A Tale of Many Explanations: Towards An Explanation Generation System for Robots

Mohan Sridharan
Electrical and Computer Engineering
University of Auckland, NZ
m.sridharan@auckland.ac.nz

Ben Meadows
Department of Computer Science
University of Auckland, NZ
bmea011@aucklanduni.ac.nz

Zenon Colaco
Electrical and Computer Engineering
University of Auckland, NZ
zncolaco@gmail.com

ABSTRACT

A fundamental challenge in robotics is to reason with incomplete domain knowledge to explain unexpected observations, and partial descriptions of domain objects and events extracted from sensor observations. Existing explanation generation systems are based on ideas drawn from two broad classes of systems, and do not support all the desired explanation generation capabilities for robots. The objective of this paper is to first compare the explanation generation capabilities of a state of the art system from each of these two classes, using execution scenarios of a robot waiter assisting in a restaurant. Specifically, we investigate KRASP, a system based on the declarative language Answer Set Prolog, which uses an elaborate system description and observations of system behavior to explain unexpected observations and partial descriptions. We also explore UMBRA, an architecture that provides explanations using a weaker system description, a heuristic representation of past experience, and other heuristics for selectively and incrementally searching through relevant ground literals. Based on this study, this paper identifies some key criteria, and provides some recommendations, for developing an explanation generation system for robots that exploits the complementary strengths of the two classes of explanation generation systems.

CCS Concepts

•Computing methodologies → Knowledge representation and reasoning; Cognitive robotics; Reasoning about belief and knowledge; Causal reasoning and diagnostics; *Nonmonotonic, default reasoning and belief revision; Logic programming and answer set programming;*

Keywords

Explanation Generation, Robotics, Answer Set Prolog, Heuristic Guidance, Cognitive Systems

1. INTRODUCTION

Robots[1] equipped with multiple sensors are being used to assist humans in different applications such as disaster rescue and healthcare. These robots receive an incomplete and inaccurate description of the domain based on the information extracted from the sensor data. The robots also receive useful commonsense information (e.g., "books are typically in the library") that holds in all but a few exceptional situations (e.g., cookbooks may be in the kitchen), but it is challenging to represent and reason with this information. To assist humans, robots thus need the ability to represent and reason with different descriptions of domain knowledge and uncertainty. One fundamental challenge for such robots is the generation of explanations for unexpected observations (e.g., of action outcomes) and for partial descriptions (e.g., of the scene) extracted by processing sensor data. Existing explanation generation systems in the artificial intelligence literature draw on ideas that can be broadly categorized into two classes based on how they represent and reason with domain knowledge, and how and to what extent they use heuristic guidance. However, neither class of systems nor approaches that combine ideas from these two classes, support reliable, efficient, incremental and partial explanation generation for robots. The first objective of this paper is to explore and compare the capabilities of a representative approach from each of these two classes of explanation generation systems. Specifically, the systems we compare are:

- KRASP, based on the declarative language Answer Set Prolog (ASP), supports non-monotonic logical reasoning with incomplete knowledge, including default knowledge. KRASP generates explanations based on the system description, observations of system behavior, and minimal use of heuristics.
- UMBRA is a custom engine that uses abductive inference guided by heuristic rules. It does not fully support non-monotonic logical reasoning, but interpolates observations and background information with domain axioms and constraints to incrementally generate explanations.

This comparative study uses execution scenarios of a robot waiter assisting in a restaurant by greeting and seating people at tables, and by delivering orders and clearing tables.

The second objective of this paper is to use this study to identify criteria, and provide recommendations, for developing an explanation generation system for robots that exploits the complementary strengths of the two classes of systems.

2. RELATED WORK

Explanation generation has been formulated in different ways in the robotics and artificial intelligence (AI) research communities. It has been formulated as diagnostics in the logic programming community, while it has been considered as an instance of abductive inference in the broader AI community. Existing approaches can be

[1] We use terms "robot" and "agent" interchangeably in this paper.

SAC 2016, April 04 - 08, 2016, Pisa, Italy

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3739-7/16/04…$15.00

DOI: http://dx.doi.org/10.1145/2851613.2851705

broadly categorized into two classes based on how they represent and reason with domain knowledge, and to what extent they use heuristic guidance. One class uses elaborate system descriptions and observations of system behavior, with minimal heuristics, to explain unexpected occurrences [9, 21]. Approaches in the other class are limited in their ability to represent and reason with system descriptions, and depend more on heuristic representation of experience and intuition to generate explanations [15, 17]. Other approaches have combined ideas from these two classes of systems.

Existing approaches for explanation generation have drawn ideas from different disciplines to represent and reason with domain knowledge. Some approaches have used research in multiagent collaboration towards shared objectives [23]. Other algorithms have built on research in collaborative planning, incorporating other agents' intentions in plans, and modeling mental states [5] or using logic programming to make inferences about the outcomes of plans [18]. Research in activity and plan recognition tends to focus on discerning and reasoning about agents' observed behavior and top-level tasks instead of state-action information [20]. Some approaches have emphasized hierarchical structures, interleaving multiple agents' actions [11] or capitalizing on prior knowledge [1]. More recent work has developed cognitive architectures that use default reasoning to support inference over problem state descriptions [4], while other work has combined probabilistic and first-order logic representations for abductive reasoning [19].

