Explainable Reasoning for Human-Robot Collaboration

RSMG 2

Oliver Michael Kamperis
School of Computer Science
University of Birmingham
United Kingdom
oxk312@student.bham.ac.uk
Supervisor: Dr. Mohan Sridharan

April 29, 2019

ABSTRACT

Robots are becoming increasingly common-place and capable of performing ever more complex tasks in human society. However, the use of robots to collaborate with humans in complex dynamic domains such as healthcare and hospitality still poses many fundamental open problems.

This thesis focuses specifically on the problem of enabling a robot to explain its knowledge, beliefs and decision making processes to humans. This ability, called *Explainable Agency* [Langley et al. 2017], requires four distinct functional abilities; (1) explain decisions it made during plan generation, (2) report which actions it executed, (3) explain how actual events diverged from a plan and how it adapted in response, (4) communicate its decisions and reasons to humans at desired level of abstraction.

This thesis will significantly expand upon the capabilities of existing explanation generations systems for robotics applications. It will create an architecture for explainable reasoning and learning for human-robot collaboration which fully supports all of the desired functional abilities of an explainable agent. This report describes the work done so far and planned future work towards this architecture. The current implementation of the architecture is ASP based and makes use of both Clingo [Gebser et al. 2014] and SPARC [Balai et al. 2013a] in an attempt to combine the complimentary strengths of both languages.

Contents

1 Introduction

Autonomous agents such as *Intelligent Autonomous Robots* (IR) and self-driving vehicles are becoming increasingly prevalent in human society. Development of such agents is now receiving tremendous amounts of attention from a widespread international research community. *Artificial Intelligence* (AI) and *Intelligent Systems* (IS) have now found their way into almost every aspect of our everyday lives and our dependency upon such systems is growing rapidly in the modern world. However, as these systems gain in intelligence and autonomy to become capable of performing ever more complex and important tasks, it becomes increasingly critical that they be able to interact and collaborate effectively with humans [Sridharan and Meadows 2019].

Trust has been identified as a key requirement for humans to allow robots to operate without constant and close supervision [Simon et al. 2019]. However, before robots will be trusted by humans, they must be capable of; clarifying every single aspect of their decision making-processes, justifying their actions, and clearly describing their functional capabilities in terms that humans find intuitive [Langley 2019, Sheh 2017b]. Furthermore, if humans are to collaborate effectively with robots, humans will demand at least the same level of transparency and clarity of communications from the robot as they would from another human [Miller 2019]. If the robot unexpectedly fails to complete a task it will also likely be put under significantly greater scrutiny than the equivalent human under the same circumstances [Kim and Hinds 2006]. Such issues thus require that robots be fully capable of explaining their knowledge, beliefs and decisions on demand and at the appropriate level of abstraction.

This ability, also called *Explainable Agency*, has the potential to make the behaviour of robots more transparent, but still remains a fundamental open problem in robotics [Langley et al. 2017, Langley 2016]. Realising explainable agency requires a robot have the following four high-level functional abilities:

1. Explain decisions it made during plan generation, including alternative choices and selection criteria
2. Report actions executed whilst following the plan, at the desired level of abstraction and verbosity
3. Explain how actual events diverged from a plan and how it adapted in response
4. Communicate its decisions and reasoning in ways that make contact with human concepts

It is challenging to achieve this objective because such a robot will be equipped with many different descriptions of knowledge and uncertainty. The robot usually has access to rich commonsense domain knowledge that holds in all but a few exceptional circumstances, such as "books are usually in the library, but cookbooks may be in the kitchen". The robot can also extract information from its sensors, and the associated uncertainty is often modelled probabilistically, such as "I am 90% certain I saw the robotics book on the desk". Additionally, humans often do not have the time or expertise to provide elaborate feedback or comprehensive domain knowledge to robots. In fact humans are rarely capable of explaining little more than a very coarse description of the dynamics of a domain [Keil 2003]. Furthermore, robots may evaluate many choices using many different performance criteria during plan generation and will acquire new knowledge during plan execution which may complement or contradict its existing beliefs. Other issues arise when a robot is given a task which violates its own constraints or may lead to an undesirable outcome. Under such circumstances a robot may reject or alter goals given by human participants [Dannenhauer et al. 2018].

Producing a robot capable of explainable reasoning and learning for human-robot collaboration will therefore require us to enable the robot to represent, reason with, and revise different descriptions of incomplete domain knowledge and probabilistic representations of uncertainty at different levels of abstraction.

There has been significant research on human-robot interaction, but these emphasise joint activity in pursuit of a common goal [Langley et al. 2017]. This thesis focuses on the more challenging scenario of when the robot receives initial instructions, carries out a plan with little to no human interaction or supervision and then must explain itself in retrospect of completing, or possibly failing to complete its task. This poses significant, as yet unsolved, design challenges regarding the way in which the robot internally stores and structures its knowledge and memory of past events, and how it extracts relevant explanations from these [Sridharan and Meadows 2019, Fox et al. 2017, Meadows et al. 2016].

Recent literature has identified the key elements of explainable agency and essential functional abilities of an explainable agent. Such studies have sought to guide researchers to creating explainable robotic systems. Yet to date, there has been no such system that is capable of fulfilling all, or even most, of these requirements. There has also been significant work that has identified the nature of explanations to be generated and how to produce relevant, context specific explanations that humans find satisfying and intuitive. Yet again, little work has taken seriously the need to ensure that explanations are relevant, intuitive and actually make contact with human concepts. Furthermore, many studies have stated the importance of rigorously evaluating the effectiveness of the proposed explanation generation system to ensure explanations can be properly understood by human collaborators. However, most practitioners only demonstrate that their systems can produce *some* explanation, rather than evaluating the system based on the judgements of others.

1.1 Motivations

Human-robot collaboration in complex dynamic domains such as healthcare, hospitality, retail, industry and disaster rescue poses many as yet unsolved problems. There has been substantial work on enabling robots to plan and act reliably in large, open, partially observable and stochastic domains. However, for robots to be deployed to any advantage in human society, they must also be capable of *explainable agency*. This still remains a fundamental open problem in robotics and thus forms the primary motivation of this thesis.

The major advantages of explainable agency for robotics applications are summarised well by [Sheh 2017b], as:

- *Trust* - Humans tend to trust systems that they believe they understand.
- *Accountability* - When things go wrong people will demand to know if robots are to blame.
- *Ease of Debugging* - It is easier to discover and fix failures when decision making process are transparent.
- *Training* - Explanations can help to teach users to perform a similar task or control a robot to achieve this task.
- *Discrepancy Resolution* - When a human and a robot disagree, the robot should be able to either convince the user it is correct or give the user enough information to correct the error.

Many current approaches to acquiring expertise, such as Deep Learning, provide autonomous agents with powerful functional abilities. Unfortunately, these systems produce opaque models whose behaviour is difficult to interpret and cannot be explained directly [Sheh 2017a] and despite recent work on explaining the predictions of any classifier [Ribeiro et al. 2016] this still remains an open problem in AI. This problem has not gone unnoticed by the AI community and has lead to the emergence of Explainable Artificial Intelligence (XAI)[1] [XAI 2017]. The lack of transparency of current approaches is seen as one of the most major impediments to the advancement of AI and IS and as such XAI is now receiving world-wide attention from researchers [Adadi and Berrada 2018, Došilović et al. 2018]. This has lead to major investments from official bodies, such as the Defense Advanced Research Projects Agency (DARPA) launching the XAI program in the USA attempting to encourage increased interest in the field [DARPA 2016].

However, most recent work on explainability, including DARPA's XAI program, focuses primarily on predicting the content of machine learning models. This work is mostly concerned with providing practitioners with tools to evaluate the performance and behaviour of trained models. Crucially, such tools do not support the type of interactive and on demand explanations required for human-robot collaborative tasks, and are thus not sufficient for this purpose. Explainable agency has in contrast received significantly less attention from practitioners both in research and industry.

Furthermore, current approaches to XAI fundamentally lack in their ability to explain the decision making processes of a model in human understandable terms [Doran et al. 2017]. It is critical that there be no ambiguity about a model's decision making processes. Leaving this open to interpretation allows different conclusions to be drawn about why a decision was made, depending upon the human collaborator's knowledge of the capabilities of the model.

It therefore seems clear that current XAI systems are fundamentally insufficient for supporting explainable agency for human-robot collaboration. To tackle this problem significantly more work is required. The next sub-sections outline the reasons why explainable agency is so crucially important for human-robot collaboration in the real-world.

1.1.1 Capacity for Explainable Agency

The capacity for explainable agency for human-robot collaboration in the modern world has been described as "not an academic exercise" [Langley et al. 2017], implying the need for it is obvious. Yet despite the fact that nearly all areas of AI, including those relevant to robotics applications, now have mature technology to offer, only very recently has significant work be done on producing explainable robotic systems.

