Towards an architecture for discovering domain dynamics

Sridharan, Mohan; Meadows, Ben

In the *International Workshop on Planning and Robotics (PlanRob) at the International Conference on Automated Planning and Scheduling (ICAPS)*, Pittsburgh, USA, June 19, 2017.

Towards an Architecture for Discovering Domain Dynamics: Affordances, Causal Laws, and Executability Conditions

Mohan Sridharan and Ben Meadows
Department of 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 a combined architecture that enables interactive and cumulative discovery of axioms governing action capabilities, and the preconditions and effects of actions in the domain. Specifically, Answer Set Prolog is used to represent the incomplete domain knowledge, and to reason with this knowledge for planning and diagnostics. Unexpected outcomes observed during plan execution trigger reinforcement learning to interactively discover specific instances of previously unknown axioms and to revise the existing axioms. Furthermore, a decision tree induction approach based on the relational domain representation constructs generic versions of the discovered axioms, which are then used for subsequent reasoning. The architecture's capabilities are illustrated and evaluated in a simulated domain of a robot moving objects to specific places or people in an indoor domain.

1 Introduction

Consider a robot[1] assisting humans by finding and moving desired objects to particular locations in an office building. In such complex, dynamic, and potentially resource-constrained environments, the robot will require a significant amount of domain knowledge, e.g., about domain objects, events and its own capabilities. At the same time, it will 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. Domain knowledge may include commonsense knowledge, including default knowledge such as "books are usually in the library" that holds in all but a few exceptional circumstances, e.g., cookbooks are in the kitchen. The robot also extracts information from its sensors and actuators, which is typically associated with numerical representations, e.g., probabilistic representations of uncertainty 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. Such domain axioms typically describe the preconditions and expected outcomes of actions that can be executed in the domain. The axioms also include knowledge of action capabilities, also known as *affordances*. With reference to an action, we define an affordance as a combination of attributes of object(s) and agent(s) involved in the action (Gibson 1986), e.g., the affordance of a person climbing a stair is described in terms of the stair's height with reference to the person's leg length (Warren 1984).

A fundamental challenge with these different types of knowledge possessed by a robot is that the knowledge is usually incomplete, and often needs to be revised over time. 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 augment and revise the different types of knowledge. Towards addressing this challenge, the architecture described in this paper seeks to enable interactive and cumulative discovery of domain axioms. Although our architecture can be used to discover both axioms governing actions performed by the robot and axioms governing exogenous actions, we focus on the former in this paper and assume that any knowledge of exogenous actions is limited to that encoded a priori—we leave the exploration of exogenous actions as a direction for further research. We discuss the following key characteristics of the architecture:

- For planning and diagnostics, an action language-based descriptions of 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 observations during plan execution are considered to indicate the existence of previously unknown knowledge about 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,

[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.

and to generalize across these candidates. These generic axioms are included in the ASP program and used for subsequent reasoning.

Given the focus on the ability to discover axioms corresponding to different types of knowledge, we abstract away the uncertainty in perception and do not describe the probabilistic reasoning component of the architecture. We illustrate the architecture's non-monotonic logical reasoning and axiom discovery capabilities in a simulated domain that has a robot assisting humans by delivering desired objects to particular locations or people in an indoor domain.

The remainder of this paper is organized as follows. 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 and a discussion of future work in Section 5.

2 Related Work

This section reviews some related work in logic programming, probabilistic reasoning, and relational learning, in the context of robotics.

