Logical Markov Decision Programs and the Convergence of Logical TD(λ)

Kristian Kersting and Luc De Raedt

Institute for Computer Science, Machine Learning Lab
Albert-Ludwigs-University, Georges-Köhler-Allee, Gebäude 079,
D-79110 Freiburg i. Brg., Germany
{kersting,deraedt}@informatik.uni-freiburg.de

Abstract. Recent developments in the area of relational reinforcement learning (RRL) have resulted in a number of new algorithms. A theory, however, that explains why RRL works, seems to be lacking. In this paper, we provide some initial results on a theory of RRL. To realize this, we introduce a novel representation formalism, called logical Markov decision programs (LOMDPs), that integrates Markov Decision Processes (MDPs) with Logic Programs. Using LOMDPs one can compactly and declaratively represent complex MDPs. Within this framework we then devise a relational upgrade of TD(λ) called *logical* TD(λ) and prove convergence. Experiments validate our approach.

1 Introduction

In the past few years, there has been a lot of work on extending probabilistic and stochastic frameworks with abilities to handle objects and relations, see [6] for an overview. From an inductive logic programming or relational learning point of view, these approaches are upgrades of propositional representations towards the use of relational or computational logic representations.

The first contribution of the present paper extends this line of research towards decision making. We introduce a novel representation formalism, called *logical Markov decision programs* (LOMDPs), that combines MDPs with *logic programming.* The result is a flexible and expressive framework for defining MDPs that are able to handle structured objects as well as relations. For MDPs, such a framework, grounded in computational logic, has been missing. Only [4] report on combining MDPs with Reiter's situation calculus. However, it is more complex, and model-free reinforcement techniques have not yet been addressed. LOMDPs share - with the other upgrades of propositional representations - two advantages. First, logical expressions (in the form of clauses, rules or transitions) may contain variables and as such make *abstraction* of many specific *grounded* transitions. This allows one to compactly represent complex domains. Secondly, because of this abstraction, the number of parameters (such as rewards and probabilities) is significantly reduced. This in turn allows one - in principle - to speed up and simplify the learning because one can learn at the *abstract* rather than at the *ground* level.

Many fascinating machine learning techniques have been developed under the name *reinforcement learning* (RL) in the context of MDPs over the last few decades, cf. [30]. Recently, there has also been an increased attention for dealing with relational representations and objects in RL, see e.g. [10, 13]. Many of these works have taken a practical perspective and have developed systems and experiments that operate in relational worlds. At the heart of these systems there is usually a function approximator (often a logical regression tree) that is able to assign values to sets of states and to sets of state–action pairs. So far, however, a theory that explains why this approach works seems to be lacking. The second contribution of the present paper is a first step in the direction of such a theory. The theory is based on a notion of abstract states and abstract policies represented by logical expressions. An abstract state represents a set of concrete states and an abstract policy is then a function from abstract states to actions. All ground states represented by the same abstract state are essentially assigned the same action. This is akin to what happens with *relational RL using logical regression trees* (RRL-RT) [10] [1], where each leaf of the regression tree represents an abstract state and where states classified in the same leaf obtain the same value or action. The convergence result presented is, to the best of our knowledge, the first presented in the context of relational reinforcement learning.

We proceed as follows. Mathematical preliminaries are reviewed in Section 2. The LOMDP framework is introduced in Section 3. Section 4 defines abstract policies, and Section 5 introduces the general framework of *generalized relational policy iteration* to learn abstract policies. In Section 6, we devise LTD(λ) for evaluation abstract policies. LTD(λ) is experimentally validated in Section 7. Before concluding, we discuss related work.

2 Preliminaries

In this section, we introduce some of the basic terminology of relational logic and MDPs, cf. [14] and [30].

Logic: A *first-order alphabet* Σ is a set of relation symbols $\mathtt{r}$ with arity $m \geq 0$, and a set of functor symbols $\mathtt{f}$ with arity $n \geq 0$. If $n = 0$ then $\mathtt{f}$ is called a constant, if $m = 0$ then p is called a proposition. An *atom* $\mathtt{r}(\mathtt{t_1}, \ldots, \mathtt{t_m})$ is a relation symbol $\mathtt{r}$ followed by a bracketed n-tuple of constants or variables. A conjunction is a set of atoms. A substitution $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$ $\{X/tex\}$, is an assignment of terms t_i to variables V_i. A conjunction A is said to be θ-subsumed by a conjunction B, denoted by $B \leq_\theta A$, if there exists a substitution θ such that $B\theta \subset A$. A term, atom or clause E is called *ground* when it contains no variables, i.e., $vars(E) = \emptyset$. A substitution θ is the *most general unifier* $\mathrm{mgu}(a, b)$ of atoms a and b iff. $a = b\theta$ and for each substitution θ' s.t. $a = b\theta'$, there exists a substitution γ such that $\theta' = \theta\gamma$. The *Herbrand base* of Σ, denoted as hb_Σ, is the set of all ground atoms constructed with the predicate and functor symbols in the alphabet Σ.

[1] RRL is sometimes used as short hand for Džeroski *et al.*'s approach. To distinguish it from relational reinforcement learning we will use RRL-RT as short hand.

Notation: Atoms are written in lower case a, sets of atoms in upper case A, and sets of sets of atoms in bold, upper case $\mathbf{A}$. To highlight that a (resp. A, $\mathbf{A}$) may not be ground, we will write $\mathbb{a}$ (resp. $\mathbb{A}$, $\mathbb{A}$).

Markov Decision Processes (MDP): A MDP is a tuple $\mathbf{M} = (S, A, \mathbf{T}, \lambda)$. Here, S is a set of system states. The agent has available a finite set of actions $A(z) \subseteq A$ for each state $z \in S$ which cause stochastic state transitions. For each $z, z' \in S$ and $a \in A(z)$ there is a transition T in $\mathbf{T}$, i.e., $z' \xleftarrow{p:r:a} z$. The transition denotes that with probability $P(z, a, z') := p$ action a causes a transition to state z' when executed in state z. For each $z \in S$ and $a \in A(z)$ it holds $\sum_{z' \in S} P(z, a, z') = 1$. The agent gains an expected next reward $R(z, a, z') := r$ for each transition. If the reward function R is probabilistic (mean value depends on the current state and action only) the MDP is called *nondeterministic*, otherwise *deterministic*. In this paper, we only consider MDPs with stationary transition probabilities and stationary, bounded rewards. A (stationary) deterministic policy $\pi : S \mapsto A$ is a set of expressions of the form $a \leftarrow z$ for each $z \in S$ where $a \in A(z)$. It denotes a particular course of actions to be adopted by an agent, with $\pi(z) := a$ being the action to be executed whenever the agent is in state z. We assume an infinite horizon and also that the agent accumulates the rewards associated with the states it enters. Future rewards are discounted by $0 \leq \lambda < 1$. The value of a policy π is the solution of the following system of linear functions $V_\pi(z) = \sum_{z' \xleftarrow{p:r:a} z \in \mathbf{T}} p \cdot [r + \lambda \cdot V_\pi(z')]$. A policy π is optimal if $V_\pi(z) \geq V_{\pi'}(z)$ for all $z \in S$ and policies π'. A (stationary) nondeterministic policy π maps a state to a distribution over actions. The value of π is then the expectation.

