Advances in Cognitive Systems 6 (2018) 1–20 Submitted 6/2018; published 8/2018

Knowledge Representation and Interactive Learning of Domain Knowledge for Human-Robot Interaction

Mohan Sridharan M.SRIDHARAN@BHAM.AC.UK
School of Computer Science, University of Birmingham, Birmingham, UK B15 2TT

Ben Meadows BMEA011@AUCKLANDUNI.AC.NZ
Department of Electrical and Computer Engineering, The University of Auckland, Auckland, NZ

Abstract

This paper describes an integrated architecture for representing, reasoning with, and interactively learning domain knowledge in the context of human-robot collaboration. Specifically, Answer Set Prolog, a declarative language, is used to represent and reason with incomplete commonsense knowledge about the domain. Non-monotonic logical reasoning with this knowledge identifies knowledge gaps and guides the interactive learning of relations that represent actions, and of axioms that encode affordances and action preconditions and effects. Learning uses decision-tree induction and relational reinforcement learning based on observations obtained from active exploration, reactive action execution, and human (verbal) descriptions. The learned actions and axioms are used for subsequent reasoning. The architecture's capabilities are illustrated and evaluated on a simulated robot assisting humans in an indoor domain.

1. Introduction

Consider robots[1] assisting humans in an office or a warehouse by delivering objects to particular locations. Information about such domains often includes commonsense knowledge, especially default knowledge that holds in all but a few exceptional circumstances, e.g., "books are usually in the library, but cookbooks are in the kitchen". Domain knowledge may also include some understanding of action preconditions and effects, and action capabilities, i.e., affordances. Human participants can help address the knowledge gaps, but these humans may not have the time and expertise to provide comprehensive domain information or elaborate feedback. The robots will thus need to reason with incomplete domain knowledge and revise this knowledge over time. The architecture described in this paper is a step towards these capabilities; it implements the following tenets:

- Knowledge elements encode symbolic content about objects, relations modeling domain attributes and actions at different levels of abstraction, and axioms composed of these relations.
- Knowledge elements are revised non-monotonically by reasoning with knowledge and observed outcomes of actions that may be immediate or delayed.

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

- Affordances are relations defined jointly over the attributes of agents and objects in the context of particular actions.
- Reasoning, learning, and interaction are coupled; values of state-action pairs are revised using observations obtained from active exploration and reactive action execution.

The combination of these tenets is novel, and our implementation exploits the complementary strengths of declarative programming, probabilistic reasoning, and interactive learning. We have explored subsets of these tenets in prior work (Sridharan & Meadows, 2017; Sridharan et al., 2017b). In this paper, we focus on the interplay between representation, reasoning and learning, and abstract away some aspects of our overall architecture, e.g., we merge some levels of representation and do not model perceptual uncertainty probabilistically. We then describe the following capabilities of the architecture:

- Incomplete domain knowledge described in an action language is translated into a relational representation in Answer Set Prolog (ASP) for inference, planning and diagnostics. ASP-based reasoning also limits interactive learning to the relevant part of the domain.
- Previously unknown actions' names, preconditions, effects, and objects over which they operate, along with the associated affordances, are learned using decision-tree induction and relational reinforcement learning based on observations from active exploration, reactive action execution, and verbal cues from humans.

The novelty in comparison with our prior work lies in learning actions along with the preconditions, effects and affordances, and in doing so interactively through relational inference and reinforcement using active exploration, reactive action execution, and human (verbal) inputs. We evaluate these capabilities in the context of a simulated robot delivering objects to particular people or places in an indoor domain. For instance, a robot equipped with our architecture and deployed to assist humans in an office could learn interactively that it cannot pick up heavy objects with its electromagnetic arm, and that grasping a brittle cup will cause the cup to break. We measure the robot's ability to acquire the missing knowledge reliably, and the resultant ability to compute minimal and correct plans for assigned goals.

We present the details of our approach in the pages that follow. We first review related work (Section 2) and describe key components of our architecture (Section 3). After this, we present our experimental evaluation of the framework (Section 4) and end with some closing thoughts (Section 5).

2. Related Work

Agents deployed in complex domains often have to represent and reason with incomplete domain knowledge, and learn from observations. Early work used a first-order logic representation and incrementally refined the action operators but did not allow for different outcomes in different contexts (Gil, 1994). With such approaches, it is also difficult to perform non-monotonic logical reasoning, or to merge new, unreliable information with existing beliefs. Research in logics has provided non-monotonic reasoning formalisms, e.g., ASP has been used in cognitive robotics (Erdem & Patoglu, 2012) and other applications (Erdem et al., 2016). Researchers have combined ASP with inductive learning to monotonically learn causal laws (Otero, 2003), and expanded the theory of actions to learn and revise system descriptions (i.e., domain knowledge) represented as ASP programs (Balduccini, 2007; Law et al., 2018). In parallel, cognitive architectures have been developed to reason with hierarchical knowledge in first-order logic and process perceptual information probabilistically to acquire domain knowledge (Laird, 2012). Cognitive architectures based on non-monotonic logic, or a combination of logic and probabilistic representations, have also been used to support reasoning and learning in robotics (Scheutz et al., 2007; Zhang et al., 2015). However, approaches based on classical first-order logic are often not expressive enough, e.g., modeling uncertainty by attaching probabilities to logic statements is not always meaningful. Algorithms based on logic programming, on the other hand, find it difficult to support one or more of the desired capabilities such as efficient and incremental learning of knowledge, learning from interactions, and reasoning with large probabilistic components. Existing algorithms and architectures also do not support generalization over learned knowledge as described in this paper.

Many formalizations have been proposed for representing, reasoning with, and learning affordances (Zech et al., 2017). Existing approaches represent affordances as possible effects of actions or behaviors (Guerin et al., 2013), or as emergent, functional and/or contextual properties based on attributes of the domain and the objects (Sarathy & Scheutz, 2016). These approaches have used logics, probabilistic reasoning or a combination of both. In recent work, we expanded on the existing work to provide a theory of affordances for cognitive systems (Langley et al., 2018) that made claims related to (a) representing the knowledge of affordances; (b) using this knowledge for plan understanding, plan generation, and the evaluation of actions in a plan; and (c) acquiring the knowledge of affordances. The architecture described in this paper seeks to explore some of these claims, e.g., that affordances are relations defined jointly over attributes of objects and attributes of agents in the context of specific actions.