Probabilistic algorithms are used widely for tasks such as reasoning and learning in robotics and AI, but these formulations, by themselves, make it difficult to reason with commonsense knowledge. In parallel, research in classical planning has provided many algorithms for representing and reasoning with commonsense knowledge. For instance, approaches based on first-order classical logic have been used for applications in robotics and AI, but they do not support desired capabilities such as non-monotonic logical reasoning and default reasoning. The logic programming community has developed many formalisms for non-monotonic logical reasoning, e.g., ASP is used by a growing international research community (Erdem and Patoglu 2012). These logical reasoning approaches, however, often require complete knowledge about the domain and the agents' capabilities. Also, these approaches, by themselves, do not support probabilistic reasoning, whereas quantitative reasoning about the uncertainty related to sensing and actuation on robots is often based on a probabilistic representation. Approaches have been developed to support both logical and probabilistic reasoning (Baral, Gelfond, and Rushton 2009; Milch et al. 2006; Richardson and Domingos 2006). The subset of these approaches based on first-order logic are often not expressive enough for certain types of knowledge, e.g., they model default knowledge and uncertainty by associating logic statements with numbers that may not be meaningful, whereas approaches based on logic programming do not support reasoning with large probabilistic components. For all these approaches, interactive discovery of knowledge continues to be an open problem.

Different approaches have been developed for representing and reasoning about action capabilities. Research in psychology indicates that humans can make accurate judgments about others' action capabilities using simple representations, without actually observing the subject perform the action(s) of interest (Ramenzoni et al. 2010). Many computational approaches have also been developed for representing and reasoning about affordances, often building on the knowledge representation and reasoning algorithms summarized above (Griffith et al. 2012; Sarathy and Scheutz 2016). Despite the existing research, open questions remain regarding the suitable definition and representation of affordances (Horton, Chakraborty, and Amant 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, a first-order logic representation and the observed effects of actions have been used to learn causal laws (Shen and Simon 1989). This approach used axioms as working hypotheses to be revised through discriminant learning when predictions fail, but only the encoded preconditions and effects of actions could be monitored. Another approach incrementally refined operators encoded in first-order logic by making any unexpected observations the preconditions or effects of operators (Gil 1994). This work focused on augmenting existing knowledge and not on revising incorrect axioms, and did not allow for the same action to lead to different outcomes in different contexts. Furthermore, these (and other such) approaches do not support generalization of acquired knowledge as described in this paper, and also have the (known) limitations of approaches based on first-order logic.

Researchers have used inductive logic with ASP to monotonically learn causal rules (Otero 2003). A maximum satisfiability framework has also been used with plan traces for refining incomplete domain models (Zhuo, Nguyen, and Kambhampati 2013). Interactive learning has also been posed as a Reinforcement Learning (RL) problem with an underlying Markov decision process (MDP) formulation (Sutton and Barto 1998). Approaches for efficient RL include sample-based planning algorithms (Walsh, Goschin, and Littman 2010), and Relational Reinforcement Learning (RRL), which combines relational representations with regression for Q-function generalization (Tadepalli, Givan, and Driessens 2004). However, existing interactive relational learning algorithms focus on planning, only generalize over the states and actions corresponding to a given planning task (Driessens and Ramon 2003), or do not support the desired commonsense reasoning.

In this paper, we present an architecture that supports automatic, interactive discovery of previously unknown knowledge governing action capabilities and the preconditions and effects of actions. We build on and extend our architectures that (a) combined declarative programming and probabilistic graphical models for planning and diagnostics in robotics (Sridharan and Gelfond 2016; Sridharan et al. 2017); and (b) integrated declarative programming with relational reinforcement learning for interactive discovery of previously unknown axioms governing action execution (Sridharan and Meadows 2016) and the agent's action capabilities (Sridharan, Meadows, and Gomez 2017).

3 Architecture Description

Figure 1 shows a block diagram of the overall architecture. For any given goal, ASP-based non-monotonic log-

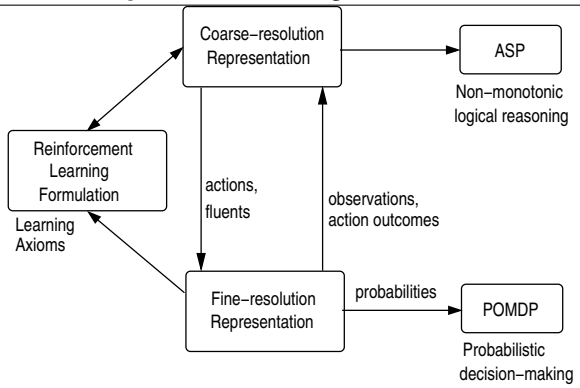

Figure 1: Architecture combines complementary strengths of declarative programming, probabilistic graphical models, and relational reinforcement learning.

