Can I Do That? Discovering Domain Axioms Using Declarative Programming and Relational Reinforcement Learning

Mohan Sridharan, Prashanth Devarakonda, and Rashmica Gupta

Department of Electrical and Computer Engineering, The University of Auckland, NZ
vdev818@auckland.ac.nz; m.sridharan@auckland.ac.nz;
rashmicy@gmail.com

Abstract. Robots deployed to assist humans in complex, dynamic domains need the ability to represent, reason with, and learn from, different descriptions of incomplete domain knowledge and uncertainty. This paper presents an architecture that integrates declarative programming and relational reinforcement learning to support cumulative and interactive discovery of previously unknown axioms governing domain dynamics. Specifically, Answer Set Prolog (ASP), a declarative programming paradigm, is used to represent and reason with incomplete commonsense domain knowledge. For any given goal, unexplained failure of plans created by inference in the ASP program is taken to indicate the existence of unknown domain axioms. The task of learning these axioms is formulated as a Reinforcement Learning problem, and decision-tree regression with a relational representation is used to generalize from specific axioms identified over time. The new axioms are added to the ASP-based representation for subsequent inference. We demonstrate and evaluate the capabilities of our architecture in two simulated domains: *Blocks World* and *Simple Mario*.

1 Introduction

Robots[1] are increasingly being used to assist humans in complex domains such as disaster rescue and health care. While it is difficult for robots to operate in such domains without considerable domain knowledge, human participants may not have the time and expertise to equip robots with comprehensive and accurate domain knowledge. Robots receive incomplete but useful commonsense knowledge about the domain, including *default* knowledge that holds in all but a few exceptional circumstances, e.g., “books are typically in the library, but cookbooks may be in the kitchen”. Robots also receive unreliable information by processing inputs from sensors such as cameras and microphones. Furthermore, the axioms governing the dynamics of the domain may be known partially and may change over time. To truly assist humans in such domains, robots need the ability to represent, reason with, and learn from, such different descriptions of knowledge and uncertainty at both the cognitive level and the sensorimotor level.

Towards addressing the challenges described above, we have developed architectures that have combined the non-monotonic logical reasoning capabilities of declarative programming with the uncertainty modeling capabilities of probabilistic graphical

[1] We use the terms “robot”, “agent” and “learner” interchangeably in this paper.

models. These architectures allow robots to represent and reason with logic-based and probabilistic representations of knowledge and uncertainty, for planning and diagnosis [1–4]. The architecture describe in this paper builds on our prior work [5, 6] to support incremental and interactive discovery of previously unknown domain axioms. The key features of the architecture are:

- An action language is used to describe the dynamics of the domain. This description and histories with initial state defaults are translated to an Answer Set Prolog (ASP) program that is solved for inference, planning and diagnostics.
- For any given goal, unexplained failures during plan execution are taken to indicate the existence of previously unknown domain axioms. The task of interactively discovering such unknown axioms is formulated as a reinforcement learning problem.
- Decision-tree regression and a relational representation are used to improve computational efficiency, and to generalize from specific axioms identified through reinforcement learning. These newly discovered axioms are added to the ASP program and used for subsequent reasoning.

These features are demonstrated in two simulated domains: *Blocks World* and the *Simple Mario* game. We show experimentally that the proposed architecture and the relational representation allow the robot to reliably discover previously unknown domain axioms more efficiently than traditional reinforcement learning.

The remainder of this paper is organized as follows. First, Section 2 summarizes prior work and describes some background material. Next, Section 3 describes the problem formulation and the proposed architecture. The experimental results are discussed in Section 4, followed by conclusions in Section 5.

2 Related Work

We motivate the proposed approach by reviewing some related work. We also provide some background information about ASP, reinforcement learning and relational representations, and their use on robots.

Probabilistic graphical models are used widely to formulate planning, sensing, navigation, and interaction, on robots [7, 8], but these formulations, by themselves, make it difficult to reason with commonsense knowledge. Research in planning has provided many algorithms for knowledge representation and reasoning on robots [9, 10], but these algorithms require considerable prior knowledge about the domain. Many of these algorithms are based on first-order logic, and do not support non-monotonic logical reasoning, default reasoning, and the ability to merge new, unreliable information with the current beliefs. Other logic-based formalisms address some of these limitations, e.g., Answer Set Prolog (ASP), a declarative language designed for representing and reasoning with commonsense knowledge [11], has been used by an international research community for cognitive robotics applications [12–14]. However, ASP does not support probabilistic models of uncertainty, and does not inherently support incremental and interactive learning from experience.

Combining logical and probabilistic reasoning is a fundamental research problem in robotics and AI. Architectures have been developed to support hierarchical representation of knowledge in first-order logic, and probabilistic processing of perceptual information [15, 16]. Existing approaches have combined deterministic and probabilistic algorithms for task and motion planning [17, 18], switched between probabilistic reasoning and first-order logic based to use semantic maps and commonsense knowledge in a probabilistic relational representation [19], and used a three-layered organization of knowledge to combine first-order logic and probabilistic reasoning for open world planning [20]. Other approaches for combining logical and probabilistic reasoning include Markov logic networks [21], Bayesian Logic [22], probabilistic first-order logic [23], first-order relational POMDPs [24], and probabilistic extensions to ASP [25, 26]. Many of these algorithms are based on first-order logic, and have the corresponding limitations, e.g., non-monotonic logical reasoning and reasoning with default knowledge are challenging. Other algorithms based on logic programming do not support all desired capabilities such as reasoning with large probabilistic components, reasoning with open worlds, and incremental and interactive learning of domain knowledge.