3 Logical Markov Decision Programs

The *logical component* of a MDP is essentially a propositional representation because the state and action symbols are flat. The key idea underlying *logical Markov decision programs* (LOMDPs) is to replace these flat symbols by abstract symbols.

Definition 1. *An abstract state is a conjunction $\mathbb{Z}$ of logical atoms, i.e., a logical query. In case of an empty conjuction, we write $\emptyset$.*

Abstract states represent sets of states. More formally, a state Z is a (finite) conjunction of ground facts over the alphabet Σ, i.e. a logical interpretation, a subset of the Herbrand base. In the blocks world, one possible state Z is $\mathtt{on(a,b)}$, $\mathtt{on(b,fl)}, \mathtt{bl(a)}, \mathtt{bl(b)}, \mathtt{cl(a)}, \mathtt{cl(fl)}$ where $\mathtt{on(a,b)}$ denotes that object $\mathtt{a}$ is on $\mathtt{b}$, $\mathtt{cl(a)}$ states that $\mathtt{a}$ is clear, $\mathtt{bl(a)}$ denotes that $\mathtt{a}$ is a block, and $\mathtt{fl}$ refers to the floor. An abstract state $\mathbb{Z}$ is e.g. $\mathtt{on(X,Y)}$, $\mathtt{bl(Y)}, \mathtt{bl(X)}$. It represents all states (over the given alphabet Σ) where a block $\mathtt{X}$ is on top of another block $\mathtt{Y}$. Formally speaking, an abstract state $\mathbb{Z}$ represents all states Z for which there exists a substitution θ such that $\mathbb{Z}\theta \subseteq Z$. Let $S(\mathbb{Z})$ denote this set of states. The substitution in the previous example is $\{\mathtt{X/a}, \mathtt{Y/b}\}$. By now we are able to define abstract transitions.

Definition 2. *An* abstract transition T *is an expression of the form* $\mathbb{H} \xleftarrow{p:r:\mathbb{a}} \mathbb{B}$ *where* $\mathrm{P(T)} := p \in [0,1]$, $\mathrm{R(T)} := r \in \mathbb{R}$, $\mathbb{a} := \mathrm{act(T)}$ *is an abstract action, and* $\mathrm{body(T)} := \mathbb{B}$ *and* $\mathrm{head(T)} := \mathbb{H}$ *are abstract states.*

We assume T to be range-restricted, i.e., $\mathrm{vars}(\mathbb{H}) \subseteq \mathrm{vars}(\mathbb{B})$, and $\mathrm{vars}(\mathbb{a}) \subseteq \mathrm{vars}(\mathbb{B})$, so that an abstract transition relies on the information encoded in the current state only. The semantics of an abstract transition[2] are:

> *If the agent is in a state Z, such that $\mathbb{B} \leq_\theta Z$, then it will go to the state $Z' := [Z \setminus \mathbb{B}\theta] \cup \mathbb{H}\theta$ with probability p when performing action $\mathbb{a}\theta$ receiving an expected next reward of r.*

For illustration purposes consider the following abstract transition, which moves block `X` from `Y` to the floor with probability 0.9:

$$\mathtt{on(X,fl),cl(X)cl(Y)} \xleftarrow{0.9:-1:\mathtt{mv\_fl(X)}} \mathtt{on(X,Y),cl(X)}$$

Applied to the state **Exp**, which is

$$\mathtt{on(a,b),on(b,fl),on(c,fl),cl(a),cl(c),bl(a),bl(b),bl(c),}$$

the abstract transition tells us that `mv_fl(a)` leads to

$$\mathtt{on(a,fl),on(b,fl),on(c,fl),cl(a),cl(b),cl(c),bl(a),bl(b),bl(c)}$$

with probability 0.9 gaining a reward of -1. One can see that this implements a kind of first-order variant of probabilistic STRIPS operators.

As LOMDPs typically consist of a set $\mathbb{T}$ of multiple abstract transitions there are two constraints to be imposed in order to obtain meaningful LOMDPs. First, let $\mathbf{B}$ be the set of all bodies of abstract state transitions in the LOMDP (modulo variable renaming). For $\mathbb{B} \in \mathbf{B}$, let $\mathbb{A}(\mathbb{B})$ denote the set of all abstract actions $\mathbb{a}$ such that $\mathbb{H} \xleftarrow{p:r:\mathbb{a}} \mathbb{B}$ is in the LOMDP. We require

$$\forall \mathbb{B} \in \mathbf{B}, \forall \mathbb{a} \in \mathbb{A}(\mathbb{B}) \sum_{\substack{\mathrm{T}\in\mathbb{T},\\ \mathrm{body(T)}=\mathbb{B},\\ \mathrm{act(T)}=\mathbb{a}}} \mathrm{P(T)} = 1.0. \tag{1}$$

This condition guarantees that all abstract successor states are specified when executing an abstract action in an abstract state and that their probabilities sum to 1. Secondly, we need a way to cope with contradicting transitions and rewards. Indeed, consider the transitions $\mathtt{e} \xleftarrow{1:-1:\mathtt{a}} \mathtt{d}$, $\mathtt{g} \xleftarrow{0.5:-2:\mathtt{a}} \mathtt{f}$ and $\mathtt{e} \xleftarrow{0.5:-2:\mathtt{a}} \mathtt{f}$, and state $Z = \{\mathtt{d},\mathtt{f}\}$. The problem with these transitions is that the first transition says that if we execute `a` in Z we will go with probability 1 to state $Z' = \{\mathtt{e},\mathtt{f}\}$ whereas the last two ones assign a probability of 0.5 to state $Z'' = \{\mathtt{d},\mathtt{g}\}$ and state $Z''' = \{\mathtt{d},\mathtt{e}\}$. To ensure that only one abstract state is firing (`d` or `f`), we assume a total order $\prec$ over all pairs of actions and rules bodies that appear in $\mathbb{T}$ and apply a conflict resolution similar to Prolog. We do a forward search

[2] We implicitly assume that an abstract action has some preconditions.

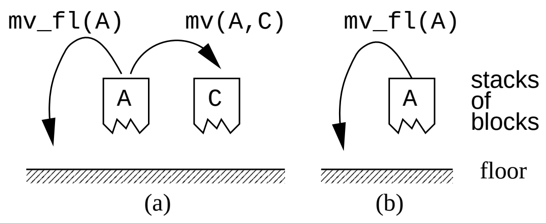


Fig. 1. The two underlying patterns of the blocks world for stacking. (a) There are at least two stacks of height > 0. (b) There is only one stack left (the goal state). The serrated cuts indicate that `A` (resp. `C`) can be on top of some other block or on the floor.

among the pairs stopping with the first matching one. For instance for $Z = \{\mathtt{d}, \mathtt{f}\}$ only the first abstract transition fires whereas for state `e, f` the last two abstract transitions fire.

By now we are able to formally define LOMDPs.