ical reasoning with a coarse2-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. For complete details about representing and reasoning with tightly-coupled transition diagrams at these two resolutions, please see (Sridharan and Gelfond 2016; Sridharan et al. 2017). Here, 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. Instead, we describe ASP-based reasoning for planning and diagnostics, and relational reinforcement learning for axiom discovery. We illustrate these capabilities of our architecture in the context of the following 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:

1. A brittle object is damaged when it is put down.
2. Delivering an unlabeled object to a sales person causes it to be labeled.
3. A damaged object cannot be delivered to a person, except to an engineer.
4. An object with a brittle surface cannot be labeled.
5. A heavy object cannot be picked up by a robot with an electromagnetic arm.
6. 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 and Inclezan 2013). Action languages are formal models of parts of natural language used for describing transition diagrams. Action language 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. Defined fluents, on the other hand, do not obey the laws of inertia and may not 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 *literal*. In AL_d, three types of statements are supported:

a **causes** l_b **if** $p_0, \ldots, p_m$ (Causal law)
l **if** $p_0, \ldots, p_m$ (State constraint)
impossible $a_0, \ldots, a_k$ **if** $p_0, \ldots, p_m$ (Executability condition)

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 representation of any domain is given by the system description $\mathscr{D}$, a collection of statement of AL_d, and history $\mathscr{H}$. The system description $\mathscr{D}$ has a sorted signature Σ and axioms that describe the transition diagram τ. The basic sorts in the signature Σ for the RA domain in Example 1 include *place*, *robot*, *entity*, *person*, *object*, *role*, *armtype*, *weight*, *surface*, *cup*, *book* 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 (of people) $\{engineer, programmer, manager\}$.

Domain attributes and actions are described in terms of the sorts of their arguments. The Σ of the RA domain in-

cludes fluents such as:

$$loc(entity, place)$$
$$obj\_status(object, status)$$
$$in\_hand(entity, object)$$

statics such as:

$$obj\_surface(object, surface)$$
$$obj\_weight(object, weight)$$
$$person\_role(person, role)$$

and actions such as:

$$move(robot, place)$$
$$serve(robot, object, person)$$
$$affix\_label(robot, object)$$

The signature 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.

The axioms of the system description $\mathscr{D}$ include causal laws such as:

$$move(rob_1, Pl) \textbf{ causes } loc(rob_1, Pl)$$
$$serve(rob_1, O, P) \textbf{ causes } in\_hand(P, O)$$
$$pickup(rob_1, O) \textbf{ causes } in\_hand(rob_1, O)$$
$$affix\_label(rob_1, O) \textbf{ causes } has\_label(O)$$

where the second axiom implies that when the robot executes the *serve* action in the context of a specific object and person, the object is in the person's hand. Although we do not describe it here, it is also possible to encode non-deterministic causal laws (Sridharan et al. 2017).

Examples of state constraints of the RA domain include:

$$\neg loc(O, L_2) \textbf{ if } loc(O, L_1),\ L_1 \neq L_2$$
$$\neg in\_hand(E, O_2) \textbf{ if } in\_hand(E, O_1),\ O_1 \neq O_2$$
$$loc(O, L) \textbf{ if } loc(E, L),\ in\_hand(E, O)$$

where the second axiom implies that any entity (robot or person) can only hold one object at a time.

Examples of executability conditions of the RA domain include the following:

$$\textbf{impossible } move(rob_1, L) \textbf{ if } loc(rob_1, L)$$
$$\textbf{impossible } pickup(rob_1, O) \textbf{ if } loc(rob_1, L_1), loc(O, L_2), L_1 \neq L_2$$
$$\textbf{impossible } serve(rob_1, O_1, P) \textbf{ if } in\_hand(P, O_2), O_1 \neq O_2$$

where the first axiom implies that a robot cannot pick up an object unless the robot and the object are in the same location.

