Advances in Cognitive Systems 5 (2017) 1-16 Submitted 4/2017; published 5/2017

An Architecture for Discovering Affordances, Causal Laws, and Executability Conditions

Mohan Sridharan M.SRIDHARAN@AUCKLAND.AC.NZ
Ben Meadows BMEA011@AUCKLANDUNI.AC.NZ
Electrical and Computer Engineering, The University of Auckland, New Zealand

Abstract

Robots assisting humans in complex domains often have to reason with different descriptions of incomplete domain knowledge. It is difficult to equip such robots with comprehensive knowledge about the domain and axioms governing the domain dynamics. This paper presents an architecture that enables interactive and cumulative discovery of axioms governing the action capabilities of the agent and the preconditions and effects of actions. Specifically, Answer Set Prolog is used to represent the incomplete domain knowledge, and to reason with this knowledge for planning and diagnostics. Unexpected state transitions during plan execution trigger reinforcement learning to interactively discover specific (i.e., ground) instances of previously unknown axioms. A decision tree induction approach and the relational representation encoded in the Answer Set Prolog program are used to generalize from these discovered axioms, providing generic axioms that revise the existing Answer Set Prolog program and are thus used for subsequent reasoning. The architecture's capabilities are illustrated and evaluated in a simulated domain that has an assistive robot moving particular objects to desired locations or people in an office.

1. Introduction

Consider a robot[1] assisting humans by finding and moving specific objects to specific locations in an office. To perform a variety of tasks in such a complex, dynamic environment, potentially under resource constraints, the robot will require a significant amount of domain knowledge, e.g., about domain objects and events. It will, however, be challenging for humans to equip the robot with comprehensive domain knowledge, or to possess the time and expertise to interpret raw sensor input and provide detailed feedback to the robot. Domain knowledge often includes commonsense knowledge, including default knowledge statements such as "books are usually in the library" that hold in all but a few exceptional circumstances, e.g., cookbooks may be in the kitchen. The information obtained from sensors and actuators is, on the other hand, typically associated with numerical representations, e.g., probabilistic representations of uncertainty in statements such as "I am 90% sure the robotics book is on the table". In addition, a robot is typically equipped with axioms governing domain dynamics, including knowledge of action capabilities that are often called affordances. We consider an *affordance* as a combination of the attributes of one or more objects and agents with

1. We use the terms "robot", "agent" and "learner" interchangeably, although an embodied agent is not essential for the learning task described in this paper.

reference to an action under consideration (Gibson, 1986). For instance, the affordance of a person climbing a stair may be described in terms of the stair's height with reference to the person's leg length (Warren, 1984). Despite these different types of knowledge possessed by a robot, one fundamental problem is that the knowledge is typically incomplete, potentially even inaccurate, and often needs to be revised in dynamic domains. For instance, if the floor of a room has just been polished, a robot's movement in this room will produce unexpected outcomes in the absence of an accurate description of the robot's movement on such surfaces. To truly assist humans in complex domains, robots thus need the ability to utilize and interactively revise the different types of knowledge.

The architecture described in this paper seeks to enable interactive and cumulative discovery of axioms governing domain dynamics. Such domain axioms typically describe the preconditions and expected outcomes of actions that can be executed in the domain, and the relationship between these actions and the properties of the object(s) and agent(s) in the domain. In this paper, we focus on the discovery of axioms governing actions performed by the robot, and assume that any knowledge of exogenous actions is limited to that encoded a priori. Our architecture can be used to discover axioms governing such exogenous actions, but we leave that as a direction for further research. We discuss the following key characteristics of the architecture:

- For planning and diagnostics, an action language-based description of the transition diagrams of the domain are translated to an Answer Set Prolog (ASP) program for non-monotonic logical reasoning, and to a partially observable Markov decision process (POMDP) for probabilistic reasoning.
- Unexpected state transitions observed during plan execution are considered to indicate the existence of previously unknown domain axioms. The discovery of these axioms, and the corresponding action capabilities, is formulated as a Reinforcement Learning problem that is informed by ASP inference.
- Decision tree-based regression with the relational representation encoded in the ASP program, and a sampling-based approach, are used to identify candidate axioms, and to generalize across these candidates. The generic axioms are then included in the ASP program for subsequent reasoning.

In this paper, we abstract away the uncertainty in perception and focus (instead) on establishing the ability of our architecture to (i) discover knowledge corresponding to action capabilities, and the preconditions and effects of actions; (ii) automatically determine the subset of the search space relevant to any given state transition; and (iii) iteratively improve the accuracy of the discovered axioms. We illustrate and evaluate these abilities in a simulated domain that has a robot helping humans by delivering desired objects to desired locations or people.

We first review related work in Section 2, and describe our architecture's components in Section 3. Experimental results are discussed in Section 4, followed by conclusions in Section 5.

2. Related Work

This section summarizes some related work in logic programming, reinforcement learning, and relational learning, in the context of robotics.

Probabilistic graphical models such as POMDP (Kaelbling et al., 1998) are used widely for reasoning tasks on robots, but these formulations, by themselves, make it difficult to reason with commonsense knowledge. Research in classical planning has provided many algorithms for knowledge representation and reasoning, but they often require complete knowledge about the domain and the agents' capabilities. Research in psychology has shown that humans can make accurate judgments about others' action capabilities using simple representations (Ramenzoni et al., 2010), and there have been many computational approaches for representing and reasoning about affordances (Griffith et al., 2012; Sarathy & Scheutz, 2016), but open questions remain regarding the suitable definition and representation of affordances (Horton et al., 2012).