Explanation generation approaches differ in how and to what extent they use heuristic guidance. For instance, approaches based on ASP have been limited to using heuristics to prioritize rules for explaining observations. Recent research has removed the need to solve ASP programs entirely anew when the problem specification changes, allowing (a) new information to expand existing programs; and (b) reuse of ground rules and conflict information to support interactive theory exploration [8]. However, these approaches cannot generate partial explanations incrementally, and efficient operation poses the challenging problem of designing better heuristics. Approaches that have developed sophisticated heuristics to reason with domain knowledge, on the other hand, have focused on finding low-cost proofs to explain observations [13], and on developing search strategies based on criteria such as parsimony and coherence [17]. These approaches tend not to support the desired capability of reasoning with commonsense knowledge.

Existing explanation generation approaches present some common challenges such as scaling, e.g., many require an exhaustive search of the space of ground literals [20], and reasoning with different (e.g., symbolic and probabilistic) descriptions of knowledge and uncertainty. Existing approaches also do not support all the desired capabilities such as efficient and incremental addition of knowledge, generating partial explanations with incomplete information, default reasoning, and use of heuristics for optimized interactive explanation generation. As a step towards developing a system that supports these capabilities, this paper compares the capabilities of two state of the art systems, and provides recommendations for the design of an explanation generation system for robots that exploits the complementary strengths of the two classes of systems.

3. SYSTEM DESCRIPTIONS

This section describes and compares the capabilities of two representative explanation generation systems. First, Section 3.1 describes the use of KRASP to represent incomplete domain knowledge and use observations of system behavior to generate explanations. Next, Section 3.2 describes the UMBRA architecture, which uses heuristic abductive inference for generating explanations.

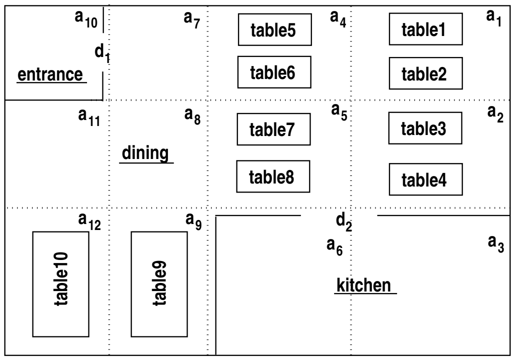

Figure 1: Map of illustrative domain used for discussion, with rooms, doors, and tables—the a_is denote areas in rooms. The humans and the robot are not shown for simplicity. © *Mohan Sridharan*

Illustrative Domain: Robot Waiter

As an illustrative example used throughout the paper, consider a robot waiter that seats people at tables in a restaurant, and delivers orders. Figure 1 provides an example map of this domain; humans and the robot waiter can be in different areas of this map. The sorts of the domain are arranged hierarchically, e.g., `location` and `thing` are subsorts of `entity`; `animate` and `inanimate` are subsorts of `thing`; `person` and `robot` are subsorts of `animate`; `object` is a subsort of `inanimate`; and `room`, `area`, `door`, and `floor` are subsorts of `location`. We consider specific rooms, e.g., `kitchen` and `dining`, and consider objects of sorts such as `table`, `chair` and `plate`, to be characterized by attributes such as `size`, `color`, `shape`, and `location`. The sort `step` is included for temporal reasoning. If, for instance, the robot waiter believes that `person1` is unattended and `table1` is unoccupied at a specific time step, but then observes `person1` at `table1` at a subsequent time step, it should infer that a waiter seated `person1` at `table1` at some point in between.

3.1 System 1: KRASP

KRASP is based on ASP, a declarative language that can represent recursive definitions, defaults, causal relations, special forms of self-reference, and language constructs that occur frequently in non-mathematical domains, and are difficult to express in classical logic formalisms. ASP is based on the stable model (answer set) semantics of logic programs and research in non-monotonic logics [9, 10]. ASP can draw conclusions from a lack of evidence to the contrary, using concepts such as *default negation* (negation by failure) and *epistemic disjunction*. For instance, unlike "$\neg a$", which implies that "*a is believed to be false*", "*not a*" only implies that "*a is not believed to be true*"; and unlike "$p \vee \neg p$" in propositional logic, "p *or* $\neg p$" is not a tautology. ASP supports non-monotonic reasoning, i.e., adding a statement can reduce the set of inferred consequences, which is a desired capability for robots. ASP also supports reasoning in large knowledge bases [24], and reasoning with quantifiers. These capabilities have led to the use of ASP architectures in robotics by an international community [22, 27]. In the remainder of this paper, we use the terms KRASP and ASP interchangeably—wherever appropriate, we highlight the differences between KRASP and a standard ASP formulation.

Knowledge Representation: KRASP's domain representation consists of a system description $\mathscr{D}$ and a history with defaults $\mathscr{H}$. $\mathscr{D}$

has (a) a sorted signature that defines the names of objects, functions, and predicates available for use; and (b) axioms that describe the transition diagram τ. To obtain $\mathscr{D}$, the syntax, semantics and representation of τ are typically described in an *action language*. For instance, action language AL has a sorted signature containing *statics*, *fluents* and *actions* [9]. Statics are domain properties whose truth values cannot be changed by actions, fluents are properties whose values are changed by actions, and actions may be executed in parallel. AL supports *causal laws*, which describe the effects of actions when certain conditions are met, *state constraints*, and *executability conditions*, i.e., conditions under which one or more actions cannot be executed—the system description $\mathscr{D}$ is thus a collection of AL statements.