Definition 3. *A logical Markov decision process (LOMDP) is a tuple* $\mathbb{M} = (\Sigma, \mathbb{A}, \mathbb{T}, \lambda)$ *where* Σ *is a first-order alphabet,* $\mathbb{A}$ *is a set of abstract actions,* $\mathbb{T}$ *is a finite set of abstract state transitions based on actions in* $\mathbb{A}$*, and* $0 \leq \lambda < 1$ *is a discount factor, such that* (1) *holds.*

Before giving the semantics of LOMDPs, let us also illustrate LOMDPs on the *stack* example from the blocks world where the goal is to move all blocks on one single stack. For the sake of simplicity, we assume that all variables denote to different objects [3]:

$$\begin{array}{lrcl}
1: & \mathtt{absorb} & \xleftarrow{\mathtt{1.0:0.0:absorb}} & \mathtt{absorb}. \\
2: & \begin{array}{r}\mathtt{on(A,fl), cl(A),} \\ \mathtt{on(C,D), cl(C), cl(B)}\end{array} & \xleftarrow{\mathtt{0.9:-1:mv\_fl(A)}} & \begin{array}{l}\mathtt{on(A,B), cl(A),} \\ \mathtt{on(C,D), cl(C).}\end{array} \\
3: & \begin{array}{r}\mathtt{on(A,C), cl(A),} \\ \mathtt{on(C,D), cl(B)}\end{array} & \xleftarrow{\mathtt{0.9:-1:mv(A,C)}} & \begin{array}{l}\mathtt{on(A,B), cl(A),} \\ \mathtt{on(C,D), cl(C).}\end{array} \\
4: & \mathtt{absorb} & \xleftarrow{\mathtt{1.0:20:stop}} & \mathtt{on(A,B), cl(A).}
\end{array}$$

If the transition probabilities do not sum to 1.0 for an abstract action then there is an additional abstract transition for staying in the current abstract state. In order to understand the LOMDP *stack*, one has to understand the abstract states that govern the underlying patterns of the blocks world, cf. Figure 1. Two abstract states (the artificial `absorb` state excluded) together with the order in which they occur cover all possible state action patterns in the blocks world [4]. Furthermore, *stack* is an episodic task, i.e., it ends when reaching the goal state. In RL, episodic tasks are encoded using *absorbing states* which transition only to themselves and generate only zero rewards. Transition 1 encodes the absorbing

[3] This can be achieved by e.g. adding `diff(X, Y)` to denote that `X` and `Y` are different.
[4] We assume that we start in a legal blocks world. Therefore, `floor` will not be moved.

state. Transitions 2 and 3 cover the cases in which there are (at least) two stacks. Finally, transition 4 encodes the situation that there is only one stack, i.e. our goal state *stack*. Here, `on(A,B)`, `cl(A)`, `bl(B)` are only used to describe the preconditions of `mv(A,B)`: the floor cannot be moved. When performing action `mv(a,b)` in state *Exp* (see above) only abstract transition 3 and the omitted abstract transition for staying in the state are firing. Similar, we can easily encode the *unstack* goal. Note that we have not specified the number of blocks. The LOMDP represents all possible blocks worlds using only 6 abstract transitions, i.e. 12 probability and reward parameters, whereas the number of parameters of a propositional system explodes, e.g., for 10 blocks there are $58,941,091$ states.

Although, as the following theorem shows, the semantics of LOMDPs are also uniquely specified in the case of functors, we will focus in this paper on functor-free LOMDPs. In this case, the induced MDP $\mathbf{M}(\mathbb{M})$ is finite.

Theorem 1. *Every LOMDP* $\mathbb{M} = (\Sigma, \mathbb{A}, \mathbb{T}, \lambda)$ *specifies a discrete MDP* $\mathbf{M}(\mathbb{M}) = (S, A, \mathbf{T}, \lambda)$.

Proof sketch: Let $\text{hb}^s_\Sigma \subset \text{hb}_\Sigma$ be the set of all ground atoms built over abstract **s**tates predicates, and let $\text{hb}^a_\Sigma \subset \text{hb}_\Sigma$ be the set of all ground atoms built over abstract **a**ction names. Now, construct $\mathbf{M}(\mathbb{M})$ from $\mathbb{M}$ as follows. The countable state set S consists of all finite subsets of hb^s_Σ. The set of actions $\mathbf{A}(Z)$ for state $Z \in S$ is given by $\mathbf{A}(Z) = \{\mathbb{a}\theta | \mathbb{H} \xleftarrow{p:r:\mathbb{a}} \mathbb{B} \in \mathbb{T}$ minimal w.r.t. $\prec$, $\mathbb{B} \leq_\theta Z\}$. We have that $|\mathbf{A}(Z)| < \infty$ holds. The probability $P(Z, a, Z')$ of a transition in $\mathbf{T}$ from Z to another state Z' after performing an action a is the probability value p associated to the unique abstract transition matching Z, a, and Z' normalized by the number of transitions of the form $Z'' \xleftarrow{a} Z$ in $\mathbb{T}$. If there is no abstract transition connecting Z and Z', the probability is zero. The bounded rewards $R(Z, a, Z')$ are constructed in a similar way but are not normalized. □

From Theorem 1 and [27, Theorem 6.2.5] it follows that for every LOMDP, there exists an optimal policy (for ground states). Finally, LOMDPs generalize (finite) MDPs because every (finite) MDP is a propositional LOMDP in which all relation symbols have arity 0.

4 Abstract Policies

Theorem 1 states that every LOMDP $\mathbb{M}$ specifies a discrete MDP $\mathbf{M}(\mathbb{M})$. The existence of an optimal policy π for MDP $\mathbf{M}(\mathbb{M})$ is guaranteed. Of course, this policy is extensional or propositional in the sense that it specifies for each ground state separately which action to execute. Specifying such policies for LOMDPs with large state spaces is cumbersome and learning them will require much effort. Therefore, we introduce *abstract policies* $\boldsymbol{\pi}$, which intentionally specify the action to take for an abstract state (or sets of states).

Definition 4. *An abstract policy* $\boldsymbol{\pi}$ *over* Σ *is a finite set of decision rules of the form* $\mathbb{a} \leftarrow \mathbb{L}$*, where* $\mathbb{a}$ *is an abstract action and* $\mathbb{L}$ *is an abstract state* [5] *and* $\text{vars}(\mathbb{a}) \subseteq vars(\mathbb{L})$.

[5] We assume that $\mathbb{a}$ is applicable in $\mathbb{L}$.

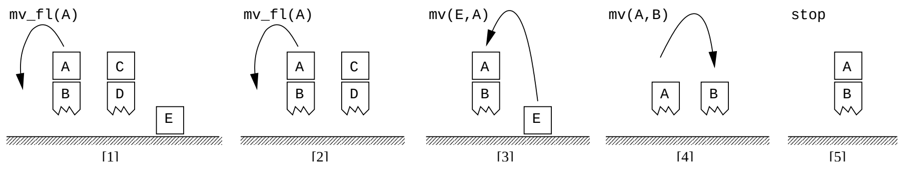


Fig. 2. The decision rules of the *unstack-stack* policy. In the figure, the decision rules are ordered from left to right, i.e., a rule fires only if no rule further to the left fires.

