In the Workshop on *Integrated Planning, Acting and Execution* at ICAPS, Delft, The Netherlands, June 25, 2018.

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

Mohan Sridharan
School of Computer Science
University of Birmingham, UK
m.sridharan@bham.ac.uk

Benjamin Meadows
Electrical and Computer Engineering
The University of Auckland, NZ
bmea011@aucklanduni.ac.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 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 probabilistic models of uncertainty, and observations from active exploration, reactive action execution, and human (verbal) descriptions. The learned actions and axioms are used for subsequent reasoning. The architecture is evaluated on a simulated robot assisting humans in an indoor domain.

1 Introduction

Consider one or more robots[1] assisting humans in an office or a home, e.g., delivering desired objects or guiding people 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 may be in the kitchen". Domain knowledge may also include some understanding of action preconditions and effects, and action capabilities, i.e., affordances. Human participants will, however, lack the time and expertise to provide comprehensive domain information or elaborate feedback. 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 addressing these open problems; it is based on the following tenets:

- Knowledge elements encode symbolical content about object constants, relations representing 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.
- Affordances are 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 we implement them using the complementary strengths of declarative programming, probabilistic reasoning, and relational learning through induction and reinforcement. In this paper, we focus on the interplay between reasoning and learning, and abstract away some aspects of our overall architecture, e.g., we flatten some levels of the representation and do not describe probabilistic modeling of perceptual uncertainty. Instead, we describe the following key capabilities:

- 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 automatically limits interactive learning to the relevant part of the domain.
- Previously unknown actions' names, preconditions, effects, and objects over which they operate, along with associated affordances, are learned using decision-tree induction and relational reinforcement learning based on observations of active exploration, reactive action execution, and verbal cues from humans.

We evaluate these capabilities in the context of a simulated robot delivering objects to particular people or locations in an indoor domain. We first describe the proposed architecture and algorithm (Section 2), followed by some results of experimental evaluation (Section 3). Then, Section 4 reviews related work, followed by a description of conclusions and future work in Section 5.

2 Proposed Architecture

Figure 1 depicts key components of the overall architecture. Incomplete domain knowledge is encoded in an action language to construct tightly-coupled relational representations at two resolutions. 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 by a partially observable Markov decision process (POMDP) that reasons probabilistically over the relevant part of the fine-resolution representation, with action outcomes and observations updating the coarse-resolution history. As stated

[1]We use "robot", "agent", and "learner" interchangeably.

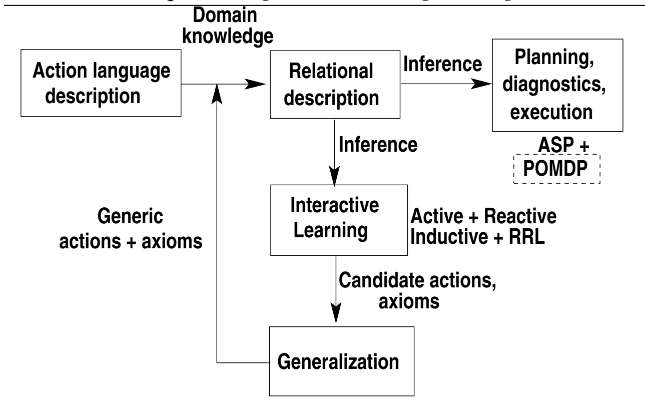

Figure 1: Architecture combines complementary strengths of declarative programming, probabilistic reasoning, and interactive learning for reasoning with and learning domain knowledge.

earlier, we abstract away the reasoning at different resolutions and the probabilistic modeling of perceptual uncertainty, and 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 obtained through active exploration, reactive execution, and human (verbal) descriptions—the learned knowledge is 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 (office, kitchen, library, workshop) in a building. Each place may have one or more 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 $weight$ $(heavy, light)$, $surface$ $(brittle, hard)$, $status$ $(intact, damaged)$, 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 about some actions or axioms (i.e., rules) 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.
- An object with a brittle surface cannot be labeled unless the robot has an electromagnetic arm.

There may be other robots that (are assumed, for simplicity, 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 knowledge of previously unknown actions and axioms. Although this domain may appear simplistic, it 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.