Domain fluents are defined in terms of the sorts of their arguments, e.g., $has\_location(thing, location)$. There are two types of fluents. *Basic* fluents obey laws of inertia and are directly changed by actions. *Defined* fluents, on the other hand, do not obey inertia laws and cannot be changed directly by actions—they are described in terms of other relations. The system description includes relations that define statics such as connections between rooms. In addition, relation $holds(fluent, step)$ implies that a specific fluent holds at a time step, and $occurs(action, step)$ is used to reason about a specific action occurring at a specific time step. Each action is defined in terms of the sorts of its arguments, e.g., $move(robot, location)$ and $pickup(robot, object)$, and domain dynamics are defined by the causal laws, state constraints and executability conditions.

Domain history $\mathscr{H}$ has records of the form $hpd(action, step)$ and $obs(fluent, boolean, step)$, which specify the occurrence of specific actions and the observation of specific fluents (respectively) at specific time steps. Unlike a standard ASP-based formulation, $\mathscr{H}$ (in KRASP) also contains prioritized defaults describing the values of fluents in their initial states, and exceptions (if any) [26]. For instance, it may be initially believed that plates are typically on a table between the kitchen and the dining room—if not there, the plates are typically on a table in the kitchen. Dirty plates are an exception, and are stacked on a different table.

The domain representation is translated into a program $\Pi(\mathscr{D}, \mathscr{H})$ in CR-Prolog, which incorporates consistency-restoring (CR) rules in ASP [3, 9]. Π includes the causal laws of $\mathscr{D}$, inertia axioms[2], closed world assumption for actions and defined fluents, reality checks, and records of observations, actions and defaults from $\mathscr{H}$. Every default is turned into an ASP rule and a CR rule that lets us assume the default's conclusion is false to restore Π's consistency.

Planning and Diagnosis: Given the CR-Prolog program Π, the ground literals in an *answer set* obtained by solving Π represent the beliefs of an agent associated with Π. Statements that hold in all such answer sets are program consequences, i.e., the result of inference using the available knowledge. It has been shown that planning can be reduced to computing answer sets of program Π by adding a goal, a constraint stating that the goal must be achieved, and a rule generating possible future actions [9]. An ASP-based architecture can also be used to explain unexpected observations [2] by reasoning about exogenous actions and partial descriptions. For instance, to reason about a door between the kitchen and the dining room being locked by a human, and to reason about a person moving away from a known location, we can introduce $locked(door)$ and $moved\_from(person, location)$ as exogenous actions, and add (or revise) axioms appropriately. The expected (i.e., ideal) observations of the values of attributes (e.g., color, shape and parts) of domain objects can also be encoded in the CR-Prolog program Π. To generate explanations, we introduce an *explanation generation* rule, *awareness* axioms for actions, and *reality check* axioms that cause inconsistencies when observations do not match expectations. Consistency can be restored by using the explanation generation rule to provide *all* explanations, or by defining a partial ordering over sets of CR-rules to generate the *minimal* explanation. For any unexpected observation, we consider candidate explanations as sets of CR rules that can be triggered, and pick the set with lowest cardinality as the *preferred* or minimal explanation [6].

Computing Answer Sets: Algorithms for computing answer sets of a logic program are *generate and test* reasoning algorithms. In their first step, they replace the program Π with its ground instantiation. Existing methods for grounding are sophisticated, e.g., they use algorithms from deductive databases, but grounding a program containing variables over large domains is still computationally expensive. The second step recursively computes the consequences of the grounded program and its partial interpretations. The algorithm considered in this paper takes as input the domain knowledge specified as a CR-Prolog program, including the sequence of observations[3]. This program is solved using disjunctive logic programming and a satisfiability solver [14]. The output (i.e., answer set) includes the minimal explanation for the observations[4].

Although not included in this study, it is possible to model heuristics or numerical costs in an ASP program. Recent answer set solvers improve computational efficiency through partial grounding and heuristic ordering of variables and rules, allow new information to expand existing programs, reuse ground rules, and support interactive query answering [8]. However, grounding and computing all consequences of a program is still expensive for complex domains, and better heuristics are needed for optimized explanation generation. The design of such systems can be informed by the insights gained from the study of systems that rely on heuristic guidance for incremental explanation generation.

3.2 System 2: UMBRA

UMBRA is a cognitive architecture that leverages abductive inference to generate explanations. Previous work has discussed its performance on cognitive tasks involving dialog [12], planning [15], and social understanding [16]. This section describes UMBRA's semantic representation and the processes that operate over it.

Working Memory: UMBRA has a *working memory* that stores information arriving from the environment, e.g., statements about an agent's behavior, and the system's inferences. This memory contains mental states, i.e., beliefs regarding factual statements, and goals comprising views about what the system intends to achieve. These are stated as logical literals with propositional structure. Each working memory element specifies (a) the agent A holding the belief or goal; (b) its content C; (c) start time for the mental state, denoting when A first entertained C; and (d) end time encoding when A stopped believing or abandoned its intention towards C. Unbound variables (implemented as skolem constants) in the arguments related to time denote unknown times.