When an event occurs which is unexpected or not immediately understandable, it is quite normal for a human to elicit an explanation as to why it happened. For explainable robotics this issue presents itself distinctly when a robot has to adapt its plans as events diverge from its expectations, either as a result of; unpredicted action or sensor failure, because assumptions it made about the state of the domain under uncertainty were incorrect, or some extreme circumstance occured which caused the robot to take drastic measures to avoid some very undesirable outcome. Robots may also often make wrong decisions, disagree with a human collaborators or do things that a human normally would not do [Fox et al. 2017]. Humans will thusly not trust robots to do any useful or important tasks unless they are able to explain themselves and justify their decisions when such events occur. Furthermore, what constitutes a satisfactory explanation to unexpected robot behaviour will vary greatly depending on the context and who is posing the query [Sheh 2017b].

[1] A system which exhibits *Explainable Artificial Intelligence* does not necessarily exhibit *Explainable Agency*.

The following examples illustrate such situations. A self-driving car is transporting a passenger to work along a familiar route. The car, in response to information telling it that the road ahead is blocked, takes an unusual turn. In response, its passenger may inquire as to why it did so, and will most likely be satisfied with the simple response of "The road ahead was blocked, so I went around to save time". However, in contrast consider, an autonomous military robot performing maneuvers in a volatile war zone. The military robot determines that the chance of successfully completing its primary objective is extremely low. Reasoning that the risk-reward trade-off is not worth while, it abandons the primary objective, to instead pursue a secondary objective of lower priority. In this case, the robot's commander will certainly demand highly detailed justifications for the robot's decision to abandon the primary objective against orders.

1.1.2 Legal Pressure for Explainable Agency

Recently concern has grown amongst governments, such as the USA, EU and China, that artificially intelligent autonomous systems may make unfair or discriminatory decisions, replicate or develop biases, and behave in unexpected ways in highly sensitive environments that can potentially put human safety at risk [Wachter et al. 2017a]. Unfortunately, the diversity and complexity of emerging AI and robotic systems significantly complicates the manner of legislating around and regulating their design and use when compared to previous automated systems. As a result of this, significant pressure from official bodies is mounting to make AI and robotics systems transparent and accountable for their actions.

The EU's General Data Protection Regulation (GDPR), and the resolution on Civil Law Rules on Robotics [EU 2017a] seem to be a result of this pressure. The GDPR apparently proposed (in articles 13-15) to mandate a "right to an explanation" about all decision making processes made by AI systems, although the policy lacks an explicit definition of the scope of information to be provided [Wachter et al. 2017b]. The Civil Law Rules on Robotics call for robotic systems to include transparency tools to explain the rationale and logic behind their decision making processes, but these regulations are equally poorly defined [EU 2017a]. The EU parliament's committee on Civil Liberties, Justice, and Home Affairs have also called for future civil laws addressing robotics to comply with the GDPR, although state that the "right to an explanation" only applies to systems "based solely on automated processing" [EU 2017b].

1.2 Structure of the Report

The rest of this report is structured as follows. First in section 2 background work regarding relevant techniques for Knowledge Representation and Reasoning (KRR) in robotics are reviewed. Seeking to identify areas of interest and lessons learnt from past work, along any key limitations and fundamental open areas for future work. Section 3 then reviews recent literature regarding the desired capabilities of explainable agents, criteria for generating relevant and context specific explanations which humans find intuitive, techniques for evaluating the performance of an explainable agent, and current Explainable Robotic Systems (XRS). Section 4 then outlines the goals of this thesis, summarising the limitations of past work, fundamental open areas for research in the field, and the key contributions of this thesis. Next sections 5 and 6 describe specific areas of interest which this thesis has been particularly exploring so far, these are Action Languages, Answer Set Programming, respectively. Section 7 then describes the current implementation of the explainable reasoning and learning architecture being produced by this thesis and some proposed novel theories. In section 8, the current proof-of-concept domain, the Robotic Hotel is described and why is was chosen as a strong domain domain for studying explainable agency, along with a review of some recent literature regarding the domain. Finally in sections 9 and 10 the work done so far and a tentative plan for future work on this thesis is summarised.

2 Knowledge Representation and Reasoning Techniques for Robotics

This section reviews past work on KRR techniques for robotics applications, identifying limitations of past work, fundamental unsolved problems, and promising avenues for future work. This section seeks to understand what KRR techniques are required by robots before latter sections discuss explainable agency. Techniques of particular interest to this thesis are discussed, and then linked to following sections which describe these in significantly more detail.

2.1 Overview of KRR Paradigms

There have been many approaches to reasoning about action effects and state changes in robotics. One of the most popular of which are Action Languages [Gelfond and Lifschitz 1998]. These are defined as; formal models of parts of natural language used for describing transition diagrams of dynamic systems. There exists a vast number of different action languages, such as STRIPS [Fikes and Nilsson 1971], $BC+$ [Babb 2015], and AL_d [Gelfond and Inclezan 2013]. There has also been a rich and successful history of their use in different robotics applications [Erdem and Patoglu 2012]. Section 5 reviews a brief history of actions languages and the basic syntax and semantics of AL_d [Sridharan et al. 2015].

There has been significant research in classical and logic programming based techniques for planning sensing and actuation on mobile robots. These have provided many algorithms for KRR in robotics applications. Logic based paradigms typically require significant prior knowledge of the domain and its dynamics, the agent's capabilities, and the preconditions and effects of its actions. Furthermore, logic based paradigms, by themselves, do not support probabilistic representations of uncertainty and attaching probabilities to logic statements is rarely a sufficient compromise. This is an important issue because a large proportion of information available to robots is represented probabilistically in order to quantitatively model the uncertainty in sensing and actuation. Many of these algorithms have used first-order logic based formalisms that do not support non-monotonic reasoning, default reasoning, or the ability to revise existing beliefs based on new unreliable information gained through sensor measurements. Non-monotonic logic paradigms such as Answer Set Prolog (ASP) address these short comings to support more elegant representation of, and reasoning with, commonsense domain knowledge [Baral 2003, Gelfond and Kahl 2014, Lifschitz 2008]. Recent ASP solvers have achieved great computational speeds due to their highly parallel [Gebser et al. 2012b] and distributed processing capabilities [Gebser et al. 2011a]. They also support incremental [Gebser et al. 2008] and multi-shot solving techniques [Gebser et al. 2019c], which allows programs to be solved efficiently even when the planning horizon is unknown or may change often. They are also capable of partial solving and grounding techniques [Gebser et al. 2015b], which removes the need to completely re-solve programs when the program changes due to revision of existing, or integration of new, knowledge. Despite these efforts to make ASP based formulations more efficient to solve, they still become computationally intractable for large domains when they are used as the only tool for representing every feature of the domain [Sridharan and Gelfond 2016]. Section 6 reviews a history of ASP systems, their use in robotics, and the syntax and semantics of two ASP systems of interest, Clingo [Gebser et al. 2014] and SPARC [Balai et al. 2013a].

Probabilistic reasoning paradigms, such as Probabilistic Sequential Decision Making Algorithms and Probabilistic Graphical Models of Uncertainty, have proved to be powerful techniques for maximising long term rewards during planning of sensing and actuation in partially observable stochastic domains by formally modelling the non-determinism of state transitions and sensor measurements [Kaelbling et al. 1998]. These include algorithms for recursive state estimation, localisation, mapping, planning and control. Particular techniques of interest include, Particle Filters, Partially Observable Markov Decision Processes (POMDPs), and Bayesian and Gaussian Networks [Thrun et al. 2005, Koller and Friedman 2009]. However, whilst probabilistic techniques such as POMDPS provide a powerful balance between accuracy and computational efficiency, they become computationally expensive when dealing with large domains and knowledge bases, even when employing state-of-art approximate solvers [Shani et al. 2013]. Unfortunately, probabilistic based KRR techniques, by themselves, do not support default knowledge and commonsense reasoning. Default knowledge is important because it provides the ability for a robot to make assumptions where it lacks complete explicit information of the domain state. The key idea is that commonsense reasoning can reduce the set of consistent possible worlds to only the most likely worlds, thus significantly reducing the complexity of planning. Reasoning efficiently with incomplete domain knowledge still however remains a fundamental challenge in robotics.

Researchers have thusly identified the distinct need to combine both logic and probabilistic based reasoning techniques, to support Integrated commonsense Reasoning and Probabilistic planning (IRP), for robotics systems. However, elegant and efficient coupling of logic and probabilistic reasoning remains a fundamental open problem in robotics [Zhang et al. 2014]. As such many techniques have been developed in attempts to solve this problem. Examples of these include Markov Logic Networks [Richardson and Domingos 2006], Bayesian Logic [Andersen and Hooker 1994], Independent Choice Logic [Poole 2000], Probabilistic First-Order Logic [Jaumard et al. 2006], First-Order Relational POMDPs [Sanner and Kersting 2010], and P-Log the probabilistic extension of ASP [Balai et al. 2019]. In the next sub-section some work towards developing mixed logical and probabilistic reasoning systems for robots are reviewed.