The recorded history $\mathscr{H}$ of a dynamic domain is usually a record of fluents observed to be true/false at a time step, i.e., $obs(fluent, boolean, step)$, and the occurrence of an action at a time step, i.e., $hpd(action, step)$. We expand this notion of history to construct a model that supports the representation of (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 if it not there, it is normally in the office" as follows:

$$\textbf{initial default } loc(X, library) \textbf{ if } textbook(X)$$
$$\textbf{initial default } loc(X, office) \textbf{ if } textbook(X), \neg loc(X, library)$$

We can also encode exceptions to this default statement such as "cookbooks are in the kitchen". Any inconsistencies introduced by observations or acquired (and encoded) knowledge, is addressed using consistency-restoring (CR) rules, as described later in this section.

Affordance representation Next, consider the representation of affordances, which can be of two types. Positive (i.e., enabling) affordances describe permissible uses of object(s) and agent(s) in actions, whereas negative (i.e., forbidding) affordances, also known as disaffordances, describe unsuitable combinations of object(s) and/or agent(s) in the context of specific actions. In this paper, we introduce the following generic definition of forbidding 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 we provide more details later). 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 the executability conditions and the forbidding affordances imply that, when the body of these axioms are true, the desired outcomes (i.e., effects) will not be achieved 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, either specify preconditions that when true will lead to the corresponding action not having the desired outcomes (negative affordance), or specify preconditions that when true will lead to the successful execution of an action that may not have been considered possible (so far). In other words, these relations remove or add elements from the set of actions available for consideration to

achieve any given goal. 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 to meet these requirements, e.g., when an executability condition is 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 4.2.

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, and incorporates CR rules (Balduccini and Gelfond 2003). We will use the terms CR-Prolog and ASP interchangeably in this paper. ASP is based on stable model semantics and non-monotonic logics, and includes *default negation* and *epistemic disjunction*, e.g., unlike "$\neg a$" that states *a is believed to be false*, "*not a*" only implies that *a is not believed to be true*, and unlike "$p \vee \neg p$" in propositional logic, "*p or* $\neg p$" is not tautologous (Gelfond and Kahl 2014). ASP can represent recursive definitions, defaults, causal relations, and constructs that are difficult to express in classical logic formalisms. The program Π thus consists of causal laws of $\mathscr{D}$, inertia axioms, closed world assumption for defined fluents, reality checks, and observations, actions, and defaults, recorded in $\mathscr{H}$. Every 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, so as to restore program consistency under exceptional circumstances. For instance, Π could include prioritized defaults encoded as ASP rules:

$$\begin{aligned} holds(loc(B,library),0) \leftarrow\ & \#textbook(B), \\ & not\neg holds(loc(B,library),0) \\ holds(loc(B,office),0) \leftarrow\ & \#textbook(B), \\ & \neg holds(loc(B,library),0), \\ & not\neg holds(loc(B,office),0) \end{aligned}$$

and the CR rules:

$$\begin{aligned} -holds(loc(B,library),0) \stackrel{+}{\leftarrow}\ & \#textbook(B). \\ -holds(loc(B,office),0) \stackrel{+}{\leftarrow}\ & \#textbook(B), \\ & \neg holds(loc(B,library),0). \end{aligned}$$

where the first CR rule implies that under exceptional circumstances, to restore program consistency, the robot can assume that a textbook is not in the library in the initial state. For planning, Π also includes the definition of a goal, a constraint stating that the goal must be achieved, and a rule generating possible future actions of the robot. Furthermore, although we do not discuss it in this paper, Π includes relations and axioms for explaining observed outcomes and partial scene descriptions.

Algorithms for computing the entailment, and for planning and diagnostics, reduce these tasks to computing the *answer sets* of the program $\Pi(\mathscr{D},\mathscr{H})$. The ground literals in an answer set represent the beliefs of an agent associated with program Π. An ASP solver is used to compute the answer sets of any given program Π. We use SPARC, a language that expands CR-Prolog to provide explicit constructs for specifying objects, relations, and their sorts (Balai, Gelfond, and Zhang 2013).