In complex domains, agents often have to start with incomplete domain knowledge, and learn from repeated interactions with the environment. Different algorithms and architectures have been developed to support this capability. For instance, Shen and Simon (1989) used a first-order logic representation and the observed effects of actions to learn causal laws—axioms were working hypotheses to be revised through discriminant learning when predictions fail, but the approach only monitored the preconditions or effects of actions that had already been encoded. Gil (1994) incrementally refined operators encoded in first-order logic by making any unexpected observations the preconditions or effects of operators—this work focused only on augmenting existing knowledge and did not consider that the same action could lead to different outcomes under different contexts. Also, approaches based on first-order classical logic do not support desired capabilities such as non-monotonic logical reasoning and default reasoning, and these (and other such approaches) do not support generalization of axioms as described in this paper.

The logic programming community has developed many formalisms for non-monotonic logical reasoning, e.g., ASP, a declarative programming paradigm (Gelfond & Lifschitz, 1988), is used by a growing international research community (Erdem & Patoglu, 2012). Researchers have used inductive logic with ASP to monotonically learn causal rules (Otero, 2003), and used a maximum satisfiability framework with plan traces for refining incomplete domain models (Zhuo et al., 2013). Since these logic programming approaches, and those based on first-order logic, do not support probabilistic models of uncertainty, approaches such as Markov logic networks (Richardson & Domingos, 2006), Bayesian Logic (Milch et al., 2006), and Plog (Baral et al., 2009) have been developed to support both logical and probabilistic reasoning. However, algorithms based on first-order logic are not expressive enough—they often model default knowledge and uncertainty by associating logic statement with numbers that may not be meaningful—and approaches based on logic programming do not support reasoning with large probabilistic components, or interactive learning of knowledge (Baral et al., 2009). Such interactive learning has been posed as a Reinforcement Learning (RL) problem with an underlying Markov decision process (MDP) (Sutton & Barto, 1998). Approaches for efficient RL include sample-based planning algorithms (Walsh et al., 2010), and Relational Reinforcement Learning (RRL), which combines relational representations with regression for Q-function generalization (Tadepalli et al., 2004). However, existing interactive relational learning algorithms focus on planning, only generalize to a given planning task (Driessens & Ramon, 2003), or do not support the desired commonsense reasoning.

The architecture described in this paper supports interactive discovery of previously unknown knowledge governing domain dynamics. We build on and extend our recent architectures that

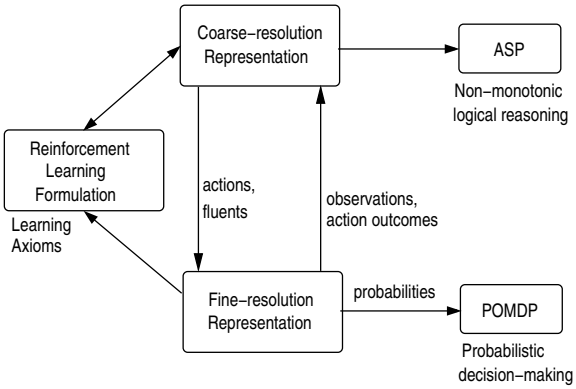

Figure 1. Architecture combines complementary strengths of declarative programming, probabilistic reasoning, and relational reinforcement learning.

(a) combined declarative programming and probabilistic reasoning for planning and diagnostics in robotics (Sridharan & Gelfond, 2016); and (b) integrated declarative programming with relational reinforcement learning for interactive discovery of axioms corresponding to previously unknown preconditions of actions (Sridharan & Meadows, 2016; Sridharan et al., 2017b).

3. Proposed Architecture

Figure 1 shows a block diagram of the overall architecture. For any given goal, ASP-based non-monotonic logical reasoning with a coarse-resolution domain description provides a sequence of abstract actions. Each abstract action is implemented as a sequence of concrete actions, using a POMDP to reason probabilistically with the relevant part of the corresponding fine-resolution system description—see (Sridharan et al., 2017a) for more details about reasoning with tightly-coupled transition diagrams at these two resolutions. In this paper, we focus on the new component of the architecture for interactively discovering domain axioms. We thus abstract away the uncertainty in perception and do not discuss probabilistic planning, use ASP-based reasoning for planning and diagnostics, and use relational reinforcement learning for axiom discovery. We illustrate the reasoning and axiom discovery capabilities using the following simulated domain.

Example 1. *[Robot Assistant (RA) Domain]*
Consider a robot that has to deliver objects to particular rooms or people. Attributes include:

- Different sorts (classes) such as $entity$, $person$, $robot$, $object$, $book$, $desk$ etc.
- Static attributes such as a human's $role$, which can be $\{engineer, manager, sales\}$; the robot's $armtype$, which can be $\{electromagnetic, pneumatic\}$; an object's $surface$, which can be $\{hard, brittle\}$; and an object's $weight$, which can be $\{light, heavy\}$.
- Fluents such as location of humans and the robot, which can be one of $office$, $kitchen$, $library$ and $workshop$; $status$ of an $object$, which can be $\{damaged, intact\}$; whether an object is being held by the robot; and whether an $object$ has been labeled.

As a partial illustration, consider a scenario with two rooms, three humans, one robot, three movable objects and five immovable objects—it has $6,946,816$ physical (object) configurations in a standard RL/MDP formulation and $55,296$ static attribute combinations that can be explored during the axiom discovery phase. In this domain, the robot may not know, for instance, that:

- A brittle object is damaged when it is put down.
- Delivering an unlabeled object to a sales person causes it to be labeled.
- A damaged object cannot be delivered, except to an engineer.
- An object with a brittle surface cannot be labeled.
- A heavy object cannot be picked up by a robot with an electromagnetic arm.
- A damaged object cannot be labeled by a robot with a pneumatic arm.