Usually, $\boldsymbol{\pi}$ consists of multiple decision rules. We apply the same conflict resolution technique as for abstract transitions. This means we assume a total order $\prec^{\boldsymbol{\pi}}$ among the decision rules in $\boldsymbol{\pi}$ and do a forward search stopping with the first matching decision rule such as in Prolog. Consider for instance the following *unstack-stack* abstract policy:

$$
\begin{array}{lrl}
\langle 1\rangle & \mathtt{mv\_fl(A)} \leftarrow & \mathtt{on(A,B),on(C,D),on(E,fl),cl(A),cl(C),cl(E).}\\
\langle 2\rangle & \mathtt{mv\_fl(A)} \leftarrow & \mathtt{on(A,B),on(C,D),cl(A),cl(C).}\\
\langle 3\rangle & \mathtt{mv(E,A)} \leftarrow & \mathtt{on(A,B),on(E,fl),cl(A),cl(E).}\\
\langle 4\rangle & \mathtt{mv(A,B)} \leftarrow & \mathtt{cl(A),cl(B).}\\
\langle 5\rangle & \mathtt{stop} \leftarrow & \mathtt{on(A,B),cl(A).}
\end{array}
$$

where we omitted the `absorb` state in front and statements that variables refer to different blocks. For instance in state **Exp** (see before), only decision rule $\langle 3\rangle$ would fire. The policy, which is graphically depicted in Figure 2, is interesting for several reasons.

1. It is close to the *unstack-stack* strategy which is well known in the planning community [29]. Basically, the strategy amounts to putting all blocks on the table and then building the goal state by stacking all blocks from the floor onto one single stack. No block is moved more than twice. The number of moves is at most twice the number of blocks.
2. It perfectly generalizes to all other blocks worlds, no matter how many blocks there are.
3. It cannot be learned in a propositional setting because here the optimal , propositional policy would encode the optimal number of moves.

The meaning of a decision rule $\mathbb{a} \leftarrow \mathbb{L}$ is that

> *if the agent is in a state Z such that $\mathbb{L} \leq_\theta Z$, then the agent performs action $\mathbb{a}\theta$ with uniform probability, i.e., $1/\sum_{\mathbb{a}\leftarrow\mathbb{L}\in\boldsymbol{\pi}} \mid \{\theta' \mid \ \mathbb{L} \leq_{\theta'} Z\} \mid$, denoted by $\boldsymbol{\pi}(Z)$.*

Let $\mathbb{L} = \{\mathbb{L}_1, \ldots, \mathbb{L}_m\}$ be the set of bodies in $\boldsymbol{\pi}$ (ordered w.r.t. $\prec^{\boldsymbol{\pi}}$). We call $\mathbb{L}$ the *abstraction level* of $\boldsymbol{\pi}$ and assume that it covers all possible states of the LOMDP. This together with the total order guarantees that $\mathbb{L}$ forms a partition of the states. The equivalence classes $[\mathbb{L}_1], \ldots, [\mathbb{L}_m]$ induced by $\mathbb{L}$ are inductively defined by $[\mathbb{L}_1] = S(\mathbb{L}_1)$, $[\mathbb{L}_i] = S(\mathbb{L}_i) \setminus \bigcup_{j=1}^{i-1}[\mathbb{L}_j]$, for $i \geq 2$. Because $\mathbb{L}$ generally does not coincide with $\mathbb{B}$, the following proposition holds.

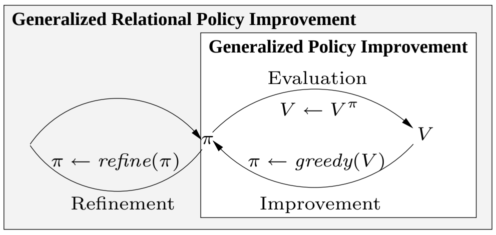

Generalized Relational Policy Improvement
Generalized Policy Improvement
Evaluation
$V \leftarrow V^{\pi}$
V
π
$\pi \leftarrow refine(\pi)$
$\pi \leftarrow greedy(V)$
Refinement
Improvement

Fig. 3. Generalized relational policy iteration which accounts for different abstraction levels. It is an upgrade of generalized policy iteration for traditional reinforcement learning as illustrated in [30]. *greedy* denotes the greedy policy computed from the current value function, see [30].

Proposition 1. *Any abstract policy* $\boldsymbol{\pi}$ *specifies a* nondeterministic *policy* π *at the level of ground states.*

5 Generalized Relational Policy Iteration

The crucial question is now, how to learn abstract policies? According to Sutton and Barto [30], almost all reinforcement learning systems follow the so called *generalized policy iteration* (GPI) scheme shown in Figure 3. It consists of two interacting processes: *policy evaluation* and *policy improvement*. Here, evaluating a policy refers to computing the value function of the current policy, and policy improvement refers to computing a new policy based on the current value function. Indeed, GPI cannot directly be applied to learn abstract policies. Different abstraction levels have to be explored. Thus, one needs an additional process which we call *policy refinement*. The resulting *generalized relational policy improvement* (GRPI) scheme is illustrated in Figure 3.

Generally speaking, policy refinement traverses the space of possible abstract policies. To do so, one can apply ILP techniques [24, 11]. For instance, we can refine the *unstack-stack* policy by adding a refined variant of decision rule $\langle 1 \rangle$,

$$\langle 0 \rangle \quad \texttt{mv\_fl(A)} \leftarrow \texttt{on(A,B)}, \texttt{on(C,D)}, \texttt{on(D,floor)}, \texttt{on(E,fl)}, \texttt{cl(A)}, \texttt{cl(C)}, \texttt{cl(E)}.$$

One can, however, do even better. When we have a model of the domain, it can be e.g. used to score different refinements (e.g. measuring the *influence* of a refinement of one state on the remaining state [25, 22]). If we do not have a model of the domain, we can employ the experience we already have, i.e., the states visited. This approach is followed by Džeroski *et al.* [10] within RRL-RT. To implement the policy refinement, Džeroski *et al.* employ logical regression trees for (abstract state-action) value function approximations. Starting with some initial abstraction level, RRL then integrates *evaluation*, *improvement*, and *refinement* in that it uses episodes to build a regression tree. Thus, RRL-RT can be seen as an instance of GRPI.

Empirically, RRL-RT has been proven to work very well on a wide range of domains such as blocks world and Digger. In the present paper, we will provide a first step in explaining why it works so well. More precisely, we will focus on the *relational evaluation problem* within GRPI approaches and prove convergence for an upgrade of TD(λ).

6 Logical TD($\boldsymbol{\lambda}$)

The *relational evaluation problem* considers how to compute the state-value function $V^{\boldsymbol{\pi}}$ for an arbitrary abstract policy $\boldsymbol{\pi}$. In this paper, we focus on model-free approaches. Model-free approaches do not know the reward and the transition functions in advance when computing the value of an abstract policies from experiences $\langle X_t, a_t, Y_t, r_t\rangle$. Furthermore, in contrast to traditional model-free approaches, maintaining values for all states of the underlying MDP $\mathbf{M}(\mathbb{M})$ is not feasible.

The basic idea to come up with a relational evaluation approach is to define the expected reward of $\mathbb{L} \in \mathbb{L}$ to be the average expected value for all the states in $[\mathbb{L}]$. This is a good model because if we examine each state in $[\mathbb{L}_i]$, we make contradictory observations of rewards and transition probabilities. The best model is the average of these observations given no prior knowledge of the model. Unfortunately, it has been experimentally shown that (already model-based) reinforcement learning with function approximation does not converge in general, see e.g. [5]. Fortunately, this does not hold for averagers. To prove convergence, we reduce the "abstract" evaluation problem to the evaluation problem for $\mathbf{M}(\mathbb{M}) = (S, A, \mathbf{T}, \lambda)$ with state aggregation (see e.g. [17, 28, 22]) with respect to $[\mathbb{L}_1], \ldots, [\mathbb{L}_m]$. For ease of explanation, we will focus on a *TD*(0) approach [6], see e.g. [30]. Results for general *TD*(λ) can be obtained by applying Tsitsiklis and Van Roy's results [32].