In the absence of comprehensive domain knowledge, appropriate plans may not be found and the execution of plan steps may have unexpected outcomes. In the RA domain, a robot moving a brittle cup to the kitchen may not know that putting the cup down is going to damage it—the unexpected outcome will only be observed after the action is completed. In this paper, *we focus on discovering such axioms that encode knowledge corresponding to causal laws, executability conditions, and forbidding affordances*. Including these axioms in Π will improve the quality of plans computed for achieving any given goal.

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 of this formulation 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. Each element in S grounds the domain attributes, and whether the last action to be executed had the expected outcome(s). Such a formulation mimics the experiences that a robot acquires incrementally and interactively as it performs the assigned tasks, and provides a principled approach to assign credit to current or previous state-action combinations for the observed transition(s). Note that the definition of the reward functions used in this approach has to be different when discovering axioms corresponding to the different types of knowledge. For instance, high immediate reward is to be provided:

- When an action's expected effects are not observed, if the focus is on discovering executability conditions.
- When effects in addition to those expected for an action are observed, if the focus is on discovering causal laws.
- When the expected and unexpected effects of an action are observed, if the focus is on discovering affordances.

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

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 a term $f(x_1,\ldots,x_n,Y)$ 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*.

Once we know the relevant object constants, we can 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 the unexplained transition, 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 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 the basic sorts of $\Sigma(T)$ are also in $\Sigma(T)$.

The axioms of $\mathscr{D}(T)$ consist of 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 in the original system description $\mathscr{D}$ maps to a transition in the system description $\mathscr{D}(T)$ relevant to the unexpected transition of interest—see (Sridharan et al. 2017) for complete details about establishing this relationship between transition diagrams. 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. Actions included in the RL formulation are (in a similar manner) the ground actions of $\mathscr{D}(T)$. Furthermore, it is possible to pre-compute or reuse some of the information used to construct the system description relevant to any given transition.

Coming back to our example of the target action $a_{tg} = serve(rob_1, cup_1, person_1)$, the relevant system description $\mathscr{D}(T)$ (with a small set of domain objects) includes thirteen atoms formed of relevant fluents:

$in\_hand(rob_1, cup_1)$, $item\_status(cup_1, damaged)$,
$item\_status(cup_1, intact)$, $labelled(cup_1, false)$,
$labelled(cup_1, true)$, $loc(person_1, kitchen)$,
$loc(person_1, library)$, $loc(person_1, office)$,
$loc(person_1, workshop)$, $loc(rob_1, kitchen)$,
$loc(rob_1, library)$, $loc(rob_1, office)$,
$loc(rob_1, workshop)$

eight possible (relevant) ground actions:

$affix\_label(rob_1, cup_1)$, $move(rob_1, kitchen)$,
$move(rob_1, library)$, $move(rob_1, office)$,
$move(rob_1, workshop)$, $pickup(rob_1, cup_1)$,
$putdown(rob_1, cup_1)$, $serve(rob_1, cup_1, person_1)$

and nine atoms formed of relevant static attributes:

$arm\_type(rob_1, electromagnetic)$, $surface(cup_1, hard)$,
$obj\_weight(cup_1, heavy)$, $obj\_weight(cup_1, light)$,
$role(person_1, engineer)$, $role(person_1, manager)$,
$role(person_1, sales)$, $surface(cup_1, brittle)$,
$arm\_type(rob_1, pneumatic)$

The system can then restrict its discovery process to focus on the possible combinations of relevant domain attributes.

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 space of relevant states and actions 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)—the motivation for constructing this tree is to relationally represent the robot's experiences. The path from the root to a leaf node corresponds to a partial 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—some of this 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 axioms. 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 will be 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, as described in 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. In a similar manner, causal laws have the occurrence of an action in the body and specific effects of executing the action in the head.