Many tasks that require the agent to learn from repeated interactions with their environment have been posed as Reinforcement Learning (RL) problems [27] and modeled as Markov Decision Processes (MDPs). It is challenging to design RL algorithms that scale to complex domains, and allow the transfer of knowledge between related tasks or domains. Relational reinforcement Learning (RRL) combines relational representations of states and actions with regression for Q-function generalization [28]. An RRL formulation enables the use of structural similarities, and the reuse of relevant experience in related regions of the state-action space [29]. Existing approaches, however, use RRL for planning, and generalization, e.g., with a decision tree [30], is limited to a single MDP corresponding to a specific planning task. Furthermore, these approaches do not fully support the ability to reason with commonsense knowledge.

We have designed architectures that combine the complementary strengths of declarative programming and probabilistic graphical models for planning and diagnostics in robotics [1–3]. In this paper, we abstract away the unreliability of perception, and combine declarative programming with RRL for incrementally and interactively discovering axioms, and generalizing across individual axiom instances. Unlike prior work that used inductive logic and ASP to monotonically learn causal rules [31], or integrated ASP with RL for discovering domain axioms [5], we use relational representation for generalization. Unlike existing work in RRL that primarily focuses on planning, our approach uses relational representations for discovering domain axioms, and generalizes across different MDPs, i.e., different decision making tasks in the domain.

3 Proposed Architecture

This section describes the proposed approach for incrementally and interactively discovering previously unknown axioms governing domain dynamics. The overall architecture is shown in Figure 1. For any given goal, ASP-based non-monotonic reasoning with a coarse-resolution domain description provides a sequence of abstract actions. Each such action is implemented as a sequence of concrete actions, using a partially observable Markov decision process (POMDP) to probabilistically model the relevant part of the fine-resolution description obtained by refining the coarse-resolution description. In this paper, we abstract away the uncertainty in perception for simplicity,

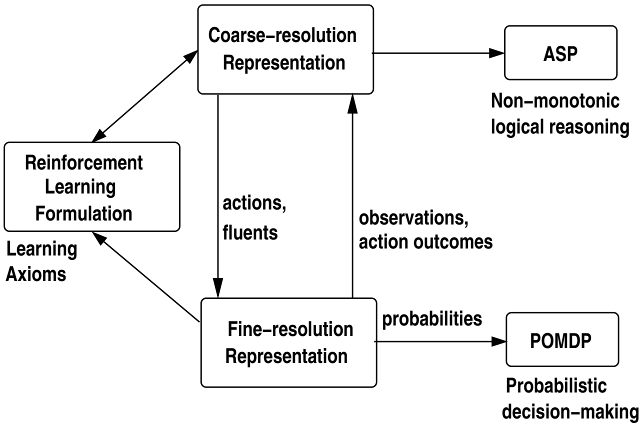

Coarse-resolution Representation
ASP
Non-monotonic logical reasoning
Reinforcement Learning Formulation
Learning Axioms
actions, fluents
observations, action outcomes
Fine-resolution Representation
probabilities
POMDP
Probabilistic decision-making

Fig. 1. The overall architecture integrates the complementary strengths of declarative programming, probabilistic graphical models, and reinforcement learning.

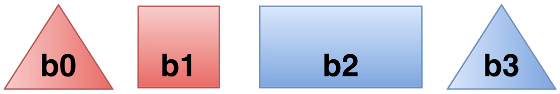
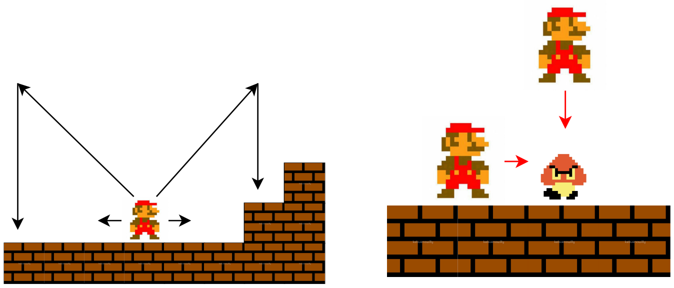


Fig. 2. Blocks world scenario with four blocks.

and thus do not discuss probabilistic planning. Instead, we represent the domain at a single resolution, use ASP-based reasoning with commonsense knowledge for planning and diagnosis, and focus on RRL-based interactive discovery of domain axioms. We illustrate the capabilities of this architecture using two simulated domains.

