in which the environment is modelled as some kind of MDP. See for related surveys and overviews (Blythe, 1999; Boutilier et al., 1999; Weld, 1999) on (decision-theoretic) planning, (Aler et al., 2000) and (Boutilier, 1999) on knowledge representation in MDPs and (Sowa, 1999) for general knowledge representation, (Muggleton and Raedt, 1994), (De Raedt and Kersting, 2003) and (De Raedt and Kersting, 2004) on (probabilistic) inductive logic programming and furthermore (van Otterlo, 2002), (van Otterlo and Kersting, 2004) and (Tadepalli et al., 2004) for previous descriptions of relational reinforcement learning. Additionally, for more general overviews on the surrounding areas of reinforcement learning and machine learning, we refer the reader to (Puterman, 1994; Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998) and (Langley, 1996; Mitchell, 1997; Alpaydin, 2004) respectively.

2. http://eecs.oregonstate.edu/research/rrl/
3. http://www.cs.waikato.ac.nz/∼ kurtd/rrfrl/
4. http://www.dagstuhl.de/05051/index.en.phtml

In this survey we take a reinforcement learning perspective, which means that we identify various forms of *relational* MDPs and corresponding abstraction formalisms. We are particularly interested in all different ways standard elements in the MDP formalism – such as value functions, policies and transition functions – can be translated into a relational (or first-order) form. The description of the existing techniques will not be chronological, but instead using the conceptual layout of the standard RL literature in that we distinguish between model-free, model-based, hierarchical learning etc.

Outline. The first part of this paper introduces the standard Markov Decision Process framework. In Section 2 we will first introduce the basic setting and some standard algorithms for computing optimal value functions and policies. In Section 2.3 a number of techniques are described that aim at solving larger MDPs by using various forms of *abstraction*, such as *factored representations* and *hierarchies*. These methods are briefly mentioned because most work in relational reinforcement learning can be viewed as first-order upgrades of these types of abstractions and algorithms. The second part of the paper, starting in Section 3, describes the relational setting for MDPs and discusses motivations for relational representations. In this section also some relational abstraction formalisms are mentioned and ways to learn them. Section 4 contains the actual survey of existing approaches. In order, we will describe *intermediate*, *model-free*, *modelling* and *model-based* approaches. After that methods are described that *guide* or *bias* the learning process, and methods that are complementary for example because they learn transition models. The survey ends with methods that use hybrid techniques and approaches that use relational reinforcement learning in hierarchical and agent systems, and in multi-agent systems. Implications of the surveyed methods for the field of relational reinforcement learning are described in Section 5, accompanied by a number of challenges for further research. The paper ends with conclusions in Section 6.

2. Solving Markov Decision Processes

Markov Decision Processes (Puterman, 1994) are an intuitive and fundamental formalism for decision-theoretic planning (Boutilier et al., 1999; Boutilier, 1999), reinforcement learning (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998; Kaelbling et al., 1996) and other learning problems in stochastic domains. In this model, an environment is modelled as a set of states and actions can be performed to control the system state. The goal is to control the system in such a way that performance is maximized. Many problems such as (stochastic) planning problems, learning robot control and game playing problems have successfully been modelled in terms of an MDP.

2.1 Markov Decision Processes

A *Markov Decision Process* (MDP) is defined as a tuple $M = \langle S, A, T, R \rangle$, where S is a finite set of *states*, A is a finite set of *actions*, T is a *transition function* and R is a *reward function*. The system described by the MDP can be *controlled* using the set of actions. By applying action a in a state s, the system makes a transition from s to a new state s', based on a probability distribution over the set of possible transitions. The transition function T is defined as $T : S \times A \times S \rightarrow [0, 1]$, i.e. the probability of ending up in state

s' after doing action a in state s is denoted $T(s, a, s')$. It is required that for all actions a, and all states s and s', $T(s, a, s') \geq 0$ and $T(s, a, s') \leq 1$. Furthermore, for all states s and actions a, $\sum_{s' \in S} T(s, a, s') = 1$, i.e. T defines a *proper probability distribution over possible next states*. The set of actions that can be applied in a state s is denoted $A(s)$. The reward function R is defined as $R : S \rightarrow \mathbb{R}$. It attaches a *reward* to each state, i.e. a value that is obtained due to being in some state. State-action, or state-action-state reward functions can be defined similarly. However, for reasons of simplicity we will mainly work with state-based reward functions throughout the paper.

A distribution over possible start states is assumed, which defines the probability that the system will be initialized in some specific start state. Starting from state s the system progresses through a sequence of states, based on the actions performed. In *episodic* tasks, there is a specific subset $G \subseteq S$, denoted *goal state area* containing states where the process ends. This is usually modelled by means of *absorbing states*, e.g. states from which every action results in a transition to that same state with probability 1. When entering an absorbing state, the process is reset and restarts in a new starting state. In *infinite horizon tasks* the process does not necessarily stop and there is no designated goal area. The system being controlled is *Markovian* if the result of an action does not depend on the previous actions and visited states (history), but only depends on the current state (s_i and a_i are the state and action at time i), i.e. $T(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = T(s_{t+1} \mid s_t, a_t) = T(s_t, a_t, s_{t+1})$.

Generating a *solution* for a given MDP $M = \langle S, A, T, R \rangle$ consists of computing a policy that is maximizing the long-time reward sequence. A deterministic *policy* $\pi : S \rightarrow A$ for M specifies which action $\pi(s) = a$ will be executed when the agent is in $s \in S$. Every policy is associated with a *value function*. Let $V^\pi : S \rightarrow \mathbb{R}$ be the value function for a fixed policy π. For each state $s \in S$ the value $V^\pi(s)$ denotes the *expected cumulative reward that will be obtained by starting in state s and following the actions suggested by π*. In a discounted, infinite horizon MDP this is expressed by

$$V^\pi(s) = E_\pi[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0 = s] \tag{1}$$

Note that in this *discounted* case, rewards obtained later are discounted more than rewards obtained earlier. This is regulated by the *discount factor* γ. The discount factor ensures that – even with infinite horizon – the sum of the rewards obtained is finite. The discount factor can take any value between 0 and 1. In episodic tasks, i.e. in tasks where the horizon is finite, the discount factor is not needed or can equivalently be set to 1. The expression in Equation 1 can recursively be defined in terms of a so-called *Bellman Equation* (Bellman, 1957) for all states $s \in S$.

$$V^\pi(s) := R(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^\pi(s') \tag{2}$$

V^π is the unique solution for this set of equations. Note that multiple policies can have the same value function, but for a given policy π V^π is unique. One way to compute a solution is using linear programming problem with one equation per state (Puterman, 1994). Often an iterative algorithm is used. One starts with an initial arbitrary assignment of values for

V_0 and then in each iteration one estimates the $n+1$-*steps-to-go value function* V_{n+1} using the estimates of V_n. That is, for each state $s \in S$ the following value is computed:

$$V^{\pi}_{n+1}(s) := R(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^{\pi}_{n}(s') \tag{3}$$

In principle, in infinite-horizon cases, one has to iterate an infinite number of times. In practice, one can stop once the difference between two successive value functions is small. For example, a stopping criterion can be $\forall s \in S. |V^{\pi}_{n+1}(s) - V^{\pi}_{n}(s)| < \epsilon$, where V^{π}_{n+1} and V^{π}_{n} are subsequent value functions. We will address the issue of iterative value function estimation again in the following section on solution techniques.

The goal for any given MDP is to find a *best* policy, i.e. the policy that receives the most reward. This means maximizing the value function of Equation 1 for all states $s \in S$. An *optimal policy*, denoted π^*, is such that $V^{\pi^*}(s) \geq V^{\pi}(s)$ for all $s \in S$ and all policies π. It can be proven that the optimal solution $V^* = V^{\pi^*}$ satisfies the following Bellman Equation:

$$V^*(s) = R(s) + \gamma \max_{a \in A} [\sum_{s' \in S} T(s, a, s') V^*(s')] \tag{4}$$

Solving this set of equations can be done in an iterative manner, similar to the computation of the value function *for a given policy* such as expressed in Equation 3. That is, we can turn the previous Bellman optimality criterium into an update rule:

$$V_{n+1}(s) := R(s) + \gamma \max_{a \in A} [\sum_{s' \in S} T(s, a, s') V_n(s')] \tag{5}$$

To select an optimal action given the optimal state value function V^* the following rule can be applied:

$$\pi^*(s) := \arg\max_{a} [R(s) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s')] \tag{6}$$

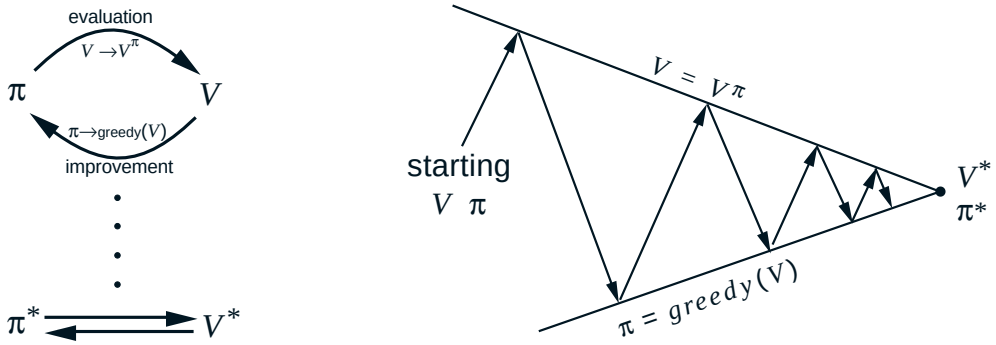
Analogously to state value functions, *action value functions* can be defined. A Q-function $Q : S \times A \rightarrow \mathbb{R}$ is a function mapping state-action pairs to values. The value $Q^{\pi}(s, a)$ is the value of performing action a in state s and following policy π afterwards.

Q-functions are useful because they make the weighted summation over different alternatives (such as in Equation 2) using the transition function unnecessary. No forward-reasoning step is needed to compute an optimal action in a state. This is the reason that in model-free approaches, i.e. in case one does not possess information about T and R, Q-functions are learned instead of V-functions (e.g. in reinforcement learning, see (Sutton and Barto, 1998)). The relation between Q^* and V^* is given by

$$Q^*(s, a) = R(s) + \gamma \cdot \sum_{s' \in S} T(s, a, s') V^*(s') \tag{7}$$

$$\text{and} \quad V^*(s) = \max_{a} Q^*(s, a) \tag{8}$$

Now, analogously to Equation 6, optimal action selection can be simply put as:



(a) Generalized Policy Iteration

(b) 2 processes

Figure 2: a) The algorithms from Section 2.2 can be seen as instantiations of a *Generalized Policy Iteration* (GPI) loop (pictures taken from Sutton and Barto, 1998). The *policy evaluation* step estimates V^π, the policy's performance. The *policy improvement* step improves the policy π based on the estimates in V^π. b) The gradual convergence of both the value function and the policy to optimal versions.

$$\pi^*(s) := \arg\max_a Q^*(s,a) \tag{9}$$

That is, the best action is the action that has the highest expected utility based on possible next states resulting from taking that action.

2.2 Solving Markov decision processes

Solving a given MDP means computing an optimal policy π^* that assigns the best action to every state in the state space. To achieve this, many *dynamic programming* and *reinforcement learning* methods exist. The basic distinction between these methods is whether they assume (perfect) knowledge about the transition function T and the reward function R. In the following two sections we will describe briefly some basic algorithms. The general form of all these algorithms consists of two processes, see Figure 2. One is the estimation of *values* for states and actions. The other is the improvement of *policies*

2.2.1 Dynamic Programming: Model-based Solution Techniques

Many model-based solution techniques known as *dynamic programming* (DP) exist. The main idea is to compute value functions and derive optimal policies from them, usually in an iterative way. Two classical DP are *policy iteration* and *value iteration*. We will briefly consider them in turn.

Policy iteration (PI) (Howard, 1960) starts with an arbitrary initialized policy π_0. Then a sequence of iterations follows in which the current policy is evaluated after which it is improved. The first step, the *policy evaluation*, step computes V^{π_k}, making use of Equation 5 in an iterative way. The second step, the *policy improvement* step, computes π_{k+1} from π_k

Algorithm 1 Policy Iteration (Howard, 1960)

Require: $V(s) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$
  {Policy Evaluation}
  repeat
    $\Delta := 0$
    for each $s \in S$ **do**
      $v := V(s)$
      $V(s) := R(s) + \gamma \cdot \sum_{s'} T(s, \pi(s), s')V(s')$
      $\Delta := \max(\Delta, |v - V(s)|)$
  until $\Delta < \sigma$
  {Policy Improvement}
  policy-stable := true
  for each $s \in S$ **do**
    $b := \pi(s)$
    $\pi(s) := R(s) + \gamma \arg\max_a [\sum_{s'} T(s, a, s')V(s')\ ]$
    if $b \neq \pi(s)$ **then** policy-stable := false
  if policy-stable **then** stop; **else** go to Policy Evaluation

using V^{π_k}. For each state, using equation 6, the optimal action is determined. If for all states s, $\pi_{k+1}(s) = \pi_k(s)$, the policy is *stable* and the policy iteration algorithm can stop. The complete algorithm can be found in Algorithm 1. When an MDP is finite, i.e. the state and action sets are finite, policy iteration converges after a finite number of iterations. Each policy π_{k+1} is a strictly better policy than π_k unless in case $\pi_k = \pi^*$ but then the algorithm stops. And because for a finite MDP, the number of different policies is finite, policy iteration converges in finite time. In practice, it usually converges after a small number of iterations. Although policy iteration computes the optimal policy for a given MDP in finite time, it is relatively inefficient. In particular the policy evaluation step is computationally expensive. Value functions for all intermediate policies $\pi_0, \ldots, \pi_k, \ldots, \pi^*$ are computed, which involves multiple sweeps through the complete state space per iteration.

In contrast, *value iteration* (VI) does not compute each value function until convergence, but performs the necessary updates on-the-fly. In essence, it combines a truncated version of the policy evaluation step with the policy improvement step, which is essentially Equation 2 turned into one update rule:

$$V_{t+1}(s) := R(s) + \max_a [\sum_{s'} T(s, a, s')V_t(s')] \tag{10}$$

$$:= \max_a Q_{t+1}(s, a). \tag{11}$$

Based on Equations (10) and (11), the VI algorithm (see Algorithm 2) can be stated as follows: starting with a value function V_0 over all states, one iteratively updates the value of each state according to (10) to get the next value functions V_t ($t = 1, 2, 3, \ldots$). VI is guaranteed to converge in the limit towards V^*, i.e. the Bellman optimality Equation (4) holds for each state. A deterministic policy π for all states $s \in S$ can be computed using Equation 6.

Algorithm 2 Value Iteration (Bellman, 1957)

Require: initialize V arbitrarily (e.g. $V(s) := 0, \forall s \in S$)
  repeat
    $\Delta := 0$
    for each $s \in S$ **do**
      $v := V(s)$
      for each $a \in A(s)$ **do**
        $Q(s,a) := R(s) + \gamma \cdot \sum_{s'} T(s,a,s')V(s')$
      $V(s) := \max_a Q(s,a)$
      $\Delta := \max(\Delta, |v - V(s)|)$
  until $\Delta < \sigma$

Modified policy iteration (MPI) (Puterman and Shin, 1978) strikes a middle ground between value and policy iteration. Value iteration essentially combines both the policy estimation and the policy improvement steps into one update. Policy iteration on the other hand, separates the two steps and computes them both in the limit. MPI maintains the two separate steps, but both steps are not necessarily computed in the limit. The key insight here is that for policy improvement, one does not need an *exactly* evaluated policy in order to improve it. For example, the policy estimation step can be approximative after which a policy improvement step can follow. In general, both steps can be performed quite independently *by different means*. For example, instead of iteratively applying the Bellman update rule from Equation 11, one can perform the policy estimation step by using a sampling procedure such as *Monte Carlo* estimation (Sutton and Barto, 1998). Policy iteration and value iteration are both the extreme cases of modified policy iteration.

Many other versions of value and policy iteration have been proposed. A large class under the name of *asynchronous* methods differ from the standard VI and PI by the order – and also the specific parts in that order - in which updates are performed. Asynchronous methods based on value estimation update values of states or state-action pairs not in one sweep throughout the state space but instead update values in specifically targeted regions, e.g. along simulated trajectories such as common in *reinforcement learning* approaches. Asynchronous value iteration methods perform updates in a non-fixed order. The *real-time dynamic programming technique* by (Barto et al., 1995) combines forward search with dynamic programming. For many of these asynchronous methods, convergence can be assured under general conditions such as that all states should be updated infinitely often.

2.2.2 Reinforcement Learning: Model-Free Solution Techniques

In contrast with the algorithms discussed in the previous section, *model-free* methods do not rely on the availability of priori known transition and reward models, in short a *model of the MDP*. The lack of a model generates a need to *sample* the MDP to gather statistical knowledge about this unknown model. Many model-free *reinforcement learning* techniques exist that probe the environment by doing actions, thereby estimating the same kind of state value and state-action value functions as model-based techniques. One of the most basic and popular methods to estimate Q-value functions in a model-free fashion is the Q-learning algorithm by Watkins and Dayan (1992). Algorithm 3 shows its outline.

Algorithm 3 Q-Learning (Watkins and Dayan, 1992)

Require: discount factor γ, learning parameter α
  initialize Q arbitrarily (e.g. $Q(s,a) := 0, \forall s \in S, \forall a \in A$)
  for each episode **do**
    s is initialized as the *starting* state
    repeat
      choose an action $a \in A(s)$ based on an exploration strategy
      perform action a
      observe the new state s' and received reward r
      $Q(s,a) := Q(s,a) + \alpha[R(s) + \gamma \cdot \max_{a' \in A(s')} Q(s',a') - Q(s,a)]$
      $s := s'$
    until s' is a **goal** state

One important aspect of model-free algorithms is that there is a need for *exploration*. Because the model is unknown, the learner has to try out different actions to see their results. A learning algorithm has to strike a balance between *exploration* and *exploitation*, i.e. in order to gain a lot of reward the learner has to exploit its current knowledge about good actions, although it sometimes must try out different actions to explore the environment for possibly better actions. The most basic exploration strategy is the ϵ-greedy policy, i.e. the learner takes its current best action with probability $(1-\epsilon)$ and a (randomly selected) other action with ϵ probability. There are many more ways of doing exploration, see (Wiering, 1999) for an overview.

Algorithms such as Q-learning, but also variants such as *SARSA* (Rummery and Niranjan, 1994; Sutton, 1996), are guaranteed to converge to the optimal value function (and policy) if all distinct values are stored in a backup table. The following sections however, will describe situations where these convergence results do not hold, due to *abstraction*.

Algorithms such as Q-learning employ *bootstrapping* to compute values: the estimate of some value is update with the estimated value of a successor state. Other algorithms that use more unbiased estimates are *Monte Carlo* techniques. They keep frequency counts of transitions and rewards and base their values on these estimates.

Indirect methods strike a balance between model-based and model-free learning. They are essentially model-free, but learn a transition and reward model in parallel with model-free reinforcement learning, and use this model to do more efficient value function learning. An example of this is the Dyna model by Sutton (1991). Another method in which model learning proves useful is *prioritized sweeping* (Moore and Atkeson, 1993).

2.3 Dealing with Large State Spaces : Abstraction

The basic framework of Markov decision processes and solution techniques as described in the previous section assume that all states and actions, as well as transition matrices, policies and value functions, are explicitly represented. Although useful for theoretical analysis, most domains are too large to represent explicitly. For example, for a state space of size 10^5 one needs to store 10^5 state values (and many more action values) and a state transition matrix contains 10^{10} entries for each action. Even if storage of these huge numbers is no problem, convergence of algorithms such as Q-learning and value iteration demands

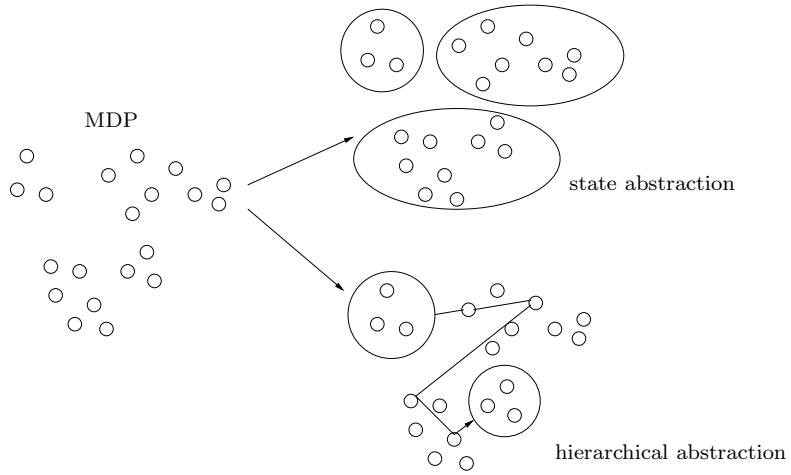


Figure 3: Abstraction in MDPs. The left figure depicts the state space of an MDP. The top right figure shows a typical *state aggregation* approach in which sets of states of the original MDP are aggregated into abstract states. The new MDP contains much fewer states. Usually the action set does not change, but because of the minimized state space, fewer values have to be learned. The down right figure shows a typical way of hierarchical abstraction. An abstract action (sub-policy) can be learned to represent the action sequence leading from one abstract state to the other.

a considerable number of updates. Methods such as MPI cut down on the numbers of necessary updates to some extent, and in RL many efficient *exploration* methods exist that focus sampling and updates on values that are most important for rapid convergence to a good or optimal policy. More sophisticated methods such as *prioritized sweeping* (Moore and Atkeson, 1993) or the general class of *real-time dynamic programming* methods (Barto et al., 1995) share this philosophy.