For literals that are true over a temporal interval, the relevant times may be unknown. UMBRA has a third form of mental state predicate for such literals, *constraint*, which specifies relations such as inequalities or temporal orderings, e.g., *during(skol1, time1, skol2, skol3)*. Working memory elements may refer to domain-level literals, e.g., *at-location(person1, area1, time2)*, but may also be em-

[2]A basic fluent retains its truth value between steps unless there is evidence to the contrary.

[3]For simplicity, in the case studies in this paper, all observations are given concurrently and analyzed incrementally as required.

[4]It also provides a minimal set of actions for a planning task.

bedded. A mental state may thus refer to another mental state rather than a domain predicate, e.g., *belief(robot1, goal(waiter1, seat(person1, table1, skol5), time3, skol6), time4, skol4)* implies that robot1 believes from time 4 to time *skol4* that waiter1 has the goal (from time 3 to time *skol6*) to seat person1 at table1 at time skol5. Such embeddings could in principle be arbitrarily deep, but studies suggest that people cannot readily use more than the fourth-order mental state nesting that UMBRA models [7].

Axiom Representation: UMBRA also has a *long-term memory* containing conceptual rules and skills that encode generalized knowledge about states and actions. A *conceptual rule* is equivalent to a Horn clause associating a predicate in the head with a relational situation described in the body. A *skill* associates a predicate in the head with a set of preconditions, postconditions, invariants, constraints, subtasks, and operators. The two types of axioms are not always distinguished during processing. Higher-level predicates are defined in terms of lower-level ones, so that long-term memory is organized similarly to a hierarchical task network or a concept grammar. Axioms often include logical constraints that specify orderings on times associated with antecedents or assert the non-equivalence of two elements. Much of an agent's axiomatic knowledge deals with its goals and beliefs about the environment. However, as social axioms include goal-directed evaluation and alteration of others' mental models [16], this knowledge may also involve embedded mental states. It is difficult to encode prioritized defaults (and exceptions to defaults) in such a representation, due to (a) its binary truth values; (b) focus on structural knowledge; and (c) emphasis on assumption as a process over content encoded within axiomatic knowledge.

Explanation: UMBRA is implemented as a layer above SWI-Prolog [25]. It builds on Prolog's support for relational logic and embedded structures to perform directed abductive inference governed by heuristic rules, and to make plausible assumptions instead of strict derivations. The system receives a sequence of observations as inputs and uses them to construct an explanation. Since robots receive observations sequentially, the explanation mechanism operates in an incremental manner[5], building on earlier inferences by chaining from observations and heads of rules, rather than Prolog-style top-down chaining from queries. It is not necessary to repeatedly reconstruct explanations or perform extensive belief revision—the explanation at any given time may include fewer or more than one top-level activity. The domain axioms, when bound to elements in the working memory, explain observed events. Since memory elements may be used in multiple rule applications, an explanation is a proof lattice of a directed acyclic graph.

Processing: UMBRA operates over a series of *input* cycles in which it receives and adds input observations to working memory as beliefs (with start times equivalent to the current time), then enacts one or more low-level *inference* cycles. Each of these stages involves a rule application that adds beliefs to working memory.

In an inference cycle, UMBRA first identifies each piece of axiomatic knowledge (rule) R with at least one component C that can unify with some element E in working memory. For each such R-E pair, it generates the partially instantiated head H by the unification of C with E. The system creates a set of candidate rule instances for each partially instantiated rule head H, unifying as many rule components with working memory elements as possible, and making the minimum number of default assumptions to complete R. Should R's components include unbound variables, UMBRA inserts skolem constants into component literals, e.g., to denote unknown times that satisfy a temporal relation in a constraint. When these variables are instantiated, any rule instance with a literal that is inconsistent with working memory is eliminated.

[5]As before, observations are given concurrently in the case studies, and analyzed incrementally, for simplicity.

UMBRA uses heuristic rules to determine which domain axioms to apply, and whether further search should continue or a new cycle should begin. The set of candidate rule instances is pruned using heuristic rules to retain instances that can (a) incorporate currently unexplained elements; (b) improve explanation by incorporating multiple rule elements that are otherwise disparate; or (c) be applied deductively without assumptions. These rule instances I are ranked by cost, using an evaluation function that considers the proportion and number of I's antecedents that the system needs to assume, and the proportion of candidate explanation elements not yet explained by other rule instances. The explanation is then extended by adding inferences from the lowest-cost rule instance to the working memory.

The input cycle ends if the rule instance exceeds a limit on the total number of assumptions per cycle. Since each cycle incrementally extends the explanation to account for the observations, the output is an account of the input in terms of the instantiated axioms. In producing this explanation, candidate literals are generated in a local manner that amounts to a heuristic search with rules for a one-step look-ahead. UMBRA is thus not guaranteed to find a minimal explanation or to refrain from unnecessary inferences. Since explanations are expanded by extending working memory monotonically, UMBRA does not fully support non-monotonic reasoning.

In the performance tasks discussed in Section 4, we use a standard parameterization of UMBRA, with assumption limit of 2, and runs limited to 12 input cycles [15]. Since UMBRA's ability to reason over nested expressions has been explored extensively, our test cases usually involve only direct beliefs about the world.

4. DISCUSSION OF SCENARIOS

In this section, we compare the capabilities of the two systems described above, using scenarios from the robot waiter domain in Section 3. As stated earlier, the robot waiter assists in a restaurant by greeting and seating people at tables, and by delivering orders and clearing tables.