These statements correspond to different types of knowledge encoded as causal laws, affordances etc, as described later. The objective is to construct and include suitable axioms in the ASP program.

3.1 Knowledge Representation

In our architecture, the transition diagrams of any given domain are described using an *action language* AL_d (Gelfond & Inclezan, 2013). Action languages are formal models of parts of natural language used for describing transition diagrams. AL_d has a sorted signature containing three *sorts*, namely *statics*, *fluents* and *actions*. Statics are domain properties whose truth values cannot be changed by actions, whereas fluents are domain properties whose truth values can be changed by actions. Fluents are of two types, *basic* and *defined*. Basic 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. Actions are defined as a set of elementary operations. A domain property p or its negation $\neg p$ is a domain *literal*. Three types of statements are allowed:

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

where a is an action, l is a literal, l_b is a basic literal, and $p_0, \ldots, p_m$ are domain literals.

Domain Description The domain is represented by system description $\mathscr{D}$, a collection of statement of AL_d, and history $\mathscr{H}$. $\mathscr{D}$ has a sorted signature Σ and axioms that describe the transition diagram τ. The basic sorts in Σ for the RA domain include *place*, *robot*, *entity*, *person*, $object$, *role*, *weight*, $surface$, *cup* etc. Sorts that are subsorts of other sorts, e.g., *cup* and *book* are subsorts of $object$, and *person* and *robot* are subsorts of *entity*, are arranged hierarchically. Furthermore, the signature includes specific instances of sorts, e.g., we have robot rob_1, places $\{office, workshop, kitchen, library\}$, and roles $\{engineer, programmer, manager\}$.

Domain attributes and actions are described in terms of their arguments' sorts. The RA domain's Σ includes fluents such as $loc(entity, place)$, $obj\_status(object, status)$, and $in\_hand(entity, object)$; statics such as $obj\_surface(object, surface)$ and $obj\_weight(object, weight)$; and actions such as $move(robot, place)$, $serve(robot, object, person)$, and $affix\_label(robot, object)$. Σ also includes the sort $step$ for temporal reasoning, and the relation $holds(fluent, step)$ to state that a particular fluent holds true at a particular timestep. System description $\mathscr{D}$ includes axioms such as:

$$move(rob_1, Pl) \textbf{ causes } loc(rob_1, Pl)$$
$$\neg in\_hand(E, O2) \textbf{ if } in\_hand(E, O1),\ O1 \neq O2$$
$$\textbf{impossible } pickup(rob_1, O) \textbf{ if } loc(rob_1, L1), loc(O, L2),\ L1 \neq L2$$

The recorded history $\mathscr{H}$ of a dynamic domain is usually a record of fluents observed to be true or false at a time step, i.e., $obs(fluent, boolean, step)$, and the occurrence of an action at a time step, i.e., $hpd(action, step)$. Our model of history also allows us to represent (prioritized) defaults describing the values of fluents in their initial states. For instance, we can encode a statement such as "books are usually in the library", and encode exceptions, e.g., "cookbooks are in the kitchen".

Affordance Representation Positive affordances describe permissible uses of objects in actions, whereas negative affordances describe unsuitable combinations of objects, agents, and actions. In this paper, we introduce the following generic definition of forbidding (i.e., negative) affordances:

$$aff\_forbids(ID, A) \textbf{ if } not\ fails(ID, A), forbidding\_aff(ID, A)$$
$$\neg occurs(A, I) \textbf{ if } aff\_forbids(ID, A)$$

where the "not" in the first statement represents default negation (on which more details later), and the second statement implies that action A cannot occur if it is not afforded, which depends on whether suitable conditions have been defined to arrive at this conclusion. Any action can have one or more such relations defined with unique IDs. For instance, if we know that a robot with an electromagnetic arm cannot pick up a heavy object, the following statements may be included in $\mathscr{D}$:

$$forbidding\_aff(id1, pickup(R, O))$$
$$fails(id1, pickup(R, O)) \textbf{ if } not\ obj\_weight(O, heavy)$$
$$fails(id1, pickup(R, O)) \textbf{ if } not\ arm\_type(R, electromagnetic)$$

where the *pickup* action is not afforded if the object is heavy and the robot's arm is electromagnetic.

The representation of knowledge in our architecture brings up some subtle issues. First, for any given action, the axioms for both executability conditions and forbidding affordances imply that (when the body of these axioms are true) the desired outcomes will not be observed because not all of the action's preconditions are satisfied—the action should then not be included in a plan. However, there are key differences in the type of knowledge encoded by executability conditions and affordances, and how this knowledge is represented. Affordance relations, once discovered, can also encode additional outcomes that were not previously known. Also, for any particular action, each affordance is defined in terms of the attributes of the objects operated by an agent, or of an agent

and an object involved in this action. An executability condition does not have these requirements, e.g., when discovered and in use, the plans computed for any given goal are a subset of the plans obtained in the absence of this condition. Second, the representation of affordances as relations between domain properties and actions, similar to the representation of actions, is distributed, e.g., we can define multiple affordance relations for any action. The advantages of this representation, e.g., information reuse and ease of plan explanation, are discussed in Section 4.1.