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 information about domain attributes 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 specific 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 axiom corresponding to a disaffordance relation in the RA domain, which prevents a robot with an electromagnetic arm from trying to pick up a heavy object. Assume that a candidate axiom has been constructed from an example leaf whose path to the root represents a partial state description that includes:

$loc(book_1, workshop)$, $loc(rob_1, workshop)$,
$obj\_weight(book_1, heavy)$, $arm\_type(rob_1, electromagnetic)$

and that a Q-value of 9.5 is associated with this example. 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 can then choose the final set of axioms to be added to the system description, i.e., the CR-Prolog program.

Filtering Candidate Axioms The final step of the axiom discovery process is generalization, i.e., the identification of the most generic form of candidate axioms with a sufficiently high likelihood of representing correct knowledge about the domain. 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. For instance,

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 axiom is incorrect. To this end, the simulation takes a random state where the target action is known to succeed, and makes minimal changes to the

domain 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 are not guaranteed to retract all incorrect ones. The remaining candidate axioms, after suitably replacing the object constants with variables, are included in the CR-Prolog program that is used for reasoning in the subsequent steps.

4 Experimental Setup and Results

In this section, we first state the claims about our architecture's capabilities (Section 4.1). Next, we illustrate some of these capabilities using an execution trace, and discuss some key advantages of the representation of knowledge in our architecture (Section 4.2). We then summarize and discuss the results of experimental evaluation in a simulated domain (Section 4.3).

4.1 Claims

We posit and evaluate the following six central claims about our architecture's capabilities:

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

We discuss the first claim qualitatively, and evaluate the other five claims quantitatively. We consider six target axioms—two each of affordances, executability conditions, and causal laws—including those discussed in Example 1. We conducted trials of 1000 repetitions apiece, providing them to the domain but removing them from the system's domain model. We examined the output axioms which the system discovered in each trial. Although each of these tests focused on a single target axiom, we have demonstrated the ability to discover axioms simultaneously elsewhere (Sridharan, Meadows, and Gomez 2017). We also conducted trials in which we explored different fractions of the search space, ignoring relevance, comparing 300 trials for each of these explorations (50 repetitions for each of the six target axioms). We used precision and recall as measures of accuracy. Furthermore, we allowed the system to test each discovered axiom in simulation, and report the resulting effects of this checking on initial accuracy.

4.2 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 the following literals:

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

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, 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 the robot from (in the future) including a *putdown* action in a plan involving a brittle object.

Let us now also consider the first 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, 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 also relates a partial state description to the actions that can (or cannot) be performed in states that build on this state description. When grounded, such a representation is close to language structures likely to be used for responses, for instance in diagnostics and plan explanation. It will thus be easy to translate and use this knowledge to make statements of the form "lifting this large cylinder with this small robot could work as long as 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 label applicator is known to work on hard surfaces".
- 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 the domain knowledge includes affordance relations that describe the ability of individual robots, with different strength levels, to pick up (or not pick up) objects of different weights and surfaces. Questions of the form "what objects can weak robots pick up?", or "which robots can pick up cups?" can be answered quickly by expanding Definition 2 to automatically consider only the affordance relations and attributes relevant to such questions. Furthermore, it will also 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 a pickup action and the hammer affords a swing action, by the agent.

- Third, the distributed representation will simplify inference and information reuse. 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 robot's arm does not match the weight of the object, can also be used to infer the robot's inability to perform similar actions such as opening a heavy door or closing a large window. This relates to research in psychology which indicates that humans can judge the intent and action capabilities of others without actually observing them perform the target actions (Ramenzoni et al. 2010).

Initial results of experimental evaluation do support the benefits listed above—future work will design and conduct extensive experimental trials to gather quantitative results in support of the first claim.

4.3 Experimental Results

Our prior work proposed different architectures for discovering forbidding affordances (Sridharan, Meadows, and Gomez 2017) and for executability conditions (Sridharan and Meadows 2016). Here, we evaluated whether a single architecture can discover these forms of knowledge as well as the knowledge of causal laws.

Discovering different types of knowledge: In our experiments, the average recall and precision over the entire set of target axioms, 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" was discovered 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 second claim, that our architecture can discover knowledge corresponding to affordances, executability conditions, and causal laws.