Interactive task learning is a general framework for acquiring domain knowledge, using labeled examples or reinforcement signals obtained from domain observations, demonstrations, or instructions from humans (Chai et al., 2018; Laird et al., 2017). It can be viewed as building on early work on joint search through the space of hypotheses and observations (Simon & Lea, 1974). Algorithms for interactive task learning can be used to learn action models or search control rules that prevent particular actions from being considered under certain circumstances. Dynamic domains often require a learner to explore actions whose outcomes may be delayed, and Reinforcement learning (RL) elegantly addresses the credit assignment problem in such cases. RL is also appropriate for learning different types of knowledge that direct attention during both planning and learning. Approaches such as relational reinforcement learning (RRL) have been developed for efficient generalization in complex domains (Driessens & Ramon, 2003; Tadepalli et al., 2004). However, interactive learning algorithms, including those that exploit relational representations and different function approximations, tend to focus on a single planning task at a time and do not support the commonsense reasoning capabilities desired in robotics (Bloch & Laird, 2017; Driessens & Ramon, 2003). One exception was our prior work that combined the principles of non-monotonic logical reasoning and RRL to discover domain axioms (Sridharan & Meadows, 2017; Sridharan et al., 2017b). The architecture described in this paper significantly extends these capabilities to support interactive learning

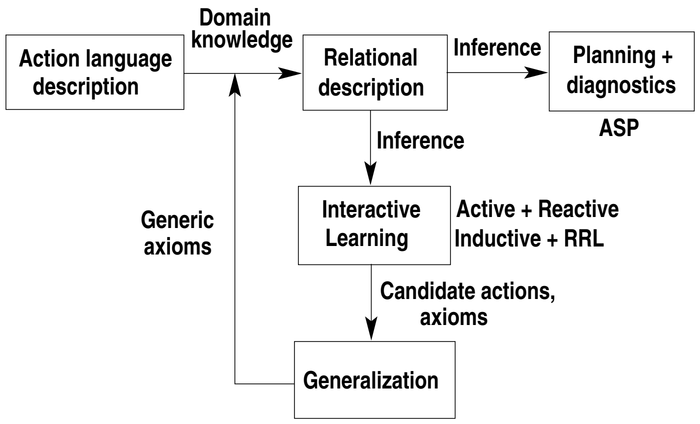

Figure 1. Architecture combines complementary strengths of declarative programming, probabilistic reasoning, and interactive learning for reasoning with and learning domain knowledge.

of actions along with their preconditions, effects and affordances, based on relational inference and reinforcement using active exploration, reactive action execution and human (verbal) inputs.

3. Proposed Architecture

Figure 1 shows key components of the overall architecture. Domain knowledge is encoded in an action language to construct tightly-coupled relational representations at two resolutions, with the fine-resolution representation defined as a refinement of the coarse-resolution representation. For any given goal, reasoning with commonsense knowledge at the coarse resolution provides a sequence of abstract actions. Each abstract action is implemented as a sequence of concrete actions using a partially observable Markov decision process (POMDP) that reasons probabilistically over the relevant part of the fine-resolution representation. The observations obtained during this implementation are added to the coarse-resolution history and used for subsequent reasoning. In this paper, we abstract away the reasoning at different resolutions and the probabilistic modeling of perceptual uncertainty. As stated earlier, we make these simplifications to better focus on the interplay between representation, reasoning, and learning. The relational representation is thus translated into an ASP program for planning and diagnostics. ASP-based reasoning also guides the interactive learning of actions, affordances, and the preconditions and effects of actions. This learning uses observations from active exploration, reactive execution, and human (verbal) descriptions—the learned knowledge is then used for subsequent reasoning. We use the following domain to illustrate our architecture's capabilities.

Example 1. *[Robot Assistant (RA) Domain]* A simulated robot/learner finds, labels, and delivers objects to people or places. Each place may have instances of objects such as $desk$, $book$, cup and $computer$. Each human has a particular $role$ (e.g., $engineer$, $manager$, $salesperson$). Objects are characterized by attributes such as $weight$ ($heavy$, $light$), $surface$ ($brittle$, $hard$), $status$ ($intact$, $damaged$), and $labeled$ ($true$, $false$). The robot's arm has a $type$ ($electromagnetic$,

$pneumatic$). The actions available to the robot include $pickup$, $putdown$, $move$, $label$, and $serve$, but it may not know some actions or axioms governing domain dynamics such as:

- A pneumatic arm cannot be used to serve a brittle object.
- Serving an object to a salesperson causes it to be labeled.
- Object with a brittle surface cannot be labeled unless the robot has an electromagnetic arm.

For simplicity, any other robots in the domain are assumed to have identical capabilities and cannot communicate with the learner. Humans and the learner can observe these robots. Humans can verbally describe other robots' activities, e.g., "Robot labeled the hard, hefty item" to help the learner acquire previously unknown actions and axioms. This domain becomes complex as the number of ground instances of objects and their attributes increases, e.g., there were $\approx$18,000 combinations of ground static attributes and $\approx$11 million combinations of ground fluent terms in an instantiation of the domain that we use in our experiments.

3.1 Knowledge Representation and Reasoning

We first describe the action language encoding of the domain dynamics, and its translation to CR-Prolog programs for knowledge representation and reasoning.

Action Language. Action languages are formal models of parts of natural language used for describing transition diagrams of dynamic systems. Our architecture uses action language $\mathcal{AL}_d$ (Gelfond & Inclezan, 2013) to describe the transition diagrams of the domain. $\mathcal{AL}_d$ has a sorted signature with *statics*, i.e., domain attributes whose truth values cannot be changed by actions, *fluents*, i.e., domain attributes whose truth values can be changed by actions, and *actions*, a set of elementary operations. Fluents can be *basic*, which obey inertia laws and can be changed by actions, or *defined*, which do not obey the laws of inertia and are not changed directly by actions. A domain attribute or its negation is a *literal*. $\mathcal{AL}_d$ allows three types of statements:

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

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

Domain Representation: Signature and Axioms. The domain representation consists of system description $\mathcal{D}$, which is a collection of statements of $\mathcal{AL}_d$, and history $\mathcal{H}$. $\mathcal{D}$ has a sorted signature Σ and axioms that describe the transition diagram τ. Σ defines the basic sorts, and domain attributes and actions. Basic sorts of the RA domain include $place$, $robot$, $object$, $entity$, $book$, $weight$, and the sort $step$ for temporal reasoning; the sorts are arranged hierarchically, e.g., $robot$ and $object$ are subsorts of $entity$. Σ includes ground instances of sorts, e.g., $office$, $workshop$, $kitchen$, and $library$ are instances of sort $place$. Domain attributes and actions are described in terms of the sorts of their arguments. The RA domain has fluents such as $loc(entity, place)$, the location of the robot and objects, with the locations of humans and other robots (if any) modeled as defined fluents whose values are obtained from external sensors; and $in\_hand(robot, object)$, which denotes whether a particular object is in the robot's hand. Static attributes of the RA domain include $arm\_type(robot, type)$, $obj\_status(object, status)$ etc. Actions of the RA domain include $move(robot, place)$, $pickup(robot, object)$, and $serve(robot, object, person)$. The signature Σ also includes a relation $holds(fluent, step)$ that implies a particular fluent is true at a particular timestep.