However, not much use is made of the inherent *structure* a domain or its description imposes. For example, in many domains, the transition function only depends on some aspects of the current state, i.e. the changes in the current state depend on a *local context* of the agent. For a robot trying to move to the door of the room it is currently in, it is usually not important what color the door has. One can make use of this by not incorporating this information in the transition function. In Blocks World , if the policy specifies to move block `a` on block `b`, the only important aspects of the state are that both blocks are clear. Another example is when a robot tries to move from room 1 to room 2, it has to generate a lot of movement actions in order to get there. However, when trying to execute the plan to get to room 2 from room 1 to first pick up a box and then deliver it in room 3, the specific movement actions are not important at that level in the plan.

Various ways of *abstraction* in reinforcement learning methods have been proposed (see Figure 3 for examples). The central idea is to make use of the *inherent structure* in the MDP itself. For example, one can make use of the sparsity of transition matrices, value functions

can be approximated by function approximators and action sequences can be abstracted in sub-policies.

2.3.1 Function Approximation

One – widely studied and validated – way of dealing with large or continuous MDPs is the use of *function approximators*. Estimating values for states or state-action pairs can be performed on individual states and actions. However, in many cases it is not necessary to store a value $V(s)$ for every state s in a lookup-table but instead a parameterized function can be used. That is, $V(s) \equiv F(s, \theta)$ and the complexity is now related to the dimensionality of θ and not to the size of the state space. The nature of the function F determines a tradeoff between learnability and the obtainable precision of the mapping from states to values. An oftenly used form of function approximation is a set of *basis functions* $\varphi_1 \ldots \varphi_n$ where the value of a state can be defined as $\sum_{i=1}^{n} w_i \varphi_i$ and the *weights* w_i are the parameters to be learned (that is, θ).

Many function approximators have been applied to the problem of solving MDPs, such as *neural networks* (Bertsekas and Tsitsiklis, 1996), *decision trees* (Chapman and Kaelbling, 1991), *support vector machines* (Dietterich and Wang, 2002) and *kernels* (Ormoneit and Sen, 2002).

2.3.2 Hierarchical and Temporal Abstraction

Another – more recent – area of MDP abstraction consists of *hierarchical* methods. These methods focus attention on the *sequential* and *temporal* aspects of a task. Consider for example a robot learning soccer. One can model the full problem of soccer as one large MDP, but it makes more sense to distinguish the various subtasks in a typical soccer domain. Dribbling with the ball, shooting, moving and tackling can be considered as *subskills* or *subpolicies*. Additionally they can be learned separately and reused in different contexts. A player that has learned to tackle one opponent can possibly apply this skill in tackling another opponent.

Hierarchical methods have been explored mainly in the last decade. The common factor in all these methods is that abstraction over *action sequences* is applied. The usual framework of MDPs is extended to *semi-markov decision processes* SMDPs (see Barto and Mahadevan, 2003), in which actions can have variable *duration*, or, an abstract action can consist of a sequence of more primitive actions. A so-called *task hierarchy* structures the policy space for learning. Each node in the hierarchy represents an action, where this action can be a primitive action (a leaf node in the hierarchy: the action belongs to the original formulation of the MDP) or a sequence of actions abstracted in a sub-policy (an internal node of the hierarchy: an abstract action). This (learned) sub-policy can higher up the hierarchy be used as an atomic action. For example, on a low level, the motors of a robot are controlled by small control actions like `increase-right-velocity` but a sequence of this kind of actions can be abstracted in the abstract action `move-to-lab`. Abstract actions can have parameters for further abstraction purposes.

Well-known examples of hierarchical abstraction are MAXQ (Dietterich, 2000) (see Figure 4) and options (Sutton et al., 1999). Both methods are prototypical for many hierarchical MDP abstraction techniques; the hierarchy is constructed in a manual fashion and

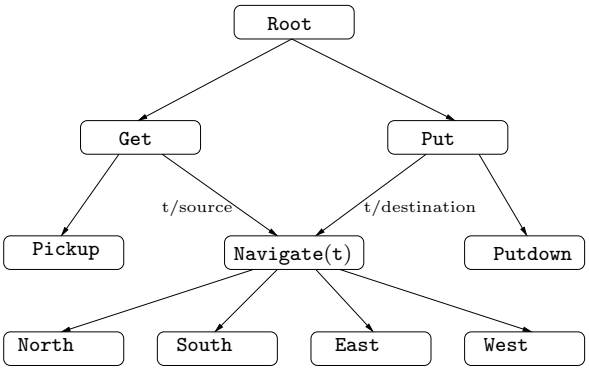


Figure 4: An example of hierarchical abstraction: the MAXQ task hierarchy (Dietterich, 2000). The hierarchy constrains the policy space in a taxi domain in which a passenger has to be picked up at the source location and put down on the destination location. The leaves of the tree contain basic actions in the domain (such as `North`) and the inner nodes represent sub-policies that are constructed using actions and subpolicies lower in the tree. Note that the `Navigate` sub-policy is reused for both getting to the passenger and delivering him, and also that it is parameterized using the source or destination location.

value functions are learned within the hierarchy. The structure in the value function can be given relative to the hierarchy. Methods that *learn* the hierarchy itself include HQ (Wiering and Schmidhuber, 1997), HEXQ (Hengst, 2002) and HASSSLE (Bakker and Schmidhuber, 2004). In model-based frameworks, such as HEXQ, properties of the transition model are used in order to subdivide the original MDP into several sub-MDPs. The model-free method HQ was designed specifically for *partially observable* contexts (POMDPs).

Hierarchical abstraction can be combined with state abstraction methods (see for example Dietterich, 2000; Bakker and Schmidhuber, 2004). For example, a robot's sub-policy for moving forward does not need information about which room it is currently in, whereas on a higher level, e.g. a sub-policy for moving from the kitchen to a library, this information is important.

2.3.3 Representational Abstraction and Factored Representations

A third direction in MDP abstraction research is concerned with the *representation* of the MDP itself. In classical planning, usually a representation is used that "intensionally" represents the problem at hand, in which *descriptions* of objects are in terms of properties they have, instead of the objects themselves. For example, a STRIPS representation describes in an abstract manner how the action operates. Most planning systems use these kinds of representations. A number of systems has been proposed that make use of these intensional descriptions in the context of MDPs.

Factored Representations One of the problems of large MDPs is the size of the transition matrix which is quadratic in the size of the state space for each action, whereas it is often the case that an action will only affect a small subset of domain features. In a *fac-*

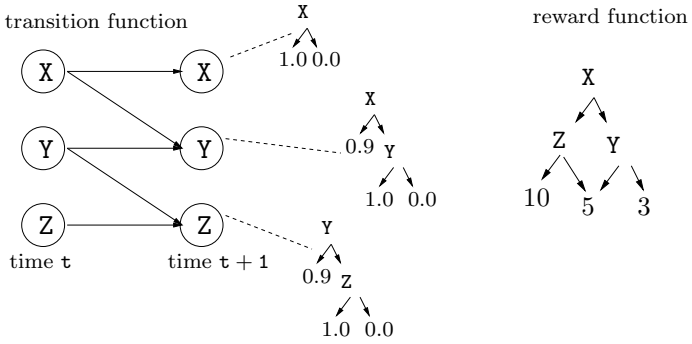


Figure 5: An example of a compact factored representation. The left figure shows a 2-time slice Dynamic Bayesian Network representation of the probabilistic dependencies between state variables between time steps, for a given action. The conditional probability tables are replaced by compact tree-based representations. The right figure shows a compact representation of the state reward function.

tored MDP the set of states is described using a set of random variables $X = \{X_1, \ldots X_n\}$ where each X_i takes on values in some finite domain $\mathrm{Dom}(X_i)$. A state $\vec{x}$ holds a value $x_i \in \mathrm{Dom}(X_i)$ for each variable X_i. This renders the number of states exponential in the number of variables, making it impractical to represent transition functions, value functions and policies explicitly in tables. For this reason, dynamic Bayesian networks (DBN) (Ghahramani, 1998) and decision trees are used to compactly represent these functions.

Boutilier et al. (see Boutilier et al., 2000a, for an overview) introduced *factored representations* in which the transition probability function is represented by means of DBNs. Techniques such as *policy iteration* and *value iteration* can be performed over the tree structure of policies and value functions, which means that computation is focused on the necessary parts of the state space, *without explicit state enumeration*. This enables efficient solution in very large state spaces. The key element here is the use of a technique called *decision-theoretic regression* (DTR) (Boutilier et al., 1999). Augmented with *algebraic decision diagrams*, the SPUDD algorithm (Hoey et al., 1999) is able to solve MDPs with hundreds of millions of states optimally. SPUDD computes intensional descriptions of value functions that have only a couple of hundreds of distinct values, though. Dietterich and Flann (1997) showed similar techniques in reinforcement learning settings by highlighting the intimate relationship between Bellman value backups and *explanation-based generalization* (Minton et al., 1989). Factored representations have also been used in the other reinforcement learning work (see for example Sallans, 2002; Guestrin, 2003; Guestrin et al., 2003b). Theoretical results by Allender et al. (2002) show that factored representations are not guaranteed to be compact, but in practice they often are.

Model Minimization Another, related line of research is concerned with the role of abstraction in itself. *Model minimization* aims at first computing a maximally abstract intensional model of a given MDP, after which standard dynamic programming techniques can be used to solve the minimized MDP. Oftenly, these techniques are applied in the context of factored representations. Givan et al. (2003) take a principled approach and introduce

a *stochastic bi-simulation* measure to group states based on their action transitions and reward distributions. Kim and Dean (2003) describe minimization techniques that allow for abstract states to be *value inhomogeneous*. Related work by Ravindran and Barto (2003) defines *homomorphisms* between the original and the minimized MDP and also considers *temporal abstractions* such as used in hierarchical RL. Work by Dearden and Boutilier (1997) uses a STRIPS model which is then used to minimize the MDP model by discarding irrelevant features.

Adaptive Resolution Reinforcement Learning In RL, function approximation over a set of features can be seen as a naive, implicit way of minimization. The minimization is built-in by the choice of features and the function class to which the function approximator belongs. Work that explicitly introduces model-minimization in an online fashion is usually referred to as *adaptive resolution* techniques. This work is characterized by an iterative procedure in which the abstract representation is modified in parallel with value learning methods that drive the representation modification process. This can also be seen as a modified (or even generalized) policy iteration process (see Section 2) in which the abstract representation of the value function and policy can change constantly. Sampled state-action pairs or solution traces are used to build compact models of the underlying MDP. Examples are the incremental tree-building *G-algorithm* (Chapman and Kaelbling, 1991), *growing neural networks* (Grossmann, 2000), *utile distinctions* (McCallum, 1996), *decision-boundary partitioning* (Reynolds, 2000) and *cognitive economy* (Finton, 2002).

2.3.4 Other Abstraction Methods

The previously described methods form a core class of representational formalisms for scaling up solution algorithms for MDPs. A number of other classes of MDP abstraction methods exist, although we will not describe them but refer the reader to the literature.

A large body of literature deals with *partially observable MDPs* (POMDP) (Kaelbling et al., 1998) in which the states are not fully observable. This has obvious connections with various abstraction methods in that abstraction essentially intends to not observe some parts of the state. Other abstraction methods search for policies directly (usually applied to POMDPs) (Peshkin, 2003), for example using evolutionary techniques (Moriarty et al., 1999) or by using *gradient descent* techniques on parameterized policy representations (Sutton et al., 2000). A recently started direction in RL is based on *predictive representations of states* (PRS) (Littman et al., 2001), in which states are represented in terms of predictions over the future.

2.4 Reflections on Abstractions

The various abstractions outlined in the previous sections solve the original MDP without enumerating the state space explicitly. The central problem is *relating the values of the underlying structures to the abstractions used*. For example, in hierarchical solution techniques, the value of a sub-policy can be related to the cumulative values of the actions that make up the sub-policy. More information about the role of representation, as well as the effects of abstraction techniques, see (Finton, 2002; Koenig and Liu, 2002; Levner et al., 2002; Boutilier, 1999)

One important aspect is that abstractions can introduce *partial observability*. When too much detail is abstracted away, important features needed to make the right decisions can be obscured. Technically, the problem is not *Markov* anymore on the abstract level. All classical solution techniques that have been developed for MDPs assume that an optimal decision does not depend on the history of the performed actions and perceived states, i.e. the *Markov property*. Partially observable MDPs (POMDPs) are much more complex and a different set of solution techniques has been developed, although they are usually capable of solving POMDPs optimally consisting of only few states (Kaelbling et al., 1998).

Abstraction additionally influences the *convergence* of algorithms. Convergence in MDP contexts means that an algorithm for computing value functions and policies is guaranteed to achieve stability in the learned structure. Often one loses convergence guarantees when using abstraction. Some convergence results are known for specific combinations of function approximators and solution techniques for MDPs (Bertsekas and Tsitsiklis, 1996; Papavassiliou and Russell, 1999). Other restricted classes of approximation techniques, under the name of *aggregation* and *averaging* (Gordon, 1995; Wiering, 2004) allow for stronger convergence results in the face of abstraction (Singh et al., 1995). Additionally, convergence when using abstraction does not have to mean converging to an *optimal* solution. Policies are optimal *on the level of abstraction*; it depends on the quality of this level whether the policy is also optimal on the level of individual states.

3. First-Order Representations and Relational MDPs

Despite theoretical results for classical and abstracted MDPs and practical successes of various abstraction mechanisms for MDPs, these techniques do not scale up to even larger domains consisting of *objects* and *relations*. Nevertheless, most – if not all – complex, real world domains are most naturally represented in terms of objects. In fact, *"it is hard to imagine a truly intelligent agent that does not conceive of the world in terms of objects and their properties and relations to other objects"* (Kaelbling et al., 2001). Classical MDP settings represent states and actions in terms of discrete symbols, or by factored representations using propositional or attribute-value variables.

In this section we will introduce relational representations to upgrade MDPs to *relational* MDPs (RMDP). A key issue is that for using these *rich representations*, new ways for abstracting states, value functions, policies etc. have to be used. Most of them use *logical formalisms*, although other methods such as *structured kernels* and relational upgrades of Bayesian networks are used as well. One way of dealing with relational MDPs is to see them as the *semantic* layer of their logical abstractions. The relational representation of MDPs separates the underlying RMDPmore strongly from the logical abstraction than in propositional representations where the abstraction uses the same language (i.e. the propositions). This makes *knowledge representation* a key issue in RRL.

3.1 The Need for Relational Representations

Knowledge representation is a key element in artificial intelligence. However, much of the research in *machine learning* (Mitchell, 1997; Alpaydin, 2004) has been concerned with propositional representations only, and is very much concerned with *statistical approaches*. An exception is formed by *inductive logic programming* (Muggleton and Raedt, 1994; Dze-

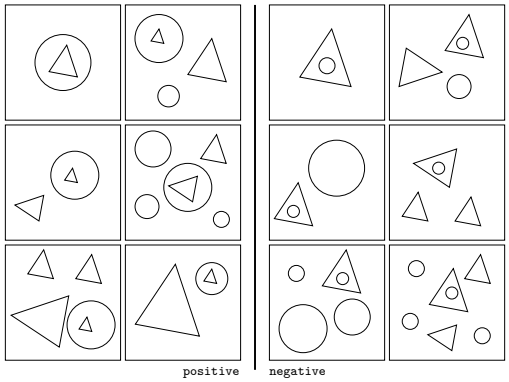


Figure 6: A *Bongard problem* (Bongard, 1970). The task in this problem is build a concept of the left example pictures such that no picture from the right side is covered. Bongard problems are difficult to model propositionally, whereas a relational representation can handle these problems quite naturally. (see also Section 3.1.)

roski and Lavrac, 2001b) that is mainly concerned with learning logical descriptions, but largely did not consider probabilistic aspects of learning. The recent direction of *probabilistic logic learning* (PLL) (De Raedt and Kersting, 2003, 2004) tries to reconcile statistical and logical approaches to deal with large, relational and uncertain domains. RRL has to be seen as fitting in this direction, and extending it with a *utility-based* framework. Reinforcement learning in relational worlds involves dealing with logical abstractions, with uncertainty, and with utilities.

Because of the similarities in ideas by ILP, PLL and RRL, most of the reasons for moving to relational representations apply to all fields equally. In many cases (e.g. for finite domains), intrinsically relational domains can be modeled in terms of a propositional representation. For example, the game of chess can – in principle – be represented in propositional form by using propositions such as `whiteKingOnSameLineAsBlackKing` and `NumberOfBlackPawnsIsNotFive`. However, the game is more naturally represented in a relational form, using for example `on`(`blackKing`, `A4`), `on`(`whiteKing`, `A7`), and `sameLine`(`blackKing`, `whiteKing`). Furthermore propositionalization comes with a number of problems. First, the number of objects should be fixed, and propositions for all relations between all objects should be constructed. This generates a huge number of propositions. Second, no generalization is possible over objects or relations. Third, an *order* has to be imposed on the objects and relations. In Figure 6 a *Bongard* problem (Bongard, 1970) is depicted. To model this domain, all objects have to be ordered such that propositions describing the scene can be constructed. This renders the representation ad hoc, and no generalization to similar problems is possible. For more on the relations between propositional and relational representations see (De Raedt, 1997; van Laer and de Raedt, 2001).

There are a number of additional reasons for modelling RL problems using relational formalisms. Lookup tables or propositional representations are not able to represent the *structural* aspects of states and actions in relational domains such as Blocks World and chess. Relational representations also allow for a general and intuitive way of specifying and

using *knowledge*. Many standard agent architectures and theories use subsets or extensions of first-order logic (e.g. such as modal logic) (Wooldridge, 2002). And as some argue (Kaelbling et al., 2001), a representation should enable representing and reasoning about *objects*. We argue that this is even more evident when relating to existing agent architectures and programming languages. One has to be able to represent objects and relations in our language if an *intentional stance* (Dennett, 1987) is taken, identifying *cognitive* notions like *beliefs*, *desires*, *intentions* and *emotions*.

3.2 Modelling Relational MDPs

As indicated in the previous sections, we will use ground *relational atoms* to describe the states of the MDP. Environments can be described in terms of *objects* and *relations* between objects. A consequence is that the MDP is not described in terms of discrete states, or factored into propositions, but instead we introduce a *relationally factored* MDP (RMDP).

3.2.1 Relational Representations

Let us start with some notation. A relational *alphabet* Σ is a set of relation symbols $\mathtt{p}$ with arity $m \geq 0$, and a set of constants $\mathtt{c}$. An *atom* $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_m)$ is a relation symbol $\mathtt{p}$ followed by a bracketed m-tuple of terms $\mathtt{t}_i$. A *term* is a variable $\mathtt{X}$ or a constant $\mathtt{c}$. A *conjunction* A is a set of atoms. The set of variables in a conjunction A is denoted as vars(A). A substitution θ is a set of assignments of terms to variables $\{\mathtt{X}_1/\mathtt{t}_1, \ldots, \mathtt{X}_n/\mathtt{t}_n\}$ where $\mathtt{X}_i$ are variables and all $\mathtt{t}_i$ are terms. A substitution can be applied to an atom, a term or a conjunction. For example, let C be the conjunction $\mathtt{on}(\mathtt{X},\mathtt{Y}), \mathtt{cl}(\mathtt{X})$ and let θ be $\{\mathtt{X}/\mathtt{a}\}$, then $C\theta \equiv \mathtt{on}(\mathtt{a},\mathtt{Y}), \mathtt{cl}(\mathtt{a})$.

A term, atom or conjunction is called *ground* if it contains no variables. For example, $\mathtt{on}(\mathtt{a},\mathtt{b})$ is a ground atom, whereas the conjunction $\mathtt{cl}(\mathtt{X}), \mathtt{cl}(\mathtt{a})$ is not ground. Conjunctions are implicitly assumed to be *existentially quantified*, i.e. the conjunction $\mathtt{cl}(\mathtt{X}), \mathtt{on}(\mathtt{X},\mathtt{Y})$ should be read as $\exists\mathtt{X}\exists\mathtt{Y}(\mathtt{cl}(\mathtt{X}) \wedge \mathtt{on}(\mathtt{X},\mathtt{Y}))$. A conjunction A is said to be θ-subsumed by a conjunction B, denoted by $A \preceq_\theta B$, if there exists a substitution θ such that $B\theta \subseteq A$. Let C_1 be the conjunction $\mathtt{on}(\mathtt{X},\mathtt{Y}), \mathtt{cl}(\mathtt{X})$ and C_2 be $\mathtt{on}(\mathtt{c},\mathtt{d}), \mathtt{on}(\mathtt{d},\mathtt{e}), \mathtt{cl}(\mathtt{c}), \mathtt{cl}(\mathtt{f})$, then C_2 is subsumed by C_1, i.e. $C_2 \preceq_\theta C_1$ where $\theta \equiv \{\mathtt{X}/\mathtt{c},\mathtt{Y},\mathtt{d}\}$.