2.2 Combined Logical and Probabilistic KRR

Recent work has combined P-Log with POMDPs, as applied to dialog with a mobile robot, in an attempt to combine commonsense reasoning and probabilistic planning techniques [Zhang and Stone 2015]. The idea is that commonsense reasoning can directly output a probability distribution which can be used by a probabilistic planner. The technique involves using P-Log to generate informative priors, by eliminating state variables that affect the priors but are irrelevant to the optimal policy, for the POMDP planning solver. Their resulting system, called CORPP, showed significant improvements in both efficiency and accuracy of the resulting POMDP policy with this technique. However, this technique does not take full advantage of ASP and ignores CR-Prolog, an extension of ASP with consistency restoring rules [Balduccini and Gelfond 2003]. It is therefore not capable of indirect exceptions to defaults and cannot fully deal with resolving a complete and consistent initial domain state based on the robot's observations. Similar work has developed a framework for HRI implemented in OpenDial, which combined P-Log with MLNs based on the concept of probabilistic rules which define mappings between logical conditions and probabilistic effects [Lison 2015].

More recent work has expanded on CORPP to produce the Interleaved Commonsense Reasoning and Probabilistic Planning system iCORPP [Zhang et al. 2017a, Zhang and Stone 2017]. This system decomposes a planning problem into two sub-problems, commonsense reasoning and probabilistic planning. Which they describe as focusing on; "understanding the world" and "accomplishing the task", respectively. This technique reduces the state space in which probabilistic planning is preformed, thus increasing efficiency of the POMDP solver, and allows the robot to reason about exogenous changes of domain attributes. Exogenous changes are changes that the robot did not make itself.

Other work has developed an architecture that reasons with tightly coupled transition diagrams of any given domain at different resolutions. Fine resolution descriptions are defined as a refinement of the coarse resolution description of the domain [Zhang et al. 2015, Sridharan and Gelfond 2016]. The action language AL_d is extended, to allow non-deterministic causal laws (amoungst other additions), and encoded into ASP at the coarse resolution. This allows non-monotonic logical reasoning, which includes default knowledge, to provide a high level sequence of abstract actions which deterministically guide the robot's planning. Each such abstract action is then implemented as a sequence of concrete actions by zooming to and reasoning probabilistically, using a POMDP, over just the relevant part of the fine resolution description. The system allows the high level ASP reasoning system to conduct complex reasoning tasks, such as explaining observations that contradict beliefs by hypothesising the occurrence of exogenous actions and generating indirect exceptions to initial state defaults. Such reasoning is impossible for probabilistic planners to preform. The notion of refinement and zooming also substantially reduces the complexity of the POMDP planner in a similar manner to the work reviewed above. This is because the POMDP policy only concerns itself with the specific parts of the domain description relevant to each abstract action. A similar technique, removes the POMDP planner used for executing each abstract action, and instead sensor observations perform incremental Bayesian updates to a probability distribution over just the subset of the domain relevant to each particular action [Colaco and Sridharan 2015].

A mobile robot planning architecture has been developed which combines first-order logic and probabilistic reasoning for open world planning which can; deal with uncertain and incomplete domain knowledge, and handle and explain task failure intelligently [Hanheide et al. 2017]. The robots' knowledge was divided into layers based on three distinct types, instance, default and diagnostic, where higher level knowledge can modify lower level knowledge. The architecture represents its information flow via a three layer system (independent of the knowledge layers), these are the; competence, belief and deliberative layers. They present a powerful system for assumptive planning by assigning probabilities to them to avoid the robot initially making unlikely assumptions. Solving four important robotic planning problems; planning in the face of uncertainty, planning in open worlds, explaining task failure and verifying those explanations.

There has also been work towards an architecture for KRR and learning for human-robot collaboration which has combined non-monotonic logical reasoning (ASP), probabilistic graphical models (POMDPs) and relational reinforcement learning [Sridharan et al. 2015, Sridharan and Meadows 2016, Sridharan and Meadows 2018]. They use a similar technique as above, employing tightly coupled transition diagrams a domain at different resolutions. When incomplete knowledge of domain dynamics results in plan failure, the various resolutions of the domain representation are used to formulate a reinforcement learning problem to incrementally discover any unknown rules governing such dynamics. Similar work also supports interactive learning from verbal cues alongside reinforcement learning [Sridharan 2016]. However, this work does not take into account that plan failure may be a result of an unknown exogenous actions.

Another fundamental problem in robotics is providing the ability for a robot to reason about its goals and intentions [Dannenhauer et al. 2018]. Robots may be given tasks which may violate one or more hard constraints, or lead to very undesirable outcomes based on its performance criteria. A robot with such goal-reasoning capabilities may adopt, abandon, or alter sub-goals which form part of some larger task. Goal adaptation may occur suddenly and part way through a task due to observations or commands from humans [Langley 2019]. Recent work has produced an ASP based framework for goal-reasoning robots which can reason about an agents "mental" state and deal with intentional actions [Rocio Gomez and Riley 2018]. They claim increased reliability and efficiency over previous work.

3 Explainable Robotics Systems

This section begins by reviewing recent literature regarding the desired capabilities of an explainable agent, this more clearly defines what the specification of an explainable agent is, and thus the goals of this thesis. It then outlines various criteria for generating effective explanations and evaluating such explanations. Finally some recent explainable robotics systems are reviewed, analysing their capabilities against those that have been identified in previous sub-sections. Content from this section provides information towards development of a design specification for the architecture which is to be produced, which will serve as a guidance for the design and evaluation of the architecture throughout this thesis.

3.1 Capabilities of an Explainable Agent

The desired capabilities of an explainable agent have been given by [Langley et al. 2017], as:

1. Given a *complex set of objectives* that require an agent's extended activity over time;
2. Given *background knowledge* about categories, relations, and activities that are relevant to these objectives;
3. Produce *records of decisions* made during plan generation, execution, and monitoring in pursuit of these aims;
4. Produce *summary reports*, in human accessible terms, of the agent's mental and physical activities;
5. Produce *understandable answers to questions* that are posed about specific choices and the reasons for them.

They maintain that explainable agents must behave well along the four functional abilities outlined in section 1 and have identified three primary elements of explainable agency to guide researchers. These are that explainable agents must:

1. *Represent knowledge in a way that supports explanation*, including terms that describe domain properties, states and actions in terms that humans find intuitive, as well as presenting the choices that were available during planning, selections made by the agent, and criteria used to make its decisions.
2. *Have an episodic memory* that records states and actions considered during planning, justifications for action selection, a record of action execution and state changes, and events that resulted in plan revision.
3. *Be able to access and extract content from episodic memory to answer queries* about their experiences. This requires the ability to interpret a natural language, retrieve relevant structures from memory, and report answers terms that humans find comprehensible.

Recent work [Sridharan and Meadows 2019] has identified the following guiding principles for designing explanation generation systems and generating effective explanations:

1. Explanations should *present context-specific information* relevant to the task, domain and the question under consideration, at an appropriate level of abstraction.
2. Explanations should be able to *describe knowledge, beliefs, actions, goals, decisions, rationale for decisions, and underlying strategies* or models in real-time.
3. Explanation generation systems should *have minimal task or domain-specific components*.
4. Explanation generation systems should *model and use human understanding and feedback to inform its choices* while constructing explanations.
5. Explanation generation systems should *use knowledge elements that support non-monotonic revision* based on immediate or delayed observations obtained from active exploration or reactive action execution.

3.2 Nature of Explanations

There has been significant work from researchers in robotics, AI and the social sciences regarding criteria for generating effective explanations that make contact with human concepts. However, constructing an explanation that answers a query in a manner that human's find intuitive is not a trivial task and still presents a fundamental open problem for human-robot collaboration. This issue is amplified by the difficulty of correctly computing the level of detail an explanation should present, based on the context and who is posing the query.

A major issue arises from the problem that it is generally very easy for a robot to explain far too much in answer to a query [Sheh 2017b]. An intelligent robot may internally store and process vast amounts of knowledge in order to complete a task. A human performing the same task may also process a similar amount of knowledge but is rarely consciously aware of doing so. As a result of this phenomenon, an explainable robot that explains its decisions in the same manner as its low-level internal encoding, will not produce an intuitive explanation.

Humans generally describe things and converse with each other in a high-level abstract manner. Thusly, robots must process its internal knowledge to generate an explanation in a form that makes contact with human conscious thoughts. Even an expert is unlikely to be able to understand a robot that does cannot do this.