Axioms of the RA domain capture the domain's dynamics. These axioms include causal laws, state constraints and executability conditions such as:

$$\begin{aligned}
&move(rob_1, L) \ \textbf{causes} \ loc(rob_1, L)\\
&serve(rob_1, O, P) \ \textbf{causes} \ in\_hand(P, O)\\
&loc(O, L) \ \textbf{if} \ loc(rob_1, L), \ in\_hand(rob_1, O)\\
&\textbf{impossible} \ pickup(rob_1, O) \ \textbf{if} \ loc(rob_1, L_1), \ \ loc(O, L_2), \ L1 \neq L2
\end{aligned}$$

The history $\mathcal{H}$ of a dynamic domain is usually a record of fluents observed to be true or false at a particular time step, i.e., $obs(fluent, boolean, step)$, and the occurrence of an action at a particular time step, i.e., $occurs(action, step)$. This notion was expanded to represent defaults describing the values of fluents in the initial state (Sridharan et al., 2017a), e.g., "books are usually in the library and if not there, they are normally in the office" is encoded as:

$$\begin{aligned}
&\textbf{initial default} \ loc(X, library) \ \textbf{if} \ book(X)\\
&\textbf{initial default} \ loc(X, office) \ \textbf{if} \ book(X), \ \ \neg loc(X, library)
\end{aligned}$$

We can also encode exceptions to these defaults, e.g., "cookbooks are in the kitchen".

Domain Representation: Affordances. We define affordances, i.e., action capabilities, as relations between attributes of robot(s) and object(s) in the context of particular actions. Negative (i.e., forbidding or dis-) affordances describe unsuitable combinations of objects, robots, and actions. Positive affordances describe permissible uses of objects in actions by agents, including exceptions to the executability conditions. We represent affordances in a distributed manner, as follows:

$$\begin{aligned}
\textbf{impossible} \ A \ &\textbf{if} \ aff\_forbids(ID, A)\\
aff\_forbids(id_i, A) \ &\textbf{if} \ \ldots\\
\textbf{impossible} \ A \ &\textbf{if} \ \ldots, \ \ not \ aff\_permits(ID, A)\\
aff\_permits(id_j, A) \ &\textbf{if} \ \ldots
\end{aligned}$$

The first two statements imply that action A cannot occur if it is not afforded, and specify the conditions (i.e., attributes of robot and objects) under which the action is not afforded. The last two statements imply that action A that is not considered during planning due to an executability condition may have a positive affordance as an exception, and encode the exception. Note that each action can have multiple indexed affordances; as discussed later, such a distributed representation improves generalization and simplifies inference.

Reasoning with Domain Knowledge. The reasoning tasks of the robot associated with a system description $\mathcal{D}$ and history $\mathcal{H}$ include planning, diagnostics and inference. To perform these tasks, the domain representation is first translated into a program $\Pi(\mathcal{D}, \mathcal{H})$ in CR-Prolog[2], a variant of ASP that incorporates consistency restoring (CR) rules (Balduccini & Gelfond, 2003). ASP is based on stable model semantics, and supports *default negation* and *epistemic disjunction*, e.g., unlike "$\neg a$" that states *a is believed to be false*, "$\mathtt{not}\ a$" 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. ASP can represent recursive definitions and constructs that are difficult to express in classical logic formalisms, and it supports non-monotonic logical reasoning, i.e., the ability to revise previously held conclusions based on new evidence. These are desirable capabilities for robotics domains.

The program Π includes the signature and axioms of $\mathcal{D}$, inertia axioms, reality checks, and observations, actions, and defaults from $\mathcal{H}$. Every default also has a CR rule that allows the robot to assume the default's conclusion is false under exceptional circumstances, to restore consistency. Π also includes other appropriate definitions and axioms, e.g., for planning, we add a definition of goal, and axioms that force the system to identify and include actions in the plan until the goal is achieved. Algorithms for computing the entailment, and for planning and diagnostics, then reduce these tasks to computing *answer sets* of the CR-Prolog programs. Each such answer set represents the set of inferred beliefs of an agent associated with the program.

Reasoning with incomplete or incorrect knowledge may overlook valid plans, find suboptimal plans, or provide plans whose execution has unintended outcomes. For instance, suppose the robot in the RA domain is asked to deliver textbook $book_1$ to the $office$. It uses default knowledge to compute the plan

$$\begin{aligned}
&1)\ \ move(rob_1, library),\\
&2)\ \ pickup(rob_1, book_1),\\
&3)\ \ move(rob_1, office),\\
&4)\ \ putdown(rob_1, book_1).
\end{aligned}$$

This does not succeed because, unknown to the robot, its electromagnetic arm cannot pick up the heavy book. This exemplifies the kind of unknown knowledge we have taken as our task to learn. We next describe the interactive learning of such knowledge.

3.2 Interactive Learning

In complex domains, learning previously unknown actions and related axioms in the most generic form may require many labeled examples, and it may be rather difficult to obtain such labeled examples. Also, humans may not have the time and expertise required to provide labeled examples, and an action's effects may be observed immediately or after a delay. To address these challenges, our architecture includes two schemes for interactive acquisition of labeled examples:

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

(i) active learning from verbal cues provided by humans; and

(ii) relational reinforcement learning that considers delayed rewards to mimic cumulative learning based on observations from active exploration or reactive action execution.

Learning from Human Interaction. The first interactive learning scheme uses verbal input provided by humans describing the observed behaviors of other robots in the domain. This scheme makes the following assumptions:

- Other robots in the domain have the same sensing and actuation capabilities as the learner;
- The human description of observed behavior focuses on one action at a time; also, this description may be ambiguous but it is not intentionally incorrect; and
- The learner can generate logic statements corresponding to the observed attributes of robot(s) or object(s) in the domain.

These assumptions are reasonable for many robotics domains, and they significantly simplify the learner's interaction with humans.