The *Herbrand base* of Σ, HB^Σ, is the set of all ground atoms which can be constructed with the predicate symbols and constants of Σ. An *interpretation* is a subset of HB^Σ. For example, with $\Sigma \equiv \{\mathtt{p}/2\} \cup \{\mathtt{a},\mathtt{b}\}$ the Herbrand base $\mathrm{HB}^\Sigma \equiv \{\mathtt{p}(\mathtt{a},\mathtt{a}), \mathtt{p}(\mathtt{a},\mathtt{b}), \mathtt{p}(\mathtt{b},\mathtt{a}), \mathtt{p}(\mathtt{b},\mathtt{b})\}$ and $\{\mathtt{p}(\mathtt{a},\mathtt{a}), \mathtt{p}(\mathtt{b},\mathtt{b})\}$ is an interpretation.

For further details on (relational) logic and (inductive) logic programming we refer the reader to (Lloyd, 1991; Flach, 1994; Nienhuys-Cheng and de Wolf, 1997; Bratko, 2001; Dzeroski and Lavrac, 2001b).

3.2.2 Relational Markov Decision Processes

RRL differs from traditional MDP-based research in that the underlying MDP is now factored into relational atoms. Solving these *Relational Markov Decision Processes* (RMDP) involves the use of logical languages to abstract over relational states and actions, and the definition of abstract value functions and policies.

There are many *implicit* and *explicit* formalizations of relational versions of MDPs. Implicit formalizations (see for example (Guestrin et al., 2003a; Guestrin, 2003; Kersting and De Raedt, 2004; Kersting et al., 2004; Fern et al., 2003, 2004; Yoon et al., 2002; Boutilier et al., 2001)) use a (fragment of a) first-order language to specify abstract definitions of e.g. transition functions and as a consequence, – in an implicit way – an underlying, ground relational MDP is specified; i.e. it consists of the semantic level of the language used. Explicit formalizations (see for example (Mausam and Weld, 2003; van Otterlo and Kersting, 2004; Roncagliolo and Tadepalli, 2004; van Otterlo, 2004)) explicitly define new versions of MDPs where the states and actions are defined using ground relational atoms; a first-order language is then typically used for abstraction over this MDP definition. Both types of formalizations are clearly interchangeable and the main important aspect is the *relation* between the logical abstraction language and the underlying MDP it models. In this paper we use a simple definition of an RMDP (van Otterlo, 2004):

Definition 3.1 (Relational MDP) *Let* Λ *be a logic and* Υ *be a theory in* Λ*. Let* $\mathtt{P}$ *be a set of predicates in* Λ*,* $\mathtt{C}$ *a set of constants in* Λ *and let* $\mathtt{A}'$ *be a set of special action predicates in* Λ*. A* relationally factored Markov decision process *(RMDP) is defined as* $\langle \mathtt{S}, \mathtt{A}, \mathtt{T}, \mathtt{R} \rangle$*, where* $\mathtt{S} \equiv \{\mathtt{s} \in \mathrm{HB}^{\mathtt{P} \cup \mathtt{C}} \mid \mathtt{s} \models \Upsilon\}$*,* $\mathtt{A} \equiv \{\mathtt{a} \in \mathrm{HB}^{\mathtt{A}' \cup \mathtt{C}} \mid \mathtt{a} \models \Upsilon\}$*, and* $\mathtt{T}$ *and* $\mathtt{R}$ *are defined as usual by* $\mathtt{T} : \mathtt{S} \times \mathtt{A} \times \mathtt{S} \rightarrow [0, 1]$ *and* $\mathtt{R} : \mathtt{S} \rightarrow \mathbb{R}$*.*

This means that based on some logic Λ, a set of predicates $\mathtt{P}$ and a set of constants $\mathtt{C}$ together define the possible states of the system (i.e. the interpretations) and the (domain-dependent) background theory Υ defines which states are possible for a given domain. In a similar fashion a set of actions $\mathtt{A}$ can be defined based on Λ. The transition function $\mathtt{T}$ and the reward function $\mathtt{R}$ are defined as usual, denoting transition probabilities between states (which are interpretations now) and denoting rewards for transitions.

Example 3.1 (Blocks World RMDP) *As an example, based on the predicates* $\mathtt{on/2}$ *and* $\mathtt{cl/1}$*, the domain* $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{floor}\}$ *and the action set* $\mathtt{move/2}$ *we can define the* Blocks World *containing* 5 *blocks with* $|S| = 501$ *legal states.* $\mathtt{T}$ *can be defined such that it complies with the standard dynamics of the* Blocks World $\mathtt{move}$ *operator (possibly with some probability of failure) and* $\mathtt{R}$ *can be defined such that sets a positive reward for states that satisfy some goal criterium and* 0 *otherwise.*

A concrete state s_1 *is* $\{\mathtt{on}(\mathtt{a}, \mathtt{b}), \mathtt{on}(\mathtt{b}, \mathtt{c}), \mathtt{on}(\mathtt{c}, \mathtt{d}), \mathtt{on}(\mathtt{d}, \mathtt{e}), \mathtt{on}(\mathtt{e}, \mathtt{floor}), \mathtt{cl}(\mathtt{a})\}$ *in which all blocks are stacked. The action* $\mathtt{move}(\mathtt{a}, \mathtt{floor})$ *moves the top block* $\mathtt{a}$ *on the floor. The resulting state* s_2 *would be* $\{\mathtt{on}(\mathtt{a}, \mathtt{floor}), \mathtt{cl}(\mathtt{a}), \mathtt{on}(\mathtt{b}, \mathtt{c}), \mathtt{on}(\mathtt{c}, \mathtt{d}), \mathtt{on}(\mathtt{d}, \mathtt{e}), \mathtt{on}(\mathtt{e}, \mathtt{floor}), \mathtt{cl}(\mathtt{b})\}$ *unless there is a probability that the action fails, in which case the current state stays* s_1*. If the goal would be to have all blocks stacked, state* s_1 *would get a positive reward and* s_2 0*.*

In contrast to the standard definition of MDPs the state space is now implicitly defined. This is due to the fact that it depends on the instantiation of the domain which states are possible. Two Blocks World problems, one where $\mathtt{C}$ is $\{\mathtt{a}, \mathtt{b}, \mathtt{floor}\}$ and another where $\mathtt{C}$ is $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{floor}\}$ differ in the number of states, although the other constituents of the RMDPs ($\mathtt{T}$, $\mathtt{R}$) are very similar. The total set of interpretations based on a set of predicates and a domain can be much larger than the actually used state space of the RMDP. For example, in Blocks World no block can be on top of itself, i.e. $\neg \exists s \in \mathtt{S}.\mathtt{on}(\mathtt{X}, \mathtt{X}) \in s$.

The size of the state space for an RMDP can grow extremely large with only a modest number of relations. As an example, for a Blocks World with 5 blocks, $|\mathtt{S}| = 501$ and for 10 blocks, $|\mathtt{S}| \sim 59$ million. If $|\mathtt{C}| = k$ and we denote the arity of a relation $\mathtt{p} \in \mathtt{P}$ by $\alpha(\mathtt{p})$ then the number of interpretations is $\prod_{\mathtt{p}\in\mathtt{P}} 2^{k^{\alpha(\mathtt{p})}}$. For a propositional representation this number is 2^n for n state propositions. Given these sizes, it becomes practically impossible solving RMDPs at the level of ground atoms. For that reason, we will discuss logical abstractions (over RMDPs) in the following section.

3.2.3 Logical Abstraction and Partitions

Conceptually, one can view a ground RMDP as the *semantics* of a logical representation level, which can consist of various forms of logical languages.

The main method of abstraction is to compactly represent sets of interpretations using a logical language and an appropriate semantics. An important notion in logic is a definition of *truth*. For example, a conjunction $\exists \mathtt{X}.\ \mathtt{on}(\mathtt{X},\mathtt{a}), \mathtt{clear}(\mathtt{X})$ means that *there is a clear block, denoted* X*, that is on block a*. In all interpretations (worlds) where there is a block on block a this formula *holds*, or, the interpretation is said to be a *model* of the formula. The state $\{\mathtt{clear}(\mathtt{b}), \mathtt{on}(\mathtt{b},\mathtt{a}), \mathtt{on}(\mathtt{a},\mathtt{c}), \mathtt{on}(\mathtt{c},\mathtt{floor})\}$ is a model of the conjunction; we can find a substitution for X, namely $\mathtt{X}/\mathtt{b}$. The notation $\varphi \models \psi$ says that if an interpretation is a model of φ then it is a model of ψ too. The well-known *modes ponens* rule in logic can be stated as $(\varphi \rightarrow \psi), \varphi \models \psi$. For example $(\mathtt{on}(\mathtt{X},\mathtt{Y}) \rightarrow \neg\mathtt{clear}(\mathtt{Y})), \mathtt{on}(\mathtt{X},\mathtt{Y}) \models \neg\mathtt{clear}(\mathtt{Y})$ where $\neg\varphi$ means that φ does not hold.

Although the semantics is defined in a *model-theoretic* way (i.e. $\models$), practical systems use *proof-theoretic* methods to check whether an interpretation is modelled by some formula in the logic. The notation $\varphi \vdash \psi$ means that there exist a *derivation* in the logical language we use that starts from φ and ends with ψ. This derivation makes use of rules (axioms) and the whole sequence is called a *proof* and the process *deduction*. For many logics, one can prove that the proof-theoretic ($\vdash$) and model-theoretic ($\models$) notions coincide, i.e. the logic is then said to be *sound* and *complete*. The proof-theoretic way enables the use of automatic *syntactic* procedures, that manipulate logical expressions. A language such as *Prolog* (Bratko, 2001) is essentially doing this. Many systems in this survey use θ-subsumption (Plotkin, 1970) for checking whether an interpretation is a model of a logical abstraction, because it is an efficient manner implementing $\vdash$ and can be proven sound and complete under reasonable assumptions.

For a basic understanding of the methods in this survey it suffices to know what it means for a logical expression to model (or *cover*) an interpretation, and that most practical systems are proof-theoretic. For a starting point to a more formal introduction to logical methods we refer the reader to (Sowa, 1999) and (Brachman and Levesque, 2004).

The main purpose of logical abstractions is to *group* states or state-action pairs of the underlying RMDP. As in many abstraction methods for MDPs (see Section 2.3), these abstractions are typically *partitions*, defining sets of states that have the same value, have the same (optimal) action, or have the same transition characteristics. Here four of the most common ways of defining logical partitions will be briefly considered. Various logical languages can be employed such as *Horn logic*, *situation calculus* and *description logic*. Relational abstractions can typically take the following forms:

- **Decision Lists.** Many systems in this survey use a set of rules with decision list semantics for abstraction. For example, consider the following *abstract* policy for a Blocks World . It encodes the optimal policy for reaching states where $\texttt{on}(\texttt{a},\texttt{b})$ holds:
  $\texttt{r}_0:\ \texttt{on}(\texttt{a},\texttt{b}) \rightarrow \texttt{noop}$
  $\texttt{r}_1:\ \texttt{onTop}(\texttt{X},\texttt{b}) \rightarrow \texttt{move}(\texttt{X},\texttt{floor})$
  $\texttt{r}_2:\ \texttt{onTop}(\texttt{Y},\texttt{a}) \rightarrow \texttt{move}(\texttt{Y},\texttt{floor})$
  $\texttt{r}_3:\ \texttt{clear}(\texttt{a}),\texttt{clear}(\texttt{b}) \rightarrow \texttt{move}(\texttt{a},\texttt{b})$
  The rules should be read from top to bottom. Given a state, the first rule where the abstract state applies generates an optimal action. Many of the model-free (see Section 4.2) and modelling (see Section 4.3) approaches use decision lists for abstraction.

- **Explicit Partitions.** Decision lists make use of the *order* imposed on the rules. If *negation* is part of the logical language used, *explicit* partitions can be defined. For example, if we have as a first rule φ, then a second rule can be denoted $\neg\varphi \wedge \psi$. In the example above, we could define a partition starting with
  $\texttt{r}_3:\ \texttt{clear}(\texttt{a}),\texttt{clear}(\texttt{b}) \rightarrow \texttt{move}(\texttt{a},\texttt{b})$
  and add the following rule denoting what to do if `a` and `b` are not both clear[5]:
  $\texttt{r}_4:\ \text{not}(\texttt{clear}(\texttt{a}),\texttt{clear}(\texttt{b})),\texttt{onTop}(\texttt{X},\texttt{b}) \rightarrow \texttt{move}(\texttt{X},\texttt{floor})$
  Systems such as SDP and FOALP (see Section 4.4) use *case* statements consisting of complex logical formulas defining explicit partitions.

- **First-Order Trees.** Tree representations are smart (and efficient) represention of decision lists, and transformations exists either way. See Figure 7 for an example in which the space[6] is partitioned using logical tests in the nodes in the tree such as $\texttt{on}(\texttt{A},\texttt{B})$?. Nodes can share variables. A tree representation of the decision list policy in the example above can differ in the order of the tests, based on statistics of occurrence, e.g. states in which $\texttt{clear}(\texttt{a}),\texttt{clear}(\texttt{b})$ holds are less frequent, so an optimized tree will test for this property lower in the tree.

- **Kernels, First-Order Features and Distances.** Some other systems such as RIB and KBR (see Section 4.2) define abstraction levels by inducing high-level (first-order) features, or by using *kernels* and *distances* on relational states. Generalization and abstraction is then performed in the new high-dimensional space spanned. These abstractions allow for advanced function approximators to be used, although the induced abstraction levels lose *comprehensibility*. Additionally, a large class of probabilistic abstraction methods have been proposed (De Raedt and Kersting, 2003), of which we will see examples later.

An additional feature of first-order languages for abstraction is the possibility of using *background knowledge*, thereby extending the language used for abstraction. The `onTop/2` relation used in the examples can be defined by the user (in terms of `on/2` and `clear/1`) and used to create powerful, compact abstractions. First-order background knowledge can

5. Note that in this example, the $\texttt{onTop}(\texttt{X},\texttt{Y})$ relation is defined in terms of the fact that Y is not clear, for at least the block `X` is above `Y`.
6. Actually, the tree in the figure partitions the state-action space.

provide a natural means to enhance the language for abstraction levels, and can be used in various induction algorithms.

3.2.4 Learning Logical Abstractions

The field of *inductive logic programming* (Dzeroski and Lavrac, 2001b; Muggleton and Raedt, 1994; Lavrac and Dzeroski, 1994; Dzeroski and Lavrac, 2001b; Flach, 1994; Dzeroski, 2003) has developed many methods to learn logical abstractions. In relational learning, the hypothesis space contains logical formulae and 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 of abstractions. The search process can be guided by providing *bias* in the form of *background knowledge* or search restrictions (*mode declarations*). Examples are methods to learn trees (Blockeel et al., 1998; Blockeel and de Raedt, 1998), decision lists (Mooney and Califf, 1995) and in general *logic programs* (Lloyd, 1991; Nienhuys-Cheng and de Wolf, 1997). Examples of (numerical) *regression* methods include (Karalic and Bratko, 1997) and TILDE-RT (Blockeel et al., 1998)

Lately, the field of ILP has branched out in the direction of probabilistic approaches, thereby combining logic and probability (Halpern, 1990). Many approaches in this new field of *probabilistic logic learning* (PLL) (De Raedt and Kersting, 2003, 2004) can be seen as upgrades (van Laer and de Raedt, 2001) of propositional, probabilistic machine learning algorithms to the relational case, just like TILDE (Blockeel and de Raedt, 1998) can be seen as a logical upgrade of propositional tree-learning algorithms. The field of PLL has developed many methods that can deal with the intrinsic probabilistic nature of RL problems, such as relational upgrades of Bayesian networks and Hidden Markov Models, Markov Models, naive Bayes classifiers (Flach and Lachiche, 1999), and probabilistic modelling methods such as *probabilistic relational models* (Getoor et al., 2001; Sanghai et al., 2003) and *Markov logic networks* (Domingos and Richardson, 2004). Other work has explored more powerful logics in machine learning (Lloyd, 2003), e.g. the tree-learner ALKEMY.

A related area is the use of relational upgrades of *neural networks* (Browne and Sun, 2001; Botta et al., 1997) and related combinations of *symbolic* and *subsymbolic* methods in RL (Sun and Peterson, 1998; Sun, 1998).

The methods mentioned in this section are a sample of the large class of methods that can learn relational abstractions. Many of them can – and many of them have been – be used for relational RL.

3.2.5 Logical Abstractions in Markov Decision Processes

The goal in research in relational RL is finding an (optimal) policy $\hat{\pi}^* : S \rightarrow A$ on an *abstract level*. Because learning the policy in the ground RMDP is not an option, various routes can be taken in order to find $\hat{\pi}^*$. One can first construct *abstract value functions* and deduce a policy from that. One might also inductively learn the policy from optimal ground traces. Relational abstraction over RMDPs induces a number of *structural* learning tasks. The intrinsic *probabilistic* nature of (R)MDPs, and also the notion of *concept drift* (Maloof, 2003) in RL approaches demand powerful learning techniques. We will now mention some of the learning tasks for RRL and possible techniques.

1. **Structural Models.** Relational abstractions can be used to model various substructures, and each of them can – in principle – be learned. Many logical representations can be learned using techniques from ILP (Muggleton and Raedt, 1994) and an increasing number of them enables modelling uncertainty (see De Raedt and Kersting, 2003, 2004, for an overview).

   (a) **Value functions.** Abstract value functions can abstract *state* or *state-action* value functions. For example, $\mathtt{V}(\mathtt{on}(\mathtt{X},\mathtt{floor})) = 10$ denotes that all states in which there is some block on the floor, get a value of 10. Some *relational regression* techniques have been used (e.g. see Section 4.2). In general they can be upgraded versions of RL-based regression algorithms (for example Driessens et al., 2001) or adapted for use in RRL (for example Driessens and Ramon, 2003). Overall relational regression algorithms that can handle *concept drift* (Maloof, 2003) are needed.

   (b) **Action definitions.** Action learning can be very useful to obtain an abstract model of the underlying RMDP (Pasula et al., 2004, but see also Section 4.5). For example, the action definition $\mathtt{on}(\mathtt{X},\mathtt{Y}),\mathtt{cl}(\mathtt{X}),\mathtt{cl}(\mathtt{Z}) \xleftarrow{\mathtt{move}(\mathtt{X},\mathtt{Y})} \mathtt{cl}(\mathtt{X}),\mathtt{cl}(\mathtt{Y}),\mathtt{on}(\mathtt{X},\mathtt{Z})$ states that applying $\mathtt{move}(\mathtt{X},\mathtt{Y})$ can be performed if $\mathtt{X}$ and $\mathtt{Y}$ are both clear and that in the resulting state $\mathtt{X}$ is on $\mathtt{Y}$. In some domains, one can also consider learning *preconditions* only. Once an action model is learned, deductive techniques can be used to learn value functions (see Section 4.4).

   (c) **RMDP abstraction.** Relational abstractions can be defined over substructures of the underlying RMDP itself (see Section 4.3). For example, $\{\mathtt{cl}(\mathtt{X}),\mathtt{on}(\mathtt{X},\mathtt{Z}),\mathtt{cl}(\mathtt{Y}),\langle\mathtt{move}(\mathtt{X},\mathtt{Y}),\mathtt{move}(\mathtt{Y},\mathtt{X})\rangle\}$ represents an abstract state with two abstract actions. The modeling approaches in Section 4.3 define such abstractions but they can be learned as well (e.g. see Morales, 2004b).

   (d) **Policies.** Policies consist of a mapping from states to actions. There are several ways to induce policies. One way is to use an inductive algorithm on ground samples of state-action pairs. These may be obtained by solving small domains (Lecoeuche, 2001) or by maximization using an abstract value function (e.g. P-learning in Dzeroski et al., 2001). The policy induction itself can be seen as a *supervised learning* problem (e.g. see Khardon, 1999b). Pure *deduction* is another option, used in many of the model-based approaches in Section 4.4. Most policies in RRL have crisp representations, but it would be interesting to consider overlapping or soft (i.e. probabilistic) relational policy abstractions too.

2. **Parameters.** Tasks **1a**, **1b**, **1c** and **1d** deal with *structural* learning tasks. However, most structures contain parameters. Model-free approaches (see Section 4.2) usually solve both problems together whereas the modeling (see Section 4.3) approaches focus on the parameter estimation. By adding structure learning mechanisms to the latter one, iterative 2-phase learning algorithms can be developed separating structure and parameter learning, i.e. in an *generalized policy iteration* (see Figure 2) fashion.

3. **Hierarchies and Program Constraints.** In a hierarchical task (see Section 2.3.2), abstraction is not only performed on the structure of the current state, but also on

the structure of the task. Logical abstractions naturally allow parameterized subpolicies, something that happens on a more ad-hoc fashion in propositional approaches. Hierarchical approaches are also related to the connection between *learning and reasoning*. Using learned task hierarchies in *plan libraries* with corresponding reasoning mechanisms in logic-based agent architectures can be seen as bridging the gap between learning and reasoning (van Otterlo et al., 2003). *Program constraints* in logical agents can be used to *bias* RRL, by defining a set of logical behaviors (subpolicies), see also Section 4.6

For more information about relational and first-order representations of MDPs, see (Kaelbling et al., 2001; van Otterlo and Kersting, 2004; Tadepalli et al., 2004; Boutilier, 2001).