ASP-based Inference The domain representation is translated into program $\Pi(\mathscr{D},\mathscr{H})$ in CR-Prolog, a variant of ASP that allows us to represent and reason with defaults and their exceptions by introducing consistency-restoring (CR) rules (Balduccini & Gelfond, 2003). ASP is based on stable model semantics and non-monotonic logics, and supports *default negation* and *epistemic disjunction*, e.g., unlike "$\neg a$" that states *a is believed to be false*, "*not a*" only implies *a is not believed to be true*, and unlike "$p \vee \neg p$" in propositional logic, "*p or* $\neg p$" is not tautologous (Gelfond & Kahl, 2014). 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. Planning and diagnostics are reduced to computing answer sets of CR-Prolog programs. In our case, the CR-Prolog program Π consists of causal laws of $\mathscr{D}$, inertia axioms, closed world assumption for defined fluents, reality checks, and observations, actions, and defaults in $\mathscr{H}$. Each default is turned into an ASP rule and a CR rule that allows the robot to assume that the default's conclusion is false, under exceptional circumstances, to restore program consistency.

With incomplete knowledge, the execution of the plan steps may have unexpected transitions. In the RA domain, a robot moving a brittle cup to the kitchen, without knowing that putting the cup down is going to damage it, will find that trying to affix a label to this cup fails. In this paper, *we focus on discovering such axioms corresponding to causal laws, executability conditions, and forbidding affordances, which will improve the quality of plans computed.*

3.2 Axiom Discovery

This section describes the steps of the axiom discovery process. We begin with the formulation of axiom discovery as a reinforcement learning (RL) problem, followed by the decision-tree regression approach to construct a relational representation of the experiences obtained during RL. We then describe the construction of candidate axioms from the tree, and the validation of the candidate axioms to produce generic axioms to be included in the CR-Prolog program.

RL and Tree Induction When executing an action produces an unexpected transition, i.e., it does not produce the expected observations and/or produces new unexpected observations, the state description described by the action's effects becomes the goal state in a relational reinforcement learning (RRL) problem—the objective is to find state-action pairs that are likely to lead to analogous "error" states. First consider the standard RL formulation and the underlying Markov decision process (MDP) defined by the tuple $\langle S, A, T_f, R_f \rangle$:

- S is the set of states;
- A is the set of actions;

- $T_f : S \times A \times S' \rightarrow [0,1]$ is the state transition function;
- $R_f : S \times A \times S' \rightarrow \Re$ is the reward function.

In the RL formulation, functions T_f and R_f are unknown (to the agent), and each element in S grounds the domain attributes, and whether the last action to be executed had the expected outcome(s). Such an RL-based formulation mimics the interactive acquisition of experiences on a robot as it performs the assigned tasks, and provides a principled approach to assign credit to current or previous state-action combinations for observed transitions. The definition of the reward functions used in the approach for discovering axioms is different for the different types of knowledge. High immediate reward is provided:

- When an action's expected effects are not observed, for executability conditions.
- When effects in addition to those expected for an action are observed, for causal laws.
- When expected and unexpected action effects are observed, for forbidding affordances.

One key benefit of ASP-based reasoning is that we can define and automatically compute the states and actions relevant to a given transition, eliminating parts of the search space irrelevant to the discovery of the desired knowledge. This is equivalent to automatically constructing the system description $\mathscr{D}(T)$, which is the part of $\mathscr{D}$ relevant to the desired transition T. To do so, we first define the object constants relevant to the unexplained failure—this is a revised version of the definition in (Sridharan & Gelfond, 2016; Sridharan et al., 2017a).

Definition 1. *[Relevant object constants]*
Let a_{tg} be the *target action* that when executed in state σ_1 did not result in the expected transition $T = \langle \sigma_1, a_{tg}, \sigma_2 \rangle$. Let $relCon(T)$ be the set of object constants of signature Σ of $\mathscr{D}$ identified using the following rules:

1. Object constants from a_{tg} are in $relCon(T)$;
2. If $f(x_1,\ldots,x_n,y)$ is a literal formed of a domain property, and the literal belongs to σ_1 or σ_2, but not both, then $x_1,\ldots,x_n,y$ are in $relCon(T)$;
3. If body B of an executability condition of a_{tg} contains an occurrence of $f(x_1,\ldots,x_n,Y)$, a term whose domain is ground, and $f(x_1,\ldots,x_n,y) \in \sigma_1$ then $x_1,\ldots,x_n,y$ are in $relCon(T)$.

Constants from $relCon(T)$ are said to be *relevant* to transition T. For instance, if the target action in RA domain is $a_{tg} = serve(rob_1, cup_1, person_1)$, with $loc(rob_1, office)$, $loc(cup_1, office)$ and $loc(person_1, office)$ in state σ_1, the relevant object constants will include rob_1 of sort *robot*, cup_1 and $person_1$ of sort *thing*, and $office$ of sort *place*.

We then define the relevant system description $\mathscr{D}(T)$ as follows.

Definition 2. *[Relevant system description]*
The system description $\mathscr{D}(T)$ relevant to the transition $T = \langle \sigma_1, a_{tg}, \sigma_2 \rangle$ that resulted in unexplained failure, is defined by the signature $\Sigma(T)$ and axioms. The signature $\Sigma(T)$ is constructed as follows:

1. Basic sorts of Σ that produce a non-empty intersection with $relCon(T)$ are in $\Sigma(T)$.
2. For basic sorts of $\Sigma(T)$ that correspond to the range of a static attribute, all domain constants of the sort are in $\Sigma(T)$.
3. For basic sorts of $\Sigma(T)$ that correspond to the range of a fluent, or domain of a fluent or a static, domain constants that are in $relCon(T)$ are in $\Sigma(T)$.
4. Domain properties restricted to basic sorts of $\Sigma(T)$ are in $\Sigma(T)$.