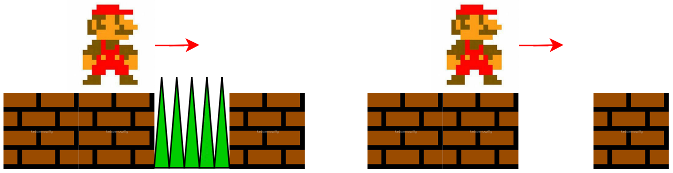
1. **Blocks World (BW):** a tabletop domain where the agent's objective is to stack blocks characterized by different colors, shapes, and sizes, in specific configurations on a table. Figure 2 illustrates a scenario with four blocks, which corresponds to ≈ 70 states under a standard RL/MDP formulation [28]. In this domain, the agent may not know, for instance, that a block should not be placed on a prism-shaped block, and any corresponding action should not be attempted.
2. **Simple Mario (SM):** a simplified version of the popular Mario game, where the agent (*mario*) has to navigate between specific locations while avoiding obstacles and hazards. The domain has ≈ 80 states and 4 parametrized actions in a standard MDP formulation. Figure 3(a) shows the safe actions (moving or jumping left or right), while the unsafe actions are shown in Figure 3(b) (colliding with a monster), Figure 4(a) (landing on spikes), and Figure 4(b) (landing on an empty space).



(a) Safe actions

(b) Unsafe action: collision

Fig. 3. Examples of safe actions (e.g., moving, jumping to unoccupied locations) and unsafe actions (e.g., collision with a monster) in the Simple Mario domain.



(a) Moving to spike block

(b) Moving to air block

Fig. 4. Examples of unsafe actions that cause episode termination (e.g., moving to a block made of *spike* or *air* material) in the Simple Mario domain.

3.1 Knowledge Representation

The transition diagram of our illustrative domain is described in an *action language* AL_d [11]. Action languages are formal models of parts of natural language used for describing transition diagrams. AL_d has a sorted signature containing three *sorts*: *statics*, *fluents* and *actions*. Statics are domain properties whose truth values cannot be changed by actions, while fluents are properties whose truth values can be changed by actions—actions refer to a set of elementary actions. Fluents are of two types: *inertial fluents* obey the laws of inertia and are changed directly by actions, whereas *defined fluents* do not obey the laws of inertia and cannot be changed directly by actions—they are changed based on other fluents. A domain property p or its negation $\neg p$ is a domain literal. AL_d allows three types of statements:

$$\begin{array}{ll} a \ \textbf{causes} \ l_{in} \ \textbf{if} \ p_0, \ldots, p_m & \text{(Causal law)} \\ l \ \textbf{if} \ p_0, \ldots, p_m & \text{(State constraint)} \\ \textbf{impossible} \ a_0, \ldots, a_k \ \textbf{if} \ p_0, \ldots, p_m & \text{(Executability condition)} \end{array}$$

where a is an action, l is a literal, l_{in} is an inertial fluent literal, and $p_0, \ldots, p_m$ are domain literals. The causal law states that action a causes inertial fluent literal l_{in} if the literals $p_0, \ldots, p_m$ hold true. A collection of statements of AL_d forms a system description.

The domain representation consists of a system description $\mathscr{D}$ and history $\mathscr{H}$. $\mathscr{D}$ consists of a sorted signature Σ and axioms used to describe the transition diagram τ. The sorted signature Σ is a tuple that defines the names of objects, functions, and predicates available for use in the domain. For instance, the sorts of the BW domain include elements such as *block*, *place*, *color*, *shape*, *size*, and *robot*, whereas the sorts of the SM domain include elements such as *location*, *block*, *material*, *size*, *direction* and *thing*. When some sorts are subsorts of other sorts, e.g., *agent* (i.e., *mario*) and *monster* may be subsorts of *thing*, they can be arranged hierarchically.

We describe the fluents and actions of the domain in terms of the sorts of their arguments. The BW domain's fluent $on(block, place)$, defined in terms of the sorts of the arguments, states that a specific block is at a specific place. This is an inertial fluent that obeys the laws of inertia. There are some statics for block attributes $has\_color(block, color)$, $has\_shape(block, shape)$ and $has\_size(block, size)$. The action $move(block, place)$ moves a block to a specific place (*table* or on top of another block). In the SM domain, the fluents are the location of *mario* and the *monster* (assuming there is only one monster)—we reason about the former and assume the latter is defined fluent known at all times, i.e., the inertial fluent is $loc(agent, block)$. Mario can move to the left or the right by one position, or jump to the left or right by up to three positions, which are represented as actions $move(mario, dir)$ and $jump(mario, dir, numpos)$ with direction (*left*, *right*) and number of positions $(1, 2, 3)$ as arguments. Statics describe block attributes, e.g., $has\_material(block, material)$, and location attributes, e.g., $right\_of(block, block)$.