4. Survey of Existing Approaches: State of the Art of RRL

Since the seminal work by Džeroski et al. (1998) an increasing number of systems has been proposed in the literature that aim at solving RMDPs. Although a large variety of representations and algorithms have been proposed, most of them can be understood in terms of the algorithms described in Section 2 and the representations in Section 3. A simple view can be taken in that it is just the underlying representation that is changed; the algorithmic part can be largely taken from the standard MDP framework. Although superficially true, the added benefits and complexities of first-order domains demand more than this. First-order domains can induce quite naturally huge, or even *infinite*, domains. Special representational devices need to be employed to deal with this. Furthermore, most logical representations have no natural *distances* or *gradients*. Many propositional techniques for generalization in RL make use of these properties in for example neural networks. For logical representations, new concepts are needed for this. Furthermore, the new opportunities to use *background knowledge* and possibilities to *transfer* learned knowledge in *comprehensible* form that first-order representations naturally incorporate, contrasts with the *tabula rasa* style of learning typical for work in RL.

The survey of existing techniques in this section will highlight a number of approaches that address these issues. We will first distinguish between *intermediate*, *model-free*, *modelling* and *model-based* methods. The model-free and model-based methods follow the patterns described in Section 2. After that, we will describe methods that tackle parts of the problem (such as learning transition models) or methods that incorporate RRL methods in other types of problems (such as the combination of learning and planning). We will then move to the broader context of agents and multi-agent systems in which action formalisms are used that aim at solving similar problems as RRL does.

4.1 Intermediate Representations

Relational representations are more difficult to manipulate and learn than propositional representations. This is the reason why some work has studied representations strictly simpler than relational, that can still be employed in relational domains.

Deictic Representations The approach by Finney et al. (2002) makes use of a *deictic representation* (DR). Technically, DRs are propositional, but because of a special semantics, they are useful in relational contexts. DRs are much 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 use *markers* as *place-holders* for objects and special actions move the markers around. These markers enable some generalization over objects. For example, that *the object in my hand* drops when I open my hand, can be applied to all objects that I can put in my hand. The main advantage of DRs is that they avoid the arbitrary naming of objects.

Although DRs are a smart way to avoid relational representations, they introduce the difficult problem of *partial observability*: in exchange for focusing on only a few things, one 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. Experimental results on Blocks Worlds using the G-algorithm (Chapman and Kaelbling, 1991) and neural networks showed that especially exploration on the extended action space (i.e. the added actions for moving markers around) makes the value function approximation very hard.

Constructive Function Approximation Recently Irodova and Sloan (2005) describe experiments on Blocks World problems using cleverly engineered, propositional and linear value function approximation. Employing propositional features and linear VFA yields very fast run-times compared to other relational RL systems. However, this – and the possibility of generalizing to larger domains – is heavily dependent on ad-hoc, domain-dependent feature engineering.

Related approaches exist under the general name of *constructive function approximation* (CFA). Clark and Thornton (1997) (but see also Thornton, 2000) explored the statistical properties of the interactions and dependencies between propositional features. Utgoff and Precup (1998) describe methods that are based on that and which can – if applied to RL problems – be classified as *adaptive resolution* techniques (see Section 2.3). CFA methods build complex features from simple ones by exploring the relational interactions between features. The *many-layered learning* method described by Utgoff and Stracuzzi (2002) builds a large number of layers of features, each one building on the previously introduced features, in a card game in which complex relations are needed.

4.2 Model-Free Methods

The first class of relational methods that we describe consists of methods that learn logical abstractions of value functions and policies without prior knowledge of either the transition probabilities or the reward function of the underlying RMDP. A common aspect is that these methods have to *sample* the RMDP in order to learn value functions. Most methods learn a logical abstraction level that can be used for abstracting value functions. A wide variety of *representational abstractions* appears, such as exact logical partitions using trees, weighted combinations of first-order features and structured kernels. The API framework by Fern et al. (2003) forms an exception in that it does not learn an explicit value function representation, but directly induces a logical policy representation. However, sampling is still an important part of this method too, for generating learning examples for policy induction.

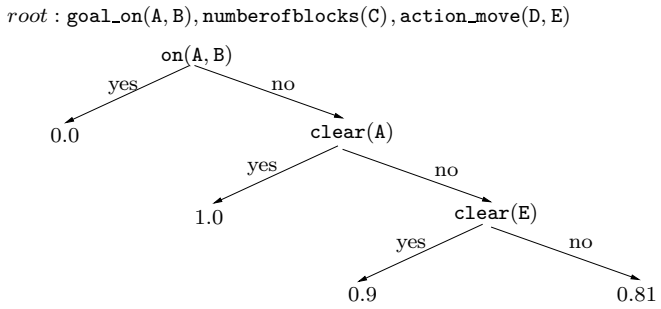


Figure 7: A logical Q-tree. The root of the tree contains the current goal for the learner, the size of the world and the action considered. The inner nodes of the tree contain tests in the form of logical atoms that test for different properties of the current state. The leafs of the tree contain the Q-value of the for the currently considered action in the current state.

We will start with the earliest work on RRL by Džeroski et al. (1998) and continue with surveying other model-free methods. The general structure of most methods can be seen as a variant of the Q-learning algorithm (see Algorithm 3), and more general as a form of generalized policy iteration (see Figure 2) A significant aspect in all methods is that either the policy evaluation or the policy improvement phases (or both) include *structure-learning* operations. This can happen *before* learning (feature induction), *during* learning (based on a batch of gathered samples) or *interleaved* with sampling (based on reward statistics). Methods for the induction of logical structures make heavily use of well-known algorithms from the ILP literature. In fact, the first RRL method was a direct combination of a relational tree learner with Q-learning.

The RRL System The RRL[7] method (Džeroski et al., 1998; Dzeroski et al., 2001) is a combination of RL and ILP (Dzeroski and Lavrac, 2001a). It combines standard Q-learning with a relational regression algorithm, TILDE-RT (Blockeel et al., 1998).

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, based on 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 (see Algorithm 3).

After each episode a decision tree is induced from the example set. The resulting tree contains nodes that are basically Prolog-queries. Nodes in the tree can share variables. The representation language used for the Q-trees employs background knowledge (in the form of definitions of extra predicates to derive new facts from states) and a *declarative bias*. Individuals (i.e. constants) are not referred to in the tree itself directly, but only through the variables of the goal. See Figure 7 for an example of such a tree. It can be transformed into the following Prolog program (essentially a decision list):

7. Although the general area of solving RMDPs is now termed *Relational Reinforcement Learning*, this first method uses exactly the same name. We will use the acronym RRL for this system.

```
qvalue(0) :- goal_on(A,B), numberofblocks(C),
     action_move(D,E), on(A,B), !.
qvalue(1.0) :- goal_on(A,B), numberofblocks(C),
     action_move(D,E), clear(A), !.
qvalue(0.9) :- goal_on(A,B), numberofblocks(C),
     action_move(D,E), clear(E), !.
qvalue(0.81).
```

One problem with Q-functions is that they encode a distance to the goal. A Q-function represents more information than actually needed for selecting an optimal action. For this reason, P-trees are learned. For each state s occurring in the training set, all possible actions in that state are evaluated. For each action a with the highest Q-value in that state, a training example $\langle s, a, 1\rangle$ is generated and for all other actions a' an example $\langle s, a, 0\rangle$. A new tree is learned to *classify* actions as *optimal* (1) or *not optimal* (0). This new P-tree represents an the best policy relative to the Q-tree, and will in general be less complex.

As was experimentally verified in deterministic Blocks Worlds , 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).

Learning Decision List Policies Lecoeuche (2001) introduced a system similar to RRL, except that it uses *first-order decision lists* instead of trees. The system assumes a small domain in which a ground Q-value function can be learned (or generated by other means) after which a ground policy is computed. This policy – in the form of state-action pairs – is then used as input for the decision list learner FOIDL (Mooney and Califf, 1995). The goal is induce a compact first-order policy representation able to generalize over states and actions. The language is a simple relational language in which each policy rule is essentially a Horn clause with an action head.

The system is tested in a *batch* and a *semi-batch* manner. In the first, a ground policy already exists and an abstract policy is induced. The second case alternates between a learning phase and a policy induction phase. After each learning phase, a decision list policy is induced which is then used to bias the next learning phase. At all times, the system keeps ground representations of the current value function and policy, and in addition an abstract policy representation. The abstract policy used for guiding the learning process generalizes over unseen states and actions, and therefore speeds up learning. The system is tested in a dialogue system application which was used in earlier work on reinforcement learning.

Incremental RRL: the TG-algorithm The previous two methods use (semi-) batch learning on ground examples or value functions and policies. This generates the need to store much information and does not make full use of abstraction. The first fully incremental relational RL system is the TG-algorithm (Driessens et al., 2001) and must be seen as a direct extension of the previously introduced RRL system.

The TG-algorithm solves a number of important problems of RRL. 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, generalization is not done in an efficient way. When updating

Algorithm 4 Feature construction (Walker et al., 2004; Srinivasan, 2000)

Require: A training set of state-action-Q-value triples of the Q-function to be learned
  for each distinct action a create p ensembles using **do**
    for each ensemble create m features using **do**
      stochastically create a feature f (Srinivasan, 2000) such that
        i) it covers more than 25% and less than 75% of the samples
        ii) it differs significantly in truth values from the other features so far
      add f to the current predictor
    Build a model using the m features utilizing an appropriate regression algorithm

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.

The algorithm is a first-order extension of the G-algorithm (Chapman and Kaelbling, 1991). TG *incrementally* builds the same kind of trees as TILDE-RT. Each leaf in the tree contains statistics about Q-values and the number of positive matches of examples. A node is split when it has seen enough examples and a test on the node's statistics becomes significant with high confidence. Generating new tests is much more complex than in the propositional case. Although a declarative language bias limits the number of possible tests, many possible splits exist due to newly introduced variables and tests cannot be undone. Due to incomplete experience at the start of learning, the possibility of introducing tests irrelevant for the final (possibly optimal) policy, is inevitable. Still, 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.

First-Order Feature Construction The approach by Walker et al. (2004) separates the structural induction of the representation from the actual value function estimation. First a set of first-order features – represented by relational conjunctions over state atoms – is induced. These are then used as input for a regression algorithm that estimates Q-value functions per action. The method resembles standard function approximation methods for RL (see Section 2.3.1), using first-order features and regularized kernel regression for learning the value function. However, instead of Q-learning, Monte Carlo estimates using hand-coded policies are used as training examples.

The use of (semi-) random features in value function approximation was already described in the propositional case by Sutton and Whitehead (1993). The use of a first-order language requires a more principled approach to generate a set of features that sufficiently covers the state space. One approach for this was described by Srinivasan (2000) and a general outline for the feature construction can be found in Algorithm 4. More specifically, an *ensemble* of feature sets is created, for increased robustness.

The method was tested in a keep-away subtask of (simulated) robot soccer in which 3 players have to keep a ball from being captured by 2 defenders. States are described by for example $\texttt{dist}-\texttt{to}-\texttt{ball}(\texttt{P1},\texttt{Dist},\texttt{Game},\texttt{Step})$ and $\texttt{angle}-\texttt{between}(\texttt{P1},\texttt{P2},\texttt{Ang},\texttt{Game},\texttt{Step})$ and actions such as `hold` and `pass`.

Relational Instance Based (RIB) The previous methods use a crisp logical abstraction for representing value functions, policies or state features. The *relational instance based*

regression method RIB by Driessens and Ramon (2003) uses *instance-based learning* (Aha et al., 1991) on ground states. The Q-function is represented by a set of well-chosen experienced examples. To look-up the value of a newly encountered state-action pair, a *distance* is computed between this pair and the stored pairs, and the Q-value of the new pair is computed as an average of the Q-values of pairs that it resembles. Special care is needed to maintain the right set of examples, by throwing away, updating and adding examples.

Instance-based regression for Q-learning has been proposed for propositional representations before but the challenge in relational domains is defining a *distance* between two interpretations. For RIB, a *domain-specific* distance has to be defined beforehand. For example, in bwp, the distance between two states is computed by first renaming variables, by comparing the stacks of blocks in the state and finally by the *edit distance* (e.g. how many actions are needed to get from one state to another). Other background knowledge or declarative bias is not used, as the representation consists solely of ground states and actions. RIB was tested on Blocks Worlds in a similar fashion as RRL.

TRENDI: Combining TG and RIB Recently, the methods TG and RIB were combined by Driessens and Dzeroski (2005), making use of the strong points of both methods. The TG is a so-called *model-building* algorithm in that it builds an explicit, structural model of the value function. This model is dependent on the language bias and can only build up a coarse approximation. The RIB method is not dependent on a language bias and the instance-based nature is better suited for regression. However, it does suffer from large numbers of examples that have to be processed. The combined algorithm – TRENDI – builds up a tree like TG but uses an instance-based representation in the leaves of the tree. Because of this, new splitting criteria are needed. Because both the language bias for TG and RIB are needed for TRENDI, more things have to be specified. However, on deterministic Blocks World examples, it is shown that the new algorithm performs better on some aspects (such as computation time) than its parent techniques.

Kernels for Structured Data (KBR) Compared to RIB, Gärtner et al. (2003) take a more principled approach to distances between relational states and use *graph kernels* and *Gaussian processes* for value function approximation in relational reinforcement learning (RRL-KBR). Whereas Walker et al. (2004) use kernel regression over the set of induced first-order features, KBR defines a kernel on the actual states and actions. A simple view on kernels is that they define a distance between instances, which are relational interpretations representing states and actions in this approach. Each state-action pair is represented as a *graph* and a product kernel is defined for this class of graphs. The kernel is wrapped into a Gaussian radial basis function, which can be tuned to regulate the amount of generalization. Ramon and Driessens (2004) examined the numerical stability of this approach. RRL-KBR was tested on Blocks World problems in a similar fashion as RRL.

Higher Order Logic Q-learning Cole et al. (2003) upgrade the representation language even further to *higher-order logic* (HOL) (Lloyd, 2003). HOL is more powerful than the representational languages used so far. The idea in this approach is similar to the RRL system: a logical tree representation is induced to represent a Q-function, in this case by the HOL tree learner ALKEMY. An advantage of the approach is that a powerful language can be used to specify the domain and provide bias for the learner. A disadvantage is that it involves much design and specification for even the simplest applications. Similar

Algorithm 5 Approximate Policy Iteration (API)

Initialize policy π
repeat
  $D :=$ **Draw-Training-Set**$(\pi, \ldots)$ {(evaluation)}
  $\pi :=$ **Learn-Decision-List**(D) {improvement}
until π satisfactory
return π

Algorithm 6 Draw-Training-Set

$D := \emptyset$; $E :=$ a set of states
for each state $s_0 \in E$ **do**
  $s := s_0$
  for $i := 1$ to h **do**
    $Q_\pi(s) :=$ **Policy-Rollout**$(\pi, h, \ldots)$
    $a :=$ action maximizing $Q_\pi(s, a)$
    $D := \langle s, \pi(s), Q_\pi(s) \rangle \cup D$
    $s :=$ state sampled from $T(s, a)$
return D

to the P-trees in RRL, policies are primary because the Q-trees do not generalize well. Experiments on Blocks World show that HOL can generate comprehensible policies, even when performed in a changing environment.

Relational Naive Bayes (SVRRL) The recent approach SVRRL by Sanner (2005) is concerned with model-free learning in undiscounted, finite-horizon domains in which there is a single terminal reward for failure or success. These assumptions enable viewing the value function as a *probability of success*, such that it can be represented as a *relational naive Bayes network*. The structure and the parameters of this network are learned simultaneously. Two structure learning approaches are described for SVRRL and in both relational features (ground relational atoms) can be combined into joint features if they are more informative than the independent features' estimates. SVRRL was tested on Backgammon and converged to a competitive level of play using a small number of training games.

The approach differs from some method in the previous paragraphs in that it does not learn a purely logical structure of the value function. It has in common with KBR and the feature induction method by Walker et al. (2004) that it uses a more direct approximation. The assumption on the independency of features might be strong for some domains, although it makes the system very efficient and robust.

QLARC A related approach by Croonenborghs et al. (2004) is the QLARC approach (Q-Learning Agent with Reasoning Capabilities). The authors talk about *informed* RL in which the learner tries to use as much knowledge as can be gathered in the environment, in this case by learning features that predict a probability of achieving a reward or the truth of other features in the next state. First, features close to the goal area are learned, and subsequently features that are correlated with already discovered features. The dependencies between features are represented by probabilistic rules, for ex-

ample $\texttt{prev\_state}(\texttt{R},\texttt{S}),\texttt{move}(\texttt{R},\texttt{S},\texttt{A},\texttt{B}) \rightarrow \texttt{on}(\texttt{S},\texttt{A},\texttt{B})$ denoting that doing action $\texttt{move}(\texttt{A},\texttt{B})$ in state S makes it likely that in the next state $\texttt{on}(\texttt{A},\texttt{B})$ holds. An approximation of the expected reward can be obtained from a look-ahead using the rules. Initial experiments on Blocks World show that the *partial model* (as opposed to a full transition model) increases performance compared to the model-free RIB method.

Approximate Policy Iteration (API) Most of the previous systems focus on the *evaluation* step in the GPI loop (see Figure 2) in that they focus on Q-value function estimation. In contrast, the *Approximate Policy Iteration* (API) method by Fern et al. (2003) focuses directly on the induction of *policies* and can be seen as an extension of the work by Martin and Geffner (2000) to stochastic domains. The motivation for P-trees in the RRL system Džeroski et al. (1998) was that policies are typically less complex and they generalize better. Especially in relational domains, value functions are hard to represent exactly (e.g. see Kersting et al., 2004). Take for example the goal of clearing block `a` in a colored Blocks World (e.g. $\texttt{clear}(\texttt{a})$). A value function has to represent all the colors of the blocks in the stack above `a`. A policy merely has to represent whether there is something above `a`.

The API method takes this as a main motivation and does not keep a direct representation of the value function. Instead, it keeps an explicit representation of the policy itself which is then used for sampling (policy evaluation) based on which the policy is improved (policy improvement). The policy is represented using *taxonomic syntax expressions* which are similar to other types of relational representations. The policy evaluation step is carried out by the simulation technique of *policy rollout* (Bertsekas and Tsitsiklis, 1996). This uses simulation of the current policy starting from states s and actions a to estimate $Q(s,a)$. The Q-value estimations can be used as examples for the policy selection phase, in which a relational classifier is used to learn optimal actions for states, in a similar fashion as the mentioned P-trees (see (simplified) Algorithms 5 and 6). The classifier is induced by a greedy decision list learner that was used in previous work (Yoon et al., 2002) which will be described in the next section on model-based approaches.

The API approach works very well in practice on large, probabilistic planning domains. In fact it dominated much of the probabilistic track of the International Planning Competition in 2004 (see Yoon et al., 2004). The choice for a policy-driving learning approach seems the main reason for this success. An additional important assumption is that the language used for representing policies enables the policies to generalize well. This claim is not instantiated theoretically, but the practical results make a strong case. Additionally, the method also assumes a *strong simulation model* – actions can be sampled in any state – although this does not seem any problem for computer-simulated tasks.

Reflections on model-free methods[8] Model-free learning in relational domains is difficult for at least two reasons. First, learning and especially representing value functions is difficult. The API method shows that by focusing on policies directly this problem can be side-stepped. Although RRL and the method by Lecoeuche (2001) learn explicit relational representations of policies too, they first have to go through the difficult phase of first representing the value function in explicit (or even ground) form. Second, the difficult nature of value function approximation in general is due to *concept drift*; the examples are dependent on the policy, and this policy is constantly subject to change. Not many relational learning

8. The work on RRL, TG, RIB and KBR has also been described in the recent thesis by Driessens (2004).

methods exist that can handle concept drift, and for this new techniques have to be developed in order to apply these in (relational) RL contexts. Still, we have described a number of methods that deal with this problem effectively. Most of them use alternating phases of parameter estimation and structure adaption. The TG and SVRRL algorithms adapt the current representation in a more incremental fashion. They have many similarities with adaptive resolution techniques (see Section 2.3).

An additional remark on representation concerns some of the promises of relational RL: *comprehensibility* and *transfer* of learned structures to new domains. A number of methods (such as RRL, TG, API) choose an explicit, logical representation of learned structures which can be examined and possibly transferred to other domains. Methods using kernels, where the featural abstraction is wrapped into a kernel regression engine, however, are less suitable for this. This does not have to be considered to be a disadvantage, because they aim at using powerful approximation engines. Nevertheless, the amount of comprehensibility or generality of learned structures will influence the possibility of understanding the acquired knowledge or even reusing them in other domains, or as subtask of larger tasks (such as in hierarchical RL and general agent architectures). Although the HOL approach generates very comprehensible policies, a question remains whether such a powerful language – which is harder to manipulate – is needed for most RRL applications.