4.1 Basic Approach to Explanation

Scenario-1 presents a basic anomaly: at time 2, $robot1$ believes $person1$ is unattended, and that $table1$ is unoccupied in Figure 1. Then at time 3, $robot1$ observes that $person1$ is at $table1$.

Inference in the KRASP program containing these statements and the system description identifies an inconsistency because the expected observations, obtained by propagating initial beliefs with inertia axioms, do not match actual observations. Consistency is restored by invoking a CR rule that provides a minimal explanation: $expl(waiter\_seated\_person(person1, table1), 2)$, i.e., an exogenous action that a waiter seated $person1$ at $table1$ at time 2.

UMBRA is able to combine its beliefs with the observations provided, and apply the conceptual rule for exogenous actions, needing only to assume $waiter\_seated\_person$ actually occurred. Since this incorporates all working memory elements, no more inputs are given and no further reasoning is attempted.

Assume that $robot1$ has prior ($default$) knowledge that unattended people usually wait near the entrance (in $area10$ in Figure 1), and knows that $person1$ is unattended initially (at time 0). If $robot1$ is asked to seat $person1$ at time 2, a plan generated by inference in KRASP will have the robot go to $area10$ because it continues to believe that $person1$ is still next to the entrance. Now, when $person1$ is observed at $table1$ at time 3, one possible explanation is that *person*1 directly walked to *table*1 to meet friends, and is thus an exception to the initial state default. The non-monotonic reasoning capability of KRASP enables the robot to draw such conclusions, and to build a different model of history to revise the result of previous inferences. UMBRA does not fully support such default reasoning, and cannot perform non-monotonic reasoning.

This scenario illustrates that efficiency, accuracy (including both good precision and good recall), and the ability to revise previous beliefs, are key criteria for explanation generation.

4.2 Multiple Concurrent Goals

Although considerable work in explanation generation has focused on identifying a top-level goal, a robot in social scenarios must maintain beliefs about agents who may follow separate plans. We next consider scenarios with tasks performed concurrently.

Scenario-2 extends Scenario-1 with statements: *robot*1 searches for *person*1 in *area*2 at time 1, then greets *person*1 (at *table*1) at time 3. As before, inference in KRASP invoked a CR rule to infer: $expl(waiter\_seated\_person(person1, table1), 2)$, i.e., *person*1 was seated at *table*1 by a waiter at time 2, and reasoned to account for other observations. UMBRA also produced the correct "search" and "greet person" rule instances but, in the following cycles, preferred conceptual rules that it was able to prove cheaply, e.g., to explain why *table*1 was considered "occupied". It also inferred variations of rule instances with different time constraints, primarily due to a lack of effective ways to implement axiomatic constraints as opposed to persistent constraints in working memory. This phenomenon was also seen in some other scenarios. When run for additional cycles, UMBRA ran out of cheap conceptual rules and correctly inferred *waiter_seated_person*.

Scenario-3 is a variation of Scenario-1 that features a second, unrelated anomaly. In addition to the statements in Scenario-1, at time 2, *robot*1 believes that *dish*1 is empty and located at unoccupied *table*1 in *area*1 in Figure 1. At the same time, *robot*1 travels from the kitchen door to *table*1 with the goal of clearing *dish*1. At time 3, the robot observes *person*1 at *table*1, and observes that *dish*1 is not at *table*1. These observations make sense if a waiter cleared *dish*1 *and* seated *person*1 at *table*1. Computing the answer set of the KRASP program restores consistency by invoking exogenous actions: $expl(waiter\_seated\_person(person1, table1), 2)$ and $expl(waiter\_cleared(dish1), 2)$, i.e., the correct explanations are generated. UMBRA also made most of the correct inferences, but was again distracted by cheaper conceptual rule applications instead of inferring *waiter_seated_person*. Reducing the preference for cheaper rules allowed UMBRA to make the right inferences.

These scenarios help identify that leveraging knowledge effectively to explain multiple simultaneous behaviors or events, is a key criterion for explanation generation systems.

4.3 Strategies for Explanation Selection

Explanation generation searches for a model of the world that is as similar as possible to the ground truth. Some approaches, such as UMBRA, guide search with heuristic measures of similarity such as minimality and coherence. Others, such as KRASP, search the space of known objects and axioms. As the number of literals and (grounded) instances increase, inference in ASP becomes computationally expensive. Even with (recent) ASP-based systems that support interactive exploration, generating incremental and partial explanations is challenging. Scaling to complex domains is difficult for UMBRA too, but heuristic measures improve tractability.

A strategy for belief maintenance is necessary when the system receives input dynamically (because observations may contradict assumptions), or if it is incremental and non-complete (due to local optima). Typical approaches for belief maintenance re-run the process from scratch, or maintain multiple competing hypotheses. Since KRASP performs a more elaborate search, standard usage would re-plan from scratch for incremental inputs. UMBRA, on the other hand, is non-rigorous, heuristic, and incremental. It delays deciding in the face of ambiguity, seeking the lowest-hanging fruit, and committing when its parameterization allows; after that, it can only rebuild explanations from scratch.