Algorithm 1 sketches logical *TD*(0). Given some experience following an abstract policy $\boldsymbol{\pi}$, LTD(0) updates its estimate $\widehat{V}$ of V. If a nonterminal state is visited, then it updates its estimate based on what happens after that visit. Instead of updating the estimate at the level of states, LTD(0) updates its estimate at the abstraction level $\mathbb{L}$ of $\boldsymbol{\pi}$.

To show convergence, it is sufficient to reduce *LTD*(0) to *TD*(0) with soft state aggregation [28]. The basic idea of soft state aggregation is to cluster the state space, i.e., to map the state space S into clusters $c_1, \ldots, c_k$. Each state s belongs to a cluster c_i with a certain probability $P(c_i|s)$. The value function then is computed at the level of clusters rather than states. Logical TD(0) is a special case of soft state aggregation. To see this recall that the abstraction level $\mathbb{L}$ partitions the state space S. Thus, one can view the abstract states in $\mathbb{L}$ as clusters where each state $Z \in S$ belongs to only one cluster $[\mathbb{L}_\mathbf{i}]$, i.e., $P([\mathbb{L}_\mathbf{i}] \mid Z) = 1$ if $Z \in S([\mathbb{L}])$; otherwise $P([\mathbb{L}_\mathbf{i}] \mid Z) = 0$. Furthermore, the state set S and the action set A of $\mathbf{M}(\mathbb{M})$ are finite, and the agent is following

[6] A similar analysis can be done for model-based approaches. Gordon [17] showed that value iteration with an averager as function approximator converges within a bounded distance from the optimal value function of the original MDP.

```
 1: Let $\boldsymbol{\pi}$ be an abstract policy with abstraction level $\mathbb{L}$
 2: Initialize $\widehat{V}_0(\mathbb{L})$ arbitrarily for each $\mathbb{L} \in \mathbb{L}$
 3: repeat (for each episode)
 4:   Initialize ground state $Z \in S_{\mathbf{M}(\mathbb{M})}$
 5:   repeat (for each step in episode)
 6:     Choose action $a$ in $Z$ based on $\boldsymbol{\pi}$ as described in Section 4, i.e.,
          i. select first decision rule $\mathbb{a} \leftarrow \mathbb{L}$ in $\boldsymbol{\pi}$ which matches according to $\prec^{\pi}$,
          ii. select $a$ uniformally among induced ground actions.
 7:     Take action $a$, observe $r$ and successor state $Z'$ as described in Section 3, i.e.,
          a. select with probability $p$ abstract transition $\mathbb{H} \xleftarrow{p:r:\mathbb{a}'} \mathbb{B}$ in $\mathbb{T}$
             where $(\mathbb{a}', \mathbb{B})$ matches $(a, Z)$ first according to $\prec$,
          b. select $Z'$ uniformally among all induced successor states.
 8:     Let $\mathbb{L}' \in \mathbb{L}$ be the abstract state first matching $Z'$ according to $\prec^{\pi}$
 9:     $\widehat{V}(\mathbb{L}) := \widehat{V}(\mathbb{L}) + \alpha \cdot (r + \lambda \cdot \widehat{V}(\mathbb{L}') - \widehat{V}(\mathbb{L}))$
10:     Set $Z := Z'$
11:   until $Z$ is terminal
12: until converged or some maximal number of episodes exceeded
```

Algorithm 1: Logical TD(0) where α is the learning rate and $\widehat{V}(\mathbb{L})$ is the approximation of $V(\mathbb{L})$.

a nondeterministic policy. Therefore, the assumptions of the following Theorem are fulfilled.

Theorem 2 (adopted from Singh et al. [28], Corollary 2**).** *TD*(0) *with soft state aggregation applied to* $\mathbf{M}(\mathbb{M})$ *while following a policy* π *converges with probability one to the solution of the following system of equations:* $\forall \mathbb{L}_i \in \mathbb{L}$: $V([\mathbb{L}_i]) =$

$$\sum_{Z \in S} P^{\pi}(Z \mid [\mathbb{L}_i]) \left[R^{\pi}(Z) + \lambda \sum_{\mathbb{L}_j \in \mathbb{L}} P^{\pi}(Z, [\mathbb{L}]_j) V([\mathbb{L}_j]) \right] \tag{2}$$

From this, it follows that *LTD*(0) converges, i.e. *LTD*(0) applied to a LOMDP $\mathbb{M}$ while following an abstract policy $\boldsymbol{\pi}$ at abstraction level $\mathbb{L}$ converges with probability one to the solutions of the system of equations (2).

Note that for arbitrary abstraction levels, while Theorem 2 shows that *LTD*(0) learning will find solutions, the error in the (ground) state space will not be zero in general. Equation (2) basically states that an abstract policy $\boldsymbol{\pi}$ induces a process $\mathbf{L}$ over $[\mathbb{L}_1], \ldots, [\mathbb{L}_m]$ whose transition probabilities and rewards for a state $[\mathbb{L}_i]$ are averages of the corresponding values of the covered ground states in $\mathbf{M}(\mathbb{M})$, see also [20]. Due to that, the process $\mathbf{L}$ appears to a learner to have a non-Markovian nature. Consider the following LOMDP

$$1\text{: } \mathsf{q} \xleftarrow{\mathsf{1.0:0.0:a}} \mathsf{p}, \mathsf{q} \qquad 2\text{: } \emptyset \xleftarrow{\mathsf{1.0:1.0:a}} \mathsf{p} \qquad \text{and} \qquad 3\text{: } \mathsf{p} \xleftarrow{\mathsf{1.0:0.0:a}} \emptyset.$$

and the abstraction level $\mathbb{L} = \{\mathsf{p}, \mathsf{q}, \emptyset\}$. Here, $\mathbf{L}$ will assign the same probabilities and rewards to the transitions from $[\mathsf{q}]$ to $[\mathsf{p}]$ and from $[\emptyset]$ to $[\mathsf{p}]$, namely the ones of transition 3. Consequently, the values for [q] and [$\emptyset$] are the same in **L** as the next state is the same, namely [p]. **M**($\mathbb{M}$), however, assigns different values to both as the following traces show: $\mathtt{q}\xrightarrow{1.0:0.0:a}\mathtt{p}, \mathtt{q}\xrightarrow{1.0:1.0:a}\mathtt{q}\ldots$ and $\emptyset\xrightarrow{1.0:0.0:a}\mathtt{p}\xrightarrow{1.0:0.0:a}\emptyset\ldots$ Nevertheless, *LTD*(0) converges at the level of $\mathbb{L}$ and can generalize well even for unseen ground states due to the abstraction.

To summarize, Theorem 2 shows that temporal-difference evaluation of an abstract policy converges. Different policies, however, will have different errors in the ground state space. In the context of GRPI (see Section 5), this suggests to use refinement operators to heuristically reduce the error in the ground state space. Applied on RRL-RT, this reads as follows.