The axioms of $\mathscr{D}(T)$ are those of $\mathscr{D}$ restricted to the signature $\Sigma(T)$. Building on our example of a state transition with $a_{tg} = serve(rob_1, cup_1, person_1)$, $\mathscr{D}(T)$ would not include other robots, cups or people that may exist in the domain. It can be shown that each transition corresponding to the original system description $\mathscr{D}$ maps to a transition corresponding to $\mathscr{D}(T)$—see (Sridharan et al., 2017a) for details. States of $\mathscr{D}(T)$, i.e., literals formed of fluents and statics in the answer sets of the corresponding ASP program, are states in the RL formulation for axiom discovery, while ground actions of $\mathscr{D}(T)$ are the actions of the RL formulation. Furthermore, it is possible to pre-compute or reuse the information used to construct the system description relevant to any given transition.

Restricting exploration to just the relevant system description still leaves some open problems. For instance, Definitions 1 and 2 may not capture deeper relationships in the construction of the axioms. Also, in domains with complex relationships between objects, the state space may still be so large that exploration may need to be limited to a fraction of this space during the RL trials. Furthermore, Q-learning (by itself) does not generalize to relationally equivalent states, making it computationally expensive to conduct RL trials in complex domains.

We exploit the relational representation encoded in the ASP program to address some of these problems. Specifically, we use a relational representation to generalize to relationally equivalent states. After one or more episodes of Q-learning, all visited state-action pairs and their estimated Q-values are used to incrementally update a binary decision tree (BDT) that relationally represents the robot's experiences. The path from the root to a leaf node corresponds to a partial state description of a state-action pair (s,a). Internal nodes correspond to boolean *tests* of specific domain attributes or actions, and determine the node's descendants. The remainder of the state description is stored at the leaf, and some of this information may be transferred to a new node (for variance reduction) when the BDT is updated. The revised tree is used to compute a new policy, eliminating the need to completely rebuild the tree after each episode. The incremental inductive learning of the BDT draws on the algorithm by Driessens and Ramon (2003). In each Q-learning episode, the system stochastically decides to attempt either a random action or the one preferred by the current policy, ignoring actions currently invalidated by known constraints or affordances. Each action application also updates the information stored at a relevant leaf. Over time, the system assigns a higher value to outcomes perceived to be similar to the originally encountered unexpected transition. Since these errors may appear in the context of different combinations of domain attributes, these combinations are varied during RL trials, and the BDT reflects the exploration of different, but similar, MDPs. Q-learning episodes terminate when the Q-values stored in the BDT converge. For large, complex domains, we assume that when the number of explored attribute-value combinations reaches a fraction of the total number of possible combinations, the RL trials are halted.

Constructing Candidate Axioms The next step constructs candidate axioms for causal laws, executability conditions and forbidding affordances, each of which has a known structure—see Section 3.1. For instance, any executability condition has the non-occurrence of an action in the head, with a body of (pre)condition(s) that prevent this action from being included in a plan.

To construct these axioms, the system examines each leaf from the induced BDT, extracts a partial state-action description using its path to the root, and aggregates the stored domain attribute information from this description. Branches with low Q-values or corresponding to an action that did not result in the observed transition are eliminated. The resulting structures include information on the mean and variance of the stored Q-value, based on the different samples clustered under each leaf. Each structure's statements about object attributes holding or not holding are partitioned into two subsets, and all possible pairwise combinations of those subsets are elicited, producing unique tuples, each of which is the basis of a candidate. These store (a) the amassed Q-value; (b) the total variance; and (c) the number of training samples that influenced the candidate.

The system estimates the quality of the candidate from the Q-values of relevant samples it has experienced. It makes a number of random sample draws from the BDT, proportional to the size of the tree and the number of attributes not used as tests, without replacement. Each sample is a full state description of information at the leaf and along the path to the root. Each such state description that matches a candidate axiom adds to its Q-value, variance and count. Consider the target affordance in the RA domain, which prevents a robot with an electromagnetic arm from trying to pick up a heavy object. Assume that a candidate affordance has been constructed from an example leaf whose path to the root describes the partial state $loc(book_1, workshop)$, $loc(rob_1, workshop)$, $obj\_weight(book_1, heavy)$, $arm\_type(rob_1, electromagnetic)$, and that the example predicted Q-value = 9.5. One resulting candidate can be written as:

---

positive: [$obj\_weight(book_1, heavy)$, $arm\_type(rob_1, electromagnetic)$]
negative: []
Q-sum: 9.5, Count: 1, Mean: 9.5

Of the random samples drawn during candidate quality estimation, only some will match the candidate's partial state description, e.g.:

---

positive: [$loc(book_1, workshop)$, $obj\_weight(book_1, heavy)$, $obj\_status(book_1, intact)$]
negative: []
Q: 9.9

The system uses this sample to update the candidate:

---

positive: [$obj\_weight(book_1, heavy)$, $arm\_type(rob_1, electromagnetic)$]
negative: []
Q-sum: 19.4, Count: 2, Mean: 9.7

When all the candidates have been found, the system chooses the final set of axioms.