Recent work [Sridharan and Meadows 2019] has made progress towards tackling this problem, they propose three fundamental axes upon which to characterise and construct explanations:

1. *Representation abstraction* - This axis models the abstraction level at which an explanation is generated, the robot may use a coarse or fine resolution domain description to construct the explanation, i.e. it may reason about being located generally within a room or its exact location within a room.
2. *Communication specificity* - This axis models what the robot focuses on while communicating with the human, at one end of this axis the explanation will be brief, describing the specific properties of the domain that lead to its decisions, and on the other end of the axis the explanation will be more elaborate, describing the full state of the domain and how each property effects its performance criteria.
3. *Communication verbosity* - This axis models the comprehensiveness of the response provided, for example either; stating the final action a robot took before completing its goal, stating every single action it executed to achieve the goal, or stating all actions along with preconditions and expectations for the effects of each action.

The fundamental idea is that these axes formally model the level of detail which explanations present. An explainable robot should be able to construct an explanation by choosing a suitable position upon these axes, revise this choice based on human feedback, and thus learn what constitutes a relevant explanation to the context.

A recent study [Miller et al. 2017] has indicated that most work on XAI and XRS has only used the researchers' own intuition of what constitutes an effective explanation and that researchers should take seriously the need to exploit the large bodies of research from the social sciences regarding how to generate explanations that properly satisfy the needs of the intended target users. Four major findings from the social sciences regarding explanation generation are described in section 1.2 of [Miller 2019], these are summarized as:

1. *Explanations are contrastive* - They are sought in response to particular counterfactual cases, e.g. people do not just ask why some event P happened, but rather why X happened instead of some other event Y.
2. *Explanations are selected in a biased manner* - People rarely expect an explanation that consists of an actual and complete cause of an event and usually select, based on cognitive bias, one cause from an often infinite number of possible causes to be the explanation.
3. *Probabilities often don't matter* - Referring to probabilities or statistical relationships in explanations is less effective then referring to causal relationships, and the most likely explanation is not necessarily the best explanation from everyone's perspective.
4. *Explanations are social* - They transfer knowledge presented as part of a social conversation or interaction, note that this does not demand that explanations be given in a natural language.

3.3 Evaluation of an Explainable Agent

Explainable agency is motivated by a human desire to understand robots. As such the evaluation of explainable agents must focus primary on the judgements of humans [Langley et al. 2017, Miller et al. 2017]. If explanations are not satisfactory from a human's perspective then the robot is not sufficiently transparent and is thus not fit for purpose.

Furthermore, when robots are deployed into the real world they will collaborate with many different humans with varying levels of understanding about robotics. It is not realistic to expect all collaborators to have the level of understanding as someone with university degree in robotics. Therefore XRS systems should be evaluated not just by practitioners in research institutes and industry, but by the general public as well. Failure to do this properly means the evaluation does not actually represent the perspectives of the target users.

An explainable agent can be evaluated via subjective and objective measures such as:

- Subjective measures:
  - Peoples' opinions of the suitability and clarity of various explanations.
  - Peoples' ratings of a degree they trust the robot to preform complex tasks after explaining its capabilities.
- Objective measures:
  - Peoples' ability to correctly summarise a plan that a robot has described to them.
  - Peoples' ability to predict a robot's behaviour based only on communication with the robot on prior tasks.

3.4 Current Explainable Robotics Systems

Explainable Planning (XAIP) has been introduced by [Fox et al. 2017] preforming tests in simulation of space rover and AUV robots. The architecture they propose can explain; why the planner chose particular actions over others, why the planner's decision was the best choice at the time of planning, why it believed certain actions could not be done, and why/if it needed to re-plan after taking sensor measurements. They characterise some of the questions that need to be explained, but maintain that there is no clear way to define what constitutes a good explanation. This system however lacks in its ability to abstract an explanation to the level of detail required for the context, explanations are instead presented similarly to how content is internally represented by the robot.

The XAI-PLAN system has also been introduced by [Borgo et al. 2018] which is implemented on top of the ROSPlan framework [Cashmore et al. 2015], which provides an API for robotic planners employing PDDL models in the Robot Operating System (ROS) [Quigley et al. 2009]. Their system can provide explanations about initial plans and supports contrastive explanations by allowing alternative actions to be suggested by the user. Whilst this work provides a more transparent interface to the generic ROSPlan framework, it can only deal with classical planning problems and provides no support for processing explanations to create human understandable queries at varied levels of abstraction.

The notions of plan explicability and predictability have been proposed by [Zhang et al. 2017b], evaluating their theories on a robot arm on the blocks world domain. They attempt to produce a general solution for high level explainable task planning for robots with goal-reasoning capabilities, using their plan explicability and predictability measures to generate plans that can be easily understood by humans. This work has been continued by [Sreedharan et al. 2017] producing a novel planning technique called *MEGA* (Multi-model Explanation Generation Algorithm). They claim their techniques bring together techniques from past research on both explainable planning and explanation generation and demonstrate this on a robot involved in a supervised reconnaissance task. They claim that explainable agents do not need to focus on "explaining the obvious", indicating that robots do not necessarily need to explain every single detail of the reasoning behind their choices as humans are naturally capable of "connecting the dots". However, these systems are based on human-aware planning. This technique causes generated plans to depart, often in a very costly manner, from the optimal plan in favour of a plan that can be more easily interpreted by humans. Such a technique is rarely be sufficient or desirable for many real-time applications of robotics. Furthermore, it is hard for such systems to know when a plan is sufficiently explicable, meaning that plans can be revised even though the original could already have been explained effectively. They are also not capable of providing explanations at different levels of abstraction.

A comparative study between two contrasting explanation generation systems, KRASP and UMBRA, has been preformed by [Meadows et al. 2016]. The first of these is a ASP based system and the second is a cognitively inspired heuristic based system. They point out that ASP based systems have many advantages over heuristic based systems, but heuristic based systems do have some complementary capabilities. Particularly, heuristic based systems are more capable of scaling to larger, more complex domains. ASP based systems are extremely domain independent, but cannot explain anything beyond the domain dynamics that have been explicitly encoded by the designer, they also struggle from the limitation that they must always compute full-models, this means that they cannot generate partial explanations. ASP based systems also represent content in a way that innately supports explanation because the resulting answer sets are by their very nature easy to interpret. More recent work has created more powerful ASP based reasoning systems for explanation generation [Colaco and Sridharan 2015, Sridharan and Meadows 2019]. These have also incorporated the ideas of refinement and zoom from [Zhang et al. 2015, Sridharan and Gelfond 2016].

There has been work on translating the decisions taken by agents based on reinforcement learned neural networks [van der Waa et al. 2018, Lee 2019] into human understandable terms. Allowing such an agent to describe its expectations for the effect of a single action, as well as its entire policy. They found that people generally seek explanations that describe the general overall plan and far more rarely ask about single actions. Similar work has introduced a framework capable of interpreting the history of a reinforcement learned agent's interaction with a domain [Sequeira et al. 2019]. However, this is mostly concerned with describing learned strategies, preferences and goals. These systems do not support the vast majority of the desired capabilities of an explainable agent, indicating that reinforcement learned neural networks are currently not at all sufficient to support explainable agency for human-robot collaboration.

An interesting future project is the UK Robotics and Artificial Intelligence Hub for Offshore Robotics for Certification of Assets (ORCA Hub) project has recently been launched [Hastie et al. 2018]. The ORCA Hub seeks to use teams of robots to operate a completely autonomous offshore energy field entirely controlled, inspected and maintained from the shore. Their goal is to create interpretation and explanation systems for autonomous control systems, including black-box models, such as Deep Neural Networks, and grey-box models, such as Bayesian Networks. If they are successful, such systems may prove highly advantageous to explainable agency in the future.

4 Goals and Contributions of this Thesis

There is sufficient motivation to achieve explainable agency for human-robot collaboration. However, there are many as yet unsolved problems that must be overcome by practitioners working towards this objective. Yet there exists substantial literature from researchers in robotics and the social sciences to guide this work.

There has been many attempts at creating explainable robotic systems. Most of these systems are capable of contrastive explanations. However they all either; do not fully describe their expectations for the effect of each action in a plan and how these relate to their performance criteria, or instead explain far too much detail to point were they cannot be understood even by an expert. Very few systems are capable of selecting and revising the level of abstraction of an explanation in real-time, and do not use human feedback to inform its choices when constructing explanations. Furthermore, many use highly domain dependant attributes or do not support one or more of the desired KRR capabilities such as non-monotonic revision of knowledge and beliefs. Recent literature regarding these systems is also generally very vague in specifying the way in which queries can be posed to the system and whether explanations are actually generated in time commensurate with the needs of human-robot collaboration. Generally current systems are good at retrospective explanations to simple queries such as "Why did you choose X action over Y action?". However, they are not capable of more complex explanations such as; "How did your final plan differ from your original plan and why?". Techniques which sacrifice plan optimally in favour of plan explicability seem useful in only a small number of circumstances. It may well be more comfortable for humans if a residential service robot plans around the "human in the loop" such their plans are more explicable, but their is a much wider range of applications where such a technique is disadvantageous, such as the majority of industrial, commercial and transport applications. It seems clear that we must first focus on making the behaviour of a robot which acts optimally more explainable before we can begin to work towards adapting a robot's behaviour to be more explicable.