2.1 Knowledge Representation and Reasoning

We first describe the action language encoding of 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. We use action language $\mathcal{AL}_d$ (Gelfond and Inclezan 2013) to describe the transition diagrams at different resolutions. $\mathcal{AL}_d$ has a sorted signature with *statics*, *fluents* and *actions*. Statics are domain attributes whose truth values cannot be changed by actions, whereas fluents are domain attributes whose truth values can be changed by actions. Fluents can be *basic* or *defined*. Basic fluents obey the laws of inertia and can be changed by actions. Defined fluents do not obey the laws of inertia and are not changed directly by actions—their values depend on other fluents. Actions are defined as a set of elementary operations. A domain attribute *p* or its negation $\neg p$ 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 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 $place$, $robot$, $role$, $book$, $weight$, $status$ etc, which are arranged hierarchically, and sort $step$ for temporal reasoning. Σ includes ground instances of sorts, e.g., $\{office, workshop, kitchen, library\}$ 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 include $arm\_type(robot, type)$, $obj\_status(object, status)$ etc, and actions include $move(robot, place)$, $pickup(robot, object)$, and $serve(robot, object, person)$. The representation also includes a relation $holds(fluent, step)$ that implies a particular fluent is true at a particular timestep.

Axioms of the RA domain include causal laws, state constraints and executability conditions such as:

$move(rob_1, L)$ **causes** $loc(rob_1, L)$

$serve(rob_1, O, P)$ **causes** $in\_hand(P, O)$

$loc(O, L)$ **if** $loc(rob_1, L),\ in\_hand(rob_1, O)$

impossible $pickup(rob_1, O)$ **if** $loc(rob_1, L_1),$
$loc(O, L_2), L1 \neq L2$

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. 2017), e.g., "books are usually in the library and if not there, they are normally in the office" is encoded as:

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

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 executability conditions that prevent the use of the corresponding action during planning. In $\mathcal{AL}_d$, we represent affordances in a distributed manner, as follows:

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

The first two statements say that action A cannot occur if it is not afforded, and specify the conditions (i.e., attributes of robot and object) under which the action is not afforded. The last two statements say that an action A that is not considered during planning due to an executability condition may have a positive affordance as an exception, and define the positive affordance. Each action can have multiple affordances indexed by the *id*s. This representation of knowledge improves generalization, and can simplify inference.

ASP-based inference The domain representation is translated into a program $\Pi(\mathcal{D}, \mathcal{H})$ in CR-Prolog[2], a variant of ASP that incorporates consistency restoring (CR) rules (Balduccini and 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*, "$not\ a$" only implies *a is not believed to be true*, and unlike "$p \lor \neg p$" in propositional logic, "$p\ or\ \neg p$" is not tautologous. ASP can represent recursive definitions 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. 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. Each *answer set* of an ASP program represents the set of beliefs of an agent associated with the program. Algorithms for computing the entailment, and for planning and diagnostics, reduce these tasks to computing answer sets of CR-Prolog programs.

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

Reasoning with incomplete or incorrect knowledge may overlook valid plans, find suboptimal plans, or provide plans whose execution has unintended outcomes. For instance, the robot in the RA domain is asked to deliver textbook $book_1$ to the $office$. It uses default knowledge to compute the plan $move(rob_1, library)$, $pickup(rob_1, book_1)$, $move(rob_1, office)$, $putdown(rob_1, book_1)$. This does not succeed because (unknown to the robot) its electromagnetic arm cannot pick up the heavy book. We next describe the interactive learning of such unknown knowledge.

2.2 Interactive Learning

Obtaining labeled samples to learn previously unknown actions, and axioms is difficult in complex domains, and humans may have limited time and expertise. Also, the effects of actions may be observed immediately or after a delay. We thus enable the robot to interactively acquire labeled examples. To speed up learning and to simulate learning without running many trials on a robot, we introduce two schemes: (i) active learning from verbal cues provided by humans; (ii) relational reinforcement learning based on observations from active exploration or reactive action execution. We describe these schemes below.

Learning from Human Interaction To acquire domain knowledge from the verbal cues provided by humans describing the observed behaviors of other robots, the learner makes the following assumptions:

- Other robots have the same capabilities as the learner;
- Learner can generate logic statements corresponding to attributes of robot(s) or object(s) in the observed action;
- Humans correctly describe one activity at a time.

These assumptions are reasonable for many robotics domains, and simplify 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 $labeled(book_1)$. 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 <word, POS> 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., $< is, VBZ > < labeling, VBG >$ 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 $book(book_1)$ and $obj\_weight(book_1, heavy)$ 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.