Effect of directed exploration: 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 values of 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 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. Furthermore, our implementation of the construction of the relevant search space can also be made more efficient. These results thus support the third claim, that limiting exploration to the space relevant to any given unexpected transition improves the computational efficiency of axiom discovery.

Robustness to perceptual noise: Next, we evaluated the fourth claim, i.e., whether axiom discovery is robust to perceptual noise, which we interpreted as the noise having a negative but non-catastrophic impact on performance. We introduced noise in the form of a fixed chance for an action to have an unexpected outcome in the form of the removal or addition of a single literal formed of a random fluent of the desired resultant state. We performed 500 repetitions for each of the six target axioms in the RA domain, and repeated this for 10 different levels of noise between $0-20\%$. The corresponding recall and precision scores are summarized in Figures 2 and 3. We observe that the addition of noise affects both precision and recall, with a more significant effect on precision. However, note that errors were predominantly false positives corresponding to over-specifications of the target axioms, e.g., they included executability conditions that would correctly prevent an action from being considered during planning but were not in the most general form possible. In addition, these false positives were incrementally eliminated as the robot performed a series of validations tests on the discovered axioms, as described below.

In the *International Workshop on Planning and Robotics (PlanRob) at the International Conference on Automated Planning and Scheduling (ICAPS)*, Pittsburgh, USA, June 19, 2017.

Condition	Causal laws		Executability conditions		(Dis)Affordances	
	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 (see Example 1) under consideration. On average, discovering causal laws increases the number of feasible plans to achieve any given goal, whereas discovering executability conditions and disaffordances decreases the number of plans that can be used to achieve any given goal.

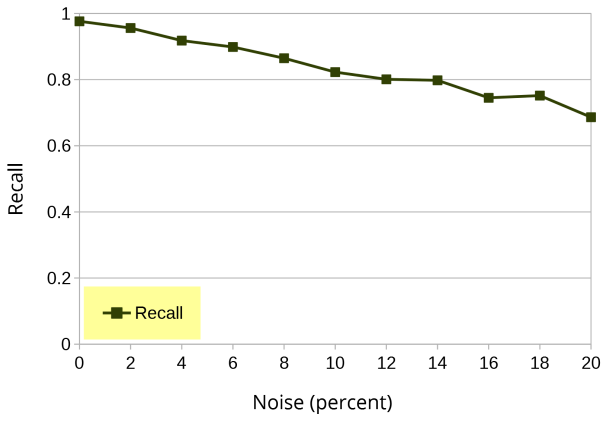

Figure 2: Recall scores as a function of added noise; although added noise reduces accurate recall of axioms, many errors correspond to over-specifications that are filtered by validation tests.

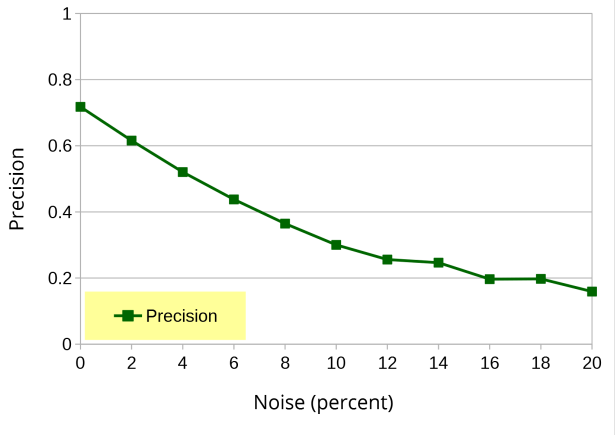

Figure 3: Precision scores as a function of added noise; although added noise affects precise discovery of axioms, many errors correspond to over-specifications that are filtered by validation tests.

Effect of validation tests: Our recent work showed that precision increased with the number of tests conducted to validate the discovered axioms (Sridharan, Meadows, and Gomez 2017). In our current work, we observed that there is a similar improvement in precision with our architecture that supports the discovery of axioms corresponding to different types of knowledge 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 (i.e., 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 fifth claim, that including validation in the loop of planning, execution, and learning, improves the accuracy of the discovered knowledge.