The learner solicits human input when available and receives a transcribed verbal description of an action and observations of the action's consequences, e.g., the learner may receive "The robot is labeling the fairly big textbook." and $\mathit{labeled}(\mathit{book}_1)$. We use the Stanford log-linear part-of-speech (POS) tagger (Toutanova et al., 2003). We employ a left, second-order sequence information model to determine each word's POS tag and append it to the word. In our example, the output is a string such as "The_DT robot_NN is_VBZ labeling_VBG the_DT fairly_RB big_JJ textbook_NN", where "VB" represents a verb, "NN" is a noun etc. The learner transforms this string to $\langle \mathit{word}, \mathit{POS} \rangle$ pairs, and transforms the sentence's verb into first-person present-tense using rules from a lemma list (Someya, 1998) and WordNet (Miller, 1995), e.g., $\langle \mathit{is}, \mathit{VBZ} \rangle$ $\langle \mathit{labeling}, \mathit{VBG} \rangle$ becomes the verb "label". The learner also marks each noun phrase as a sequence of zero or more adjectival terms followed by a noun, discarding other interleaved words. Our example sentence's noun phrases are *robot* and *big textbook*. Nouns signify object sorts and adjectival terms signify values of static attributes. To determine terms' referents, WordNet relations such as *linked synsets* are used to find a synonym that is also a domain symbol, e.g., "big" and "heavy" share a WordNet synset, *heavy* is an attribute value, and $\mathit{book}(\mathit{book}_1)$ and $\mathit{obj\_weight}(\mathit{book}_1, \mathit{heavy})$ are domain attributes. The matched domain symbols combine to refer to particular objects. We require static attributes' values to be disjoint sets, and each noun phrase to signify an existing object—these are true by design in our domain.

Next, the robot constructs an action literal from the verb and object referents; in our example, this is $\mathit{label}(\mathit{rob}_1, \mathit{book}_1)$. The arguments' lowest-level sorts are assumed to be the valid arguments of the action, e.g., $\mathit{label}(\#\mathit{robot}, \#\mathit{book})$. If this candidate action does not match any known action literal, the robot lifts the literal, its arguments and the observed action consequences. This forms the basis for constructing candidate causal laws and generalizing over time. For instance:

$$\mathit{label}(\mathit{rob}_1, \mathit{book}_1)\ \textbf{causes}\ \mathit{labeled}(\mathit{book}_1)$$

is lifted to:

$$\mathit{label}(R, B)\ \textbf{causes}\ \mathit{labeled}(B)$$

If, on the other hand, the new literal matches an existing one, the first common ancestor of each argument's sort is found. For instance, if the learner knows $label(\#robot, \#cup)$ and finds that $label(\#robot, \#book)$ has matching consequences, it will generalize over these specific instances to obtain the action $label(\#robot, \#object)$. This method for learning from interaction with humans adapts existing natural language processing methods to work with our representation. It helps the learner acquire a previously unknown action's name, and the sorts of objects the action operates on. However, this knowledge is not sufficient because the learner may still not know axioms that govern the domain dynamics related to this action. This missing knowledge is acquired using the second learning scheme below.

Relational Reinforcement Learning. The second learning scheme enables axiom discovery by active exploration of the transition corresponding to a particular action, or by exploration in response to unexpected and unexplained transitions. To explore a particular transition, the resultant state is set as the goal of a reinforcement learning (RL) problem, i.e., the objective is to find state-action pairs most likely to lead to this (and other similar) states. The underlying MDP is defined by a set of states (S), set of actions (A), state transition function $T_f : S \times A \times S' \rightarrow [0, 1]$, and reward function $R_f : S \times A \times S' \rightarrow \mathfrak{R}$. Similar to classical RL formulations, T_f and R_f are unknown to the agent. Each state has ground atoms formed of the domain attributes (i.e., fluent terms and statics), and a boolean literal describing whether the most recent action had the expected outcome. Each action is a ground action of the system description. The functions T_f and R_f are constructed from statistics collected in an initial training phase; T_f is a probabilistic model of the uncertainty in state transitions, while R_f provides instantaneous rewards for executing particular actions in particular states. The RL formulation is constructed automatically from the system description—Sridharan et al. (2017a) describe a method for translating an ASP-based system description to a representation for probabilistic sequential decision making.

The values of state-action pairs are estimated in a series of episodes, until convergence, using the Q-learning algorithm (Sutton & Barto, 1998). In each episode, the agent executes a sequence of actions chosen by an ϵ-greedy algorithm with eligibility traces. The combinations of states and actions invalidated by existing axioms are not explored. Each episode terminates when a time limit is exceeded or the target action succeeds. The physical configuration of objects is then reset to its state from the beginning of the episode, and a new episode begins. Similar to other standard RL formulations, such a formulation can become computationally intractable for complex domains. A key advantage of our architecture is that ASP-based reasoning can be used to automatically restrict the object constants, domain attributes and axioms *relevant* to the desired transition, i.e., to those that influence or are influenced by the transition, which significantly reduces the search space. This notion of relevance is based on the following desiderata regarding the relations that may appear in a discovered axiom:

- For any static attribute that may exist in the body of the discovered axiom, we wish to explore all possible elements in the range of the attribute, e.g., for action $serve(rob_1, cup_1, person_1)$, all possible weights of cup_1 and roles of $person_1$ are explored.

- For any fluent that may appear in the body of the axiom, we wish to explore only those elements in the range of the fluent that occur in the state before or after the state transition. Any other element cannot, by design, be influenced by this transition anyway.

ASP-based reasoning is used to encode these requirements and automatically construct the system description $\mathcal{D}(T)$, the part of $\mathcal{D}$ relevant to the transition T. To do so, we first define the object constants relevant to the transition of interest. These definitions are adapted from the definitions introduced in Sridharan et al. (2017a).

Definition 1. *[Relevant object constants]*
Let a_{tg} be the *target action* that when executed in state σ_1 resulted in the unexpected transition $T = \langle \sigma_1, a_{tg}, \sigma_2 \rangle$. Let $relCon(T)$ be the set of object constants of Σ of $\mathcal{D}$ identified using the following rules:

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

Object constants from $relCon(T)$ are said to be *relevant* to T. For instance, consider the action $a_{tg} = serve(rob_1, cup_1, person_1)$ in the RA domain, with $loc(rob_1, office)$, $loc(cup_1, office)$, and $loc(person_1, office)$ in σ_1. Then, the object constants relevant to the transition T include rob_1, cup_1, $person_1$, and $office$.