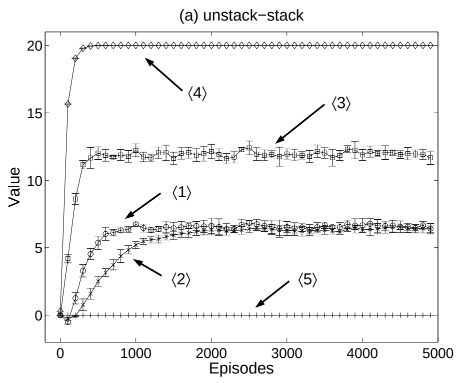
1. Because each logical regression tree induces a finite abstraction level, temporal-difference evaluation of a fixed regression tree converges.
2. Relational node/state splitting (based on the state sequences encountered so far) is used to heuristically reduce the error in the ground state space.

7 Experiments

Our task was to evaluate abstract policies within the blocks world. This task was motivated by the experiments in relational reinforcement learning (RRL) [10] and by the fact that the blocks world is the prototypical toy domain requiring relational representations. In contrast to the experiments reported by [10] on RRL, we exclusively use the standard predicates `on`, `cl`, and `bl`. [10] also needed to make use of several background knowledge predicates such as `above`, `height` of stacks as well as several directives to the first-order regression tree learner. Another difference to our approach is that RRL induces the relevant abstract states automatically using a regression tree learner. Our goal, however, was not to present an overall GRPI system but to put the following hypotheses to test:

H1 LTD(0) converges for finite abstraction levels.
H2 Using LTD(0), abstract policies can be compared.
H3 LTD(0) works for actions with multiple outcomes.
H4 Relational policy refinement is needed.
H5 Variance can be used as a heuristic state-splitting criterion.

We implemented *LTD*(0) using the Prolog system YAP-4.4.4. All experiments were run on a 3.1 GHz Linux machine and the discount factor λ was 0.9, and the learning rate α was set to 0.015. We randomly generated 100 blocks world states for 6 blocks, for 8 blocks, and for 10 blocks using the procedure described by [29]. This set of 300 states constituted the set *Start* of starting states in all experiments. Note that for 10 blocks a traditional MDP would have to represent $58,941,091$ states of which $3,628,800$ are goal states. The result of each experiment is an average of five runs of 5000 episodes where for each new episode we randomly selected one state from *Start* as starting state. For each run, the value function was initialized to zero. Note that in all experiments, the abstract policies and value functions apply no matter how many blocks there are.



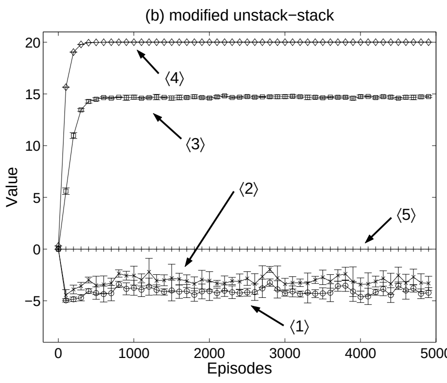


Fig. 4. Learning curves for TD(0) on the evaluation problem **(a)** for the *unstack-stack* policy and **(b)** for the *modified* unstack-stack policy. The predicted values are shown as a function of number of episodes. These data are averages over 5 reruns; the error bars show the standard deviations.

Experiment 1: Our task was to evaluate the *unstack-stack* abstract policy for the *stack* LOMDP, both introduced above. The results are summarized in Figure 4 **(a)** and clearly show that hypothesis **H1** holds. The learning curves show that the values of the abstract states converged, i.e., *LTD*(0) converged. Note that the value of abstract state ⟨5⟩ remained 0. The reason for this is that, by accident, no state with all blocks on the floor was in *Start*. Furthermore, the values converged to similar values in all runs. The values basically reflect the nature of the policy. It is better to have a single stack than multiple ones. The total running time for all 25000 episodes was 67.5 seconds measured using YAP's build-in `statistics(runtime, _)`.

Experiment 2: In reinforcement learning, *policy improvement* refers to computing a new policy based on the current value function. In a relational setting, the success of, e.g., computing the greedy policy given an abstract value function depends on the granularity of the value function. For instance, based on the last value function, it is not possible to distinguish between `mv_fl(A)` and `mv(A, B)` as actions in decision rule ⟨1⟩ because both would get the same expected values. To overcome this, one might refine the abstraction level (see experiment 5) or evaluate different policies at the same abstraction level.

In this experiments, we evaluated a *modified* "unstack-stack" policy in the same way as in the first experiment. It differed from the "unstack-stack" policy in that we do not perform `mv_fl(A)` but `move(E, A)` in the decision rule ⟨1⟩. The results are summarized in Figure 4 **(b)**. Interestingly, the values of abstract states ⟨1⟩ and ⟨2⟩ dropped from approximately 5 to approximately -4. This shows that *unstack-stack* is preferred over this policy. Furthermore, the total running time of all 25000 episodes increased to 97.3 seconds as the average length of an episode increased. This clearly shows that hypothesis **H2** holds.

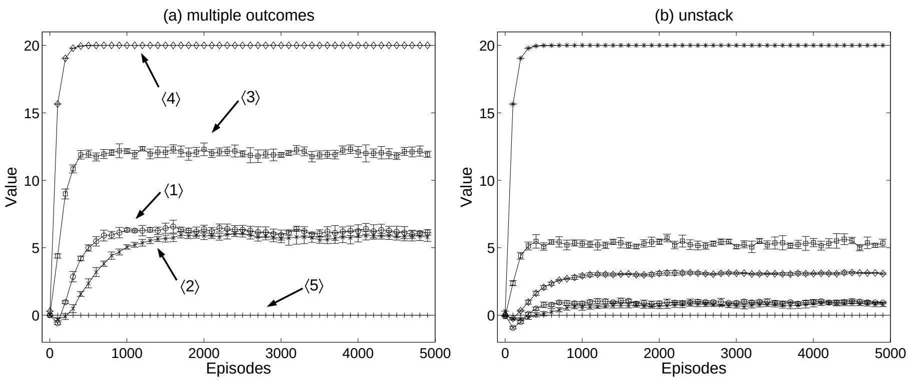


Fig. 5. Learning curves for TD(0) on the evaluation problem for the *unstack-stack* policy where **(a)** the actions of the underlying LOMDP have multiple outcomes and **(b)** the underlying LOMDP encoded the *unstack* problem. The predicted values are shown as a function of number of episodes. These data are averages over 5 reruns; the error bars show the standard deviations.

Experiment 3: We reran the first experiment where the underlying LOMDP now encoded that `mv_fl` and `mv` have multiple outcomes. For action $\mathtt{mv\_fl}(\mathtt{A},\mathtt{fl})$, block `A` may fall on `C`, and for action $\mathtt{mv}(\mathtt{A},\mathtt{C})$ the block `A` may fall on the floor `fl`. Again, LTD(0) converged as Figure 5 **(a)** shows. This shows that hypotheses **H3** holds.

Experiment 4: We reran the first experiment, i.e., we evaluated the *unstack-stack* policy, but now the underlying LOMDP encoded the *unstack* problem. Again, LTD(0) converged as Figure 5 **(b)** shows. The running time over all 25000 episodes, however, increased to 317.7 seconds as the underlying LOMDP was more complex. This shows that hypothesis **H1** holds.