4.3 Parameter-Learning in Model-Free Approaches

A second class of models side-steps the difficult task of structural induction by *modelling* parts of logical abstraction level. Abstractions over transition functions, policies or parts of the underlying MDP are defined before learning, whereas the learning phase itself is concerned with parameter-learning (such as values, probabilities). The assumption is that a designer can provide these abstractions based on prior knowledge of the domain, or that they can be learned in a separate learning phase before parameter-learning. One advantage of this is more stable learning and it leaves room for better analysis of the interaction between the abstraction level and performance. Additionally, it enables the derivation of error bounds and convergence proofs.

Logical Markov decision programs (LOMDP) The *Logical Markov decision programs* (LOMDP) by Kersting and De Raedt (2003, 2004) introduce a first-order probabilistic STRIPS-like language for compactly and declaratively specifying Markov decision processes over relational domains. A LOMDP consists of a set of action definitions, whereas the reward function is left implicit. For example, the transition rule

$$\texttt{on(X,Z)}, \texttt{cl(X)}, \texttt{cl(Y)} \xrightarrow{0.9:\texttt{move(X,Y)}} \texttt{on(X,Y)}, \texttt{cl(Z)}, \texttt{cl(X)}$$

specifies that the action `move(X,Y)` will lead from the abstract (pre) state on the right to the abstract (post) state on the left, with probability 0.9. The exact, ground, action depends on unification of pre-state with the current ground state. An ordering on the transition function definition ensures that the semantics of LOMDPs is defined, and a representation theorem is proven that every LOMDP induces a finite underlying RMDP.

An abstract policy in this approach is an ordered set of rules $\texttt{State} \rightarrow \texttt{Action}$. This set of state-action rules can be seen as an abstraction of the Q-table for the underlying RMDP. The Q-learning equivalent for this representation is called LQ-learning, basically learning

Q-values for all state-action rules. Experiments on Blocks World show that if a correct abstraction level is given, the approach is able to learn good or optimal policies.

CARCASS The CARCASS abstraction method, introduced by van Otterlo (2003, 2004), aims at abstracting the state-action space of an RMDP. The representation language consists of conjunctions of first-order literals (with negation), possibly augmented with background knowledge predicates. The abstraction consists of an ordered list of rules $\texttt{State} \rightarrow \{\texttt{Action}_1, \ldots, \texttt{Action}_n\}$. Each of these rules represents an abstract state with a number of possible actions. For example, the rule $\texttt{cl(X)}, \texttt{cl(Y)}, \texttt{on(Y,a)} \rightarrow \{\texttt{move(X,Y)}, \texttt{move(Y,X)}\}$ represents the abstract state in which two blocks (X and Y) are clear and X can be put on Y or vice versa.

Two learning methods are proposed. The first one is Q-learning on the abstracted state-action space. The algorithm resembles standard Q-learning, except that one has to respect the order of the rules (e.g. take the first one that applies) and for action selection one i) chooses an abstract action based using an exploration strategy and ii) a random instantiation that complies with the current state is taken. For example, using the above rule for the state $\{\texttt{on(b,a)}, \texttt{cl(b)}, \texttt{cl(c)}, \texttt{cl(d)}\}$ and the action $\texttt{move(X,Y)}$, a random selection for either $\texttt{move(c,b)}$ or $\texttt{move(d,b)}$ is made and this ground action is performed. The second learning method makes use of the fact that the abstraction level actually defines a new state space consisting of abstract states associated with a set of actions that can be performed. This enables the estimation of transition and reward models of the abstract RMDP. This model is then used in a *prioritized sweeping* (PS) algorithm for more efficient learning than standard Q-learning. PS does not only update the current Q-value, but uses the model to propagate value updates back through the model in order to speed up convergence. CARCASS was shown effective on Blocks World problems and the game *Tic–Tac–Toe*.

Relational Q-Learning (RQ) Another representation for Q-learning on a relational abstraction level was proposed by Morales (2003). In this representation so-called *r-states* are defined by sets of first-order relations that partition the state space of the RMDP into abstract states. Similar *r-actions* partition the action space. Each *r-action* is defined by a precondition that specifies its applicability in states. The *r-states* and *r-actions* in fact define a new, minimized, abstract state-action space, if the designer takes care of i) that the state space defines a strict partition (each state of the RMDP belongs to exactly one abstract state) and ii) that if an abstract action is applicable to state belonging to some abstract state, it should be applicable in all states belonging to that abstract state.

On the minimized, abstract state-action space a relational version of Q-learning can be naturally applied, in a similar fashion as in the CARCASS representation. The representation was tested in maze problems, the Taxi domain (Dietterich, 2000) and chess (King-Rook vs. King) (Morales, 2003). Furthermore, it was also used in a power generation application (Reyes et al., 2003). Recently, the representation was used in a *flight simulator* application in which the actual policy was learned through *behavioral cloning* (Morales, 2004a). Although RQ-learning uses a fixed abstraction level, some ideas on how to learn the structural definitions of these abstractions were mentioned by Morales (2004b).

Stochastic Policies for POMDPs Itoh and Nakamura (2004) describe an approach for partially observable domains in which policies are represented in a relational decision list

fashion. The hand-coded policies can make use of a memory consisting of a limited number of memory bits. The algorithm is tested in a maze-like domain where planning is sometimes useful and the problem is to learn when it is useful. This is done by treating the policy as stochastic where the probabilities for the policy rules are used for exploration and learned via gradient descent techniques.

Probabilistic Relational Models (PRM) The approach by Guestrin (2003) (see also Guestrin et al. (2003a)) improves on the representation of value functions by using the Probabilistic Relational Model framework (PRM) (Getoor et al., 2001). A PRM models a probability distribution over a first-order representation. A crucial assumption of the approach is that the relations between objects are *static*. While this may not look unrealistic for domains such as the subtask of the computer game *Freecraft* on which the approach is tested, the majority of tasks such as used in planning systems or in any of the other RRL systems – including Blocks World – involves relations that change over time.

In this limited setting, the global value function is assumed to be decomposed additively into local value functions for each object, so-called *class-based value functions*. It is assumed that all objects have a certain *class* and the local value functions can vary from class to class. The local, class-based value functions are assumed to only depend on some of the properties of those objects and objects that might be directly related to them. The method uses an efficient linear programming procedure to find appropriate weights for the linear combination of the local object properties. If all assumptions – including the ones described about the additive nature of the value functions – hold, the method can be guaranteed to approximate the true value function within some bound.

The work provides bounds for the generalization to worlds with different numbers of domain objects and presents good empirical results in large domains. It shares with the other modelling approaches the difficulties of the modelling involved, in this case the definition of basis functions that can be combined into a global value function.

Reflections on modelling approaches The modelling methods study learning algorithms on a priori defined structures. Learning amounts to estimating parameters for these structures. One of the advantages is that by using *static* structures, one can analyze the relation between the abstraction levels and the underlying RMDP. Because the relational aggregations define new, minimized state (and action) spaces, there are obvious relations with *model minimization* (Givan et al., 2003) and *averagers* (Gordon, 1995). RQ, LOMDP and CARCASS have independently introduced similar abstraction levels for RMDPs. RQ and CARCASS use slightly more powerful state abstractions in that they handle negation and arbitrary state predicates (e.g. by using background knowledge). In general, all three use abstraction to transform the underlying RMDP into a much smaller abstract MDP, which can then be solved by modifications of traditional RL algorithms. In fact, the abstraction levels are *exact aggregations* (i.e. partitions) of state-action spaces. Kersting et al. (2004) show that if one considers only state abstractions (and uses only ground actions) existing convergence proofs for TD(λ)-learning will apply to a state value function on the abstracted state space. However, the more general case of abstract actions (such as used in LOMDP policies and CARCASS) has not been considered yet. RQ-learning in contrast, will converge for abstract actions as well, given the fact that *separate* state and action partitions are defined which can be considered as a new, discrete MDP.

Most of the abstraction levels in this section can be mapped onto abstractions used in the model-free approaches (see Section 4.2). For example, a Q-tree in the RRL method uses a similar representation as e.g. CARCASS. This means that the evaluation method from CARCASS could be applied to the Q-tree and that parameters for the induced Q-trees might be learned using RQ. Other learning methods from ILP might be used as well, for example one could use specific learning techniques to learn structural aspects of PRMs for use in class-based value functions. Morales (2004b) proposes possible techniques to be used in RQ, but in general, much cross-fertilization is possible between the approaches.

4.4 Model-Based Methods

The model-free and modelling approaches are *inductive*, in a broad sense. These methods do not have enough information about the underlying RMDP and have to use *samples* and *inductive methods* to generate logical structures or to estimate parameters' values. In contrast, the *model-based* methods in the this section can in general be classified as *deductive*. That is, given an abstract model of the underlying RMDP, optimal value functions and policies can be computed by *logical deduction* using the model as a theory. Instead of (possibly unsound) generalization using induction, the model-based methods can be seen as employing *justified generalization*, assuming correctness of the given model. Although deduction is enough to compute exact, optimal value functions and policies, the computational burden to perform all necessary (structural) operations is very high. For this reason, recently some approaches have introduced *inductive* procedures that give up exactly representing value functions, thereby making the trade-off between optimality and both computational speed and the possibility of solving larger problems. These will be described in the second part of this section.

We will start with a description of the computational and conceptual core of most of the exact algorithms – *a first-order value iteration algorithm* – and introduce the exact methods, after which we will describe the approximate algorithms.

4.4.1 Exact Algorithms

In model-based approaches, a transition model can be generally defined as a set of action definitions, where for each action a set of k rules pre $\xrightarrow{p_i}$ post$_i$ is defined (and $\sum_{i=1}^{k} p_i = 1$). A value function for a horizon of length t is defined as $V^t : S \rightarrow \mathbb{R}$ which naturally induces a *partition* $\mathcal{P}(V^t)$ of the state space S as $\{(\Sigma_1^t, v_1^t), \ldots (\Sigma_n^t, v_n^t)\}$ so that for all $s \in S$ there is exactly one Σ_i^t such that $s \in \Sigma_i^t$ and $V^t(s) = v_i^t$.

Remember from Section 2 that the value of a state $s \in S$ is defined as $V^{t+1}(s) = \max_{a \in A} Q^{t+1}(s,a)$ and that $Q^{t+1}(s,a) = \gamma \cdot \sum_{s' \in S} T(s,a,s') V^t(s')$ (reward addition omitted), where t denotes the planning horizon. In other words, based on the estimates for V^t (t-steps-to-go horizon), one can compute the estimates for V^{t+1} ($(t+1)$-steps-to-go horizon), extending the horizon with one step. This forms the basis of the value iteration algorithm (see Algorithm 2). The fact that the model is defined in terms of sets can be used to define backup operators and Bellman equations (see Section 2) that operate on the level of *sets* of states (and state-action pairs). This idea forms the core of most model-based solution techniques for RMDPs.

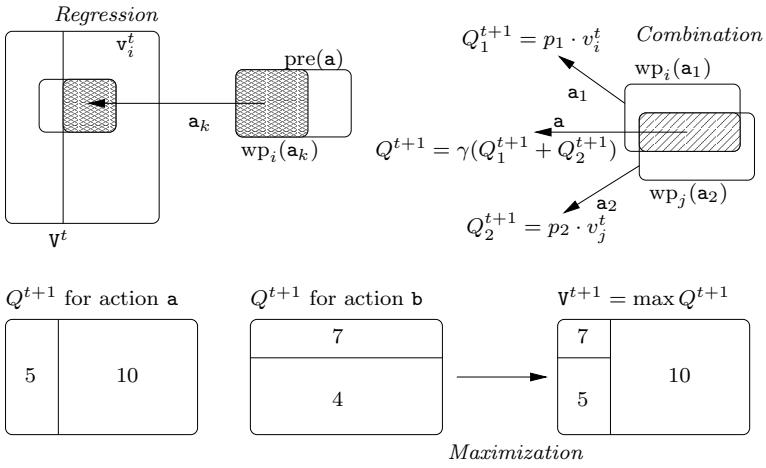


Figure 8: Decision-theoretic regression (DTR). A *structured Bellman backup* consists of three steps. The *regression* step considers the conditions under which an action makes certain state properties true. The *combination* step combines different (probabilistic) outcomes of the same action. The picture shows the combination of two outcomes of one probabilistic action. Finally, the *maximization* intersects the partitions induced by the various actions and takes the maximum value for each intersection. Note that, although not depicted here, to complete the backup the Q-partitions (or V-partition) have to be intersected with the partition induced by the reward partition. See the text for explanation.

The core problem now is to compute an abstract value function V^{t+1} from the abstract value function V^t. This can be done using three steps: *regression*, *combination* and *maximization* (see Figure 8). We will consider each of them in turn.

- **Regression** The first step is to compute $Q^{t+1}(s, a)$ for each distinct action outcome. Let us for a moment only consider one outcome of an action A that is defined by $\mathtt{R} \equiv \mathrm{pre} \xrightarrow{p_k} \mathrm{post}_k$ (for the k-th outcome of action a). For computing the value of $Q^{t+1}(s, a_k)$ we can only consider sets of states that are subsets of the sets in $\mathcal{P}(V^t)$ because other sets are value inhomogeneous and *the* value is not defined. Let (Σ_i^t, v_i^t) be in $\mathcal{P}(V^t)$. The set of states that lead to states in Σ_i^t by applying a_k can be found using *regression*, which actually considers the *inverse* operation of a_k. The set of states in Σ_i^t that can be reached by a_k is defined by $\mathrm{post}_k \cap \Sigma_i^t$. By using $\mathtt{R}$ we derive an abstract state $\mathrm{pre}' \subseteq \mathrm{pre}$ that represents the set of states from which we will reach a state in $\mathrm{post}_k \cap \Sigma_i^t$ when a_k is applied in that state. In other words, the value region Σ_i^t is *pushed through* the action a_k to get the *weakest precondition* pre', denoted $\mathrm{wp}_i(a_k)$. For example, regression of $\mathtt{on}(\mathtt{l}_1), \mathtt{off}(\mathtt{l}_2)$ through the action rule

$$\mathtt{off(X)}, \mathtt{off(Y)} \xrightarrow{p:\mathtt{turn-on(X)}} \mathtt{on(X)}, \mathtt{off(Y)}$$

gives the state $\mathtt{off}(\mathtt{l}_1), \mathtt{off}(\mathtt{l}_2)$. Because the value of $\mathrm{post}_k \cap \Sigma_i^t$ is exactly v_i^t, $Q(\mathrm{wp}_i, a)$ can be computed as $p_k \cdot v_i^t$. To summarize, the regression computes the conditions

such that an action a leads to some part Σ_i^t in the V^t-partition, i.e. the conditions such that a makes $\text{post}_k \cap \Sigma_i^t$ true.

- **Combination**. Let us assume that we have computed the regression of all parts of $\mathcal{P}(V^t)$ through all action rules for some action a. That is, we have computed all $\text{wp}_i(a_k)$ (i ranging over $\mathcal{P}(V^t)$ and k over outcomes of a). The Q-value of an abstract action a depends on all of its outcomes a_k. Let a have m different outcomes (i.e. a is defined by m rules), let $|\mathcal{P}(V^t)| = n$ (i.e. V^t consists of n abstract states) and $z_1, \ldots z_m$ be such that all $z_i \in \{1, \ldots, n\}$ ($i = 1 \ldots m$). The abstract state $S \equiv \text{wp}_{z_1}(a_1) \cap \ldots \cap \text{wp}_{z_m}(a_m)$ consists of all states such that applying abstract action a leads with probability p_i to $\Sigma_{z_i}^t$ ($i = 1 \ldots m$). This, in turn, makes it easy to compute the value of $Q(S, a)$ as $\sum_{i=1}^m \gamma p_i Q(\text{wp}_i(a_k), a)$. All possible combinations of different outcomes for an action a will induce a state-action pair (S, a) such that the Q-value for applying action a in state S is defined. If $S \equiv \text{wp}_{z_1}(a_1) \cap \ldots \cap \text{wp}_{z_m}(a_m) = \emptyset$ then this particular combination of wps (outcomes) is not possible.

- **Maximization**. Recall that $V^{t+1}(s) = \max_{a \in A} Q(s, a)$. The previous combination step leaves us with a set of partitions of the state-action space $\mathcal{P}(Q_i^{t+1})$ (with i ranging over the abstract actions). The *intersection* of these partitions forms $\mathcal{P}(V^{t+1})$. Each $(\Sigma_i^{t+1}, v_i^{t+1})$ is defined such that Σ_i^{t+1} is a part of the intersection of the state-action partitions and v_i^{t+1} is the maximum value of the values of $\mathcal{P}(Q_i^{t+1})$ in that intersection.

The three operations *regression*, *maximization* and *combination* together compute a complete Bellman backup operator that computes an abstract value function V^{t+1} from an abstract value function V^t. The regression part is common in regression-based planners and in *explanation-based generalization* (Minton et al., 1989). However, the stochastic nature of the actions in an (R)MDP render the maximization and combination steps necessary in order to cope with multiple outcomes under a probability distribution. This is generally referred to as *decision-theoretic regression* (DTR) (Boutilier et al., 1999).

The *regression* step in DTR is highly representation-dependent. In some action languages (such as Situation Calculus Reiter, 2001) regression is a standard operation. For DBN factored dynamics, regression involves probabilistic inference over random variables. In general, regression from a value partition essentially *refines* the partition by identifying regions that make (or do not make) some properties of the state true. The *combination* step is a computationally demanding problem for most representations, because it involves combining many weakest preconditions into a Q-value partition. For propositional representations, it can be done relatively efficiently, because there is only a small (finite) number of individual propositions. For logical representations, it becomes necessary to *simplify* formulas. For example, the combination of $\exists \texttt{X}.\ \texttt{p}(\texttt{X})$ and $\exists \texttt{Y}.\ \texttt{p}(\texttt{Y})$ may yield $\exists \texttt{XY}.\ \texttt{p}(\texttt{X}) \wedge \texttt{p}(\texttt{Y})$ although this reduces to $\exists \texttt{Z}.\ \texttt{p}(\texttt{Z})$ (because $\texttt{X}$ and $\texttt{Y}$ can refer to the same constants). The more complex the representation language, the more complex these reduction operations are. The final step of *maximization* again depends on the representation. Intersecting the partitions is similar to combining abstractions. Fine-grained partitions yield many combinations that have to be generated. Furthermore, maximizing involves throwing away abstractions that are dominated (in value) by other ones. Deciding whether an abstraction

models the same set of states but is dominated by some other in value, degenerates to deciding entailment which gets harder the more complex the representation language is.

Symbolic Dynamic Programming (SDP) *Symbolic dynamic programming* (Boutilier et al., 2001; Boutilier, 2001) is the first model-based algorithm for RMDPs, based on a probabilistic version of the *situation calculus* (SC) language (Reiter, 2001; McCarthy, 1963). SC is a full first-order logic for specifying and implementing dynamical systems. The language is centered around *situations*, representing *sequences of actions*. For example, the situation $\texttt{clear}(\texttt{a}, \texttt{do}(\texttt{move}(\texttt{a}, \texttt{floor}), \texttt{s}_0))$ denotes that block a is clear after doing action $\texttt{move}(\texttt{a}, \texttt{floor})$ in the initial situation $\texttt{s}_0$. All atoms (*fluents*) have a situation term attached.

Formalizing a domain consists of specifying so-called *successor state axioms* (SSA). An SSA is of the form $\texttt{F}(\tilde{\texttt{x}}, \texttt{do}(\texttt{action}, \texttt{s})) \equiv \Phi_{\texttt{F}}(\tilde{\texttt{x}}, \texttt{a}, \texttt{s})$, *completely* characterizing the truth values of the fluent $\texttt{F}$ in the next situation $\texttt{do}(\texttt{action}, \texttt{s})$, using the formula Φ_F. Action applicability is formalized using precondition axioms. Regression forms a core part of the logic and is naturally defined on the structure of SSAs. In fact, the truth value of a fluent is determined by regression to the initial state $\texttt{s}_0$. The probabilistic extension of SC allows for multiple, probabilistic outcomes of actions.

SDP solves a first-order MDP specified in SC and produces a *logical description* of the optimal value function and policy. It follows the general outline of decision-theoretic regression, and an optimal value function can be found purely by deduction, without explicit state enumeration. Although the regression step of DTR is a natural concept in the language, the combination and maximization steps demand powerful theorem provers for simplification of formulas, given that SC is a complex language. Due to this, the computational complexity does not allow for doing this automatically, and the examples on a logistics domain are computed by hand. However, SDP elegantly formalizes the general idea of first-order decision-theoretic regression and it defines (like RRL did for the model-free approaches) the first model-based algorithm for RMDPs.

Value Iteration in Fluent Calculus (FOVIA) The FOVIA method (Großmann et al., 2002; Holldobler and Skvortsova, 2004; Karabaev and Skvortsova, 2004) is on the surface very similar to the SDP approach, in that it replaces the logical language by another full first-order logic – the *fluent calculus* (Thielscher, 1998) – and keeps the conceptual structure of structured value iteration intact. However, by making use of only a subset of the language, an efficient algorithm has been developed for automatically solving RMDPs. An important difference with SC is that FOVIA uses a *state-based* representation, in much the same way as previous model-free and modelling approaches.

FOVIA (and its implementation FCPlanner) uses a state representation referred to as CN-states, which are conjunctions of (possibly negated) logical atoms. Causality (e.g. for action dynamics) can be reversed in order to obtain a regression procedure. The maximization and combination steps of DTR are efficiently implemented because of two reasons. First, the representation of states (the CN-states) is much simpler than in SC. Second, *subsumption* between states is used to prune away value-dominated abstract states.