Definition 2. *[Relevant system description]*
The system description relevant to the desired transition $T = \langle \sigma_1, a_{tg}, \sigma_2 \rangle$, i.e., $\mathcal{D}(T)$, is defined by signature $\Sigma(T)$ and axioms. The signature $\Sigma(T)$ is constructed to comprise the following:

1. Basic sorts of Σ that produce a non-empty intersection with $relCon(T)$.
2. All object constants of basic sorts of $\Sigma(T)$ that form the range of a static attribute.
3. The object constants of basic sorts of $\Sigma(T)$ that form the range of a fluent, or the domain of a fluent or a static, and are in $relCon(T)$.
4. Domain attributes restricted to basic sorts of $\Sigma(T)$.

Axioms of $\mathcal{D}(T)$ are those of $\mathcal{D}$ restricted to $\Sigma(T)$. For $a_{tg} = serve(rob_1, cup_1, person_1)$ in our current example, $\mathcal{D}(T)$ does not include other robots, cups or people in the domain. It can be shown that for each transition in the transition diagram of the system description $\mathcal{D}$, there is a transition in the transition diagram of $\mathcal{D}(T)$. States of $\mathcal{D}(T)$, i.e., literals comprising fluents and statics in the answer sets of the ASP program, are states in the RL formulation, and actions are ground actions of $\mathcal{D}(T)$. Furthermore, information used to construct $\mathcal{D}(T)$ for any given T can be cached and reused for computing the system description relevant to other similar transitions.

Once the $\mathcal{D}(T)$ relevant to the target transition T has been identified, the RL formulation is constructed as before to compute the values of state-action combinations. The extent to which computing $\mathcal{D}(T)$ reduces the search space depends on the relationships between the domain attributes and axioms. For instance, although there are several thousand static attribute combinations and more than a million object configurations in our instantiation of the RA domain, computing $\mathcal{D}(T)$

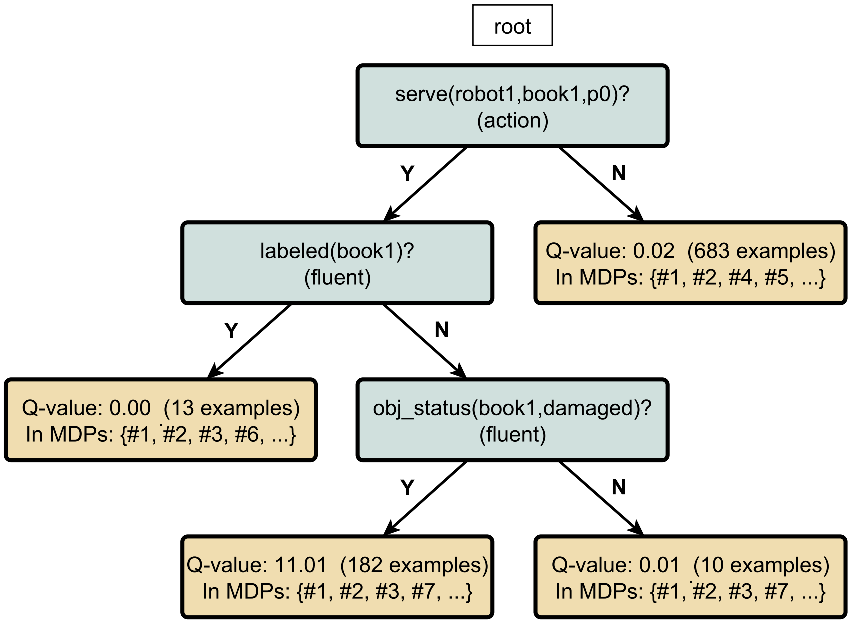
root
serve(robot1,book1,p0)?
(action)
Y
N
labeled(book1)?
(fluent)
Q-value: 0.02 (683 examples)
In MDPs: {#1, #2, #4, #5, ...}
Y
N
Q-value: 0.00 (13 examples)
In MDPs: {#1, #2, #3, #6, ...}
obj_status(book1,damaged)?
(fluent)
Y
N
Q-value: 11.01 (182 examples)
In MDPs: {#1, #2, #3, #7, ...}
Q-value: 0.01 (10 examples)
In MDPs: {#1, #2, #3, #7, ...}

Figure 2. Illustrative example of a binary decision tree (BDT) with nodes representing tests of domain literals. The BDT is constructed incrementally over time.

often reduces the space of attribute combinations to as few as 12 for the $serve$ action. However, in other domains with complex relationships between objects, e.g., in a blocks world domain in which different physical arrangements of objects with different attributes are equivalent, computing $\mathcal{D}(T)$ may not significantly reduce the space of attribute combinations to be explored. In such cases, exploration may need to be further limited to a fraction of this restricted state space. Furthermore, Q-learning does not generalize to relationally equivalent states.

Inspired by the RRL-TG algorithm (Driessens & Ramon, 2003), we facilitate generalization to relationally equivalent states by constructing a binary decision tree (BDT) whose nodes represent tests of domain literals—Figure 2 shows an example of a BDT. Unlike the destructive branching of RRL-TG, we model the partial description of a state-action pair as a path to a leaf where we store the remaining state information. When Q-value variance is reduced by adding a test at a leaf, the BDT is expanded and used to compute policies in subsequent RL episodes. To learn generic versions of axioms, the robot explores different values of static attributes and fluent literals. ASP-based reasoning automatically selects relevant combinations to make exploration tractable, and uses sampling if the search space is too large. Unlike traditional RRL methods, the learned Q-values now represent values across different MDPs.

After learned values converge, axioms are constructed from the BDT. A partial description (path to leaf) is selected if it is associated with the high accrued value, and all subsets of its literals become candidate axioms. Since each candidate axiom could correspond to different branches of the BDT, the learner randomly draws a number of samples without replacement, considers additional

Table 1. Overall control loop used by the architecture for learning and reasoning.

```
Input: $\Pi(\mathcal{D}, \mathcal{H})$; goal description; initial state $\sigma_1$.
Output: Control signals for robot to execute.
planMode = true, learnType = 0
while true do
    Add observations to history.
    ComputeAnswerSets($\Pi(\mathcal{D}, \mathcal{H})$)
    if planMode then
        if existsGoal then
            /* Goal exists, consistent model, execute plan */
            if explainedObs then
                ExecutePlanStep()
            else
                /* Q-RRL */
                planMode = false
                learnType = 1
            end
        else
            /* Active learning */
            planMode = false
            learnType = 2
        end
    else
        /* Interrupt learning if needed */
        if interrupt then
            planMode = true
        /* Continue learning */
        else if learnType == 1 then
            ContinueRRL()
        else if learnType == 2 then
            if verbalCue then
                ContinueActiveLearn()
            else
                ContinueActiveRRL()
            end
    end
end
```

literals stored at the leaves, and alters candidates that match the sample. Candidates with sufficient support are *validated*, i.e., tested under conditions that are simulated to match the transition that triggered learning. Candidates that do not pass these tests are removed from further consideration. For instance, if a learned executability condition is correct, executing the action when literals in the body are true should not provide the expected outcome. Note that these tests are guaranteed to not eliminate any valid axioms although they may not remove all false positive candidates. The final candidates are lifted by replacing ground terms with variables, and added to the ASP program as axioms for subsequent reasoning. We refer to our RRL approach as "Q-RRL".