Validating Candidate Axioms The final step of the learning process is the generalization to a final set of axioms by selecting candidates with a sufficiently high likelihood of representing the

truth. First, candidates not refined by additional training samples after their construction are removed. Then, the candidates are ranked by the number of samples used to adjust them, and any candidates that elaborate other, higher-ranked candidates are removed. Finally, the remaining candidates undergo validation tests in simulation. For instance, a candidate executability condition, if true, should describe a case where an action will not provide the desired effects. If we can find a case that should imply a "failure" (i.e., unexpected transition) based on this axiom, but meets with "success" (i.e., expected transition) when the action is executed, the candidate is incorrect. So, the simulation takes a random state where the target action is known to succeed, and makes the minimal changes to attributes necessary to make the state match the partial state description of the candidate axiom. If executing the action only provides the expected outcomes in this adjusted state, the candidate axiom is discarded. Note that these validation tests are guaranteed *not* to retract any correct axioms, but may fail to retract some incorrect ones. The remaining candidate axioms, after suitably replacing constants with variables, are included in the ASP program for subsequent use.

4. Experimental Setup and Results

We analyzed our architecture's capabilities by evaluating the following five claims:

1. The architecture can learn symbolic knowledge structured as affordances, executability conditions, and causal laws;
2. During axiom discovery, automatically limiting search to the space relevant to any given unexpected transition improves computational efficiency;
3. Introducing validation tests in the loop of learning, planning, and execution, significantly improves the accuracy of the discovered axioms;
4. The discovered axioms help improve the quality of plans generated for any given goal; and
5. The distributed representation of different types of knowledge supports information reuse (for additional inference) and efficient information consolidation.

We discuss the last claim qualitatively, and evaluate the other four claims quantitatively. We set six target axioms—two each of causal laws, executability conditions, and affordances, as discussed in Example 1. We conducted trials of 1000 repetitions for each axiom and for combinations of these axioms, and examined the output axioms that the system discovered in each trial. In trials corresponding to particular target axioms, these axioms are used to simulate the action outcomes but are not included in the system description during axiom discovery. In other work (Sridharan et al., 2017b), we demonstrated the ability to discover two or more axioms simultaneously, and thus mostly focused on one axiom at a time to generate the results discussed below. We also conducted trials in which we explored different fractions of the search space, ignoring relevance, and compared the effect of these explorations over 300 trials for each exploration level. We used precision and recall as the measures of accuracy in all the experimental trials, and computed the time taken to discover the axioms. Furthermore, we allowed the system to conduct validation tests of each of the discovered axioms, and examined the resulting effects on accuracy.

4.1 Execution Trace and Discussion

As an illustration of the architecture's working, consider the robot in the RA domain that does not know that a brittle object will be damaged if it is put down. Suppose also that $obj\_surface(cup_1, brittle)$ and $obj\_status(cup_1, intact)$, and that the domain characteristics are otherwise as described earlier. Let the initial state therefore include:

$$in\_hand(cup_1, rob_1), \;\; loc(cup_1, workshop)$$
$$obj\_surface(cup_1, brittle), \;\; obj\_status(cup_1, intact)$$

and let the goal state description include $loc(C, kitchen)$, where C is a variable of sort *cup*, i.e., the objective is to deliver a cup to the kitchen. One possible plan to achieve this goal has two actions:

$$move(rob_1, kitchen)$$
$$putdown(rob_1, cup_1)$$

However, if this plan is executed, the second action results in an unexpected outcome (cup_1 being broken), potentially triggering the discovery process. Over a period of learning, the robot is able to add the following generic axiom to its system description:

$$putdown(R, O) \;\; \textbf{causes} \;\; obj\_status(O, damaged) \;\; \textbf{if} \;\; obj\_surface(O, brittle)$$

preventing (in the future) the inclusion a *putdown* action in a plan involving a brittle object.

Next, consider the claim about the benefits of the representation of knowledge in our architecture. Recall that action capabilities, and the preconditions and effects of actions, are encoded in a distributed manner using one or more axioms, which provides the following advantages:

- First, inference with the available information will be simplified. For instance, if a hammer has a graspable handle, this relation also holds true of the parent object class and for all other objects with graspable handles. In a similar manner, a forbidding affordance (i.e., disaffordance) that prevents the *pickup* action when the type of the arm does not match the weight of the object, can also be used to infer the inability to perform similar actions such as opening a heavy door or closing a large window. This relates to research in psychology that indicates that humans can judge the action capabilities of others without actually observing them perform the target action (Ramenzoni et al., 2010).

- Second, it will be possible to efficiently respond to queries that require consolidation of knowledge across attributes of robots or objects, by directing attention to the relevant knowledge. For instance, assume that domain knowledge includes affordance relations that describe the ability of robots, with different types of arms, to pick up (or not pick up) objects of different weights and surfaces. Questions of the form "what objects can robots with an electromagnetic arm pick up?", or "which robots can pick up cups?" can be answered efficiently by considering only the relevant affordance relations. Furthermore, it will be possible (although we do not consider it here) to develop composite affordance relations, e.g., a hammer may afford an "affix objects" action in the context of a specific agent because the handle affords to be picked up, and the hammer affords a swing action, by the agent.

- Third, it will be possible to provide more meaningful explanations of plans and inferences. Recall that affordances are statements about an action with respect to the attributes of object(s) and/or agent(s) involved in the action. Each affordance relates a partial state description to the actions that can (or cannot) be performed in that state. Such a representation is close to language structures likely to be used for providing explanations of the form "the small robot may be able to lift the large cylinder if the cylinder is not heavy", or "affixing a label to the coffee cup will not work when the cup is known to be brittle and the robot arm is only suitable for hard surfaces".