In Scenario-4, *robot*1 believes at time 0 that *person*1 is located in *area*4 in Figure 1, and travels from the *kitchen* to *area*4. At time 1, *robot*1 observes that *person*1 is not in *area*4. At time 0 and time 1, *robot*1 does *not* know whether *person*1 is unattended or seated at a table, making it difficult to arrive at a single explanation—*person*1 may not have been in *area*4, or may have been in *area*4 and moved away. Inference in the KRASP program provides two explanations, corresponding to two possible worlds that trigger the same number of rules: $expl(person\_moved\_away(person1, area4), 0)$ or $expl(false\_request(person1), 0)$. Unlike UMBRA, KRASP does not include variable heuristic costs of relaxing different rules. The first explanation implies that *person*1 was not really waiting to be seated to begin with, and the second explanation implies that *person*1 has moved away from the previous location. The dynamics of these exogenous actions need to be modeled explicitly in KRASP before these actions can be used to generate explanations. UMBRA, on the other hand, works analogously to local greedy search with one-step look-ahead and monotonic commitments. It cannot generate and directly compare different multiple-step explanations. For this scenario, UMBRA explained the absence of *person*1 by assuming that *person*1 had been there, but left the building. Based on UMBRA's heuristics, this assumption resulted in as cheap and coherent an explanation as the alternatives.

This scenario suggests that an explanation system for robots should have a formal model of explanation generation, and be flexible enough to decide when and how to maintain competing hypotheses, or to revise baseline beliefs.

4.4 Including Different Assumptions

Providing a meaningful explanation frequently involves making assumptions about different aspects of the world, e.g., the observations may include false negatives. Scenario-5 states that at time 1, *robot*1 is in the *kitchen*, and believes *person*1 is seated at *table*1 in *area*1 (Figure 1) and has ordered *pasta*. At time 1, *robot*1 picks up a dish believing it to contain *pasta*. At time 2, *robot*1 observes its own location to be *area*1. At time 3, *robot*1 observes that *person*1 is not at *table*1. The explanation, that *person*1 has moved and left the restaurant, requires the system to reason about (a) an exogenous event; (b) a domain object (a dish containing *pasta*); and (c) a movement action (i.e., robot moving from *kitchen* to *area*1).

KRASP can only reason about actions that have been modeled, and cannot reason about object instances that have not been specified. Inference in the KRASP program for this scenario provides: $expl(person\_moved\_away(person1, area1), 2)$ to explain the unexpected absence of *person*1 at *table*1. If, in addition to the *move* action, the KRASP program includes action *robot_moved_to* to reason about the robot having moved to a specific location, inference *also* generates $expl(robot\_moved\_to(robot1, area1), 2)$ as an explanation. However, there is no further reasoning about the dish that has been picked up.

UMBRA may assume any literal, but this ability causes it to sometimes make unnecessary or irrelevant assumptions. Being incremental and boundless, UMBRA is constrained only by cycle limits, its cost function, and sparsity of inputs. For Scenario-5, UMBRA successfully inferred that *robot*1 traveled from the *kitchen* to $area1$, identified the abnormality, and explained that $person1$ left the building. However, rather than assuming a new instance of a *dish* and its various properties, the low(er) cost option was to assume that $robot1$ was a *dish*! This scenario is another instance where integrated constraint-like axioms, especially if incorporated into concept rule definition, would improve representational power and give more accurate results.

4.5 Explanations with Inconsistent Inputs

So far we have discussed input omissions, but observations received by the robot may include false positives and false negatives. Scenario-6 includes such irrelevant, incorrect literals. At time 1, $robot1$ believes $person1$ has ordered *pasta*, believes $dish1$ contains *pasta*, observes $dish0$ is empty, and travels from *kitchen* to $table1$ in $area1$ of Figure 1. Once $robot1$ is in $area1$ at time 2, it observes $person1$ seated at $table1$, $dish1$ is at $table1$ and contains *noodles*, and $person2$ is seated at $table2$. Possible explanations are that the wrong dish (*noodles*) has been delivered to $person1$, or that the initial belief about $dish1$ was incorrect.

Inference in the KRASP program only explains the state of $dish1$; nothing else needs explanation. The corresponding explanation $expl(mixed\_up(dish1, noodles), 1)$ invokes an exogenous action to imply that $dish1$ has been mixed up and the wrong dish has been delivered to $person1$ instead of *pasta*. The other possibility is not considered because the domain knowledge does not include any measure of uncertainty for the initial belief about the state of $dish1$. Interestingly, if the number of time steps is increased between the initial belief and final observation of $dish1$, KRASP hypothesizes that the dish got mixed up at different points in this interval.

For UMBRA, this scenario was one of the least accurate cases, and the explanation mostly consisted of applying cheap conceptual rules. UMBRA also assumed that $robot1$ was a *person* that had left the restaurant, i.e., the most crucial anomaly went unexplained. It is often difficult to establish why a heuristic-directed reasoning process ends up making an error on a particular case, but explanation based on non-complete search can only be as good as the heuristic it uses for inference choice. Furthermore, since UMBRA aims to produce reasonable assumptions given reliable partial inputs, rather than create a maximally consistent explanation given unreliable inputs, it cannot readily redact beliefs.

A robot may encounter inputs that are more problematic than the type of false positive considered above. Consider Scenario-7, which states that: at time 2, $robot1$ believes that $table1$ is unoccupied, $dish1$ is located on $table1$ in $area1$, and $dish1$ is empty. The robot moves to $area1$ to clear $dish1$. At time 3, $robot1$ observes that there is no $dish1$ on $table1$. Inference in KRASP generates the explanation: $expl(waiter\_cleared(dish1), 2)$, i.e., that a waiter cleared $dish1$ at time 2. For this scenario, UMBRA is able to generate the correct explanation as well.