The robot constructs a literal for the action from the verb and the object referents, e.g., $label(rob_1, book_1)$. The arguments' lowest-level sorts are assumed to be the valid arguments, e.g., $label(\#robot, \#book)$. 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:

$$label(rob_1, book_1) \textbf{ causes } labeled(book_1)$$

is lifted to:

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

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 to $label(\#robot, \#object)$. 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 $R_f : S \times A \times S' \rightarrow \Re$. 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. 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. 2017) describes 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 and Barto 1998). In each episode, the agent executes a sequence of actions chosen using an ϵ-greedy algorithm and 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. 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, significantly reducing 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. 2017).

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

Constants from $relCon(T)$ are said to be *relevant* to T, e.g., for 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, the relevant object constants include rob_1, cup_1, $person_1$, and $office$.

Definition 2. *[Relevant system description]*
The system description relevant to the desired transition $T =$

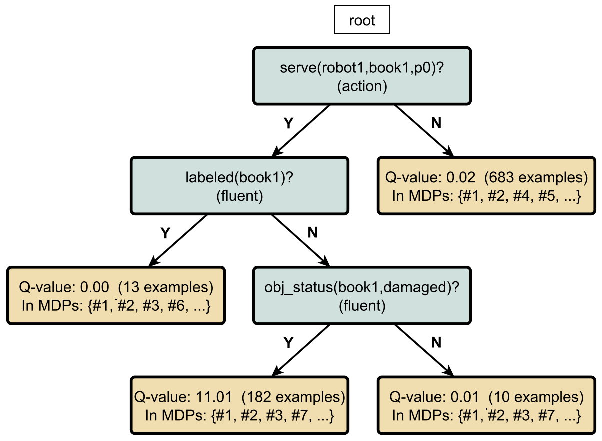
root
serve(robot1,book1,p0)?
(action)
Y
N
labeled(book1)?
(fluent)
Q-value: 0.02 (683 examples)
In MDPs: {#1, #2, #4, #5, ...}
Y
N
Q-value: 0.00 (13 examples)
In MDPs: {#1, #2, #3, #6, ...}
obj_status(book1,damaged)?
(fluent)
Y
N
Q-value: 11.01 (182 examples)
In MDPs: {#1, #2, #3, #7, ...}
Q-value: 0.01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

$\langle \sigma_1, a_{tg}, \sigma_2 \rangle$, i.e., $\mathcal{D}(T)$, is defined by signature $\Sigma(T)$ and axioms. $\Sigma(T)$ 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 formed of 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, it is possible to pre-compute or reuse some of the information used to construct $\mathcal{D}(T)$ for any given T.

Once the $\mathcal{D}(T)$ relevant to the target transition 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 $\mathcal{D}(T)$ 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, 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 and 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 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 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 this RRL approach as "Q-RRL".

Control Loop Algorithm 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.

3 Experimental Setup and Results

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

- **H1**: Active learning of actions from verbal descriptions provides a foundation for further learning;
- **H2**: Q-RRL provides a mechanism for discovering axioms related to an action;
- **H3**: Learned knowledge improves plan quality.

Algorithm 1: Overall control loop.

```
Input: $\Pi(\mathcal{D}, \mathcal{H})$; goal description; initial state $\sigma_1$.
Output: Control signals for robot to execute.
   /* Start with planning */
1  planMode = true, learnType = 0
2  while true do
3      Add observations to history.
4      ComputeAnswerSets($\Pi(\mathcal{D}, \mathcal{H})$)
5      if planMode then
6          if existsGoal then
               /* Goal exists, consistent
                  model, execute plan */
7              if explainedObs then
8                  ExecutePlanStep()
9              else
                   /* Q-RRL */
10                 planMode = false
11                 learnType = 1
12             end
13         else
               /* Active learning */
14             planMode = false
15             learnType = 2
16         end
17     else
           /* Interrupt learning if needed
              */
18         if interrupt then
19             planMode = true
           /* Continue learning */
20         else if learnType == 1 then
21             ContinueRRL()
22         else if learnType == 2 then
23             if verbalCue then
24                 ContinueActiveLearn()
25             else
26                 ContinueActiveRRL()
27             end
28     end
29 end
```

These hypotheses were evaluated in the RA domain (Example 1) in the context of two actions ($serve$ and $label$). We considered the following target axioms to be discovered by the robot, for the action $serve$:

(1) Serving an object to a salesperson causes it to be labelled (*causal law*);

(2) A damaged object cannot be served to a person who is not an engineer (*executability condition*);

(3) A robot with a pneumatic arm cannot serve a brittle object (*negative affordance*); and

(4) A damaged object cannot be served to a person who is not an engineer, unless it is labeled (*positive affordance*).

and for the action $label$:

(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*).