For the BW domain, the dynamics are defined in terms of causal laws such as:

$$move(B, L) \ \ \textbf{causes} \ \ on(B, L)$$

state constraints such as:

$$\neg on(B, L_2) \ \ \textbf{if} \ \ on(B, L_1), \ L_1 \neq L_2$$

and executability conditions such as:

$$\textbf{impossible} \ \ move(B_2, L) \ \ \textbf{if} \ \ on(B_2, L), \ B_1 \neq B_2$$

The SM domain's dynamics are defined in a similar manner, using causal laws such as:

$$move(mario, right) \ \ \textbf{causes} \ \ loc(mario, B_2), \ right\_of(B_2, B_1), \ loc(mario, B_1)$$

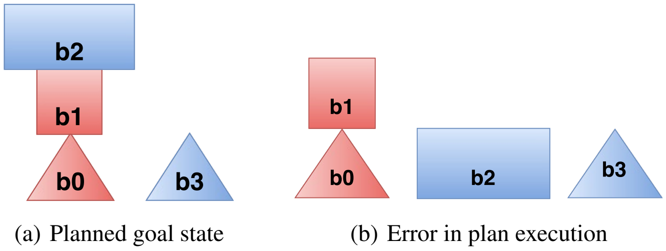
state constraints such as:

$$\neg loc(mario, B_2) \ \ \textbf{if} \ \ loc(mario, B_1), \ B_1 \neq B_2$$

and executability conditions such as:

$$\begin{aligned}\textbf{impossible} \ \ move(mario, right) \ \ \textbf{if} \ \ & loc(mario, B_1), \ right\_of(B_2, B_1), \\ & has\_material(B_2, spike)\end{aligned}$$

The recorded history of a dynamic domain is usually a record of (a) fluents observed to be true at a time step $obs(fluent, boolean, step)$, and (b) the occurrence of an action at a time step $hpd(action, step)$. Our prior architecture expanded on this view by allowing histories to contain (prioritized) defaults describing the values of fluents in their initial



(a) Planned goal state (b) Error in plan execution

Fig. 5. Illustrative example of (a) planned goal state; and (b) error in a plan execution step.

states [2, 32]. For instance, we can represent a default statement of the form "blocks are usually on the table or on another block that is on the table" and elegantly encode exceptions to such default knowledge.

The domain representation is translated into a program $\Pi(\mathscr{D},\mathscr{H})$ in CR-Prolog, a variant of Answer Set Prolog (ASP) that supports representation and reasoning with defaults and their direct and indirect exceptions [33]. This program is a collection of statements describing domain objects and relations between them, and incorporates consistency restoring (CR) rules in ASP [11][2]. ASP is based on stable model semantics and non-monotonic logics, and includes *default negation* and *epistemic disjunction*, e.g., unlike $\neg a$ that implies *a is believed to be false*, *not a* only implies that *a is not believed to be true*, and unlike "$p \lor \neg p$" in propositional logic, "p *or* $\neg p$" is not tautologous. ASP can represent recursive definitions, defaults, causal relations, and constructs that are difficult to express in classical logic formalisms. The ground literals in an *answer set* obtained by solving Π represent beliefs of an agent associated with Π; statements that hold in all such answer sets are program consequences. Algorithms for computing the entailment of CR-Prolog programs, and for planning and diagnostics, reduce these tasks to computing answer sets of CR-Prolog programs. Π consists of causal laws of $\mathscr{D}_H$, inertia axioms, closed world assumption for defined fluents, reality checks, and records of observations, actions, and defaults from $\mathscr{H}$. Every default is turned into an ASP rule and a CR rule that allows the robot to assume, under exceptional circumstances, that the default's conclusion is false, so as to restore program consistency—see [32, 2] for formal definitions of states, entailment, and models for consistent inference. Although not discussed here, the ASP program representing the current beliefs of the robot also supports other capabilities such as jointly explaining unexpected action outcomes and partial descriptions extracted from sensor inputs—see [1] for more details.

It is challenging to provide and encode all the knowledge corresponding to any given complex domain. For instance, some of the domain axioms may be unknown or may change over time, and the plans created using this incomplete knowledge may not succeed. Consider a scenario in the BM domain in which the goal is to stack three of four blocks placed on the table. Figure 5(a) shows a possible goal configuration that could be generated based on the available domain knowledge. The corresponding plan (starting with all four blocks on the table) has two steps: $move(b_1, b_0)$ followed by $move(b_2, b_1)$.

[2] We use the terms "ASP" and "CR-Prolog" interchangeably in this paper.

The robot expects to use this plan to stack the blocks as desired. Unknown to the robot, it is not possible to stack any block on top of a prism-shaped block in this domain, and execution of this plan results in failure that cannot be explained—specifically, action $move(b_1, b_0)$ does not result in the configuration shown in Figure 5(b). In this paper, we focus on discovering previously unknown executability conditions that can prevent such actions from being executed.

3.2 Relational RL for Discovering Axioms

Our approach for incremental and interactive discovery of domain axioms differs from previous work by us and other researchers. The proposed approach:

- Explores the existence of previously unknown axioms only when unexpected action outcomes cannot be explained by reasoning about exogenous actions [1].
- Uses RRL and decision tree regression for improving the computational efficiency of identifying candidate axioms, and for generalizing from specific axioms.
- Focuses on the discovery of unknown axioms by generalizing across multiple MDPs, instead of using RRL for planning, which limits generalization to a specific MDP.

Generalization and computational efficiency are key considerations for incremental and interactive learning. For instance, in the BW domain, discovery of the axiom "a red cube should not be placed on a blue prism" does not help when the tabletop has a *red prism* and *blue cube*, unless the agent realizes that the axiom it has discovered is a specific instance of the general axiom "no block should be placed on a prism-shaped block".