Experiment 5: Finally, we investigated $\mathtt{on}(\mathtt{a},\mathtt{b})$ as goal. The underlying LOMDP was

$$
\begin{aligned}
\mathtt{absorb} &\xleftarrow{\mathtt{1.0:20:stop}} \mathtt{on}(\mathtt{a},\mathtt{b}).\\
\mathtt{on}(\mathtt{A},\mathtt{fl}),\mathtt{cl}(\mathtt{A}),\mathtt{on}(\mathtt{C},\mathtt{D}),\mathtt{cl}(\mathtt{C}),\mathtt{cl}(\mathtt{B}) &\xleftarrow{\mathtt{0.9:-1:mv\_fl(A)}} \mathtt{on}(\mathtt{A},\mathtt{B}),\mathtt{cl}(\mathtt{A}),\mathtt{on}(\mathtt{C},\mathtt{D}),\mathtt{cl}(\mathtt{C}).\\
\mathtt{on}(\mathtt{A},\mathtt{C}),\mathtt{cl}(\mathtt{A}),\mathtt{on}(\mathtt{C},\mathtt{D}),\mathtt{cl}(\mathtt{B}) &\xleftarrow{\mathtt{0.9:-1:mv(A,C)}} \mathtt{on}(\mathtt{A},\mathtt{B}),\mathtt{cl}(\mathtt{A}),\mathtt{on}(\mathtt{C},\mathtt{D}),\mathtt{cl}(\mathtt{C}).\\
\mathtt{on}(\mathtt{A},\mathtt{floor}),\mathtt{cl}(\mathtt{A}),\mathtt{cl}(\mathtt{B}) &\xleftarrow{\mathtt{1.0:-1:mv(A,floor)}} \mathtt{on}(\mathtt{A},\mathtt{B}),\mathtt{cl}(\mathtt{A}).
\end{aligned}
$$

assuming that all variables denote to different objects and `absorb` omitted. If the transition probabilities do not sum to 1.0 then there is an additional abstract transition for staying in the current abstract state. Following the same experimental setup as in the first experiment but using a step size $\alpha = 0.5$, we evaluated two different policies, namely $\langle a \rangle - \langle e \rangle, \langle l \rangle$ and $\langle a \rangle - \langle l \rangle$ where

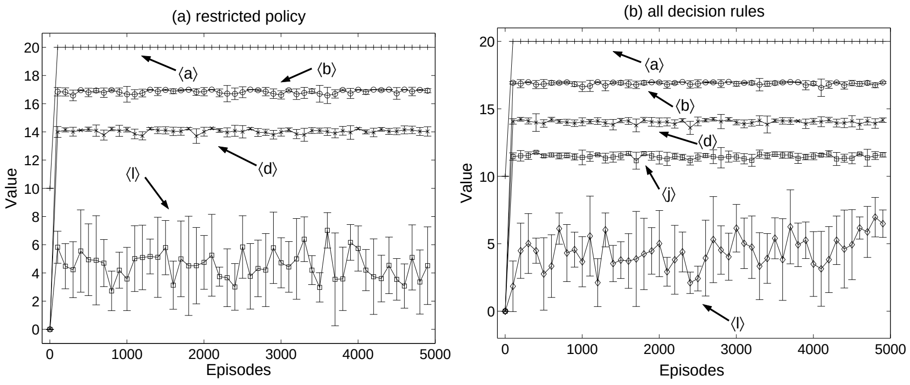


Fig. 6. Learning curves for TD(0) on the evaluation problem for the *on(a,b)* policy where the underlying LOMDP encoded the *on(a,b)* goal. The predicted values are shown as a function of number of episodes. These data are averages over 5 reruns; the error bars show the standard deviations; only state with a non zero value are shown (note that we used a finite set of starting states only). **(a)** Policy restricted to $\langle a\rangle - \langle e\rangle$ and $\langle l\rangle$. **(b)** All decision rules $\langle a\rangle - \langle l\rangle$.

$$
\begin{array}{lrl}
\langle a\rangle & \mathtt{stop} & \leftarrow \mathtt{on(a,b)}. \\
\langle b\rangle & \mathtt{mv(a,b)} & \leftarrow \mathtt{cl(a),cl(b),on(a,B)}. \\
\langle c\rangle & \mathtt{mv\_fl(b)} & \leftarrow \mathtt{cl(b),on(a,C),on(b,a)}. \\
\langle d\rangle & \mathtt{mv\_fl(A)} & \leftarrow \mathtt{cl(b),cl(A),on(a,C),on(A,a)}. \\
\langle e\rangle & \mathtt{mv\_fl(A)} & \leftarrow \mathtt{cl(a),cl(A),on(a,C),on(A,b)}. \\
\langle f\rangle & \mathtt{mv\_fl(A)} & \leftarrow \mathtt{cl(A),on(a,D),on(b,a),on(A,b)}. \\
\langle g\rangle & \mathtt{mv\_fl(b)} & \leftarrow \mathtt{cl(b),on(a,C),on(b,D),on(D,a)}. \\
\langle h\rangle & \mathtt{mv\_fl(a)} & \leftarrow \mathtt{cl(a),on(a,C),on(C,b)}. \\
\langle i\rangle & \mathtt{mv\_fl(a)} & \leftarrow \mathtt{cl(A),on(a,D),on(A,b)}. \\
\langle j\rangle & \mathtt{mv\_fl(A)} & \leftarrow \mathtt{cl(b),cl(A),on(a,C),on(A,D),on(D,a)}. \\
\langle k\rangle & \mathtt{mv\_fl(A)} & \leftarrow \mathtt{cl(a),cl(A),on(a,C),on(A,D),on(D,b)}. \\
\langle l\rangle & \mathtt{mv\_fl(A)} & \leftarrow \mathtt{cl(A),on(A,D)}.
\end{array}
$$

In both cases, LTD(0) converged as Figure 6 shows. The running time over all 25000 episodes was 4.85 seconds ($\langle a\rangle - \langle e\rangle, \langle l\rangle$) and 6.7 seconds ($\langle a\rangle - \langle l\rangle$). In both experiments, state $\langle l\rangle$ was exceptional. It obeyed a higher variance than the other states. The reason is that it acts as a kind of "container" state for all situations which are not covered by the preceding abstract states. In the refined policy, all added states showed low variances. Thus, we may iterate and refine $\langle l\rangle$ even more. The experiments show that hypotheses **H1**, **H4**, and **H5** hold and supports the variance-based state-splitting approach taken in RRL-RT [10].

8 Related Work

Within reinforcement learning (RL), there is currently a significant interest in using rich representation languages. Kaelbling et al. [19] and Finney et al. [13] investigated propositionalization methods in relational domains, namely *deictic representations* (DRs). DRs avoid enumerating the domain by using variables such as *the-block-on-the-floor*. Although DRs have led to impressive results [23, 34], Finney et al.'s results show that DR may degrade learning performance within relational domains. According to Finney et al. *relational reinforcement learning* such as RRL-RT [10] is one way to effectively learning in domains with objects and relations. The Q-function is approximated using a relational regression tree learner. Although the experimental results are interesting, RRL-RT did not explain, in theoretical terms, why it works. We provide some new insights on this.