Ultimately, there exists no system which supports all of the capabilities which have been detailed above. Every single one of these is crucial for an agent to be considered properly explainable. This thesis will significantly expand upon the capabilities of existing explanation generations systems for robotics applications. It will create an architecture for explainable reasoning and learning for human-robot collaboration which fully supports all of the functional abilities of an explainable agent outlined above. This architecture will enable robots with the abilities required to be capable of:

1. Describing its knowledge and beliefs, justifying its actions and decisions, and clarifying its capabilities, performance criteria and decision making processes.
2. Non-monotonic revision of belief states, knowledge and assumptions, explaining how and why it adapted as events diverged from expectations based on observations.
3. Representing and reasoning with both logical and probabilistic knowledge elements.
4. Reasoning about goals and intentions, explaining if and why it chose to adopt, reject or alter its goals.
5. Interactive and on-demand explanations to queries posed in real-time both during and after plan execution.
6. Understanding queries posed in natural language such that humans need not adapt communicate with robots.
7. Constructing intuitive, relevant and context-specific explanations at the appropriate level of abstraction.

5 Action Languages

Action languages are formal models of parts of natural language used for describing transition diagrams of dynamic systems [Gelfond and Lifschitz 1998]. Action languages are used to model effects of an agent's actions upon a system, and as such many have been developed and used in robotics applications. Recent versions of action languages are often viewed as high level notations of answer set programs [Babb 2015].

This section reviews some literature regarding action languages of specific interest to this thesis and introduces the basic concepts of action languages, for a more detailed explanation see chapter 8 of [Gelfond and Kahl 2014].

5.1 Review of Literature Regarding Action Languages

The first action language, STRIPS [Fikes and Nilsson 1971] was developed to deal with basic robot planning problems in the real world, such as modelling the position of objects and performing navigation in a domain. Such problems require more complex and generalised models of the domain compared to those needed to solve puzzles or games. STRIPS was able to update a robot's internal model of the domain state in order to simulate the effects of actions. It did this using first-order predicate calculus and could deal with the large number of facts and relations required for a robotic planning problem. STRIPS was successfully used on the infamous "Shakey Robot" in 1966-1972 which was the worlds first intelligent mobile robot [SHAKEY 1972]. This illustrated the potential power of action languages, paving the way their use in robotics applications.

Action language ADL [Pednault 1989] was later developed which addressed and solved a major limitation of STRIPS operators. This was that they did not allow context specific effects, meaning that all consequences of every possible agent action in all possible situations had to be listed.

An issue which troubled early action languages was the *frame problem*, which occurs from the need to specify what has not changed over time. It can be solved by finding a representation of the *inertial axiom*, which is a default stating that things usually remain the same as time passes unless they are forced to change. Both STRIPS and ADL solved the frame problem because they are able to determine which domain attributes are not affected by the execution of a particular action and should therefore keep their original value in the resulting state [Downs et al. 1994].

Action languages B (section 5 of [Gelfond and Lifschitz 1998]), C [Giunchiglia and Lifschitz 1998], and C+ (section 4 of [Giunchiglia et al. 2004]), were then introduced which further improved upon STRIPS and ADL by allowing definitions of the indirect effects of actions by describing the dependency between *fluents* (domain attributes that can change with time) and thus solved the *ramification problem*. Action language B allows recursive definitions, which play an important role in reasoning about actions and describing transitive relations between fluents. Like STRIPS and ADL the frame problem is solved in B by incorporating the inertial axiom. However, it is difficult to describe fluents whose temporal behaviour is not inertial in B. Action languages C and its extension C+ attempt to solve this issue by allowing inertial axioms to be freely included or not included, thus allowing other assumptions about the temporal behaviour of fluents to be postulated. On the other hand, recursive definitions cannot be easily expressed in C and C+.

Action language BC [Lee et al. 2013], its newer version BC+ [Babb 2015] and its probabilistic extension [Lee and Wang 2018] have also been developed in an attempt to combine the advantageous features of B and C+ by including *default rules* [Reiter 1980] which allow one to derive the conclusion of a rule from the premise if its justification can be assumed consistently. These default rules allow *causal laws* (describing action effects) which specify default values of fluents at arbitrary time steps. These languages are designed specifically to take advantage of powerful constructs provided by modern ASP solvers such as choice rules, aggregates, and abstract constraint atoms.

Action language AL_d [Gelfond and Inclezan 2009, Gelfond and Inclezan 2013] improved upon its predecessor B by making a distinction between *inertial fluents* (also called *basic fluents*) and *defined fluents*. Inertial fluents obey the laws of inertia and can be directly changed by actions, defined fluents on the other hand do not obey inertial laws and cannot be directly changed by actions, instead they are defined in terms of other fluents. AL_d also introduced the concept of a *fluent dependency graph* which formally describes the dependency between fluents in a system. The fluent dependency graph (page 170 of [Gelfond and Kahl 2014]) allows powerful assertions to be made about the validity of a system description, such as *weak acyclicity* and *well-foundedness*.

AL_d has been further extended by [Sridharan et al. 2015] in a number of ways. The newer version of AL_d supports many addition features. Non-boolean fluents allow compact modelling of system descriptions. Non-deterministic casual laws describe the stochastic effects of an agent's actions. Basic fluents are partitioned into *basic physical* and *basic knowledge* fluents, along with a similar partitioning of actions into *physical actions* and *knowledge producing actions*. And finally the observability of physical fluents is defined, describing whether their value can be directly observed by knowledge producing actions, or indirectly observed by inferring their values from other knowledge.

5.2 Syntax of AL_d

A *term* is a *constant*, *variable*, *arithmetic expression* or *function*, where variables are used as a shorthand for a collection of *ground instances* of the same statement obtained by replacing the variables by constants of the corresponding sorts. A *function* is an expression of the form $f(\bar{t}) = x$, where f is the *function symbol*, x is a constant, variable or arithmetic expression defining the the *value* of f and $\bar{t}$ is a vector of terms defining the *parameters* of f. An *atom* is an expression of the form $p(\bar{t}) = true|false$, where p is a *predicate symbol* and $\bar{t}$ is again a vector of terms defining the parameters of p. Usually in action languages, predicate symbols are omitted because they do not aid the expressiveness of the model, in section 6 the basic process of translating AL_d statements into an ASP program is outlined, including the process of adding the required predicate symbols. Each function f (and predicate p) is assigned; a positive integer n, called the *arity* of f, which defines its number of parameters, a set of sorts $s_0, ...s_n$, called the *domain* of f, which define the sorts of each parameter, and a sort s, called the *range* of f, which defines the sort of the values that may be assigned to it, for predicates these can only be Boolean values.

The *sorted signature* of a *system description* SD written in AL_d is defined by the tuple:

$$\Sigma = \langle C, F, S \rangle \tag{1}$$

Where: C is a set of sorts, F is a set of functions, all sorts and functions in C and F are defined by strings over some fixed alphabet and S is a sort hierarchy which is a directed acyclic graph whose nodes are labelled by sort names from C. An arc $\langle s_0, s_1 \rangle$ from S indicates that s_0 is a sub-sort of s_1. F contains three special sorts; *statics*, *fluents* and *actions*. Statics and fluents are *domain properties*, the value of fluents can be changed (directly or indirectly) by actions and the value of statics can never change. A *domain literal* is a parameterised domain property with an assigned value $l(\bar{t}) = x$ or its negation $l(\bar{t}) \mathrel{!}= x$. Domain literals formed by fluents are *fluent domain literals*, otherwise if they are formed by statics they are *static domain literals*. Fluent domain literals are partitioned into *inertial fluent literals* and *defined fluent literals*, as described previously. Defined fluents can only ever be assigned boolean values, whereas inertial fluents may be non-boolean. Inertial fluent literals are further partitioned into *physical fluent literals* and *knowledge fluent literals*, the prior referring to physical properties of the domain and the latter referring to the agent's own knowledge. Actions are also similarly partitioned into *physical actions* and *knowledge producing actions*, the prior can change the value of physical fluents and the latter grant the agent knowledge of the actual value of physical fluents. Physical fluent literals are also split into *directly observable* and *indirectly observable* physical fluents, as described previously. A set L of domain literals is *complete* if for all domain literals l either $l(\bar{t}) = x$ or $l(\bar{t}) \mathrel{!}= x$ is in L. L is *consistent* if there is no domain literal l where both $l(\bar{t}) = x$ and $l(\bar{t}) \mathrel{!}= x$ are simultaneously in L.