A sequence of steps is used to identify and generalize from candidate axioms. First, when a specific plan step fails, the corresponding state is considered the goal state in a RL problem, with the objective of finding state-action pairs that are most likely to lead to this error state. The RL problem uses an MDP formulation and the tuple $\langle S, A, T, R \rangle$, where:

- S: set of states.
- A:set of actions.
- $T : S \times A \times S' \rightarrow [0, 1]$ is the state transition function.
- $R : S \times A \times S' \rightarrow \Re$ is the reward function

Popular RL algorithms such as Q-learning or SARSA, which estimate $Q(s, a)$, the Q-values of state-action pairs, become computationally intractable as the state space increases in size and do not generalize to relationally equivalent states and actions. The second step uses a relational representation to support generalization. After a few episodes (i.e., iterations) of Q-learning (with eligibility traces) for a specific goal state, all state-action pairs that have been visited, along with their Q-values, are used to construct a binary (i.e., logical) decision tree (BDT). The path from the root node to any leaf node corresponds to one state-action pair, and individual nodes correspond to specific fluents—the value at the leaf node is the average of the values of all training samples that are grouped under that node. The BDT created after one iteration is used to compute the policy (based on a soft-max function [27]) in the subsequent episode. When the learning is terminated after convergence of the Q-values or after a specific number of episodes, the BDT relationally represents the experiences of the robot. Figure 6 illustrates a subset of a BDT constructed for the BW domain.

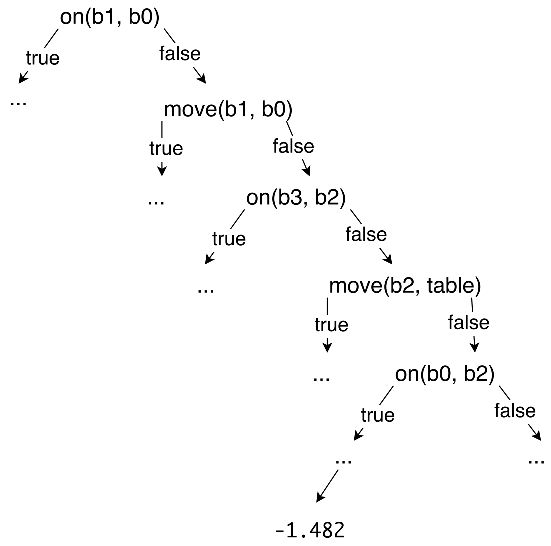


Fig. 6. Illustrative example of a subset of the binary decision tree for a specific scenario in the BW domain.

The method described above only considers generalization within a specific MDP. To truly identify general domain axioms, the third step of our approach simulates similar errors (to the one actually encountered due to plan step execution failure) and considers the corresponding MDPs as well. The Q-value of a state-action pair is now the the weighted average of the values across different MDPs. The weight assigned to a particular state-action pair in a specific MDP is inversely proportional to the shortest distance between the state and the goal state of the corresponding MDP based on the optimal policy for that MDP:

$$w_i = \frac{1/d_i}{\sum\limits_{j=0}^{N} 1/d_j}$$

where w_i is the weight of the state-action pair of MDP_i, d_i is the distance from the state to the goal state of MDP_i and N is the number of MDPs considered. Note that these similar MDPs are currently chosen randomly—future work will use the information encoded in the CR-Prolog program to direct attention to objects and attributes more relevant to the observed failure.

The fourth step identifies candidate executability constraints. The head of such an axiom has a specific action, and the body contains attributes that influence (or are influenced by) the action. We construct training samples by considering each such possible action and the corresponding attributes based on the BDT constructed as described above. These training samples are used to construct a decision tree whose root node corresponds to non-occurrence of the action, intermediate nodes correspond to attributes

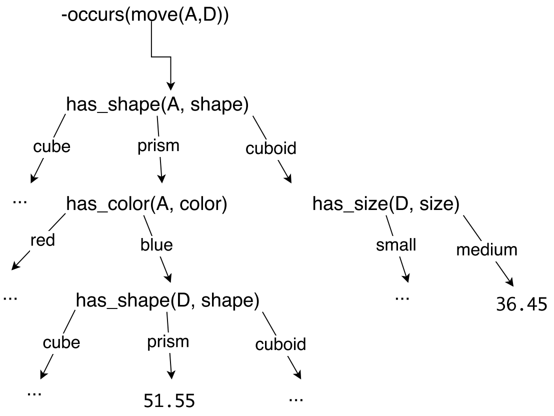


Fig. 7. Illustrative example of a decision tree constructed to represent candidate axioms related to a specific action in the BW domain.

of object involved in the action, and the leaf nodes are the average of the values of the training samples grouped under that node. Each path from the root node to a leaf is a candidate axiom with a corresponding value. Figure 7 illustrates a subset of such a tree for a specific action.

The final step considers all candidate axioms (for different actions), and uses the K-means algorithm to cluster these candidates based on their value. The axioms that fall within the cluster with the largest mean are added to the CR-Prolog program and used in the subsequent reasoning steps.