Although the language used is strictly less powerful than SC, the gain is an automated procedure to perform first-order value iteration. The method was validated on a number of Blocks World problems and it participated in the probabilistic track of the International Planning Competition 2004.

Relational Value Iteration (ReBel) The relational Bellman backup operator ReBel (Kersting et al., 2004; van Otterlo et al., 2004) too aims at solving RMDPs with a value iteration algorithm. It uses a relational language in which states are represented as conjunctions augmented with constraints. For example, the state $\texttt{clear(X)}, \texttt{clear(Y)}, \texttt{X} \neq \texttt{Y}$ denotes the states in wich at least two blocks are clear. A *constraint handler* is used to reason with domain constraints, by deducing new constraints (for example, the state $\texttt{on(X,Y)}$ entails that $\texttt{X} \neq \texttt{Y}$) and checking whether generated states are *legal* states (for example, $\texttt{on(X,X)}$ would entail `false` because no block can be *on* itself).

Value functions are efficiently represented by *overlapping* rules. No explicit partitions are represented, but because each rule has an associated value, the semantics of value functions are declarative: the value of a state is the maximum value of all the rules that cover the state. This type of representation, together with efficient subsumption checking procedures, make the combination and maximization steps of DTR highly efficient. The combination step makes use of the *greatest lower bound* (glb) of two action rules and some of the operations during DTR have been made much more efficient by using logic programming techniques such as *tabling* and *tries* (Fischer, 2005). However, because one of the aims of ReBel is to use a simple language, regression is not standardly defined. For this, a special procedure is used to perform regression on action rules that are similar to the ones used in LOMDPs (see Section 4.2), taking the constraints into account using the constraint handler. The regression step allows for the introduction of new variables to explain certain effects. For example, when regressing from $\texttt{on(a,b)}$ through a `move` action, one gets $\texttt{clear(a)}, \texttt{clear(b)}$ as a possible pre-state. However, allowing to add new variables, one gets additionally the state $\texttt{on(a,b)}, \texttt{clear(X)}, \texttt{clear(Y)}$ assuming we have moved two other blocks (`X` and `Y`).

The experimental results showed that the logistics domain from (Boutilier et al., 2001) could be solved automatically. Results in Blocks World showed an interesting feature. Because ReBel allows for the introduction of new variables, value iteration might not stop because an infinite number of abstract states can be introduced. For example, regression from $\texttt{clear(a)}$ can generate an infinite number of blocks on top of `a`. This is essentially due to an *open world semantics*; value iteration can be performed without actually knowing how large the domain is.

Reflections on exact model-based algorithms The roots of the all three approaches go back to explanation-based learning (EBL) and in fact, the specific way of using an abstract model to perform *set-based* value iteration has been used before in different ways. Under the name of *decision-theoretic* regression, it has been applied for factored representations of MDPs in structured versions of value iteration and policy iteration (see for an overview, Boutilier et al., 2000a). Other work has shown that structured versions of value and policy iteration using propositionally factored dynamics can be seen as forms of model minimization (see for an overview, Givan et al., 2003). The first approach highlighting the intimate relationship between Bellman backups and explanation-based learning in RL settings was the *explanation-based reinforcement learning* (EBRL) approach by Dietterich and Flann (1997). Other work using similar ideas for continuous state MDPs was recently described by Feng et al. (2004). All these works use very different representations, although the underlying mechanisms are quite similar. Each different representation demands new operations in order to perform regression, combination and maximization.

FOVIA and ReBel use various techniques to cope efficiently with the difficulties of the operations needed for DTR. Still the computational burden of doing exact value iteration is high. Reasoning with and simplifying logical expressions is hard. For this reason, the next series of methods use approximations in model-based settings.

4.4.2 Approximate Algorithms

The first three methods in this section are approximate versions of the exact methods described in the previous section. After that, two additional methods not performing DTR are described.

FOLAO*: Approximate FOVIA Recently Karabaev and Skvortsova (2005) extended the FOVIA approach with a heuristic search that avoids evaluating all states. Guided by an admissible heuristic, the search is restricted only to those states that are reachable from the initial state. This apparently reduces the number of DTR operations considerably. The method is related to the REBP approach by Gardiol and Kaelbling (2003) (see Section 4.5.2), by only considering the state space that is actually relevant for the given task. Furthermore, it is related to RTDP (Barto et al., 1995).

FOALP: Approximate Symbolic Dynamic Programming The SDP approach was recently extended by Sanner and Boutilier (2005) in the FOALP approach, transforming it into an *approximate* value iteration (AVI) algorithm (Schuurmans and Patrascu, 2001). Instead of exactly representing the complete value function – which can be large and fine-grained and because of that hard to compute – the authors use a fixed set of *basis functions*. Each step in AVI does not generate new logical structures, but instead a set of *constraints* on the values of the basis function contribution to the global value function are generated. The constraint generation problem is more complex in the first-order logic case than in the propositional case because of a possibly infinite domain. The constraint set can be efficiently solved using linear programming techniques and the approach was tested in a small elevator dispatch problem. An important point is that an error bound can be computed on the error that is introduced due to approximations. An issue that is still largely open is how to generate suitable, domain-dependent basis functions in an automated fashion.

Inductive Policy Selection using First-Order Regression A second approximation to the SDP approach was described by Gretton and Thiébaux (2004). The method uses the same basic setup as SDP, but the DTR procedure is only partially computed. By employing multi-step *classical* regression from the goal states, a number of structures are computed that represent abstract states. The combination and maximization steps are not performed, but instead the structures generated by regression are used in the higher-order inductive tree-learner ALKEMY (Lloyd, 2003) to induce a tree representing the value function. This combination of deductive and inductive procedures has shown to perform very well on Blocks World problems and logistics domains, and side-steps the difficulties encountered in the original SDP for which powerful theorem provers were needed.

Inductive Policy Selection Yoon et al. (2002) introduced a method for inductive policy selection that learns an ensemble of decision lists that represents a policy. The algorithm can be viewed upon as an extension of the work by Martin and Geffner (2000) and Khardon (1999b) to stochastic domains. The policies obtained from solving several small domains

are used to induce general policies that can be applied in larger problems. In fact, this algorithm has been used in the API framework (see Section 4.2) sharing the assumptions that policies often generalize well, if the policy language is well-chosen. The example policies in this framework are obtained by using a probabilistic planner. Related approaches that also reduce learning policies to *supervised learning* are (Lagoudakis and Parr, 2003; Benson, 1996; Khardon, 1999b,a; Martin and Geffner, 2000).

Upgrading from small domains The method by Mausam and Weld (2003) does not use any deductive steps on an abstract level, but generalizes from learned, ground value functions. The first step is to take a small RMDP for which the ground value function can be solved using an efficient technique such as SPUDD (Hoey et al., 1999) (see also Section 2.3). This is repeated a number of times to generate many training examples of value functions. The next step is to use a relational regression algorithm to induce an abstract state value function. In this respect, the method is similar to the work by Lecoeuche (2001) in that it upgrades from solved, small domains. One drawback is that value functions, as opposed to policies, do not generalize well because they are dependent on the domain size. Additionally, the work considers *Dynamic Object* RMDPs in which actions can generate new objects.

4.5 Guided Exploration and Transition Model Learning

We have described a number of model-based and model-free learning algorithms that can handle the typically huge state spaces that RMDPs induce. Approximations are commonly used to deal effectively with scaling up to even larger domains. However, like in the classical MDP context more methods exist that can help learning, such as smart exploration and learning models of the MDP such that these can be used in learning or planning. This section will contain some of these approaches.

4.5.1 Guiding the Learner

Driessens and Dzeroski (2004) use a hand-coded policy to provide *guidance* to the RRL system (see Section 4.2), which can be seen as a guided exploration method. The guidance policy generates biased samples that provide (semi)-optimal learning examples that are then treated as normal. If properly mixed with the standard learning mode, guidance speeds up learning. An additional benefit is that it provides feedback in case rewards are sparse (such as with a single goal state which is hard to find with random exploration). Related to guidance is the usage of *behavioral cloning* by Morales (2004a).

Fern et al. (2004) (in the context of the API framework by Fern et al. (2003), see Section 4.2) do not explore the state space, but instead use exploration on the set of domains instantiations. The systems starts by generating easy instantiations of the given domain and the difficulty of the problems is increased in accordance with the current policy's quality. The instantiations are generated by performing *random walks* in the planning domain, generating a start and goal state. The length of the random walk determines the difficulty of the task. The approach can be seen as a kind of *reward shaping*.

Heuristics can be used as a *measure of progress* for domains in which feedback is sparse. Random exploration in a state space containing only one goal state will not generate any feedback until the learner accidentally hits the goal state. Yoon et al. (2005) report on an algorithm for learning measures of progress in deterministic and stochastic planning

domains. A compact representation is used that can represent a prioritized sequence of components of the heuristic function. In Blocks World , these components can be e.g. how many blocks are already *well-placed*.

4.5.2 Learning and using Transition Models

Transition models embody *knowledge* about the environment that can be explored in various ways. Most model-free RL methods (see Sections 2.2 and 4.2) do not make explicit use of gathered knowledge about the environment's dynamics whereas a learned model could be reused by an agent in situations it has never experienced before. *Partial models* of these dynamics have been shown useful for relational RL (Croonenborghs et al., 2004). Complete models can be used in the various model-based methods in Section 4.4 and in probabilistic planning approaches.

However, learning probabilistic transition models with relational structure is difficult. Only some approaches have been proposed for propositional approaches and for the relational case even less. Koller and Pfeffer (1997) describe how to learn probabilities for fixed relational rules and a number of other methods, such as relational upgrades of Bayesian networks, can perform similar tasks. However, the induction of the *structure* of the transition models (together or before parameter learning) is difficult.

Learning Probabilistic Relational Planning Rules. The only work so far addressing the issue of learning probabilistic, relational planning rules is by Pasula et al. (2004). The approach assumes that actions will only affect a small number of properties of the world and that the number of different outcomes of one action is small. The action rules are assumed to have a STRIPS-like form, in which a *context* defines an abstract state in which the action is applied and a set of probabilistic *outcomes* that define the changes in the state if the action is applied. Learning rules consists of a three-step greedy search approach:

- **learnRules** performs a search through the set of rule sets using ILP operators,
- **induceOutcomes** finds the best set of outcomes, given a context and an action and
- **learnParameters** learns a probability distribution over a set of outcomes.

The learning process is *supervised* as it requires a dataset of state-action-state pairs taken from the domain. As a consequence, the rules are only valid on this dataset, and care has to be taken that it is representative for the domain. Experiments on Blocks World and logistics domains show the robustness of the approach.

Recently Pasula et al. (2005) extended this approach to handle domains with more complex dynamics. In many domains the assumption that each action has only a small set of outcomes, does not hold. Some action effects are highly unlikely and therefore hard to model. Consider pushing over a stack of blocks; the number of possible blocks configurations (outcomes) is extremely large. This is solved by using *noise outcomes* for actions, and the exact *structural* result of the action in that case is ignored. This effectively renders the model *partial* in return for still being able to learn and act effectively. Furthermore, the approach is able to invent new features to be used in the rules. New features are expressed in terms of already learned ones, thereby extending the concept language during learning. This is

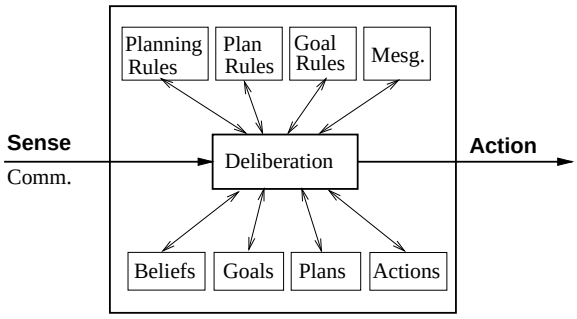


Figure 9: A typical agent architecture used in many (logic-based) agent systems. The agent receives *sensor information* from its environment (which might include visual state information, utterances by other agents etc.). In a deliberation cycle, the agent employs various mechanisms and background knowledge (such as planning rules, knowledge about the environment) to decide on an *action* based on its current beliefs, goals and capabilities.

commonly referred to as *predicate invention* in ILP, and has connections with constructive function approximation (see Section 4.1). Earlier work by Hernandez-Gardiol (2003) who used standard ILP learning on state transitions, exemplified the difficulties of the complex domains that Pasula et al. (2005) solve.

Planning with Envelopes *Relational Envelope-Based Planning* (REBP) (Gardiol and Kaelbling, 2003) lets an agent begin acting quickly within a restricted part of the full state space and to expand its envelope if resources permit. A set of probabilistic planning rules is used by a probabilistic planner to generate an initial sequence of states and actions, the envelope, leading to the goal. This sequence is turned into an RMDP, where obviously many actions lead outside the known state space. By sampling from the states one step outside the envelope, the *fringe* states, the envelope is expanded. After that, the newly added actions are evaluated and a new policy (for the states in the envelope) is computed. The aim of REBP is to compute a policy, by first generating a good initial plan and use envelope-growing to improve the robustness of the plans incrementally.

4.6 Hierarchies and Agents

The previous methods have described a wide spectrum of techniques solving RMDPs and implementing ways to represent the abstract structures mentioned in Section 3.2.5, such as value functions and policies. In this section we will discuss RRL methods that are situated in the context of more general abstraction levels, such as *hierarchies*, *agents* and *multi-agent systems*. One should view (relational) RL as a sub-component of more complex, intelligent entities (i.e. agents), in which it functions as a method for learning *behaviors*. The agent literature (Wooldridge, 2002; Weiss, 1999; Ferber, 1999) contains many examples of logical agents capable of reasoning, communicating, planning and acting (e.g. see also Figure 9). RRL methods can provide a general framework for *learning* in general agent architectures.

For more on RRL in agents and multi-agent systems we refer the reader to (Dzeroski, 2002; van Otterlo et al., 2003; Tuyls et al., 2005).

4.6.1 Hierarchical Approaches

Although hierarchical RL (Barto and Mahadevan, 2003, , see also Section 2.3.2) has become mature, hierarchical approaches for RMDPs have only recently be explored. An advantage of *relational* HRL is that parameterizations of sub-policies and goals naturally arise.

Driessens and Blockeel (2001) presented an approach for solving an RMDP with multiple goals that can be classified as hierarchical. In the computer game *Digger*, the learner has to gather *diamonds* while *avoiding monsters*. Both subgoals are first learned in isolation using the RRL system (see Section 4.2) and after that, the learner is confronted with the complete task, in which it can use the value functions both subtasks as background knowledge. This differs from standard HRL in which *sub-policies* are used to generate a hierarchy.

Aycenina (2002) uses the original RRL system to build a hierarchical RRL system. A number of subgoals is given and separate policies are learned to achieve them. When learning a more complex task, instantiated sub-policies can be used as new actions. More recently, Roncagliolo and Tadepalli (2004) use batch learning on a set of examples to learn values for a given relational hierarchy. The input to the algorithm consists of tuples of the current state, task, subtask and Q-value. The result is a decision list style Q-value function that generalizes over tasks and subtasks.

Two other systems use hybrids of *planning* and learning, which can be considered as generating hierarchical abstractions by planning, and using RL to learn concrete policies for behaviors. The approach by Ryan (2002) (see also Ryan, 2004) uses a planner to build a task hierarchy. RL is used to learn *teleo-operators* that move from one abstract state in the plan to another. A precondition of such an operator defines the application space of a behavior learnt by RL, and the postcondition a local goal for the operator. Grounds and Kudenko (2005) introduce a similar method, differing in that this method operates in terms of STRIPS plans instead of teleo-reactive planning.

4.6.2 Decision-Theoretic Agent Programming Languages

Decision-theoretic (agent) programming languages are powerful, first-order languages that can be used to axiomatize dynamic worlds. One of the most commonly used languages is *situation calculus* (Reiter, 2001; McCarthy, 1963). Its stochastic version (Reiter, 2001; Boutilier et al., 2000b) has been used in several model-based approaches for RMDPS (see Section 4.4, including *symbolic dynamic programming* (Boutilier et al., 2001)). The Golog-Language (Levesque et al., 1997) based on situation calculus can be used to specify *complex actions*, consisting of standard programming language constructs such as *conditional actions* and *procedures*. In fact, this can be seen as providing a policy space bias, or *program constraints*. Gu (2003) highlights the relation between macro-actions in situation calculus and hierarchical abstraction of (R)MDPs. Many other extensions to situation calculus and Golog have been proposed and might be used or incorporated in the RRL framework.

Icarus (Choi et al., 2004) is another powerful language – or: *cognitive architecture* – in which – among many other things – hierarchical, relational skills can be specified and learned. These skills support reactive execution and model-free reinforcement learning can

be used through the skill hierarchy. However, recent work has extended this with model-based learning techniques (Langley et al., 2004). Earlier, (Shapiro and Langley, 2002) already described the SHARSA as an extension of SARSA, adding (H)ierarchy aspects. In Icarus, a clear separation exists between control knowledge in terms of logical skills and numeric utility functions. A driving simulation shows that this enables learning a variety of agent behaviors (e.g. different driving styles) based on the same set of skills.

Yet another powerful cognitive architecture is SOAR. SOAR is strong at knowledge-rich, symbolic reasoning but weak at knowledge-lean, statistical-based learning. Recently, Nason and Laird (2004a,b) incorporated elements of relational RL into the SOAR architecture, creating SOAR-RL. SOAR-RL employs model-free RL (SARSA) in order to learn preferences for operators.

Other decision-theoretic agent programming languages that can be used to specify MDPs over relational domains are, for example, the *integrated Bayesian Agent Language* (IBAL) (Pfeffer, 2001), the *independent choice logic* (ICL) (Poole, 1996) and ALisp (Andre and Russell, 2002) but there are many more.

4.6.3 Multi-Agent Systems

Hierarchies and decision-theoretic languages are aimed at the individual agent, although many agents will be embedded in *multi-agent systems* (Ferber, 1999; Weiss, 1999). Some methods have been proposed that explicitly address multi-agent learning in relational contexts. A recent survey on RL in multi-agent contexts is (Gu and Yang, 2004).

Letia and Precup (2001) report on multiple agents, modelled as *independent reinforcement learners* in the Golog extension (Levesque et al., 1997) of situation calculus (Reiter, 2001; McCarthy, 1963). The agents do not communicate, but act in the same environment. Golog programs specify initial plans and knowledge about the environment. The complex actions in the Golog programs induce a semi-MDP, and learning is performed by model-free RL methods based on the *options* framework (Sutton et al., 1999).

Finzi and Lukasiewicz (2004a) introduce GTGolog, a *game-theoretic* extension of the Golog language. This language integrates explicit agent programming in Golog with game-theoretic multi-agent planning in Markov Games. Finzi and Lukasiewicz (2004b) define *relational Markov Games*, which are static structures in stochastic situation calculus, that can be used to abstract over multi-agent RMDPs and to compute *Nash policy pairs*.

Hernandez et al. (2004) describe an approach that combines BDI (beliefs, desires, intentions, see more on this Wooldridge, 2002) approaches with learning capabilities (logical decision trees) in a multi-agent context. Finally, Tuyls et al. (2005) discuss in a position paper issues about using the system RRL (see Section 4.2) in multi-agent contexts.

4.7 Other Approaches

Finally, in this last section we mention three approaches that do not fit easily in the classification so far. A very interesting recent approach by (Asgharbeygi et al., 2005) uses RRL methods to guide *inference* in order to adapt to the current scenario. This *adaptive logic interpreter* uses RRL techniques to focus reasoning on high-utility parts of the knowledge base. The approach deviates from all existing RRL techniques, in that it does not focus on the traditional task of action learning. Lane and Wilson (2005) focus on environments

having strong *topological* structure. Many navigational tasks have a special structure that has a natural relational representation in terms of topological relationships, such as used in *the point n distance to the south of the wall*. Policies can be reused or *relocated* by making use of the properties of the underlying metrics and topologies. Baum (1999) uses evolutionary methods to learn policies for (propositional) Blocks World . Individuals in the population are condition-action pairs, and a value for this action is learned by temporal difference learning. When an action has to be taken, each individual can bid on getting the right to perform its action, using its value as a bid. Within this *economy* of individuals, each individual should learn an accurate value of its worth, and evolution ensures that low-valued individuals are discarded and high-valued ones are propagated through the population.

5. The Challenges Ahead

The first part of this survey described the MDP setting of which the work in RLL can be seen as an upgrade to relational representations. Section 4 has surveyed a large number of approaches that have been proposed in the literature. Various systems have been proposed on the whole landscape of RL methods. In this section we will present some of the remaining challenges and directions for further research (see also van Otterlo and Kersting, 2004).

5.1 Upgrading the spectrum of RL

The most obvious of all challenges presents itself immediately. The field of RL has presented a large body of research, consisting of a large number of methods. We have described some of the main approaches in Section 2, but there are many more. A continuing challenge is to upgrade more methods to the relational domain. We would like to mention some areas that deserve attention.