5.3 Statements of AL_d

A system description SD of AL_d is a collection of the following statements over some sorted signature Σ:

1. *Deterministic Causal Laws*:

$$a \textbf{ causes } l_{in}(\bar{t}) = x \textbf{ if } body$$

2. *Non-deterministic Causal Laws*:

$$a \textbf{ causes } l_{in}(\bar{t}) = \{X : p(X)\} \textbf{ if } body$$

3. *State Constraints*:

$$l(\bar{t}) = x \textbf{ if } body$$

4. *Definitions*:

$$l_d(\bar{t}) = true \textbf{ if } body$$

5. *Executability Conditions*:

$$\textbf{impossible } a_0, ..., a_k \textbf{ if } body$$

Where: the *body* of a rule is formed by a set $l_0(\bar{t_0}) = y_0, ..., l_m(\bar{t_n}) = y_m$ of domain literals, a is an action, l_{in} is an inertial fluent literal, l_d is a defined fluent literal, l is any fluent literal, $k \geq 0$, and $m \geq -1$. If $m = -1$ then there are no literals following the keyword **if** and it can be thusly omitted.

Causal laws define the direct effects of executing actions in particular domain states. Deterministic causal laws state then if a is executed in a state satisfying *body* then the value of $l_{in}(\bar{t})$ is x in the resulting state. If a is a knowledge producing action then $l_{in}(\bar{t})$ must be a knowledge fluent. Non-deterministic causal laws state then if a is executed in a state satisfying *body* then the value of $l_{in}(\bar{t})$ may take any value from the set $\{X : p(X)\}$, which is referred to as the dynamic range of $l_{in}(\bar{t})$.

State constraints define the indirect effects of executing actions and prohibit domain states which are physically impossible. A state constraint states that the value of $l(\bar{t})$ must be x for every state that satisfies the body of the rule.

Definitions describe relations between defined fluent literals and other domain literals, where $l_d(\bar{t}) = true$ if it follows from the truth of at least one of its defining rules and otherwise $l_d(\bar{t}) = false$. Importantly no negation of a defined fluent may occur in the head of a definition.

Executability conditions define conditions under which particular actions are cannot be executed. An executability condition states that actions $a_0, ..., a_k$ cannot be simultaneously executed in a state which satisfies the body of the rule.

5.4 Initial State Default Knowledge and Histories

The default knowledge D of a system description SD of AL_d is a collection of the following statements over Σ:

1. *Initial State Default*:

$$\textbf{initial default } d(\bar{t}) : l_{in}(\bar{t}) = x \textbf{ if } body(d)$$

2. *Default Preference*:

$$\textbf{prefer}(d_1(\bar{t}), d_2(\bar{t}))$$

Where d is the unique name of a default declaration. Initial state defaults say that in an initial state satisfying *body*, the inertial fluent literal l_{in} is assigned the value of x by default. Default preferences define an anti-symmetric, transitive preference relation between defaults, stating that d_1 is preferred over d_2. Intuitively, in the case that the simultaneous application of both defaults would lead to a contradiction, d_1 is applied over d_2.

The *recorded history* Γ_{n-1} of SD up to the current time step n is set of *observations* made by an agent, describing the known value of domain properties and the occurrence of actions that have actually happened. These are defined by a collection of the following statements over Σ, where i is a time step:

$$\textbf{observed}(l(\bar{t}) = x, i) \qquad \textbf{observed}(l(\bar{t}) \mathrel{!}= x, i) \qquad \textbf{happened}(a, i)$$

A path $\langle \sigma_0, a_0, \sigma_1, ..., a_{n-1}, \sigma_n \rangle$ of a transition diagram $\tau(SD)$ is a *model* of Γ_{n-1} if for any $0 \leq i < n$:

1. $a_i = \{a : \textbf{happened}(a, i) \in \Gamma_{n-1}\}$
2. if $\textbf{observed}(l(\bar{t}) = x, i) \in \Gamma_{n-1}$ then $l(\bar{t}) = x \in \Gamma_{n-1}$
3. if $\textbf{observed}(l(\bar{t}) \mathrel{!}= x, i) \in \Gamma_{n-1}$ then $l(\bar{t}) \mathrel{!}= x \in \Gamma_{n-1}$

If Γ_{n-1} has a model then it is said to be *consistent*.

5.5 Transition Diagrams and Trajectories

In AL_d a dynamic domain defined by a system description SD is modelled by means of a *transition diagram* $\tau(SD)$. A transition diagram is a directed graph whose nodes correspond to possible states of the domain and whose arcs describe possible actions which can change the state between those connected by the arc.

A single state transition of a transition diagram can be described by $\langle \sigma_0, \{a_1, ..., a_k\}, \sigma_1 \rangle$, where $\{a_1, ..., a_k\}$ is a arbitrarily large set of actions which if executed simultaneously in a state σ_0 would result in a state σ_1. This guarantees that the representation is *Markovian*, whereby the effect of an action depends only on the current state and knowledge of any past states would carry no additional information about its effect.

A path $\langle \sigma_0, a_0, \sigma_1, ..., a_{n-1}, \sigma_n \rangle$ of the transition diagram $\tau(SD)$ represents a possible *trajectory* of the system from its initial state σ_0 to its final (goal) state σ_n. The transition diagram of a dynamic system contains within it all possible trajectories of that system.

6 Answer Set Programming

Answer Set Programming (ASP) is a well known declarative programming paradigm that has established itself as a prominent technique for knowledge representation and reasoning, particularly for knowledge intensive combinatorial optimization problems, such as planning, diagnostics, query answering, explanation generation [Kaminski et al. 2017].

ASP is defined by *stable model* semantics [Gelfond and Lifschitz 1988] and has its roots in *logic programming* and *non-monotonic reasoning* [Gelfond and Lifschitz 1991]. The central concept is to represent a problem as a *program* whose *models*, called *answer sets* correspond to possible solutions to the problem. ASP also shares a close relationship with other formalisms such as Propositional Satisfiability (SAT), Satisfiability Modulo Theories (SMT), Quantified Boolean Formulas (QBF), Constraint Programming (CP) and Planning and Scheduling [Gebser et al. 2017].

ASP has been successfully employed in a wide range of industrial and scientific applications, including a large number of robotics systems [Erdem and Patoglu 2018]. It has further been shown to be highly effective at puzzle solving and theory proving. The successfulness of ASP is generally attributed to the versatility, expressiveness, readability and elegance of its modelling language and to the widespread availability of efficient ASP solvers [Gebser et al. 2018].

ASP was originally introduced by [Gelfond and Lifschitz 1991] and has since been significantly extended with constructs which make solving of complex problems easier and more efficient. Modern ASP solvers now include many features including; heuristics, aggregates, weight and cardinality constraints, optimisation statements, and external atoms. As such an ASP language standard has been then introduced by [Francesco Calimeri 2012] in order to keep the syntax of the language's growing set of constructs consistent across the varying solvers. The input language of ASP allows complex problems to be encoded and solved in a purely declarative and highly elaboration tolerant manner.

The main component of an ASP program is the *rule*, which consists of a *head* and *body* of the form shown in rule 2.

$$head \longleftarrow body \tag{2}$$

Where the body is a logical conjunction, possibly involving negation, and the head is either an atomic formula or a logical disjunction. An ASP program is a collection of such rules. Intuitively, a rule functions as such, if the body of a rule is satisfied then so must its head. The fundamental types of rule and their functions are described in sub-section 6.3.

6.1 Review of Literature Regarding ASP Systems

ASP systems generally consist of two distinct parts, the *grounder* and the *solver* [Gebser et al. 2018]. The prior of these takes an ASP program containing *non-ground rules* and returns the equivalent program containing only *ground rules*, where an expression is said to be ground, if it contains no variables [Kaminski et al. 2017]. The latter takes a ground program and *solves* it to produce one or more *answer sets* representing the possible solutions of the program.

6.1.1 Grounders

Grounding is the process of replacing all rules containing variables with rules containing only constants. This is done by replacing all variables with all possible combinations of the corresponding constants. The grounding process typically produces a ground program of polynomial to exponential size with respect to the original non-ground input program. Modern grounders employ various techniques for producing ground programs of minimal size whilst still preserving the semantics of the original program, i.e. it is still guaranteed to produced the same answer sets once solved. One such process is called *simplification*, part of this process allows certain rules to be omitted from the program under particular conditions, hence reducing the size of the resulting program and ultimately the computational complexity of the solving process. For example, all ground atoms which do not occur in the head of any rule results in all possible ground rules which would contain that atom in the body not being grounded. One of the major problems that grounders attempt to deal with is *unsafe* rules [Balai et al. 2013a]. Unsafe rules are problematic as they can result in program rules behaving erroneously after addition of other rules which are seemingly unrelated to the original rules.