4 Experimental Setup and Results

The proposed architecture and algorithms were grounded and experimentally evaluated in the Blocks World domain and the Simple Mario domain. We describe the performance in illustrative execution scenarios in these domains. We also compare the rate of convergence of the proposed algorithm for discovering axioms, henceforth referred to as "Q-RRL", with that of traditional Q-learning.

4.1 Blocks World

As stated in Section 3, the robot's objective in the BW domain was to stack the blocks in a specified configuration. Consider the experimental trials in which the robot did not know that it was not possible to move any block on top of a prism-shaped block. We considered different scenarios with blocks of different shapes and colors (but the same size). For instance, one scenario had the following four blocks: $b0$ (Red Prism); $b1$ (Red Cube); $b2$ (Blue Cuboid); and $b3$ (Blue Prism). All blocks were initially on the table, i.e., $on(b0,table)$, $on(b1,table)$, $on(b2,table)$, and $on(b3,table)$. We provided the

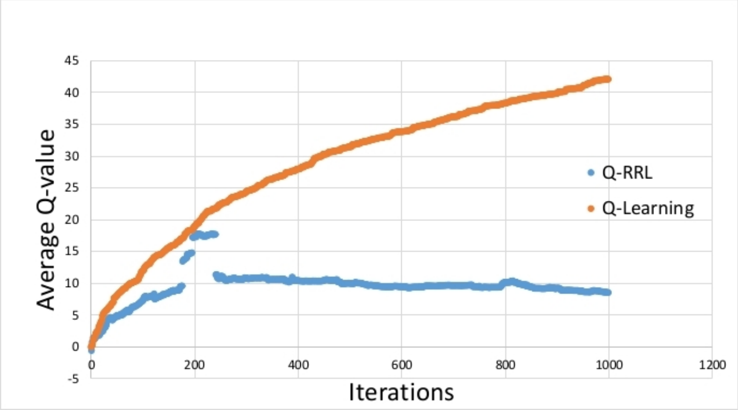


Fig. 8. Comparing the rate of convergence of Q-RRL with that of Q-learning in a specific scenario in the BW domain—Q-RRL converges much faster.

goal state description as $(on(b0,table)$, $on(b1,b0)$, $on(b2,b1)$, $on(b3,table))$, i.e., the objective was to stack three of the four blocks on the table. The plan obtained by solving the ASP program had actions $move(b1,b0)$ and $move(b2,b1)$. The action $move(b1,b0)$ fails, as expected. During Q-RRL and Q-Learning, the agent experiences a reward of $+100$ when it reaches goal state and a negative reward (i.e., cost) of -1.5 otherwise (i.e., for all other actions).

As stated earlier, RRL is triggered when executing plan steps to stack the blocks results in an unexpected and unexplained outcome. When such an error occurs, different related scenarios are simulated to generate the training samples for generalization. Figure 8 shows the rate of convergence of the average Q-value obtained using Q-RRL and Q-learning. The Q-RRL algorithm converges much faster, i.e., the optimal policy is computed in a much fewer number of iterations. Note that the rate of convergence is the performance measure in these experiments—it does not matter whether the actual average Q-values of one algorithm are higher (or lower) than the other algorithm. The following are some axioms identified during the various iterations:

$$\begin{aligned}
\neg occurs(move(A,D),I) \leftarrow & has\_shape(D,prism), has\_shape(A,cuboid),\\
& has\_color(D,blue)\\
\neg occurs(move(A,D),I) \leftarrow & has\_shape(D,prism), has\_shape(A,cube),\\
& has\_color(D,blue), has\_color(A,red)\\
\neg occurs(move(A,D),I) \leftarrow & has\_shape(D,prism), has\_shape(A,prism),\\
& has\_color(A,red), has\_color(D,blue)
\end{aligned}$$

$$\begin{aligned}
\neg occurs(move(A,D),I) \leftarrow & has\_shape(D,prism), has\_shape(A,cube),\\
& has\_color(D,red)
\end{aligned}$$

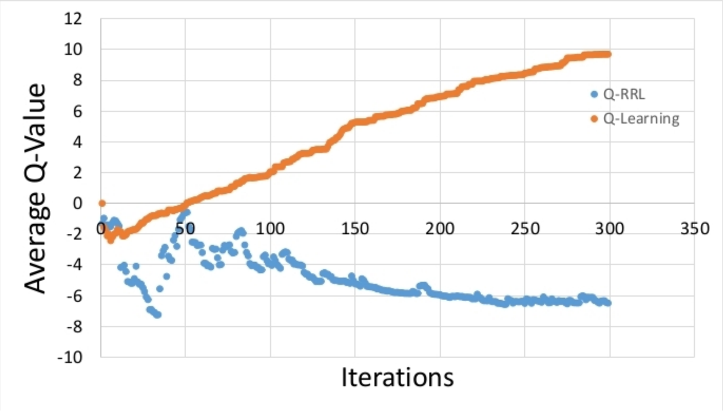


Fig. 9. Comparing the rate of convergence of Q-RRL with that of Q-learning in a specific scenario in the SM domain—Q-RRL converges much faster.