Some combinations of methods involving inductive and deductive procedures have been presented, such as the approximate methods in model-based RRL (see Section 4.4). More of these approaches might be developed by taking advantage of the strong points of both. For example, combinations of model-learning with model-based solution techniques have not yet been explored (although CARCASS representations (van Otterlo, 2004) explore model-building on fixed structures). Also, methods such as approximate FOVIA (Karabaev and Skvortsova, 2005) that implement asynchronous deductive updates focused on relevant areas of the state space, are interesting.

Partially observable MDPs (POMDPs) (Kaelbling et al., 1998) have not been explicitly addressed in relational contexts. However, the field of POMDPs has developed many exact and approximate algorithms that can be upgraded to the relational case. Perhaps a useful way to start is to make a connection with *modal* (or, epistemic) logic (see for references Wooldridge, 2002), which is a rich branch of logics dealing with modelling *belief states*. Logics such as situation calculus have already been extended with epistemic aspects. Given that this language underlies a number of approaches in this survey, this might be a start.

The most recent field within RL, that of predictive representations of state (PRS) (Littman et al., 2001) has not yet been considered in the relational domain. For this, one might have to start from temporal logics, but this direction is still completely open.

5.2 Theoretical Explorations of Relational Reinforcement Learning

The development of theoretical insights has not kept pace with empirical work. There exists a number of experimental approaches, each targeting a specific learning task or domain. However, development of theories on how and why some methods work is still in an initial phase. Some work has started in the modelling approaches to identify useful concepts that can be transferred to the inductive approaches, such as *abstraction levels*. From a logical point of view, the deductive approaches are more amenable to formal analysis, but because the inductive approaches are based on the same semantics – an RMDP – both may benefit.

5.2.1 Exploring Abstraction Levels

Depending on the complexity of the logic used, various *abstraction levels* can be defined, even over infinite domains. If the abstraction level is too general, *partial observability* might be introduced. Various logical connectives and quantifiers give more representational powers to abstraction levels – and are especially useful for the deductive approaches– but this in turn influences learning possibilities for inductive approaches. The representational power of abstraction levels determines whether or not policies can be learned or represented.

Abstraction levels can be homogeneous or inhomogeneous (Givan et al., 2003; Kim and Dean, 2003). Current deductive approaches aim at exact solutions and keep their abstractions value-homogenous. However, abstractions obtained in inductive approaches might be more compact and more eligible for generalization to other domains. An explicit connection to related work on *adaptive resolution* in propositional systems was only made by (Driessens et al., 2001) by upgrading a propositional tree learner based on state-splitting. Related work in model-free and model-based state-splitting can be exploited to develop similar techniques for relational representations. Splitting (and merging) state descriptions in relational languages, is more complex than in the propositional case though. Splitting in the joint state-action space is not considered much in the propositional case – except in continuous spaces – where usually the state space is partitioned and the action set is kept separate. For RMDPs the joint space is more important because of variables connecting states and actions.

Comprehensibility of learned abstractions is one of the promises of RRL. Although relational rules can be more comprehensible than e.g. the weights of a neural network, there are some trade-offs: consider the following abstraction in the Blocks World :

(1) $\texttt{on(a,A)}, \texttt{cl(a)}, \texttt{on(b,B)}, \texttt{on(X,b)}, \texttt{cl(X)}, \texttt{X} \neq \texttt{a}$
(2) $\texttt{on(a,A)}, \texttt{on(X,a)}, \texttt{cl(X)}, \texttt{on(b,B)}, \texttt{cl(b)}, \texttt{X} \neq \texttt{b}$
(3) $\texttt{on(a,A)}, \texttt{on(b,a)}, \texttt{cl(b)}$
(4) $\texttt{on(a,A)}, \texttt{on(X,a)}, \texttt{cl(X)}, \texttt{on(b,B)}, \texttt{cl(b)}$

The abstraction levels $\{1, 2, 3\}$ and $\{1, 4\}$ are logically equivalent. From a *minimum description length* (MDL) point of view, the latter is preferred. However, most humans will overlook that in (4) variable `X` can unify with `b`. But if we have to visually inspect 100 rules or 10, less rules are preferred. So, although logical representations may yield *comprehensible* abstractions, there are important trade-offs to be considered.

The abstractions currently used are of the *aggregation*, or *averaging* (Gordon, 1995) type. So far, *exact* versions of this are used. However, more robust (and possibly more

compact) abstractions could be explored using *soft aggregation*, where the abstractions are overlapping and aggregation is probabilistic in nature (e.g. see Sanner, 2005).

Yet another dimension is *action abstraction*. Because action abstractions are usually (syntactically) dependent on state or precondition descriptions, state and action abstractions are more tightly connected than in propositional representations. The consequences of this for modelling abstraction levels has not been addressed explicitly.

5.2.2 Proofs of Convergence

Convergence analysis is an important issue. Work on this is limited to some initial convergence results (see Ramon, 2005; Kersting and De Raedt, 2004) and error bounds (see Guestrin et al., 2003a; Sanner and Boutilier, 2005) for specific systems. Relational abstraction is essentially a form of *function approximation*, for which – especially in model-free learning – convergence guarantees are scarce. Given the fact that general relational abstraction can be viewed as *averaging* (Gordon, 1995) useful bounds might be computed for fixed abstraction levels. However, because most inductive approaches are of the *adaptive resolution* kind, i.e. the abstraction level changes during the learning process, it is largely open how to approach convergence in these contexts (but see Ramon, 2005). In general one has to distinguish between *structural* convergence, i.e. obtaining a stable abstraction level and *value* convergence within an abstraction level. Not converging on the structural level does not have to exclude computing an optimal policy (Kersting et al., 2004). For the deductive approaches one could use *stochastic bisimulation* approaches (Givan et al., 2003) to asses the difference between abstraction and RMDP. The use of *action abstraction* in general breaks many existing convergence proofs and is an important open problem.

5.2.3 Complexity Trade-Offs

The trade-off between more powerful abstractions and the increased cost of manipulating relational structures should be investigated. For some classes of problems *propositionalization* might be more efficient in terms of computation, memory or sample complexity. It is interesting to study how the relation between abstraction level and RMDP changes if the latter grows, possibly to infinite size. Furthermore, using more expressive logics such as situation calculus enables more powerful abstraction (lower space complexity) but it comes with an increased cost of manipulating and learning them. Comparing propositional and first-order languages in terms of MDP-specific criteria, such as number of episodes, number of value backups and the relative sizes of value functions and policies are interesting and mostly absent. For the inductive approaches the number of needed samples is important, whereas in the deductive approaches the cost of manipulating expressions is an important issue (e.g. theorem proving). Combinations of deduction and induction such as (Gretton and Thiébaux, 2004) are interesting in this respect. Furthermore, being able to *reason* about experience might help solve *partially observable* problems. These kind of problems have not been approached in RRL yet.

Interestingly, complexity trade-offs in relational RL can be approached from either the RL perspective or the logical perspective. From the viewpoint of RL, approximations can be introduced by making use of more efficient update schemes such as more efficient *exploration* and *asynchronous* versions of dynamic programming algorithms (such as variants of

MPI), see Section 2. In this way, updates are concentrated on regions in the state(-action) space that are actually needed to derive optimal policies. Examples are *envelopes* (REBP), heuristic search (*approximate* FOVIA) and *guidance* in the RRL system, but also efficient choices for domain instantiations such as used in the *random walks* approach in the API framework. From a logical viewpoint, approximations can be introduced by *coarser* abstraction levels. For example, the FOALP approach limits the representation of the value function to a finite set of *logical basis functions*. Related to this, the model-free approaches TG and RIB allow a small variance in the values of the abstraction level. Combinations of both types of approximations will prove vital to scaling up to larger domains.

5.3 Values vs. Policies

The overall goal RRL is to learn an optimal abstract policy $\hat{\pi}^*$. The majority of work is *value-based*. However, whereas the relational structure of the value function can be large (or infinite), a corresponding optimal policy can be very compact, by generalizing over objects *and* values (see for an example in the propositional case Anderson, 2000). As an example, consider (part of) the value function for the goal $\texttt{on}(\texttt{a},\texttt{b})$ in Blocks World as shown in figure 10. The structure of the value function is relatively complex, and might need an infinite number of abstract states. However, even in infinite worlds, an optimal policy can be stated as the very short decision list from section 3.2.3. Additionally, it can be learned from a small part of the value function. Recent work has shown that for induction of finite policies *background knowledge* can actually be a *necessity*, and not a mere *feature* (Kersting et al., 2004). The use of background knowledge enables new ways of abstraction and it has not been analyzed explicitly yet.

Optimal value functions may have infinite range and can be heavily *shattered*. This creates difficulties especially for model-free learners such as RRL. But also deductive approaches are hindered by the explosion of the value partition. For that reason, it is also interesting to consider *direct policy* learning. There are at least three interesting directions to explore: (1) Upgrading *policy gradient* techniques (Sutton et al., 2000) to the relational case seems a interesting direction. Related are *generalized expectation maximization* (GEM) approaches. In both cases however, the problem of how to *parameterize* abstract policies to derive necessary gradients should be solved. (2) The use of *evolutionary algorithms* (EA) has proved very efficient in standard RL (Moriarty et al., 1999), e.g. evolution of neural networks is a powerful way of obtaining neural networks representing control policies. In the relational case, an evolutionary learning approach (see for an introduction Divina, 2004) inducing relational abstractions can be explored for learning abstract policies. (3) As was done for learning first-order action models (Pasula et al., 2004) and policies (Yoon et al., 2002) *supervised* learning abstractions can be done using ILP methods. For this direction, more suitable contexts for obtaining optimal samples or traces should be explored. It is interesting to analyze the policy versus value trade-off in the relational setting and to also see whether approximative and probabilistic approaches are the answer to shattered value functions.

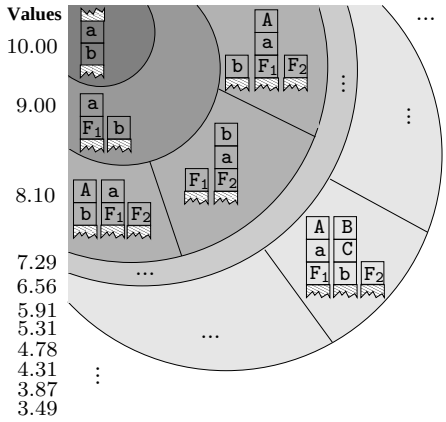


Figure 10: Parts of an abstract state value function for the goal on(a, b) after 10 iterations (Kersting et al., 2004). F_i can be a block or the floor. The fine-grained structure of the value function is prototypical for many relational domains.

5.4 Exploring Generalization

Because relational formalisms abstract over objects, it is tempting to learn policies in small domains and apply them to larger domains. However, in probabilistic domains, due to the domain *size* this might not work. Guestrin et al. (2003a) give insight into a restricted case but in general, it is an open problem to determine when generalization can be applied.

Consider for example Blocks World with probabilistic actions. Assume that one of the outcomes of move(X, Y) is that block X is moved onto somewhere else (i.e. not on Y). An abstract n-steps-to-go value function generalizes no matter how many blocks there are. However, if we consider a Blocks World where this outcome is to move onto *some other block*, it depends on how many blocks there are and how many stacks, i.e. the probability distribution over the next states depends on this (and so may be the reward). In such a context, the value function is dependent on the number of blocks. In some contexts, even the policy may depend on the domain size. In other words, *upgrading* learned value functions or even policies to larger domains might not always be valid. Insight is needed into when *upgrading* from small problem instances (RMDPs) is applicable. As shown by Kersting et al. (2004), sometimes *background knowledge* might solve the policy induction problem, but also here insight is needed how and when to apply this. Overall, changing the semantics by adding domain elements might be harmful to the abstraction level and analysis is needed, especially for the infinite case. Abstract definitions of RMDPs essentially represent whole *families* of RMDPs, depending on e.g. the size of the domain. Theory should be developed on these matters.

5.5 Proofs of Soundness & Completeness

Many different logical formalisms can be (and have been) employed for abstraction, such as Horn logic and description logics. It is important to provide *mappings* between these frameworks for comparison purposes. Also important are *representation theorems*: when in-

ducing or defining an abstraction level, one has to consider the relation between abstraction level and the underlying RMDP. For the deductive approaches, *soundness* of the reasoning process and simplification of expressions is important, e.g. ReBel (Kersting et al., 2004) uses domain constraints to enforce this. The modelling approaches too have to ensure that the models they define corresponds to a correct underlying RMDP. For the inductive approaches, *completeness* is more an issue. Induced structures should be rich enough to model the underlying RMDP.

5.6 Agents, Reasoning and Transfer

One approach to RRL is by upgrading RL techniques to relational representations. An opposite approach is to consider rich agent languages and cognitive architectures and extending them with a *utility framework* and means to learn behaviors. Methods in both these directions have been described in Section 4. An natural question is to ask where these directions collide and in what ways both fields can learn from and contribute to each other. Cognitive architectures such as SOAR have many reasoning capabilities that can be employed to reason over the experience gained in a learning process. This knowledge can be reused by forming plan libraries, or task hierarchies such that skills can be identified and transferred to other learning tasks. The use of communication in multi-agent systems would enable to transfer knowledge between agents. The policy abstractions learned by many of the systems occurring in this survey can be seen as induced, procedural knowledge that would normall have been specified by the designer of some agent system. An important overall direction is one in which RRL is not seen in isolation as just a set of methods solving RMDPs, but seen as a component in more complex agent architectures capable of learning, reasoning, planning, communicating and more.

5.7 Applications

A final important issue is convincing (real-life) *applications*. The Blocks World (Slaney and Thiébaux, 2001) is used in the majority of systems, but it should be viewed upon as – in analogy to genetics – a *Drosophila*. Among others, the following applications have also appeared in this survey: Tetris, Digger, Tic-Tac-Toe, various logistics and planning domains, Backgammon, robot soccer, Chess subgames and dialogue systems. Although this represents a wide variety of problems, most of them are small and the purpose is to show the viability of the approaches. In order to more fully show the benefits of relational representations for MDPs, one should solve larger, problems for which propositional algorithms are insufficient.

Taking inspiration from the field of *probabilistic logic learning* (De Raedt and Kersting, 2003, 2004), *webmining* and *bio-informatics* applications are challenging domains due to their size and their intrinsically relational structure. RRL upgrades of RL *Webspiders* (Rennie and McCallum, 1999) would be very interesting. Other domains in which learning and reasoning could be combined are interesting, for example in *multi-agent* contexts. Here, many interesting problems can be explored, such as communication and cooperation. Agents in these contexts are often modeled in terms of first-order languages, whereas usually behavior learning is performed using propositional languages or ad-hoc methods are applied to connect learning to the structural representations of knowledge.

Furthermore, in the line of games such as Chess, Backgammon, Tetris, Digger, it would be very interesting to approach the game of *Go* or to find applications in the general area of *computer games* (Rabin, 2002; Amir and Doyle, 2002). Recent progress in real-time shooters such as *Unreal* show the viability of logic-based approaches (Jacobs et al., 2005). A very interesting area consists of *real-time strategy* games, of which the domain taken by Guestrin et al. (2003a), – *FreeCraft* – is an example.

6. Conclusions

Relational Reinforcement Learning aims at tackling one of the main goals of artificial intelligence, namely planning, acting, learning and reasoning in richly structured, probabilistic domains. In the first years after the seminal work by Džeroski et al. (1998) some methods were introduced, but in recent years, the field has rapidly expanded by introducing a whole range of methods for model-based and model-free solution techniques. A central issue is how to trade-off the complexity of the logical abstractions and useful approximations to be used in increasingly larger problems. The approximative methods in the model-based area show many interesting techniques for this trade-off. The coming years, the field should continue to expand by upgrading more solution techniques from the classical RL framework. However, theoretical advances are needed to be able to understand better the various interactions between logic, utility and probability. Furthermore, a direction that will prove fruitful for general artificial intelligence is by connecting agents and powerful logical reasoning systems with relational RL. Some of the techniques in this paper have shown considerable progress in this respect, and it is interesting, and vital, that more work studies the interaction between reasoning and learning. Making use of lots of available knowledge breaks with the *tabula rasa* style of early work in RL and will bring the construction of even more intelligent agents closer.

We hope that this survey will inspire researchers of such diverse fields as computer science, AI, statistics, logic, and cognitive science to contribute to this exciting field.

References

D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

R. Aler, D. Borrajo, and P. Isasi. Knowledge representation issues in control knowledge learning. In P. Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning, ICML'00*, pages 1–8, 2000.

E. Allender, S. Arora, M. Kearns, C. Moore, and A. Russell. A note on the representational incompatability of function approximation and factored dynamics. In *Proceedings of NIPS'02*, 2002.

E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, Cambridge, Massachusets, 2004.

E. Amir and P. Doyle. Adventure games: A challenge for cognitive robotics. In *Proceedings of the AAAI'02 Workshop on Cognitive Robotics*, 2002.

C.W. Anderson. Approximating a policy can be easier than approximating a value function. Technical Report CS-00-101, Computer Science Department, Colorado State University, February 2000.

D. Andre and S.J. Russell. State abstraction for programmable reinforcement learning agents. In *Proceedings of AAAI'02*, 2002.

N. Asgharbeygi, N. Nejati, P. Langley, and S. Arai. Guiding inference in reactive agents through relational reinforcement learning. In *Proceedings of the International Conference on Inductive Logic Programming*, 2005.

M. Aycenina. Hierarchical relational reinforcement learning, 2002. Stanford Doctoral Symposium.

B. Bakker and J. Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. In A. Bonarini E. Yoshida F. Groen, N. Amato and B. Kröse, editors, *Proceedings of the 8-th Conference on Intelligent Autonomous Systems, IAS-8, Amsterdam, The Netherlands*, pages 438–445, 2004.

A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event systems*, 13(4):341–379, 2003.

A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

E. B. Baum. Toward a model of intelligence as an economy of agents. *Machine Learning*, 35(2):155–185, 1999.

R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.

S.S. Benson. *Learning action models for reactive autonomous agents*. PhD thesis, Stanford University, 1996.

D. P. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

H. Blockeel and L. de Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.

H. Blockeel, L. de Raedt, and J. Ramon. Top-down induction of clustering trees. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 55–63. Morgan Kaufmann, 1998.

J. Blythe. An overview of planning under uncertainty. *Lecture Notes in Computer Science*, 1600:85–110, 1999.

M. Bongard. *Pattern Recognition*. Hayden Book Company, 1970.

M. Botta, A. Giordana, and R. Piola. FONN: Combining first-order logic with connectionist learning. In *Proceedings of the Fourteenth International Conference on Machine Learning, San Francisco, CA*, pages 48–56. Morgan Kaufmann, 1997.

C. Boutilier. Knowledge representation for stochastic decision processes. *Lecture Notes in Computer Science*, 1600:111–152, 1999.

C. Boutilier. Planning and programming with first-order representations of Markov decision processes, 2001. accompanying paper for the invited talk at TARK VIII.

C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1–2):49–107, 2000a.

C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDP's. In Bernhard Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 690–697, San Francisco, CA, August 4–10 2001. Morgan Kaufmann Publishers, Inc.

C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-Theoretic, high-level agent programming in the situation calculus. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 355–362, Menlo Park, CA, July 30– 3 2000b. AAAI Press.

R.J. Brachman and H.J. Levesque. *Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers, 2004.

I. Bratko. *Prolog, Programming for Artificial Intelligence (3rd edn).* Addisson-Wesley, 2001.

A. Browne and R. Sun. Connectionist inference models. *Neural Networks*, 14(10):1331–1355, December 2001.

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

D. Choi, M. Kaufman, P. Langley, N. Nejati, and D. Shapiro. An architecture for persistent reactive behavior. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 988–995, 2004.

A. Clark and C. Thornton. Trading spaces: Computation, representation, and the limits of uninformed learning. *Behavioral and Brain Sciences*, 20(506):57–90, 1997.

J. Cole, K.W. Lloyd, and K.S. Ng. Symbolic learning for adaptive agents. In *Proceedings of the Annual Partner Conference, Smart Internet Technology Cooperative Research Centre*, 2003. http://csl.anu.edu.au/jwl/crc_paper.pdf.

T. Croonenborghs, J. Ramon, and M. Bruynooghe. Towards informed reinforcement learning. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004.

L. De Raedt. Logical settings for concept-learning. *Artificial Intelligence*, 95(1):187–201, 1997.

L. De Raedt and K. Kersting. Probabilistic logic learning. *ACM-SIGKDD Explorations, special issue on Multi-Relational Data Mining*, 5(1):31–48, July 2003.

L. De Raedt and K. Kersting. Probabilistic inductive logic programming. In *Proceedings of the 15th International Conference on Algorithmic Learning Theory (ALT-2004)*, 2004.

R. Dearden and C. Boutilier. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence*, 89(1–2):219–283, 1997.

D. C. Dennett. *The Intentional Stance*. The MIT Press, 1987.

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

T.G. Dietterich and N.S. Flann. Explanation-based learning and reinforcement learning: A unified view. *Machine Learning*, 28(503):169–210, 1997.

T.G. Dietterich and X. Wang. Batch value function approximation using support vectors. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Processing Systems 14 (NIPS'02)*, Cambridge, MA, 2002.

Federico Divina. *Hybrid Genetic Relational Search for Inductive Learning*. PhD thesis, Department of Computer Science, Vrije Universiteit, Amsterdam, the Netherlands, 2004.