One of the first grounders was *Lparse* [Syrjänen 1998]. Programs passed to *Lparse* must conform to it's ω*-restrictedness* condition [Syrjänen 2001]. This demands that each variable in a rule must occur in a positive body literal of that rule which is not mutually recursive with its head and is not unstratified and not dependent upon an unstratified predicate. Another early grounder was *Gringo* series 1 [Gebser et al. 2007]. ASP programs passed to *Gringo* must conform to it's λ*-restrictedness* condition. This condition extends ω*-restrictedness* to deal with recursion, guaranteeing a finite sized ground program. Later the *DLV* [Eiter et al. 1998] grounder and *Gringo* series 3 (and above) [Gebser et al. 2011b, Gebser et al. 2015a] were introduced. These grounders impose the simpler condition of *variable safety* which requires only that every variable in a rule appears in at least one positive body literal of that rule. These grounders successively expand a program's term base to avoid duplicate work during grounding.

6.1.2 Solvers

Solving ASP programs to compute answer sets is founded upon the concept of *Propositional Satisfiability* (SAT) and *Boolean Constraint* solving. SAT solving is based upon the classical Davis-Putnam-Logemann-Loveland (DPLL) [Davis et al. 1962] procedure but current solvers now employ conflict-driven solving procedures [Gebser et al. 2009] based on Conflict-Driven Clause Learning (CDCL) [Marques-Silva and Sakallah 1999].

The first ASP solvers were the *DLV* [Eiter and Faber 2000, Leone et al. 2006] and *Smodels* [Niemelä et al. 2000, Simons et al. 2002] solvers and were based on the DPLL procedure adjusted for ASP solving. Recent conflict-driven ASP solvers include *Clasp* [Gebser et al. 2015c] and *WASP* [Alviano et al. 2013]. *Clasp* series 1 [Gebser et al. 2012a] was the first such solver, featuring conflict-driven learning and back-jumping. *Clasp* series 2 [Gebser et al. 2012b] extended this to support parallel search via shared memory multi-threading. *Clasp* series 3 [Gebser et al. 2015c] further expanded upon its predecessors by supporting parallel processing of disjunctive logic programs, prioritised parallel optimisation, and integration of a wide range of aggregate, heuristic and meta-statements.

There has also been solvers developed for cluster based solving [Gebser et al. 2011a], GPU based solving [Dovier et al. 2015] and solvers dedicated to ASP variants P-Log [Balai et al. 2019] and CR-Prolog [Balduccini 2007].

6.1.3 Combined Systems and their Interfaces

The earliest combined system was *Lparse* coupled with *Smodels*, they provided a basic C++ library, but required you to "manually" pipe the output of the grounder to the solver. Unfortunately, both *Lparse* and *Smodels* are now deprecated, they no longer conform to the ASP-Core-2 language standard and do not support most new ASP features.

The most successful modern ASP system [Gebser et al. 2017] is *Clingo* series 5 [Gebser et al. 2014]. The Clingo system integrates the *Gringo* series 5 grounder and the *Clasp* series 3 solver together. Clingo provides an API with bindings in C, C++, Lua and Python, giving control over the grounding and solving processes [Gebser et al. 2019a, Kaminski et al. 2017]. The API provides powerful features such as incremental [Gebser et al. 2008] and multi-shot solving [Gebser et al. 2015d, Gebser et al. 2019c]. These are extremely powerful techniques for automated planning as they allow parts of the program to be iterative re-grounded with respect to a successively increasing planning horizon. This is useful because the planning horizon is rarely known and hard to estimate prior to solving, thus Clingo provides an efficient technique for determining the horizon, assuming it is finite. Clingo also removes the need to completely re-ground and re-solve programs when new statements are integrated into the program by reusing ground rules from previous calls [Gebser et al. 2015b], this is powerful for robotic planning applications because the robot may commit many observations to the program during plan execution and may thus need to resolve the program multiple times to adapt to observations that contradict its beliefs.

6.1.4 ASP with Sorts

ASP systems such as *Clingo* suffer most majorly from an inability for the user to conveniently specify the sorted signature of a system description in a manner as described in section 5.2. Early systems such as *Lparse* used the $\#domain$ meta-statement to define sorts for variables. This statement simply "locks" a variable name to a particular sort, thus when the program is grounded that variable is safe as it can only be replaced with the correct sort. However, this technique is generally not sufficient for large programs and causes major problems if different ASP programs are combined that declare different sort domains for the same variable name [Balai et al. 2013b].

Although there has been multiple attempts to solve this problem, this report focuses on the most promising of these, the *SPARC* system [Balai et al. 2013a, Balai 2018, Balai 2019]. The SPARC system is a not a grounder or solver, but rather a translator and an answer set parser. SPARC allows a sorted signature to be enforced upon an ASP program and provides an intuitive interface for easing the process of creating and debugging programs. The language of SPARC can be viewed as high level syntax for the ASP-Core-2 language standard which additionally provides the ability to explicitly declare sort definitions on top of the normal language. This means that programmers do not need to concern themselves with variable safety conditions of rules, and errors, such as misspelling a function symbol, can be more easily found and fixed. SPARC also provides a high level notation for defining CR-Rules. Unfortunately, SPARC does not allow prioritised CR-Rules, and does not fully support some features of modern ASP solvers such as body literal weight/cardinality constraints, weak constraints, optimisation statements and some meta statements.

6.2 Applications of ASP in Robotics

There has been many real-world applications of ASP to date [Esra Erdem 2016, Grasso et al. 2013]. These include robotics [Erdem and Patoglu 2018], healthcare [Dodaro et al. 2018, Erdem et al. 2011, Gebser et al. 2010] and industrial [Falkner et al. 2018] applications. This sub-section reviews a few robotics applications of ASP.

ASP has been incorporated into a planning and monitoring system for housekeeping robots [Aker et al. 2012]. They embedded commonsense knowledge, continuous geometric reasoning and temporal reasoning about durations of actions within their framework. They used the incremental ASP solver iClingo to encode the action language C+, finding it used significantly less memory compared to the causal reasoner CCALC which they used in previous work.

An extension of the Robot Operating System (ROS) [Quigley et al. 2009] which attempts to provide a seamless way to use ASP in a robotic environment has also be produced [Andres et al. 2013]. The extension, called ROSoClingo integrates the reactive ASP solver oClingo into a ROS package to provide a general way in which an ASP program can control and respond to a robot's actions. They illustrate the usage of ROSoClingo in a simulation of a robot delivering mail in an office. ROS is a very popular system for robot control, and because ROSoClingo is highly compatible with other ROS components, it poses a potentially greatly effective way of controlling a robot in a fully declarative manner.

Another study [Balduccini et al. 2014] developed an ASP based architecture for coordinating teams of Unmanned Aerial Vehicles (UAVs) in challenging real-world dynamic environments where communication between agents is unreliable. Claiming that it was the first practical application of a complete ASP-based multi-agent architecture involving a combination of centralized reasoning, decentralized reasoning, execution monitoring, and reasoning about network communications. They demonstrated that network-aware reasoning in ASP significantly improves reliability and performance of multi-agent planning. The architecture may prove highly useful in network-centric environments, which are common in many military, public safety and commercial applications.

The ASP solver WASP has also been extended with an interface allowing domain-specific heuristics to be easily embedded [Dodaro et al. 2016]. They showed that the combination of ASP and domain-specific heuristics could significantly improve the performance of ASP solvers. Their evaluation showed that the domain-heuristic-driven ASP solver found solutions for all implementations of two extremely challenging real-world applications of ASP defined by Siemens: the Partner Units Problem and the Combined Configuration Problem.

6.3 Syntax and Semantics of ASP

The following sub-section describes the basic syntax of an ASP program, enough for the reader to understand the general concept, for more details see [Francesco Calimeri 2012] for the complete ASP-Core-2 language standard, [Gebser et al. 2014, Kaminski et al. 2017, Gebser et al. 2019b, Gebser et al. 2019a] for the syntax of Clingo and its API, and [Balai et al. 2013a, Balai 2018] for the syntax of SPARC. All following rules are encoded using the syntax of Clingo following the ASP-Core-2 language standard, except that SPARC-like syntax is used for defining sorts (statements proceeded by a hash) and for defining CR-Rules due to the increased clarity of the language.

An ASP program, similar to most logic programming paradigms, consists of rules of the form:

$$a_0 \,|\, ... \,|\, a_m \text{ :- } a_{m+1} \text{ , } ... \text{ , } a_n, not\ a_{n+1} \text{ , } ... \text{ , } not\ a_o. \tag{3}$$

Where: a is an *atom* of the form $p(\bar{t})$ or its classical negation $-p(\bar{t})$, p is a *predicate symbol* and $\bar{t}$ is a vector of terms defining the parameters of p composed of function symbols and variables. In contrast to the notation of AL_d, a constant in ASP is actually a function of arity *zero*, tuples can also be represented, which are functions with no name and at least one parameter. Atoms a_1 to a_m are called head atoms, whilst a_{m+1} to a_n and $not\ an+1$ to $not\ a_o$ are called positive and negative body literals, respectively. The logical connective *not* stands for *negation as failure* (sometimes referred to as *default negation*), as such a negative body literal of the form $not\ a$ is often referred to as a *naf-literal*. In contrast to *classical negation*, which stands for "a is believed to be false", negation as failure stands for "a is not believed to be true", i.e. a can take three possible truth values, *true*, *false* or *unknown*.