As the robot explores different scenarios, there are fewer and fewer errors because actions that are impossible are no longer included in the plans that are generated. Furthermore, the robot is able to incrementally generalize from the different specific axioms to finally add an axiom to the CR-Prolog program:

$$\begin{array}{ll} \textbf{impossible} \ \ move(A,D) \ \ \textbf{if} \ \ has\_shape(D,prism) & \text{(action language)} \\ \neg occurs(move(A,D),I) \leftarrow has\_shape(D,prism) & \text{(CR-Prolog statement)} \end{array}$$

In other experimental trials, the proposed architecture and algorithms resulted in the successful discovery of other such domain axioms.

4.2 Simple Mario

In the SM domain, the agent "mario" has to travel to a specific destination, "flag", from a starting position. As described at the beginning of Section 3, moving actions move mario one position away from the current position, while jumping actions attempt to move up to three positions away. If an obstacle prevents the agent from moving to a certain location, it lands on the closest (open) position available. Collision with any angry monster in the domain terminates the episode. Furthermore, blocks in the domain are made of different materials—*brick* blocks are harmless, whereas materials such as *spike* or *air* will result in the termination of the episode. The objective is to pick actions that will not result in episode termination—any such unexpected termination that cannot be explained triggers RRL for discovering axioms.

To evaluate the ability to efficiently discover generic domain axioms in the SM domain, scenarios related to the one causing an error (e.g., with different block attributes, and different locations for *mario* and monsters) are simulated (see Section 3.2). These simulated scenarios provide the training samples necessary for generalizing from the specific axioms discovered. For instance, in one scenario, three positions that would result in episode termination were used to trigger RRL. The first and second positions involve movement to blocks with *spike* material, while the third position involves collision with an angry monster. Figure 9 compares the rate of convergence of Q-RRL with that of Q-learning as a function of the number of episodes. Similar to the results obtained in the BW domain, Q-RRL converges significantly faster than Q-learning. Over a set of episodes, the following are some generalized axioms discovered:

$$\begin{aligned}
\neg occurs(move(mario, left), I) \leftarrow\ & loc(mario, B_1),\ \ left\_of(B_2, B_1),\\
& has\_material(B_2, spike)\\
\neg occurs(move(mario, right), I) \leftarrow\ & loc(mario, B_1),\ \ right\_of(B_2, B_1),\\
& has\_material(B_2, spike)\\
\neg occurs(jump(mario, left, 1), I) \leftarrow\ & loc(mario, B_1),\ left\_neighbor(B_2, B_1),\\
& has\_material(B_2, spike)\\
\neg occurs(jump(mario, right, 1), I) \leftarrow\ & loc(mario, B_1),\ right\_neighbor(B_2, B_1),\\
& has\_material(B_2, spike)
\end{aligned}$$

These axioms specify that *mario* cannot execute a move or jump action if this action will lead it to a block with *spike* material. Similar performance was observed in other scenarios that resulted in the failure of the corresponding plans, with the successful discovery of the corresponding domain axioms.

5 Conclusion

Robots collaborating with humans in complex domains frequently need to represent, reason with, and learn from different descriptions of incomplete domain knowledge and uncertainty. The architecture described in this paper combines the complementary strengths of declarative programming and relational reinforcement learning to discover previously unknown axioms governing domain dynamics. We illustrated the capabilities of this architecture in some simulated domains, with promising results. Future work will explore the ability to discover other kinds of axioms in more complex domains. We will also compare the performance of our algorithm with other popular relational reinforcement learning algorithms. Furthermore, we will conduct experimental trials on a mobile robot after including some probabilistic models of the uncertainty in perception on robots.

Acknowledgments

This work was supported in part by the US Office of Naval Research Science of Autonomy award N00014-13-1-0766. All opinions and conclusions in this paper are those of the authors alone.