Control Loop. Table 1 describes the overall control loop for reasoning and learning in our architecture. The baseline behavior (lines 5-17) is to plan and execute actions to achieve the given goal as long as a consistent model of history is can be computed (lines 7-9). If such a model cannot be constructed, it is attributed to an unexplained, unexpected transition, and the robot triggers Q-RRL (lines 9-12) to discover the corresponding unknown axioms (lines 20-21). If there is no active goal to be achieved, the robot triggers active learning (lines 13-16) using Q-RRL (lines 25-27) or verbal descriptions obtained from a human participant (lines 23-25) to learn previously unknown actions or axioms. When in the learning mode, the robot can be interrupted if needed (lines 18-19), e.g., to pursue a new goal. An implementation of the control loop and the components described above can be found online: `https://github.com/bmeadows/actionaxiomlearning`.

4. Experimental Design and Results

In this section, we describe the results of experimentally evaluating three distinct hypotheses:

- **H1**: Verbal descriptions enable active learning of actions and causal laws, which serve as the foundation for further learning;
- **H2**: Q-RRL enables reliable discovery of axioms related to the known or learned domain actions; and
- **H3**: Learned domain knowledge improves the quality of plans computed for any given goal.

We describe execution traces in support of hypothesis $H1$, and provide quantitative results in support of hypotheses $H2$ and $H3$.

In an initial training phase, we set the values of relevant parameters in Q-RRL by trading off accurate estimation of policies against processing time, e.g., learning rate and exploration preference were fixed at 0.1. Also, up to ten validation tests were conducted to evaluate candidate axioms. Furthermore, we used domain knowledge and statistics (e.g., of the observed outcomes of multiple runs of particular actions) to compute the transition function (T_F) and reward function (R_F) that were not known to the robot, but were used to conduct realistic trials in simulation.

We report results of experiments conducted in the context of discovering two actions ($serve$ and $label$), and the corresponding axioms in the RA domain (Example 1):

1. Serving an object to a salesperson causes it to be labelled (*causal law*);
2. A person who is not an engineer cannot be served a damaged object (*executability condition*);
3. A robot with a pneumatic arm cannot serve a brittle object (*negative affordance*);

4. A damaged object cannot be served to a person who is not an engineer, unless it is labeled (*positive affordance*);
5. An object with a brittle surface cannot be labeled by a robot (*executability condition*);
6. A damaged object cannot be labeled by a robot with a pneumatic arm (*negative affordance*);
7. Labelling a light object with a pneumatic arm causes it to be damaged (*causal law*); and
8. An object with a brittle surface cannot be labelled by a robot, unless the object is heavy and the robot has an electromagnetic arm (*positive affordance*).

where the first four axioms correspond to action $serve$; the others correspond to action $label$.

The robot learned the representation of each action and the associated causal law from verbal descriptions. The robot then used Q-RRL to learn one causal law, one executability condition, one positive affordance, and one negative affordance for each of the two actions ($serve$, $label$). Axioms for each action can be discovered concurrently. Candidate axioms were constrained to have no more than two positive literals and two negative literals formed of domain attributes—this limit can be increased as needed in other domains at the expense of a corresponding increase in computational complexity. Up to 10 validation tests were conducted to evaluate candidate axioms.

We used *precision* and *recall* as the performance measures. Also, *plan quality* was measured as the ability to compute minimal and correct plans to achieve any given goal. Each value of the performance measures reported below was averaged over 1000 repetitions (e.g., for each axiom). Axioms were required to exactly match the ground truth to be counted as true positives; under-specifications (e.g., some missing literals) and most over-specifications (e.g., unnecessary literals) were considered false positives.

4.1 Execution Trace

The following execution trace supports $H1$ by illustrating learning of actions and the objects those actions operate on, using verbal cues from human participants.

Execution Example 1. *[Learning from human input]*
Suppose the robot in Example 1 does not know actions $label$ and $serve$, or the related axioms. For each action, the agent receives five grammatically-correct descriptions of the action being applied by another robot; these statements uphold our assumptions but otherwise vary arbitrarily. First consider the action $label$:

- The learner receives "A robot is labeling the lightweight cup" and the observation $labeled(cup_1)$. It parses the statement, matches it to the domain, lifts it to $label(\#robot, \#cup)$, and infers:
$$label(R, B) \textbf{ causes } labeled(B)$$
- Next, the learner receives "Robot labeled one computer", and $labeled(comp_1)$. It learns the signature $label(\#robot, \#computer)$ and generalizes over the learned signatures to obtain $label(\#robot, \#object)$.
- Further input descriptions are automatically reconciled either when specific sorts are subsumed by more general ones, e.g., when it learns from "The pneumatic robot labels the light breakable cup", or the parse results in an exact match for the action description, as in "Next the robot labeled the hard, hefty item".

Next, in the context of learning the serve action:

- The learner receives the observation $\mathit{in\_hand}(p_1, book_1)$ and "A robot serves a manual to the manager". It produces the action description $\mathit{serve}(\#robot, \#book, \#person)$ and extracts the causal law:

$$\mathit{serve}(R, O, P) \ \textbf{causes} \ \mathit{in\_hand}(P, O)$$

- Next, the learner is given "The pneumatic robot is serving the breakable cup to the clerical person over there" and $\mathit{in\_hand}(p_0, cup_1)$. Generalizing over the two examples results in $\mathit{serve}(\#robot, \#object, \#person)$. The remaining sentences, "Robot serves ledger to clerical person" and "A robot served a lightweight cup to an expert", fit the inferred structures and do not change them.

For both actions, two examples were sufficient to reach the required level of generality to model the action and an initial causal law. A key advantage of learning from verbal cues is that only a small number of examples are needed to learn the actions and the objects that they operate on. This is especially useful when actions have irreversible effects. The disadvantage is that humans are expected to provide correct descriptions of the behaviors they observe, although the robot can identify and revise any incorrect information learned and included in the ASP program.