Furthermore, the present work complements Kersting and De Raedt's [20] and Van Otterlo's [33] approaches. In [20], Kersting and De Raedt report on experiments with a relational upgrade of Q-learning. Van Otterlo devised a relational prioritized sweeping approach. Both works, however, do not consider convergence proofs.

The LOMDP formalism is related to Poole's independent choice logic [26]. Independent choice logic makes the dependencies among the probabilities explicit, but does not consider the learning problem.

From a more general point of view, our approach is closely related to *decision theoretic regression* (DTR) [3]. State spaces are characterized by a number of random variables and the domain is specified using logical representations of actions that capture the regularities in the effects of actions. Because 'existing DTR algorithms are all designed to work with *propositional* representations of MDPs', Boutilier et al. [4] proposed *first order DTR* which is a probabilistic extension of Reiter's *situation calculus*. The language is certainly more expressive than that of LOMDPs. However, it is also much more complex. Furthermore, Boutilier et al. assume that the model is given whereas in the present paper model-free learning methods have been applied.

Using a model-based approach, Yoon et al. [35] introduced a method for generalizing abstract policies from small to large relational MDPs employing description logics. The method has been extended [12] to an approximated policy iteration. Guestrin et al. [18] specify relationally factored MDPs based on probabilistic relational models [15] but not in a reinforcement learning setting. In contrast to LOMDPs, relations do not change over time. This assumption does not hold in many domains such as the blocks world.

The idea of solving large MDP by a reduction to an equivalent, smaller MDP is also discussed e.g. in [7, 16]. However there, no relational or first order representations have been investigated. Kim and Dean [22] investigate model-based RL based on non-homogenous partitions of propositional, factored MDPs. Furthermore, there has been great interest in abstraction on other levels than state spaces. Abstraction over time [31] or primitive actions [8, 1] are useful ways to abstract from specific sub-actions and time. This research is orthogonal and could be applied to LOMDPs in the future.

Finally, Baum [2] reports on solving blocks worlds with up to 10 blocks using RL related techniques. However, the language is domain-dependent and is not based on logic programming.

9 Conclusions

We have introduced a representation that integrates MDPs with logic programs. This framework allows one to compactly and declaratively represent complex (relationally factored) MDPs. Furthermore, it allows one to gain insights into relational reinforcement learning approaches. More precisely, we introduced abstract policies for LOMDPs and *generalized relational policy iteration* (GRPI) which is a general scheme for learning abstract policies. Because abstract policies basically introduce state aggregation, they can be evaluated using simple upgrades of (propositional) reinforcement learning methods such as $TD(\lambda)$-learning. Convergence has been proven and experimentally validated.

Such convergence guarantees are important because recent results on exact relational value iteration (RVI) [21] show that RVI can require an infinite abstraction level, i.e., an infinite logical regression tree in order to converge to exact values. Here, approximative approaches such as $LTD(\lambda)$ are useful, as they allow one to cut the regression tree at any level and to estimate the best values one can achieve at that abstraction level (cf. experiment 5). In other words, approximation is not only an interesting feature, but in some cases also a necessity for successful relational reinforcement learning.

The authors hope that the presented framework will be useful as a starting point for further theoretical developments in RRL. Of interest are: real-world applications; extending the language by e.g. negation and $\forall$-quantification; unordered abstract transitions; MDP-specific relational state splitting rules; applying Gordon's convergence results to RRL-RT with k-nearest-neighbour [9] (if possible); and to prove convergence of logical Q-learning [20]. The last point seems to be challenging as it also introduces action aggregation. Nevertheless, initial experiments [20] show that logical Q-learning performs well.

Acknowledgments The authors are deeply grateful Bob Givan for valuable discussions on the paper and its topic. The authors would also like to thank Martijn Van Otterlo for fruitful discussions on the topic. Also many thanks to the anonymous reviewers whose comments helped to improve the paper. The research was supported by the European Union IST programme, contract no. FP6-508861, *Application of Probabilistic Inductive Logic Programming II.*

References

1. D. Andre and S. Russell. Programmable reinforcement learning agents. In *Advances in Neural Information Processing Systems 13*, pages 1019–1025. MIT Press, 2001.

2. E. B. Baum. Towards a Model of Intelligence as an Economy of Agents. *Machine Learning*, 35(2):155–185, 1999.
3. C. Boutilier, T. Deam, and S. Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *JAIR*, 11:1–94, 1999.
4. C. Boutilier, R. Reiter, and B. Price. Symbolic Dynamic Programming for First-order MDPs. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 690–700, Seattle, USA, 2001.
5. J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: safely approximating the value function. In *Advances in Neural Information Processing Systems*, volume 7, 1995.
6. L. De Raedt and K. Kersting. Probabilistic Logic Learning. *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining*, 5(1):31–48, 2003.
7. R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, 89(1):219–283, 1997.
8. Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
9. K. Driessens and J. Ramon. Relational Instance Based Regression for Relational Reinforcement Learning. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 123–130, Washington DC, USA, 2003.
10. S. Džeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43(1/2):7–52, 2001.
11. S. Džeroski and N. Lavrač. *Relational Data Mining*. Springer-Verlag, 2001.
12. A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias. In *Proceedings of the Neural Information Processing Conference (NIPS)*, 2003.
13. S. Finney, N. H. Gardiol, L. P. Kaelbling, and T. Oates. The thing that we tried didn't work very well: Deictic representation in reinforcement learning. In *Proceedings of the Eighteenth International Conference on Uncertainty in Artificial Intelligence (UAI-02)*, 2002.
14. P. Flach. *Simply logical: intelligent reasoning by example*. John Wiley and Sons Ltd., 1994.
15. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI-99)*, pages 1300–1309, Stockholm, Sweden, 1999. Morgan Kaufmann.
16. R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147:163–224, 2003.
17. G. J. Gordon. Stable fitted reinforcement learning. In *Advances in Neural Information Processing*, pages 1052–1058. MIT Press, 1996.
18. C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing Plans to New Environments in Relational MDPs. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003.
19. L. P. Kaelbling, T. Oates, N. H. Gardiol, and S. Finney. Learning in worlds with objects. In *Working Notes of the AAAI Stanford Spring Symposium on Learning Grounded Representations*, 2001.
20. K. Kersting and L. De Raedt. Logical markov decision programs. In *Working Notes of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data (SRL-03)*, pages pp. 63–70, 2003.

21. K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes Relational. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-04)*, Banff, Alberta, Canada, July 4-8 2004. (to appear).
22. K.-E. Kim and T. Dean. Solving factored mdps using non-homogeneous partitions. *Artificial Intelligence*, 147:225–251, 2003.
23. A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden States*. PhD thesis, Department of Computer Science, University of Rochester, 1995.
24. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.
25. R. Munos and A. Moore. Influence and Variance of a Markov Chain : Application to Adaptive Discretization in Optimal Control. In *Proceedings of the IEEE Conference on Decision and Control*, 1999.
26. D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1–2):7–56, 1997.
27. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.
28. S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing 7*, pages 361–268. MIT Press, 1994.
29. J. Slaney and S. Thiébaux. Blocks World revisited. *Artificial Intelligence*, 125:119–153, 2001.
30. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.
31. R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.
32. J. N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions of Automatic Control*, 42:674–690, 1997.
33. M. Van Otterlo. Reinforcement Learning for Relational MDPs. In *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*, 2004.
34. S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45 – 83, 1991.
35. 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)*, 2002.