Now, consider two variants of Scenario-7 that feature unexpected inputs. In the first variant, new literals are added to describe a previously unknown attribute of $robot1$ or a previously unknown relationship between $dish1$ and $table1$. KRASP cannot deal with such new additions that have not been defined previously. UMBRA, on the other hand, succeeds in ignoring the foreign literals until an opportunity comes up to incorporate those observations. The second variant involves contradictory observations, e.g., at time 3, $dish0$ is observed to have state *pasta*, and $dish0$ is observed to have state *noodles*. Inference in the KRASP program cannot deal with this inconsistency and no solutions are provided. UMBRA, on the other hand, constructs the explanation when provided the inputs incrementally; when the contradictory input is received, it fails.

These scenarios highlight some key criteria for explanation generation systems (a) robustness with respect to observed false positives and false negatives; (b) ability to decide what observations to focus on; and (c) graceful failure (when failing is unavoidable).

Table 1: Comparison of KRASP and UMBRA in terms of support for some desired explanatory capabilities.

Capability	KRASP	UMBRA
High accuracy and quantity of inferences, including ability to explain concurrent behaviors/events	**Yes**	**Partial**
Leverage rule types (e.g., axiomatic constraints, default rules, HTNs, concept taxonomies) in appropriate ways	**Yes**	**Partial**
Can apply and maintain constraints	**Yes**	**Partial**
Flexible approach to ambiguity and belief maintenance	**Partial**	**No**
Different strategies for belief revision and maintain competing hypotheses	**Partial**	**No**
Formal guarantees about inferences	**Yes**	**No**
Search is scalable, and uses heuristic rules to focus on those elements with higher demand for explanation	**No**	**Partial**
Incorporate sensor data incrementally, reduce search cost through heuristics	**No**	**Yes**
Explain false positive and partial descriptions	**Yes**	**Partial**

4.6 Summary

Table 1 summarizes some desired capabilities of an explanation generation system for robots, identified based on the scenarios discussed above. Table 1 also documents the extent to which the explanation generation systems KRASP and UMBRA, as described in Section 3.1 and Section 3.2 respectively, support these desired capabilities. We observe that the two systems largely complement each other in their ability to support these capabilities.

The differences in the capabilities of the two systems are primarily due to what kind of information and knowledge they can represent, how they generate explanations, i.e., process information, and how they handle different forms of uncertainty. KRASP generates explanations using the system description and observations of system behavior. This approach is formally well-defined and provides satisfying guarantees, besides supporting appealing capabilities such as non-monotonic reasoning and default reasoning. However, it is challenging to include new relations or objects, and the system cannot reason with observation inconsistencies or produce partial explanations incrementally. This approach can also become computationally expensive if inference is not done selectively as the number of relations and grounded instances increases.

UMBRA's incremental, heuristic approach to explanation generation, on the other hand, is robust to the unexpected observations of objects, and allows guidance through heuristic rules. However, this approach is susceptible to local minima, and may face difficulties in reasoning in a targeted manner. It is also not capable of non-monotonic reasoning, and does not provide formal guarantees for the conclusions drawn. The ideal explanation generation system for robots would include the best features of these systems, e.g., it would support (a) an elegant representation for incomplete domain knowledge; (b) non-monotonic (and default) reasoning; and (c) scalable, incremental, and partial explanation generation through (well-designed) heuristic guidance.

5. CONCLUSIONS AND FUTURE WORK

Existing explanation generation systems are based on ideas drawn from two broad classes of systems, and do not support all the desired explanation generation capabilities for robots. The objective of this paper was twofold. First, we investigated a representative state of the art example from each of the two broad classes of explanation generation systems. Specifically, we compared (a) KRASP, a system based on Answer Set Prolog, which uses an elaborate system description, observations of system behavior, and minimal heuristics, to generate explanations; and (b) UMBRA, a system that generates explanations using a weaker system description, and abductive inference guided by a heuristic representation of past experience and other heuristics for selective and incremental search. We used execution scenarios of a robot waiter assisting in a restaurant to highlight the strengths and limitations of these systems. Second, we used this study to identify important criteria for explanation generation systems for robots. We have also confirmed that the two systems pose some common challenges, such as scaling and reasoning with different descriptions of uncertainty. Exploiting the complementary strengths of the two systems may provide solutions to these challenges, and this paper opens up many directions for further research.

First, we will focus on computational efficiency, scalable search, failure recovery, and improved guidance by knowledge, in UMBRA. This will include a better theory of explanation generation, using lessons learned from KRASP to support reasoning with default knowledge and a selective hierarchy of elements to explain. It will also employ better cost functions for search, and an improved approach to determine which hypotheses to maintain and when to prune the hypotheses.

Second, we will explore the design of heuristics that can improve the scalability of KRASP for complex domains. We will build on lessons learned from the study of heuristics used by UMBRA, while still retaining KRASP's ability to reason with commonsense knowledge. Such a system will operate over the same representations as KRASP, and incorporate fine-grained heuristic control for scalable, incremental generation of explanations.