Effect on plan quality: 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 constructed 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 that can be executed 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

This paper described an architecture for interactive and cumulative discovery of axioms corresponding to causal laws, executability conditions and forbidding affordances. We used Answer Set Prolog 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 the context of a robot assisting humans in an indoor domain by moving particular objects to particular places or people) indicate that our approach:

- Supports reliable and efficient reasoning and discovery of the axioms corresponding to different types of knowledge, especially when search is limited to the space relevant to the unexpected transition that triggered axiom discovery;
- Provides some robustness to perceptual noise—although noise degrades the accuracy of axiom discovery, including validation tests in the loop of planning, execution and learning helps recover from these errors; and
- Results in the discovery of axioms that when included in the system description, improves the quality of the plans found for any given goal.

The architecture opens up multiple directions for research that we seek to investigate in the future. More specifically:

- 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;
- We will investigate the discovery of affordance relations that enable the execution of specific actions by specific agents;
- We will explore active interactive discovery of axioms instead of limiting discovery to situations corresponding to unexpected state transitions during plan execution; and
- 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 descriptions of incomplete domain knowledge.

Acknowledgements

The authors thank Pat Langley for discussions related to the representation of affordances described in this paper. 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

Balai, E.; Gelfond, M.; and Zhang, Y. 2013. Towards Answer Set Programming with Sorts. In *International Conference on Logic Programming and Nonmonotonic Reasoning*.

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

Baral, C.; Gelfond, M.; and Rushton, N. 2009. Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming* 9(1):57–144.

Driessens, K., and Ramon, J. 2003. Relational Instance-Based Regression for Relational Reinforcement Learning. In *International Conference on Machine Learning*, 123–130. AAAI Press.

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

Gelfond, M., and 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(1-2):105–120.

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

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

Gil, Y. 1994. Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In *International Conference on Machine Learning*, 87–95.

Griffith, S.; Sinapov, J.; Sukhoy, V.; and 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.; and Amant, R. S. 2012. Affordances for Robots: A Brief Survey. *Avant: Journal of Philosophical-Interdisciplinary Vanguard* III(2):70–84.

Milch, B.; Marthi, B.; Russell, S.; Sontag, D.; Ong, D. L.; and 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. In *International Conference on Inductive Logic Programming*, 299–310.

Ramenzoni, V. C.; Davis, T. J.; Riley, M. A.; and Shockley, K. 2010. Perceiving Action Boundaries: Learning Effects in Perceiving Maximum Jumping-Reach Affordances. *Attention, Perception and Psychophysics* 72(4):1110–1119.

Richardson, M., and Domingos, P. 2006. Markov Logic Networks. *Machine Learning* 62(1-2):107–136.

Sarathy, V., and Scheutz, M. 2016. A Logic-based Computational Framework for Inferring Cognitive Affordances. *IEEE Transactions on Cognitive and Developmental Systems* 8(3).

Shen, W.-M., and Simon, H. 1989. Rule Creation and Rule Learning through Environmental Exploration. In *International Joint Conference on Artificial Intelligent*, 675–680.

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

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

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

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

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

Tadepalli, P.; Givan, R.; and Driessens, K. 2004. Relational Reinforcement Learning: An Overview. In *Relational Reinforcement Learning Workshop at International Conference on Machine Learning*.

Walsh, T. J.; Goschin, S.; and Littman, M. L. 2010. Integrating Sample-Based Planning and Model-Based Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*.

Warren, W. H. 1984. Perceiving Affordances: Visual Guidance of Stair Climbing. *Journal of Experimental Psychology: Human Perception and Performance* 10(5):683–703.

Zhuo, H. H.; Nguyen, T.; and Kambhampati, S. 2013. Refining Incomplete Planning Domain Models Through Plan Traces. In *International Joint Conference on Artificial Intelligence*, 2451–2457.