We provide execution traces in support of hypothesis $H1$; hypotheses H2 and H3 are evaluated quantitatively.

3.1 Experimental Setup

The initial setup included experimentally setting the values of some 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. 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.

In the experimental trials reported below, the robot learned the representation for each action and 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. Unless stated otherwise, each value of the performance measures reported below was averaged over 1000 repetitions (e.g., for each axiom). We used *precision* and *recall* as the performance measures. 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. Plan quality was measured as the ability to compute a minimal plan to achieve the desired goal.

3.2 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 the RA domain (Example 1) does not know that it can $label$ and $serve$ objects, and does not know the related axioms. For each of the actions, we gave the agent five descriptive examples of the action being applied, with descriptions that were grammatically-correct English statements that upheld our assumptions, but otherwise varied arbitrarily. First consider the $label$ action:

- The learner receives "A robot is labeling the lightweight cup", with the observation $labeled(cup_1)$. It parses the statement, matches it to the domain, lifts it to store $label(\#robot, \#cup)$, and infers:

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

- Next, the learner receives "Robot labeled one computer", and $\mathit{labeled}(\mathit{comp}_1)$. It learns the signature $\mathit{label}(\#\mathit{robot}, \#\mathit{computer})$ and generalizes over the learned signatures to obtain $\mathit{label}(\#\mathit{robot}, \#\mathit{object})$.
- 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 "A robot serves a manual to the manager" and the observation $\mathit{in\_hand}(p_1, \mathit{book}_1)$. It produces the action description $\mathit{serve}(\#\mathit{robot}, \#\mathit{book}, \#\mathit{person})$ 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 $\mathit{in\_hand}(p_0, \mathit{cup}_1)$. Generalizing over the two examples results in $\mathit{serve}(\#\mathit{robot}, \#\mathit{object}, \#\mathit{person})$. 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.

It is important to appreciate the benefits of the distributed representation used in the architecture. 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 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 of objects or robots, 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 be used to provide more meaningful explanations of decisions.

3.3 Quantitative Evaluation

We experimentally evaluated hypotheses H2 and H3.

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

Table 1: Accuracy when Q-RRL was used to discover multiple axioms corresponding to two specific actions. High recall and precision are attained, especially after candidate axioms are validated.

H2: Q-RRL enables reliable discovery of axioms We explored whether Q-RRL can learn new axioms related to a known (or newly learned) action. Results averaged over the four axioms for each action are summarized in Table 1. We observe that Q-RRL attains high recall and precision, especially after the candidate axioms are validated. The accuracy of discovering the axioms corresponding to the serve action is a little lower than that for the label action, as it is more complex, i.e., it has more arguments. 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 executability condition to make the corresponding action applicable. Note that human input is not essential for this learning—a robot could learn from experiences accumulated over time.

H3: Learning improves plan quality To evaluate hypothesis H3, we explored the effects of the discovered axioms on the system's ability to generate plans that provide the desired 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. The trials used randomized scenarios in which the target action was required to achieve the goal.

We found that adding the learned executability conditions or negative affordances resulted in 13% (serve) or 23% (label) fewer plans found. Adding the positive affordances resulted in 17% (serve) or 23% (label) more plans. These results are expected, as executability conditions and negative affordances preclude actions in some contexts, and knowledge of positive affordances serves to enable particular transitions. We performed additional trials which added or removed all the learnable axioms collectively, resulting in a difference of 19% (serve) or 58% (label) in the plans found. Furthermore, we verified that all the plans that were computed after including the target axioms were correct.

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. Given alternative runs that involve 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, e.g., 0.89 recall and 0.95 validated precision without noise, 0.69 recall and 0.53 validated precision at 10% noise, or 0.56 recall and 0.34 validated precision at 20% noise. Most false positives were merely overly-specific variations of correct axioms.