Third, we will investigate the inclusion of a probabilistic description of knowledge and uncertainty with non-monotonic logical reasoning. We will build on recent work that uses ASP and probabilistic models with robots [6, 26, 27]. Such a system would extend the representation used by KRASP to consider different possible worlds, but use heuristics to generate explanations that are highly probable. The long-term objective is to provide a reliable, efficient and well-defined approach for explanation generation, as a fundamental capability for human-robot collaboration.

6. ACKNOWLEDGMENTS

The authors thank Pat Langley and Michael Gelfond for discussions related to the two systems discussed in this paper. This work was supported in part by the US Office of Naval Research Science of Autonomy award N00014-13-1-0766. All opinions and conclusions are those of the authors alone.

7. REFERENCES

[1] D. Appelt and M. Pollack. Weighted Abduction for Plan Ascription. *User Modeling and User-Adapted Interaction*, 2:1–25, 1992.

[2] M. Balduccini and M. Gelfond. Diagnostic Reasoning with A-Prolog. *Theory and Practice of Logic Programming*, 3(4-5):425–461, 2003.

[3] M. Balduccini and M. Gelfond. Logic programs with Consistency-Restoring Rules. In *Logical Formalization of Commonsense Reasoning, AAAI SSS*, pages 9–18, 2003.

[4] P. Bello. Shared Representations of Belief and their Effects on Action Selection: A Preliminary Computational Cognitive Model. In *International Conference of the Cognitive Science Society*, pages 2997–3002, Boston, MA, 2011.

[5] C. Castelfranchi. Modelling Social Action for AI Agents. *Artificial Intelligence*, 103:157–182, 1998.

[6] Z. Colaco and M. Sridharan. What Happened and Why? A Mixed Architecture for Planning and Explanation Generation in Robotics. In *Australasian Conference on Robotics and Automation*, Canberra, Australia, December 2-4, 2015.

[7] R. Dunbar. On the Origin of Human Mind. In *Evolution and the Human Mind: Modularity, Language, and Meta-Cognition*. Cambridge University Press, 2000.

[8] M. Gebser, T. Janhunen, H. Jost, R. Kaminski, and T. Schaub. ASP Solving for Expanding Universes. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, Lexington, USA, 2015.

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

[10] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *International Conference on Logic Programming*, pages 1070–1080, 1988.

[11] R. Goldman, C. Geib, and C. Miller. A New Model of Plan Recognition. In *International Conference on Uncertainty in Artificial Intelligence*, San Francisco, USA, 1999.

[12] P. Langley, B. Meadows, A. Gabaldon, and R. Heald. Abductive Understanding of Dialogues about Joint Activities. *Interaction Studies*, 15(3):426–454, 2014.

[13] D. Leake. Focusing Construction and Selection of Abductive Hypotheses. In *International Joint Conference on Artificial Intelligence*, San Francisco, CA, 1993. Morgan Kaufmann.

[14] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[15] B. Meadows, P. Langley, and M. Emery. Seeing Beyond Shadows: Incremental Abductive Reasoning for Plan Understanding. In *AAAI Plan, Activity, and Intent Recognition Workshop*, pages 24–31, Bellevue, USA, 2013.

[16] B. Meadows, P. Langley, and M. Emery. Understanding Social Interactions Using Incremental Abductive Inference. In *International Conference on Advances in Cognitive Systems*, pages 39–56, Baltimore, USA, 2013.

[17] H. T. Ng and R. Mooney. Abductive Plan Recognition and Diagnosis: A Comprehensive Empirical Evaluation. In *International Conference on Principles of Knowledge Representation and Reasoning*, pages 499–508, 1992.

[18] E. Pontelli, T. C. Son, C. Baral, and G. Gelfond. Answer Set Programming and Planning with Knowledge and World-Altering Actions in Multiple Agent Domains. In *Correct Reasoning*, pages 509–526. Springer, 2012.

[19] S. Raghavan and R. Mooney. Abductive Plan Recognition by Extending Bayesian Logic Programs. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 629–644, Antwerp, Belgium, September 2011.

[20] M. Ramirez and H. Geffner. Probabilistic Plan Recognition

Using Off-the-Shelf Classical Planners. In *AAAI Conference on Artificial Intelligence*, Atlanta, USA, 2010.

[21] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32:57–95, 1987.

[22] Z. Saribatur, E. Erdem, and V. Patoglu. Cognitive Factories with Multiple Teams of Heterogeneous Robots: Hybrid Reasoning for Optimal Feasible Global Plans. In *International Conference on Intelligent Robots and Systems*, pages 2923–2930, Chicago, USA, 2014.

[23] K. Sycara. Multiagent Systems. *AI magazine*, 19(2):79, 1998.

[24] G. Terracina, N. Leone, V. Lio, and C. Panetta. Experimenting with Recursive Queries in Database and Logic Programming Systems. *Theory and Practice of Logic Programming*, 8(2):129–165, 2008.

[25] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12:67–96, 2012.

[26] S. Zhang, M. Sridharan, M. Gelfond, and J. Wyatt. Towards An Architecture for Knowledge Representation and Reasoning in Robotics. In *International Conference on Social Robotics*, pages 400–410, Sydney, Australia, October 27-29, 2014.

[27] S. Zhang, M. Sridharan, and J. Wyatt. Mixed Logical Inference and Probabilistic Planning for Robots in Uncertain Worlds. *IEEE Transactions on Robotics*, 31(3):699–713, June 2015.