Proof of concept experimental trials support the benefits summarized above, and future work will conduct extensive experimental trials to obtain quantitative data in support of this claim.

4.2 Experimental Results

Our prior work described different architectures for discovering forbidding affordances (Sridharan et al., 2017b) and executability conditions (Sridharan & Meadows, 2016). In this paper, we evaluated whether a single architecture can discover these forms of knowledge and the knowledge corresponding to causal laws.

In our experiments, the average recall and precision were 0.98 and 0.72 respectively. The system took a mean 5.95 time units to perform decision tree induction and a mean 0.35 time units to extract a set of axioms. We found that axioms with more clauses were more difficult to learn. Also, if axioms were structured to include specific exceptions, there were more logical over-specifications, e.g., an axiom stating that "a light, brittle object cannot be labeled" instead of "a brittle object cannot be labeled". We treat these over-specifications as false positives, and the (overall) worst case recall and precision were 0.91 and 0.64 respectively. This supports our first claim, that our architecture can discover affordances, executability conditions, and causal laws.

Next, we conducted trials to examine the benefit of limiting exploration, for any given unexpected transition, to just the relevant portion of the search space. The mean time required to compute this relevant space was 533 time units, and the mean time to discover the axioms in this reduced search space was 6.3 time units, resulting in a mean total time of 539.3 units for discovering axioms in the search space relevant to any given transition. It is possible to precompute and cache the relevant space to be explored in response to any given unexpected transition in a given domain. Recall that the RA domain has $55,296$ static attribute combinations and $6,946,816$ physical (object) configurations. For our target axioms, this space can be reduced to $8-128$ static combinations (mean $= 54$) that are relevant to any given action.

Next, we conducted trials that explored only a fraction of the original space, ignoring relevance, to discover the target axioms. At 0.01% exploration of the original space, a mean total time of 608.4 units was required to discover the axioms; the corresponding precision and recall were 0.28 and 0.86 respectively. The time taken to explore 0.01% of the original search space is therefore similar to the length of time taken to explore just the space relevant to any given transition. However, exploring only the relevant search space provides significantly higher recall and precision. Next, at 0.1% exploration of the search space (again ignoring relevance), the discovery of axioms took a mean total time of 800.4 units; the corresponding precision and recall were 0.53 and 0.99 respectively. In this

Condition	Axiom 1	Axiom 2	Axiom 3	Axiom 4	Axiom 5	Axiom 6
Without axioms	101.4	0	37.9	164.4	29.7	121.3
With axiom	110.7	11.2	24.8	75.7	23.0	86.5

Table 1. Number of plans found without and with each of the target axioms under consideration.

case, although the recall is similar to that obtained when only the relevant search space is explored, the precision is still significantly lower and the computation time requirement is significantly higher. These results thus support the claim that limiting exploration to the space relevant to any given unexpected transition improves the computational efficiency of axiom discovery.

Our recent work showed that precision increased with the number of tests conducted to validate the discovered axioms (Sridharan et al., 2017b). There is a similar improvement for the unified architecture that supports the discovery of different types of axioms in a more complex domain. These validation tests improved precision by filtering incorrect axioms. For instance, when only the search space relevant to any unexpected transition is explored, precision improved (on average) from 0.72 to 0.91 after ten validation tests. Even for the 0.01% and the 0.1% exploration of the original space (ignoring relevance), ten validation tests improved precision from 0.28 to 0.90 and from 0.53 to 0.95 respectively. These results support our third claim, that including validation in the loop of planning, execution, and learning, improves the accuracy of discovered knowledge.

Finally, we explored the effect of the discovered axioms on the quality of plans generated. We conducted 1000 paired ASP-based planning trials for each axiom, and for all the axioms, with and without the corresponding target axiom(s) in the system description. Table 1 summarizes the results for each of the six axioms, and displays some interesting trends. For instance, the set of plans found after including axioms that represent knowledge corresponding to a causal law (e.g., axiom 1 or 2 in this study) was a *superset* of the plans found without including these axioms in the system description. In other words, discovering previously unknown knowledge corresponding to a causal law (on average) increases the number of possible plans that can be executed to achieve an assigned goal. On the other hand, the set of plans found after including axioms that represent knowledge of an executability condition or a forbidding affordance (axioms $3-6$ in this study) was a *subset* of the plans found without including these axioms. In other words, discovering knowledge corresponding to a previously unknown executability condition or forbidding affordance (on average) reduces the number of plans to achieve an assigned goal. However, when axioms corresponding to different types of knowledge are considered together, the set of plans found after including these axioms is no longer a subset or superset of the plans found without including the axioms. For instance, when all six target axioms are considered together, all we can say is that 29.2 is the average magnitude of the difference in the number of plans found with and without including these axioms. Furthermore, we verified that all the plans that were computed after including all the target axioms were correct.

5. Conclusions and Future Work

The paper described an architecture that enables the interactive and cumulative discovery of axioms corresponding to causal laws, executability conditions and forbidding affordances. Answer Set

Prolog was used to represent and reason with incomplete domain knowledge for planning and diagnostics, and used decision tree induction and relational reinforcement learning to identify specific candidate axioms and generalize across these specific instances. Experimental results in a simulated domain indicate that the architecture and the underlying representation of knowledge support the (a) reliable and efficient discovery of the target axioms; and (b) use of the discovered axioms to improve the quality of plans computed for assigned goals.