4 Related Work

Agents 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). It is also difficult, with such approaches, to perform non-monotonic logical reasoning or merge new, unreliable information with existing beliefs. Research in logics has provided non-monotonic logical reasoning formalisms, e.g., ASP has been used in cognitive robotics (Erdem and Patoglu 2012). Researchers have combined ASP with inductive learning to monotonically learn causal laws (Otero 2003), and expanded the theory of actions to revise system descriptions (Balduccini 2007). Architectures have been developed to reason with hierarchical knowledge in first-order logic and process perceptual information probabilistically (Laird 2008). Many general frameworks have been developed that combine logical and probabilistic reasoning, e.g., Bayesian logic (Milch et al. 2006), first-order relational POMDPs (Juba 2016), and probabilistic extensions to ASP (Lee and Wang 2015). Algorithms 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 tend not to support some 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 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, Kruger, and Kraft 2013), or as emergent, functional and/or contextual properties based on attributes of the domain and the objects (Sarathy and Scheutz 2016). These approaches have used logics, probabilistic reasoning or a combination of both. Unlike these approaches, we build on research in psychology to formulate affordances as joint relations over attributes of one or more agents and objects in the context of specific actions (Langley, Sridharan, and Meadows 2018).

Interactive task learning is a general approach that includes learning concepts from domain observations and human demonstrations or instructions (Kirk, Mininger, and Laird 2016). It has often been posed as an RL problem, and relational RL (RRL) uses relational representations and regression for efficient Q-function generalization (Driessens and Ramon 2003; Tadepalli, Givan, and Driessens 2004). However, interactive relational learning algorithms typically limit generalization to a single planning task at a time, based on different function approximation or learning algorithms (Driessens and Ramon 2003; Bloch and Laird 2017), and do not support the commonsense reasoning capabilities desired in robotics. One exception was our prior work that combined ASP with RRL to discover some domain axioms and conditions under which specific actions cannot be executed (Sridharan, Meadows, and Gomez 2017; Sridharan and Meadows 2017). The architecture described in this paper combines the complementary strengths of declarative programming and relational learning through induction and reinforcement, for reasoning with and interactively revising incomplete domain knowledge.

5 Conclusions

This paper described an architecture for representing, reasoning with, and interactively learning actions' names, preconditions, effects, and objects over which they operate, along with associated affordances. Answer Set Prolog was used to represent and reason with incomplete domain knowledge for planning and diagnostics, and to guide interactive learning. The learning is 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 a simulated domain indicate that our architecture supports reliable and efficient reasoning, and learning of actions and axioms corresponding to different types of knowledge. Inclusion of the learned actions and axioms in the system description improves the quality of the computed plans. 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 use of the 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 the US Office of Naval Research Science of Autonomy award N00014-17-1-2434. All opinions and conclusions described in this paper are those of the authors.

References

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

Balduccini, M. 2007. Learning Action Descriptions with A-Prolog: Action Language C. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*.

Bloch, M. K., and Laird, J. E. 2017. Deciding to Specialize and Respecialize a Value Function for Relational Reinforcement Learning. In *Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*.

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.

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

Guerin, F.; Kruger, N.; and 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.

Juba, B. 2016. Integrated Common Sense Learning and Planning in POMDPs. *Journal of Machine Learning Research* 17(96):1–37.

Kirk, J.; Mininger, A.; and Laird, J. 2016. Learning Task Goals Interactively with Visual Demonstrations. *Biologically Inspired Cognitive Architectures* 18:1–8.

Laird, J. E. 2008. Extending the Soar Cognitive Architecture. In *International Conference on Artificial General Intelligence*.

Langley, P.; Sridharan, M.; and Meadows, B. 2018. Represention, Use, and Acquisition of Affordances in Cognitive Systems. In *AAAI Spring Symposium on Integrating Representation, Reasoning, Learning and Execution for Goal Directed Autonomy*.

Lee, J., and Wang, Y. 2015. A Probabilistic Extension of the Stable Model Semantics. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*.

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.

Miller, G. A. 1995. Wordnet: A lexical database for english. *Communications of the ACM* 38(11):39–41.

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.

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

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

Sridharan, M., and Meadows, B. 2017. A Combined Architecture for Discovering Affordances, Causal Laws, and Executability Conditions. In *International Conference on Advances in Cognitive Systems (ACS)*.

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*.

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*.

Toutanova, K.; Klein, D.; Manning, C.; and Singer, Y. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In *International Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 252–259.

Zech, P.; Haller, S.; Lakani, S. R.; Ridge, B.; Ugur, E.; and Piater, J. 2017. Computational Models of Affordance in Robotics: A Taxonomy and Systematic Classification. *Adaptive Behavior* 25(5):235–271.