P. Domingos and M. Richardson. Markov logic: A unifying framework for statistical relational learning. In *Proceedings of the ICML-2004 Workshop on Statistical Relational Learning and its Connections to Other Fields, (at ICML'04 Banff, Canada)*, pages 49–54, 2004.

K. Driessens. *Relational reinforcement learning*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, may 2004.

K. Driessens and H. Blockeel. Learning digger using hierarchical reinforcement learning for concurrent goals. In *Proceedings of the European Workshop on Reinforcement Learning, EWRL'01*, oct 2001.

K. Driessens and S. Dzeroski. Integrating experimentation and guidance in relational reinforcement learning. *Machine Learning*, xx(xx):xxx–xxx, 2004.

K. Driessens and S. Dzeroski. Combining model-based and instance-based learning for first order regression. In *Proceedings of the International Conference on Machine Learning ICML'05*, 2005.

K. Driessens and J. Ramon. Relational instance based regression for relational reinforcement learning. In *Proceedings of the International Conference on Machine Learning ICML'03*, 2003.

K. Driessens, J. Ramon, and H. Blockeel. Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. In *Proceedings of ECML - European Conference on Machine Learning*, volume 2167 of *LNAI*, pages 97–108. Springer-Verlag, 2001.

S. Dzeroski. Relational reinforcement learning for agents in worlds with objects. In *In Proceedings of the AISB'02 Symposium on Adaptive Agents and Multi-Agent Systems*, pages 1–8, 2002.

S. Dzeroski. Learning in rich representations: Inductive logic programming and computational scientific discovery. In *Proceedings of the Twelfth International Conference on Inductive logic programming (ILP'03)*, pages 346–349. Springer, Berlin, 2003.

S. Džeroski, L. De Raedt, and H. Blockeel. Relational reinforcement learning. In J. Shavlik, editor, *Proceedings ICML'98*, pages 136–143. Morgan Kaufmann, 1998.

S. Dzeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.

S. Dzeroski and N. Lavrac. Introduction to inductive logic programming. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 48–73. Springer-Verlag, September 2001a. ISBN 3-540-42289-7.

S. Dzeroski and N. Lavrac. *Relational Data Mining*. Springer, Berlin, 2001b.

Z. Feng, R. Dearden, N. Meuleau, and R. Washington. Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI-2004)*, 2004.

J. Ferber. *Multi-Agent Systems*. Pearson Education Unlimited, Great Britain, 1999.

A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias. In *Proceedings of the Neural Information Processing Conference (NIPS'03)*, 2003.

A. Fern, S. Yoon, and R. Givan. Learning domain-specific control knowledge from random walks. In *14Th International Conference on Automated Planning & Scheduling (ICAPS'04)*, 2004.

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

D.J. Finton. *Cognitive Economy and the Role of Representation in On-line Learning*. PhD thesis, University of Wisconsin, Madison, 2002.

A. Finzi and T. Lukasiewicz. Game-theoretic agent programming in Golog. In *Proceedings of the European Conference on AI*, 2004a.

A. Finzi and T. Lukasiewicz. Relational Markov games. In *Proceedings of JELIA'04*, 2004b.

J. Fischer. Asynchrone relationale werte iteration. Master's thesis, Albert-Ludwigs-University Freiburg, Germany, 2005. In German.

P. Flach and N. Lachiche. *1BC: A first-order Bayesian Classifier*, volume 1634 of *Lecture Notes in Articial Intelligence*, pages 14–27. Springer-Verlag, Berlin, 1999.

P. A. Flach. *Simply Logical: Intelligent Reasoning by Example*. John Wiley, 1994.

N.H. Gardiol and L.P. Kaelbling. Envelope-based planning in relational MDPs. In *Proceedings of the Neural Information Processing Conference (NIPS'03)*, 2003.

T. Gärtner, K. Driessens, and J. Ramon. Graph kernels and Gaussian processes for relational reinforcement learning. In *Proceedings of the International Conference on Inductive Logic Programming (ILP'03)*, 2003.

L. Getoor, N. Friedman, D. Koller, and B. Taskar. Probabilistic models of relational structure. In *International Conference on Machine Learning (ICML'01), Williamstown, MA*, 2001.

Z. Ghahramani. Learning dynamic Bayesian networks. In C. L. Giles and M. Gori, editors, *Adaptive Processing of Sequence and Data Structures*, pages 168–197. Springer, 1998.

R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147:163–223, 2003.

G.J. Gordon. Stable function approximation in dynamic programming. In *Proc. of ICML'95*, 1995.

C. Gretton and S. Thiébaux. Exploiting first-order regression in inductive policy selection. In *20th Conference on Uncertainty in Artificial Intelligence (UAI-04), Banff, Canada*. Morgan Kaufmann, 2004.

A. Grossmann. Adaptive state-space quantisation and multi-task reinforcement learning using constructive neural networks. In J.-A. Meyer, A. Berthoz, D. Floreano, H.L. Roitblat, and S.W. Wilson, editors, *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior, Supplement Book*, pages 160–169, 2000.

A. Großmann, S. Hölldobler, and O. Skvortsova. Symbolic Dynamic Pprogramming within the Fluent Calculus. In Naohiro Ishii, editor, *Proceedings of the IASTED International Conference on Artificial and Computational Intelligence*, pages 378–383, Tokyo, Japan, September 25-27 2002. ACTA Press. ISBN: 0-88986-358-X.

M. Grounds and D. Kudenko. Combining reinforcement learning with symbolic planning. In *Proceedings of the Fifth European Workshop on Adaptive Agents and Multi-Agent Systems, Paris*, 2005.

D. Gu and E. Yang. Multiagent reinforcement learning for multi-robot systems: A survey. Technical Report CSM-404, Dept. of Computer Science, University of Essex, 2004.

Y. Gu. Macro-actions in the stochastic situation calculus. In *Proceedings of IJCAI'03 Workshop on Nonmonotonic Reasoning, Action, and Change, August, 2003, Acapulco, Mexico.*, 2003.

C. Guestrin. *Planning Under Uncertainty in Complex Structured Environments.* PhD thesis, Computer Science Department, Stanford University, August 2003.

C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational MDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'03), Acapulco, Mexico*, 2003a.

C. Guestrin, D. Koller, and R. Parr. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 19:399–468, 2003b.

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

B. Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In C. Sammut and A. Hoffmann, editors, *Proceedings of the International Conference on Machine Learning (ICML'02), Sidney, Australia*, pages 243–250, San Francisco, 2002. Morgan Kaufmann Publishers Inc.

A.G. Hernandez, A. El Fallah-Seghrouchni, and H. Soldano. Learning in BDI multi-agent systems. In *Proceedings of CLIMA IV - Computational Logic in Multi-Agent Systems*, 2004.

N. Hernandez-Gardiol. Applying probabilistic rules to relational worlds. Master's thesis, Department of Electrical Engineering and Computer Science, Massachustts Institute of Technology, 2003.

J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In Kathryn B. Laskey and Henri Prade, editors, *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 279–288, S.F., Cal., July 30–August 1 1999. Morgan Kaufmann Publishers.

S. Holldobler and O. Skvortsova. A logic-based approach to dynamic programming. In *Proceedings of the AAAI-04 Workshop on Learning and Planning in Markov Processes - Advances and Challenges*, 2004.

R.A. Howard. *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge, Massachusetts, 1960.

M. Irodova and R.H. Sloan. Reinforcement learning and function approximation. In *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*, 2005.

H. Itoh and K. Nakamura. Towards learning to learn and plan by relational reinforcement learning. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004.

S. Jacobs, A. Ferrein, and G. Lakemeyer. Unreal Golog bots. In *Proceedings of the IJCAI'05 Workshop on Reasoning, Representation, and Learning in Computer Games*, 2005.

L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stoachastic domains. *Artificial Intelligence*, 101:99–134, 1998.

L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

L.P. Kaelbling, T. Oates, N. Hernandez, and S. Finney. Learning in worlds with objects. In *The AAAI Spring Symposium*, March 2001.

E. Karabaev and O. Skvortsova. Fcplanner: A planning strategy for first-order MDPs, 2004. ICAPS'04 planning competition, probabilistic track.

E. Karabaev and O. Skvortsova. A heuristic search algorithm for solving first-order MDPs. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI'05)*, 2005.

A. Karalic and I. Bratko. First-order regression. *Machine Learning*, 26:147–176, 1997.

K. Kersting and L. De Raedt. Logical Markov decision programs. In *Proceedings of the IJCAI'03 Workshop on Learning Statistical Models of Relational Data*, 2003.

K. Kersting and L. De Raedt. Logical Markov decision programs and the convergence of TD(λ). In *Proceedings of the Conference on Inductive Logic Programming*, 2004.

K. Kersting, M. van Otterlo, and L. De Raedt. Bellman goes relational. In *Proceedings of the International Conference on Machine Learning (ICML'04)*, 2004.

R. Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113: 125–148, 1999a.

R. Khardon. Learning to take actions. *Machine Learning*, 35(1):57–90, 1999b.

K.-E. Kim and T. Dean. Solving factored MDPs using non-homogeneous partitions. *Artificial Intelligence*, 147:225–251, 2003.

S. Koenig and Y. Liu. The interaction of representations and planning objectives for decision-theoretic planning. *Journal of Experimental and Theoretical Artificial Intelligence*, 2002.

D. Koller and A. Pfeffer. Learning probabilities for noisy first-order rules. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1316–1323, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers. ISBN 1-55860-480-4.

M.G. Lagoudakis and R. Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*, 2003.

T. Lane and A. Wilson. Toward a topological theory of relational reinforcement learning for navigation tasks. In *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*, 2005.

P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, San Francisco, 1996.

P. Langley, S. Arai, and D. Shapiro. Model-based learning with hierarchical relational skills. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004.

N. Lavrac and S. Dzeroski. *Inductive Logic Programming*. Ellis Harwood, New York, 1994.

R. Lecoeuche. Learning optimal dialogue management rules by using reinforcement learning and inductive logic programming. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, Pittsburgh, June 2001.

I.A. Letia and D. Precup. Developing collaborative Golog agents by reinforcement learning. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01)*. IEEE Computer Society, 2001.

H.J. Levesque, R. Reiter, and Y. Lesperance. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83, 1997.

I. Levner, V. Bulitko, O. Madani, and R. Greiner. Performance of lookahead control policies in the face of abstractions and approximations. *Lecture Notes in Computer Science*, 2371: 299–307, 2002. ISSN 0302-9743.

M.L. Littman, R.S. Sutton, and S. Singh. Predictive representations of state. In *Proceedings of NIPS'01*, 2001.

J. W. Lloyd. *Logic for learning: learning comprehensible theories from structured data*. Springer-Verlag, 2003.

J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1991. second edition.

M.A. Maloof. Incremental rule learning with partial instance memory for changing concepts. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2764–2769, 2003.

M. Martin and H. Geffner. Learning generalized policies in planning using concept languages. In *Proceedings of the Seventh Internatinal Conference on Knowledge Representation and Reasoning (KR 2000)*, 2000.

Mausam and D.S. Weld. Solving relational MDPs with first-order machine learning. In *Workshop on Planning under Uncertainty and Incomplete Information at ICAPS'03, Trento, Italy*, 2003.

A.K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, Rochester, New York, 1996.

J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University, 1963.

S. Minton, J. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40(1–3):63–118, 1989.

T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

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*, 3:1–24, 1995.

A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

E.F. Morales. Scaling up reinforcement learning with a relational representation. In *Proceedings of the Workshop on Adaptability in Multi-Agent Systems at AORC'03, Sydney*, 2003.

E.F. Morales. Learning to fly by combining reinforcement learning with behavioural cloning. In *Proceedings of the International Conference on Machine Learning (ICML'04)*, 2004a.

E.F. Morales. Relational state abstractions for reinforcement learning. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004b.

D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19 & 20:629–680, May 1994.

S. Nason and J.E. Laird. Soar-rl: Integrating reinforcement learning with soar. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004a.

S. Nason and J.E. Laird. Soar-rl: integrating reinforcement learning with soar. In *Proceedings of the Sixth International Conference on Cognitive Modeling*, pages 208–213, 2004b.

S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, February 1997.

D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49: 161–178, 2002.

V.A. Papavassiliou and S.J. Russell. Convergence of reinforcement learning with general function approximators. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*, pages 748–757, S.F., July 31–August 6 1999. Morgan Kaufmann Publishers.

H.M. Pasula, L. Zettlemoyer, and L.P. Kaelbling. Learning planning rules in noisy stochastic worlds. In *Proceedings of AAAI 2005*, 2005.

H.M. Pasula, L.S. Zettlemoyer, and L.P. Kaelbling. Learning probabilistic planning rules. In *14Th International Conference on Automated Planning & Scheduling (ICAPS'04)*, 2004.

L. Peshkin. Reinforcement learning by policy search. Technical Report AITR-2003-003, MIT Artificial Intelligence Laboratory, February 14 2003.

A. Pfeffer. IBAL: A probabilistic rational programming language. In *Proceedings of IJCAI'01*, pages 733–740, 2001.

G. D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

D. Poole. A framework for decision-theoretic planning I: Combining the situation calculus, conditional plans, probability and utility. In Eric Horvitz and Finn Jensen, editors, *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 436–445, San Francisco, August 1–4 1996. Morgan Kaufmann Publishers. ISBN 1-55860-412-X.

M.L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

M.L. Puterman and M.C. Shin. Modified policy iteration algorithms for discounted Markov decision processes. *Management Science*, 24:1127–1137, 1978.

S. Rabin, editor. *AI Game Programming Wisdom*. Charles River Media, Inc., Hingham, Massachusetts, 2002.

J. Ramon. Convergence of reinforcement learning using a decision tree learner. In K. Driessens, A. Fern, and M. van Otterlo, editors, *Proceedings of the ICML-2005 Workshop on Rich Representations for Reinforcement Learning*, 2005.

J. Ramon and K. Driessens. On the numeric stability of Gaussian processes regression for relational reinforcement learning. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004.

B. Ravindran and A.G. Barto. Smdp homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. In *Proceedings of IJCAI'03*, 2003.

R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, Cambridge, Massachusetts, 2001.

J. Rennie and A. McCallum. Using reinforcement learning to spider the web efficiently. In *Proceedings of ICML'99*, 1999.

A. Reyes, L.E. Sucar, E. Morales, and P. Ibarüengoytia. Relational MDPs using qualitative proportionality predicates: An application in power generation. In *NIPS'03 workshop on: Planning for the Real World: The promises and challenges of dealing with uncertainty*, 2003.

S.I. Reynolds. Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In *Proc. 17th International Conf. on Machine Learning*, pages 783–790. Morgan Kaufmann, San Francisco, CA, 2000.

S. Roncagliolo and P. Tadepalli. Function approximation in hierarchical relational reinforcement learning. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004.

G.A. Rummery and M. Niranjan. On-line Q-Learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University, Engineering Department, 1994.

S.J. Russel and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice Hall, New Jersey, 2003. 2nd edition.

M.R.K. Ryan. Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In C. Sammut and A. Hoffmann, editors, *Proceedings of the International Conference on Machine Learning (ICML'02), Sidney, Australia*, pages 522–529, San Francisco, 2002. Morgan Kaufmann Publishers Inc.

M.R.K. Ryan. *Hierarchical Reinforcement Learning: A Hybrid Approach*. PhD thesis, University of NSW, School of Computer Science and Engineering, Sidney, Australia, 2004.

B. Sallans. *Reinforcement Learning for Factored Markov Decision Processes*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, 2002.

S. Sanghai, P. Domingos, and D. Weld. Dynamic probabilistic relational models. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 2003.

S. Sanner. Simulteneous learning of structure and value in relational reinforcement learning. In *Proceedings of the ICML-2005 Workshop on Rich Representations for Reinforcement Learning*, 2005.

S. Sanner and C. Boutilier. Approximate linear programming for first-order MDPs. In *Proceedings of UAI-2005*, 2005.

D. Schuurmans and R. Patrascu. Direct value approximation for factored MDPs. In *Proceedings of NIPS'01*, 2001.

D. Shapiro and P. Langley. Separating skills from preference. In C. Sammut and A. Hoffmann, editors, *Proceedings of the International Conference on Machine Learning (ICML'02), Sidney, Australia*, pages 570–577, San Francisco, 2002. Morgan Kaufmann Publishers Inc.

S.P. Singh, T. Jaakkola, and M.I. Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 361–368. The MIT Press, 1995.

J. Slaney and S. Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125:119–153, 2001.

J. F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Thomson Learning, Stamford, Connecticut, 1999.

A. Srinivasan. A study of two probabilistic methods for searching large spaces with ilp. Technical Report PRG-TR-16-00, 2000.

R. Sun. Supplementing neural reinforcement learning with symbolic methods. In *Hybrid Neural Systems*, pages 333–347, 1998.

R. Sun and T. Peterson. Autonomous learning of sequential tasks: Experiments and analyses. *IEEE Transactions on Neural Networks*, 9(6):1217, November 1998.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. The MIT Press, 1996.

R.S. Sutton. DYNA, an integrated architecture for learning, planning and reacting. In *Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures*, pages 151–155, March 1991.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: an Introduction*. The MIT Press, Cambridge, 1998.

R.S. Sutton, D McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

R.S. Sutton and S.D. Whitehead. Online Learning with Random Representations. In *Machine Learning: Proceedings of the Tenth International Conference (ICML)*, pages 314–321, San Mateo, CA, 1993. Morgan Kaufmann.

P. Tadepalli, R. Givan, and K. Driessens. Relational reinforcement learning: An overview. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004.

M. Thielscher. Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence*, 2(3–4):179–192, 1998.

C. Thornton. *Truth from Trash: How Learning makes Sense*. The MIT Press, Cambridge, Massachusetts, 2000.

K. Tuyls, T. Croonenborghs, J. Ramon, R. Goetschalckx, and M. Bruynooghe. Multi-agent relational reinforcement learning. In *Proceedings of the LAMAS Workshop at AAMAS-2005*, 2005. position paper.

P.E. Utgoff and D. Precup. Constructive function approximation. In H. Liu and H. Motoda, editors, *Feature Extraction, Construction and Selection: A Data Mining Perspective*, volume 453 of *The Kluwer International Series in Engineering and Computer Science*, chapter 14. Kluwer Academic Publishers, 1998.

P.E. Utgoff and D.J. Stracuzzi. Many-layered learning. *Neural Computation*, 14(10):2497–2529, 2002.

W. van Laer and L. de Raedt. *How to Upgrade Propositional Learners to First Orde Logic: A Case Study*, volume 2049 of *Lecture Notes in Articial Intelligence*, pages 102–126. Springer-Verlag, 2001.

M. van Otterlo. Relational representations in reinforcement learning: Review and open problems. In E. de Jong and T. Oates, editors, *Proceedings of the ICML'02 Workshop on Development of Representations*, 2002.

M. van Otterlo. Efficient reinforcement learning using relational aggregation. In *Proceedings of the Sixth European Workshop on Reinforcement Learning, Nancy, France (EWRL-6)*, 2003.

M. van Otterlo. Reinforcement learning for relational MDPs. In A. Nowé, T. Lenaerts, and K. Steenhaut, editors, *Machine Learning Conference of Belgium and the Netherlands (BeNeLearn'04)*, pages 138–145, 2004.

M. van Otterlo and K. Kersting. Challenges for relational reinforcement learning. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004.

M. van Otterlo, K. Kersting, and L. De Raedt. Bellman goes relational (extended abstract). In *Proceedings of BNAIC'04*, 2004.

M. van Otterlo, M.A. Wiering, M. Dastani, and J.-J.Ch. Meyer. A characterization of sapient agents. In *Proceedings of the International Conference on the Integration of Knowledge Intensive Multi-Agent Systems KIMAS'03*, 2003.

T. Walker, J. Shavlik, and R. Maclin. .relational reinforcement learning via sampling the space of first-order conjunctive features. In *Proceedings of the Workshop on Relational Reinforcement Learning at ICML'04*, 2004.

C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning Journal*, 8(3/4), May 1992. Special Issue on Reinforcement Learning.

G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, Massachusets, 1999.

D.S. Weld. Recent advances in AI planning. *AI Magazine*, 20(2):93–123, 1999.

M. Wiering. *Explorations in Efficient Reinforcement Learning*. PhD thesis, Faculteit der Wiskunde, Informatica, Natuurkunde en Sterrenkunde, Universiteit van Amsterdam, 1999.

M. Wiering. Convergence and divergence in standard and averaging reinforcement learning. In J-F Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *Proceedings of the 15th European Conference on Machine Learning (ECML'04)*, pages 477–488, Berlin Heidelberg, 2004. Springer-Verlag.

M.A. Wiering and J.H. Schmidhuber. HQ-learning. *Adaptive Behavior*, 6(2):219–246, 1997.

M. Wooldridge. *An introduction to MultiAgent Systems*. John Wiley & Sons Ltd., West Sussex, England, 2002.

S. Yoon, A. Fern, and R. Givan. Inductive policy selection for first-order MDPs. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI'02, Edmonton, Canada)*, 2002.

S. Yoon, A. Fern, and R. Givan. Learning reactive policies for probabilistic planning domains, 2004. ICAPS'04 planning competition, probabilistic track.

S.W. Yoon, A. Fern, and R. Givan. Learning measures of progress for planning domains. In *Proceedings of AAAI-2005*, 2005.