The architecture opens up multiple directions for further research. First, we will explore the problem of automatically determining when to discover different types of knowledge, and thoroughly investigate the benefits of the underlying distributed representation. Second, we will explore the discovery of affordance relations that enable the execution of specific actions by specific agents. Third, we will explore active discovery of axioms instead of limiting interactive discovery to situations corresponding to unexpected state transitions. Furthermore, we will evaluate the architecture on physical robots, which will require the use of the component of the architecture that reasons about perceptual inputs probabilistically. The long-term objective is to enable robots assisting humans to represent, reason with, and interactively revise different types of incomplete domain knowledge.

Acknowledgments

This work was supported in part by the US Office of Naval Research Science of Autonomy award N00014-13-1-0766, Asian Office of Aerospace Research and Development award FA2386-16-1-4071, and US National Science Foundation grant CHS-1452460. All opinions and conclusions expressed in this paper are those of the authors.

References

Balduccini, M., & Gelfond, M. (2003). Logic Programs with Consistency-Restoring Rules. *AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning* (pp. 9–18).

Baral, C., Gelfond, M., & Rushton, N. (2009). Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming*, *9*, 57–144.

Driessens, K., & Ramon, J. (2003). Relational Instance-Based Regression for Relational Reinforcement Learning. *International Conference on Machine Learning* (pp. 123–130).

Erdem, E., & Patoglu, V. (2012). Applications of Action Languages to Cognitive Robotics. In *Correct Reasoning*. Springer-Verlag.

Gelfond, M., & Inclezan, D. (2013). Some Properties of System Descriptions of AL_d. *Journal of Applied Non-Classical Logics, Special Issue on Equilibrium Logic and Answer Set Programming*, *23*, 105–120.

Gelfond, M., & Kahl, Y. (2014). *Knowledge Representation, Reasoning and the Design of Intelligent Agents*. Cambridge University Press.

Gelfond, M., & Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. *International Conference and Symposium on Logic Programming* (pp. 1070–1080).

Gibson, J. J. (1986). *The Ecological Approach to Visual Perception*. Psychology Press.

Gil, Y. (1994). Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. *International Conference on Machine Learning* (pp. 87–95). New Brunswick, USA.

Griffith, S., Sinapov, J., Sukhoy, V., & Stoytchev, A. (2012). A Behavior-Grounded Approach to Forming Object Categories: Separating Containers From Noncontainers. *IEEE Transactions on Autonomous Mental Development*, *4*, 54–69.

Horton, T. E., Chakraborty, A., & Amant, R. S. (2012). Affordances for Robots: A Brief Survey. *Avant: Journal of Philosophical-Interdisciplinary Vanguard*, *III*, 70–84.

Kaelbling, L., Littman, M., & Cassandra, A. (1998). Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, *101*, 99–134.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D. L., & Kolobov, A. (2006). BLOG: Probabilistic Models with Unknown Objects. In *Statistical Relational Learning*. MIT Press.

Otero, R. P. (2003). Induction of the Effects of Actions by Monotonic Methods. *International Conference on Inductive Logic Programming* (pp. 299–310).

Ramenzoni, V. C., Davis, T. J., Riley, M. A., & Shockley, K. (2010). Perceiving Action Boundaries: Learning Effects in Perceiving Maximum Jumping-Reach Affordances. *Attention, Perception and Psychophysics*, *72*, 1110–1119.

Richardson, M., & Domingos, P. (2006). Markov Logic Networks. *Machine Learning*, *62*, 107–136.

Sarathy, V., & Scheutz, M. (2016). A Logic-based Computational Framework for Inferring Cognitive Affordances. *IEEE Transactions on Cognitive and Developmental Systems*, *8*.

Shen, W.-M., & Simon, H. (1989). Rule Creation and Rule Learning through Environmental Exploration. *International Joint Conference on Artificial Intelligent* (pp. 675–680). Detroit, USA.

Sridharan, M., & Gelfond, M. (2016). Using Knowledge Representation and Reasoning Tools in the Design of Robots. *IJCAI Workshop on Knowledge-based Techniques for Problem Solving and Reasoning (KnowProS)*. New York, USA.

Sridharan, M., Gelfond, M., Zhang, S., & Wyatt, J. (2017a). *A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics*. Technical report, `http://arxiv.org/abs/1508.03891`.

Sridharan, M., & Meadows, B. (2016). Should I do that? Using Relational Reinforcement Learning and Declarative Programming to Discover Domain Axioms. *International Conference on Developmental Learning and Epigenetic Robotics (ICDL-EpiRob)*. Paris, France.

Sridharan, M., Meadows, B., & Gomez, R. (2017b). What can I not do? Towards an Architecture for Reasoning about and Learning Affordances. *International Conference on Automated Planning and Scheduling (ICAPS)*. Pittsburgh, USA.

Sutton, R. L., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA.

Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational Reinforcement Learning: An Overview. *Relational Reinforcement Learning Workshop at International Conference on Machine Learning*.

Walsh, T. J., Goschin, S., & Littman, M. L. (2010). Integrating Sample-Based Planning and Model-Based Reinforcement Learning. *AAAI Conference on Artificial Intelligence*. Atlanta, USA.

Warren, W. H. (1984). Perceiving Affordances: Visual Guidance of Stair Climbing. *Journal of Experimental Psychology: Human Perception and Performance*, *10*, 683–703.

Zhuo, H. H., Nguyen, T., & Kambhampati, S. (2013). Refining Incomplete Planning Domain Models Through Plan Traces. *International Joint Conference on Artificial Intelligence* (pp. 2451–2457). Beijing, China.