References

1. Colaco, Z., Sridharan, M.: What Happened and Why? A Mixed Architecture for Planning and Explanation Generation in Robotics. In: Australasian Conference on Robotics and Automation (ACRA), Canberra, Australia (December 2-4, 2015)
2. Zhang, S., Sridharan, M., Gelfond, M., Wyatt, J.: Towards An Architecture for Knowledge Representation and Reasoning in Robotics. In: International Conference on Social Robotics (ICSR), Sydney, Australia (October 27-29, 2014) 400–410
3. Zhang, S., Sridharan, M., Wyatt, J.: Mixed Logical Inference and Probabilistic Planning for Robots in Unreliable Worlds. IEEE Transactions on Robotics **31**(3) (2015) 699–713
4. Sridharan, M.: Towards An Architecture for Knowledge Representation, Reasoning and Learning in Human-Robot Collaboration. In: AAAI Spring Symposium on Enabling Computing Research in Socially Intelligent Human-Robot Interaction, Stanford, USA (March 21-23, 2016)
5. Sridharan, M., Rainge, S.: Integrating Reinforcement Learning and Declarative Programming to Learn Causal Laws in Dynamic Domains. In: International Conference on Social Robotics (ICSR), Sydney, Australia (October 27-29, 2014)
6. Sridharan, M., Gelfond, M.: Using Knowledge Representation and Reasoning Tools in the Design of Robots. In: IJCAI Workshop on Knowledge-based Techniques for Problem Solving and Reasoning (KnowProS), New York, USA (July 10, 2016)
7. Bai, H., Hsu, D., Lee, W.S.: Integrated Perception and Planning in the Continuous Space: A POMDP Approach. International Journal of Robotics Research **33**(8) (2014)
8. Hoey, J., Poupart, P., Bertoldi, A., Craig, T., Boutilier, C., Mihailidis, A.: Automated Handwashing Assistance For Persons With Dementia Using Video and a Partially Observable Markov Decision Process. Computer Vision and Image Understanding **114**(5) (2010) 503–519
9. Galindo, C., Fernandez-Madrigal, J.A., Gonzalez, J., Saffioti, A.: Robot Task Planning using Semantic Maps. Robotics and Autonomous Systems **56**(11) (2008) 955–966
10. Varadarajan, K.M., Vincze, M.: Ontological Knowledge Management Framework for Grasping and Manipulation. In: IROS-2011 Workshop on Knowledge Representation for Autonomous Robots. (September 25, 2011)
11. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning and the Design of Intelligent Agents. Cambridge University Press (2014)
12. Balduccini, M., Regli, W.C., Nguyen, D.N.: An ASP-Based Architecture for Autonomous UAVs in Dynamic Environments: Progress Report. In: International Workshop on Non-Monotonic Reasoning (NMR), Vienna, Austria (July 17-19, 2014)
13. Chen, X., Xie, J., Ji, J., Sui, Z.: Toward Open Knowledge Enabling for Human-Robot Interaction. Journal of Human-Robot Interaction **1**(2) (2012) 100–117
14. Erdem, E., Patoglu, V.: Applications of Action Languages to Cognitive Robotics. In: Correct Reasoning. Springer-Verlag (2012)
15. Laird, J.E.: Extending the Soar Cognitive Architecture. In: International Conference on Artificial General Intelligence, Memphis, USA (March 1-3, 2008)
16. Talamadupula, K., Benton, J., Kambhampati, S., Schermerhorn, P., Scheutz, M.: Planning for Human-Robot Teaming in Open Worlds. ACM Transactions on Intelligent Systems and Technology **1**(2) (2010) 14:1–14:24
17. Kaelbling, L., Lozano-Perez, T.: Integrated Task and Motion Planning in Belief Space. International Journal of Robotics Research **32**(9-10) (2013) 1194–1227
18. Saribatur, Z., Erdem, E., Patoglu, V.: Cognitive Factories with Multiple Teams of Heterogeneous Robots: Hybrid Reasoning for Optimal Feasible Global Plans. In: International Conference on Intelligent Robots and Systems, Chicago, USA (2014) 2923–2930

19. Hanheide, M., Gretton, C., Dearden, R., Hawes, N., Wyatt, J., Pronobis, A., Aydemir, A., Gobelbecker, M., Zender, H.: Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Robot Behaviour. In: International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, Spain (July 16-22, 2011)
20. Hanheide, M., Gobelbecker, M., Horn, G., Pronobis, A., Sjoo, K., Jensfelt, P., Gretton, C., Dearden, R., Janicek, M., Zender, H., Kruijff, G.J., Hawes, N., Wyatt, J.: Robot Task Planning and Explanation in Open and Uncertain Worlds. Artificial Intelligence (2015)
21. Richardson, M., Domingos, P.: Markov Logic Networks. Machine Learning **62**(1-2) (February 2006) 107–136
22. Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D.L., Kolobov, A.: BLOG: Probabilistic Models with Unknown Objects. In: Statistical Relational Learning. MIT Press (2006)
23. Halpern, J.Y.: Reasoning about Uncertainty. MIT Press (2003)
24. Sanner, S., Kersting, K.: Symbolic Dynamic Programming for First-order POMDPs. In: AAAI Conference on Artificial Intelligence, Atlanta, USA (July 11-15, 2010) 1140–1146
25. Baral, C., Gelfond, M., Rushton, N.: Probabilistic Reasoning with Answer Sets. Theory and Practice of Logic Programming **9**(1) (January 2009) 57–144
26. Lee, J., Wang, Y.: A Probabilistic Extension of the Stable Model Semantics. In: AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning. (March 2015)
27. Sutton, R.L., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, USA (1998)
28. Dzeroski, S., Raedt, L.D., Driessens, K.: Relational Reinforcement Learning. Machine Learning **43** (2001) 7–52
29. Tadepalli, P., Givan, R., Driessens, K.: Relational Reinforcement Learning: An Overview. In: Relational Reinforcement Learning Workshop at the International Conference on Machine Learning. (2004)
30. Blockeel, H., Raedt, L.D.: Top-down Induction of First-order Logical Decision Trees. Artificial Intelligence **101**(1-2) (1998) 285–297
31. Otero, R.P.: Induction of the Effects of Actions by Monotonic Methods. In: International Conference on Inductive Logic Programming. (2003) 299–310
32. Sridharan, M., Gelfond, M., Zhang, S., Wyatt, J.: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics. Technical report, Unrefereed CoRR abstract: `http://arxiv.org/abs/1508.03891` (August 2015)
33. Balduccini, M., Gelfond, M.: Logic Programs with Consistency-Restoring Rules. In: AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning. (2003) 9–18