The distributed representation of knowledge in our architecture supports some key capabilities. First, this representation simplifies inference and information reuse. For instance, if a cup has a graspable handle, this relation also holds true for other objects with handles. If an affordance prevents the robot from picking up a heavy object, this information may be used to infer that it cannot open a large window. This capability relates to research in psychology which indicates that humans can judge action capabilities of others without actually observing them perform the target actions (Ramenzoni et al., 2010). Second, it becomes possible to respond efficiently to queries that require consolidation of knowledge across different attributes, and 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, for the agent. Finally, learning from verbal descriptions can help provide more meaningful explanations of decisions.

4.2 Quantitative Evaluation

Next we describe the results of experimentally evaluating hypotheses H2 and H3 in simulated environments.

H2: Q-RRL enables reliable discovery of axioms. We explored the ability of Q-RRL to support the learning of previously unknown axioms related to a known (or newly learned) action. Results averaged over the four axioms for each action are summarized in Table 2. We observe that Q-RRL attains high recall and precision, especially after the candidate axioms are validated. The accuracy of discovering the axioms for serve is a little lower than that for label because the action serve is more complex, i.e., it has more arguments than the action label. There were very few differences in the values of performance measures for causal laws, executability conditions and negative affordances. The recall and precision measures were a little lower for positive affordances since axioms corresponding to positive affordances are more complex. They add context to an

Table 2. Accuracy when Q-RRL was used to discover multiple axioms corresponding to two specific actions: $label$ and $serve$. Q-RRL provides high recall and precision, especially after candidate axioms are validated.

Action	**Recall**	**Precision**	**Precision (validated)**
label	0.92	0.82	0.96
serve	0.88	0.70	0.95

executability condition to make an action applicable; this action would remain inapplicable in the absence of this context. Furthermore, note that human input is not essential for the learning of axioms with Q-RRL—a robot can learn the axioms from experiences accumulated over time through active observation or reactive action execution.

H3: Learning improves plan quality. Next, we explored the effects of the discovered axioms on the system's ability to generate plans that provide the correct outcome. For each axiom of each target action, we conducted 1000 paired ASP-based planning trials with and without the corresponding target axiom in the system description. Each trial used a randomized scenario that required the target action to achieve the goal. We found that adding the learned executability conditions or negative affordances to the ASP program resulted in 13% (serve) or 23% (label) fewer plans. Executability conditions and negative affordances preclude certain actions in some contexts, i.e., these results indicate that the acquired knowledge improved the quality of the computed plans by eliminating plans that were incorrect or suboptimal. In a similar manner, learning and including knowledge of previously unknown positive affordances in the ASP program resulted in 17% (serve) or 23% (label) more plans. Since knowledge of positive affordances enables the consideration of previously unknown state transitions, these results indicate that the computed plans better exploit domain knowledge for achieving the assigned goals. We also conducted trials in which we included or removed all the learnable axioms collectively. These trials resulted in a difference of 19% (serve) or 58% (label) in the plans found. We verified that all the plans computed after including the target axioms were minimal and correct, i.e., of the best quality possible. These results thus indicate that the learning of axioms eliminates incorrect or suboptimal plans, and helps compute plans that would have not been considered in the absence of the learned axioms.

In the paired trials that included or excluded the causal laws extracted from the verbal cues, there was no measurable difference in the number of plans found. This is expected; a causal law for $serve$ produces outcomes which impact the applicability of other actions, and similarly for $label$. This will be the case for any scenario in which the plan produced does not repeat the action influenced by the causal law. In alternative runs that involved planning for a random goal, we observed that the presence or absence of causal laws had an impact on the number of plans found.

Our evaluation also included other findings. For instance, in our experiments, we found that using the ASP-based inference to guide learning makes the learning significantly more efficient. We also observed that RL with the relational representation significantly speeds up the learning in comparison with not using the relational representation. Finally, we introduced a percentage chance per action to encounter actuator noise during learning to test the system's robustness, and found a steady decline in accuracy. For instance, the recall and validated precision were 0.89 and 0.95

respectively without any noise. The values of recall and validated precision dropped to 0.69 and 0.53 respectively at 10% noise, and dropped further to 0.56 and 0.34 respectively at 20% noise. Similar to the results obtained in the absence of noise, most false positives were merely overly-specific variations of the correct axioms.

4.3 Uncertainty, Scaling, and Computational Effort

Recall that uncertainty in perception (i.e., symbol or observation) has been abstracted away in the architecture described above. This simplification helped us to better focus on the interplay between representation, reasoning, and learning, but there is uncertainty in both perception and actuation when robots are used in dynamic, partially observable domains. However, it is easy to model and address perceptual uncertainty in our architecture. As stated in Section 3, the underlying knowledge representation and reasoning architecture already supports non-monotonic logical reasoning and probabilistic reasoning (Sridharan et al., 2017a). We can introduce probabilistic models of the uncertainty in perception. Introducing such models will change the mathematical formulation of the scheme for learning from repeated trials and delayed rewards (as in Section 3.2) from an MDP to a POMDP. Although such a formulation will significantly increase the computational effort involved in learning domain knowledge, we hypothesize that learning will remain tractable with our architecture because our approach includes the use of ASP-based reasoning to automatically limit learning to the relevant parts of the domain. We leave further exploration of this idea as a potential direction for future research.

Next, we consider how our architecture will scale to more complex domains, i.e., domains with many more objects and more complex relationships between objects. ASP-based reasoning has already been shown to scale well to large domains with many objects and axioms (Erdem et al., 2016). The other key component of our architecture consists of two schemes for learning of domain knowledge (see Section 3.2). The first scheme based on processing human verbal input takes very little computational effort; this scheme can be used for real-time processing of verbal inputs. In the second scheme based on RRL, each episode of learning is not computationally expensive, but it may take multiple episodes of RL before the values (of state action combinations) converge and valid axioms can be generated, especially if the knowledge being acquired is a complex relationship between different attributes. The actual learning time can thus vary significantly depending on the domain. However, our RRL approach provides an elegant means for acquiring labeled training samples associated with a measure of relative merit, in the presence of delayed action effects. In addition, our RRL approach incorporates a notion of relevance and exploits the underlying relational representation, e.g., by using ASP-based reasoning to guide learning. Experimental results also indicate that our RRL approach shows promise for scaling to domains with many objects and more complex relationships between objects. We leave further exploration of the scalability of our learning approach as a direction for future research.

5. Conclusions

This paper described an architecture for representing, reasoning with, and interactively learning actions' names, preconditions, effects, and the objects over which these actions operate, along with associated affordances. We used Answer Set Prolog, a declarative language, to represent incomplete domain knowledge, and to perform non-monotonic logical reasoning with this knowledge for planning and diagnostics. We also demonstrated the use of ASP-based reasoning to guide the interactive learning of actions and axioms. This learning was achieved using decision-tree induction and relational reinforcement learning from observations obtained through active exploration, reactive action execution, and verbal descriptions from humans. Experimental results in the context of a simulated robot assisting humans in an indoor domain indicate that our architecture supports: (i) reliable and efficient reasoning; and (ii) learning of actions and axioms corresponding to different types of knowledge. Additionally, inclusion of the learned actions and axioms in the system description improves the quality of the plans computed for any given goal.

In the future, we will explore the learning of actions and axioms in more complex domains and evaluate the architecture on physical robots, which will require the further development and integration of an existing component that reasons about perceptual uncertainty 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

This work was supported in part by the Asian Office of Aerospace Research and Development Award FA2386-16-1-4071, and Award N00014-17-1-2434 from the US Office of Naval Research. All opinions and conclusions described in this paper are those of the authors.

References

Balduccini, M. (2007). Learning action descriptions with A-Prolog: Action language C. *Proceedings of the 2007 AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*. Palo Alto, CA: AAAI Press.

Balduccini, M., & Gelfond, M. (2003). Logic programs with consistency-restoring rules. *Proceedings of the 2003 AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning* (pp. 9–18). Palo Alto, CA: AAAI Press.

Bloch, M. K., & Laird, J. E. (2017). Deciding to specialize and respecialize a value function for Relational Reinforcement Learning. *Proceedings of the Third Multi-disciplinary Conference on Reinforcement Learning and Decision Making*. Ann Arbor, MI.

Chai, J. Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., & Xu, G. (2018). Language to action: Towards interactive task learning with physical agents. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. Stockholm, Sweden: IJCAI.

Driessens, K., & Ramon, J. (2003). Relational instance-based regression for relational reinforcement learning. *Proceedings of the Twentieth International Conference on Machine Learning* (pp. 123–130). Washington, DC: AAAI Press.

Erdem, E., Gelfond, M., & Leone, N. (2016). Applications of answer set programming. *AI Magazine*, *37*, 53–68.

Erdem, E., & Patoglu, V. (2012). Applications of action languages in cognitive robotics. In E. Erdem, J. Lee, Y. Lierler, & D. Pearce, (Eds.), *Correct reasoning*. Berlin: Springer-Verlag.

Gelfond, M., & Inclezan, D. (2013). Some properties of system descriptions of $\mathcal{AL}_d$. *Journal of Applied Non-Classical Logics*, *23*, 105–120.

Gil, Y. (1994). Learning by experimentation: Incremental refinement of incomplete planning domains. *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 87–95). New Brunswick, NJ: Morgan Kaufmann.

Guerin, F., Kruger, N., & Kraft, D. (2013). A survey of the ontogeny of tool use: From sensorimotor experience to planning. *IEEE Transactions on Autonomous Mental Development*, *5*, 18–45.

Laird, J. E. (2012). *The Soar cognitive architecture*. The MIT Press.

Laird, J. E., et al. (2017). Interactive task learning. *IEEE Intelligent Systems*, *32*, 6–21.

Langley, P., Sridharan, M., & Meadows, B. (2018). Representation, use, and acquisition of affordances in cognitive systems. *Proceedings of the 2018 AAAI Spring Symposium on Integrating Representation, Reasoning, Learning and Execution for Goal Directed Autonomy*. Stanford, CA: AAAI Press.

Law, M., Russo, A., & Broda, K. (2018). The complexity and generality of learning answer set programs. *Artificial Intelligence*, *259*, 110–146.

Miller, G. A. (1995). Wordnet: A lexical database for English. *Communications of the ACM*, *38*, 39–41.

Otero, R. P. (2003). Induction of the effects of actions by monotonic methods. *Proceedings of the Thirteenth International Conference on Inductive Logic Programming* (pp. 299–310). Szeged, Hungary: Springer.

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.

Sarathy, V., & Scheutz, M. (2016). A logic-based computational framework for inferring cognitive affordances. *IEEE Transactions on Cognitive and Developmental Systems*, *8*, 26–43.

Scheutz, M., Schermerhorn, P., Kramer, J., & Anderson, D. (2007). First steps towards natural human-like HRI. *Autonomous Robots*, *22*, 411–423.

Simon, H. A., & Lea, G. (1974). Problem solving and rule induction: A unified view. In L. W. Gregg, (Ed.), *Knowledge and Cognition*, 15–26. Oxford, UK: Lawrence Eribaum.

Someya, Y. (1998). e_lemma.txt (Version 2 for WordSmith 4).

Sridharan, M., Gelfond, M., Zhang, S., & Wyatt, J. (2017a). *A refinement-based architecture for knowledge representation and reasoning in robotics*. Unpublished manuscript, `http://arxiv.org/abs/1508.03891`.

Sridharan, M., & Meadows, B. (2017). A combined architecture for discovering affordances, causal laws, and executability conditions. *Proceedings of the Fifth International Conference on Advances in Cognitive Systems*. Troy, NY: Cognitive Systems Foundation.

Sridharan, M., Meadows, B., & Gomez, R. (2017b). What can I not do? Towards an architecture for reasoning about and learning affordances. *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling* (pp. 18–23). Pittsburgh, PA: AAAI Press.

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

Tadepalli, P., Givan, R., & Driessens, K. (2004). Relational reinforcement learning: An overview. *Papers from the Relational Reinforcement Learning Workshop at the Twenty-First International Conference on Machine Learning*. Banff, Alberta, Canada: IEEE Press.

Toutanova, K., Klein, D., Manning, C., & Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. *Proceedings of the 2003 International Conference of the North American Chapter of the Association for Computational Linguistics* (pp. 252–259). Edmonton, Canada: ACM.

Zech, P., Haller, S., Lakani, S. R., Ridge, B., Ugur, E., & Piater, J. (2017). Computational models of affordance in robotics: A taxonomy and systematic classification. *Adaptive Behavior*, *25*, 235–271.

Zhang, S., Sridharan, M., & Wyatt, J. (2015). Mixed logical inference and probabilistic planning for robots in unreliable worlds. *IEEE Transactions on Robotics*, *31*, 699–713.