A rule with exactly one atom in the head and at least one literal in the body ($m = 1$) is called a *normal* rule, if the body if a normal rule is satisfied by the answer set of the program, then the atom in the head must be in the answer set. A rule with exactly one atom in the head and no body ($m = o = 1$) is called a *fact*, all atoms occurring in facts must be in all answer sets of the program. A rule with no head and at least one literal in the body ($m = 0$) is called an *integrity constraint*, no answer set of the program may satisfy its body, because the head can never be satisfied. A rule with multiple atoms in the head and any number of literals in the body ($m > 1$) is called a *disjunctive* rule, disjunctive rules are similar to normal rules except the head is satisfied if any one of the atoms in the head occurs in the answer set of the program. A rule with at least one atom in the head enclosed in braces and any number of literals in the body ($m > 0$) is called a *choice* rule, choice rules are similar to disjunctive rules except the head is satisfied if any, all, or none of the atoms in the head are in the answer set of the program. Choice rules can be preceded by the aggregate statements $\#count$ or $\#sum$ (amoungst others) and constrained between upper and lower limits to express *cardinality constraints* and *weight constraints* respectively [Dell'Armi et al. 2003, Calimeri et al. 2005]. It is important to note that although aggregates are preceded with a hash, this does not imply they are sort declarations or meta-statements.

Two special types of rule are *weak constraints* and *consistency restoring* (CR-rules) rules. In fact CR-rules are encoded via weak constraints and a particular type of choice rule known as a generator rule, the process of doing this is described in algorithm 1 of [Balai et al. 2013a]. A CR-rule states that if the body of the rule is satisfied, then the head may be satisfied only under the condition that not doing so would result in inconsistency. Thus the resulting answer set of a CR-Prolog program should contain a minimal number of occurrences of head atoms entailed only from CR-rules.

6.4 Encoding AL_d in ASP

Translation of a system description SD of AL_D into ASP is a relatively simple process. This is because AL_d is designed as a high level notation for ASP programs. This sub-section describes the general concept behind this process, there are many other rules that form an ASP program of an AL_d system description, only the most critical are covered here, for full details see sections 8, 9 and 10 of [Gelfond and Kahl 2014].

First the required predicates must be declared, these are; $holds(F, V, I)$, $is(S)$ and $occurs(A, I)$ (and their classical negations). Where F is a fluent domain literal, V is a value from F's range, S is a static domain literal, A is an action and I is a time step. These say that a fluent F holds a value V at step I, static S always holds, and action A occured at step I, respectively. This notation is sufficient to encode AL_d statements into ASP, as follows:

1. For every *Causal Law*:
   - Replace a **causes** $l_{in}(\bar{t}) = x$ **if** *body*
   - With $holds(l_{in}(\bar{t}), x, I+1)$:- $occurs(a, I), body.$
2. For every State Constraint:
   - Replace $l(\bar{t}) = x$ **if** *body*
   - With $holds(l(\bar{t}), x, I)$:- $body.$
3. For every *Definition*:
   - Replace $l_d(\bar{t}) = true|false$ **if** *body*
   - With $holds(l_d(\bar{t}), true|false, I)$:- $body.$
4. For every *Executability Condition*:
   - Replace **impossible** $a_0, ..., a_k$ **if** *body*
   - With :- $occurs(a_0, I), ..., occurs(a_k, I), body.$

Some general axioms must also be added, including the inertial axiom (rules 4 and 5), which solves the frame problem as described earlier, and the closed world assumption for defined fluents and actions (rules 6 and 7).

$$holds(F, V, I+1) \text{ :- } holds(F, V, I),\ not\ \text{-}holds(F, V, I+1),\ \#inertial\_fluent(F),\ I < max. \tag{4}$$

$$\text{-}holds(F, V, I+1) \text{ :- } \text{-}holds(F, V, I),\ not\ holds(F, V, I+1),\ \#inertial\_fluent(F),\ I < max. \tag{5}$$

$$\text{-}holds(F, V, I) \text{ :- } not\ holds(F, V, I),\ \#defined\_fluent(F). \tag{6}$$

$$\text{-}occurs(A, I) \text{ :- } not\ occurs(A, I). \tag{7}$$

For planning problems the "simple planning module" is also required. The most interesting rule of which is the CR-rule shown in rule 8. This rule informally reads as "find minimal plans", and if encoded in SPARC and solved with Clasp, will produce answer sets representing the provably minimal number of action occurrences which satisfy the goal.

$$occurs(A, I) \text{ :+ } not\ goal(I),\ \#agent\_action(A). \tag{8}$$

The notion of encoding AL_d into ASP, including defaults, is returned to in section 7 to avoid repetition.

7 Proposed Explainable Reasoning and Learning Architecture for Robots

This section describes the current implementation of an ASP based, high-level, explainable reasoning and learning architecture for human-robot collaboration. The architecture is encoded primarily in Clingo and Python, but additionally allows the option for the AL_d system description SD to be encoded into SPARC. This is advantageous because SPARC is significantly more user friendly than Clingo and it is particularly easier to debug large programs in SPARC. Unfortunately, SPARC does not support prioritised CR-rules, Clingo's *program parts*, or *external atoms*. These features are required for the architecture; to correctly resolve contradictions between beliefs and observations, and support multi-shot solving. As such, all domain independent *modules* are encoded in Clingo, the user should never have to modify these files, and the system description can be either encoded in Clingo or SPARC according to the user's preference. Thus the architecture attempts to combine the complimentary features of Clingo and SPARC.

The work done so far on the architecture has focused primarily on adding new predicate symbols for encoding statements of AL_d, which has in turn resulted in minor changes to the notation of the statements of AL_d. These predicates serve to allow explanations describing a plan to be more easily extracted from the program's answer sets, and provide the ability to plan sensor measurements for goal-critical fluent values. Additionally, novel theories for encoding initial state defaults, quantitative descriptions of items in a domain, and optimisation over cost and reward functions are presented.

7.1 Proposed Changes to the Statements of AL_d

The following describes the proposed potential changes to the notation and terminology of statements of AL_d. The old name is preserved as the changes are minor and do not change the semantics of the language. These changes are proposed to better represent the design of the new predicates for encoding such statements. In essence the concept behind these predicates is to provide a domain independent way to extract explanations about a robot's decision making processes during planning and its understanding of the domain's dynamics. The full process of doing this has not yet been implemented into the architecture so is not described in this report.

1. *Action Effects*:

$$a \textbf{ causes } l_{in}(\bar{t}) = x \textbf{ if } body$$

2. *Executability Conditions*:

$$l_f(\bar{t}) = x \textbf{ prohibits } a \textbf{ if } body$$

3. *State Constraints*:

$$l(\bar{t}) = x \textbf{ invalidates } l_f(\bar{t}) = x \textbf{ if } body$$

4. *Fluent Dependencies*:

$$l_f(\bar{t}) = x \textbf{ defines } l_f(\bar{t}) = y \textbf{ if } body$$

5. *Concurrency Conditions*:

$$\textbf{impossible } a_0, ..., a_k$$

Where the body of any rule can be formed by any combination of static or fluent domain literals but not actions and all other notation is preserved from section 5.3. *Causal Laws* are renamed to *Action Effects* and *Definitions* are renamed to *Fluent Dependencies*. Note that concurrency conditions do not have a separate body because they encoded via an integrity constraint and thus they consist only of a body which can contain only actions. The reason for the distinction between executability conditions and concurrency conditions is due to the fact the a single executability condition cannot only prohibit one action. The current implementation does not support non-deterministic action effects.

7.2 Proposed encoding of AL_d in ASP

In this sub-section the changes to AL_d are justified with regards to the new predicates and their functionality. This is mostly concerned with making the notion of AL_d more inline with that of the ASP encoding so that they may be translated more easily by the user. The AL_d statements above all now have two atoms split via a key word (**causes**, **invalidates**, **defines**, **prohibits**) preceding the **if** keyword and the *body*, except concurrency conditions. This is because these two atoms both exist in the head of the respective ASP rule for that statement, where both atoms are enclosed in one of the new predicate symbols. Below the translation between the AL_d statements and the ASP encoding is detailed and then the function of these predicates for generating explanations is described.

1. For every *Action Effect*:
   - Replace a **causes** $l_{in}(\bar{t}) = x$ **if** *body*
   - With $fixed\_effect(a, l_{in}(\bar{t}), x)$:- $body.$
   - Or $dynamic\_effect(a, l_{in}(\bar{t}), x, I)$:- $body.$
2. For every *Executability Condition*:
   - Replace $l_f(\bar{t}) = x$ **prohibits** a **if** *body*
   - With $fixed\_condition(l_f(\bar{t}), x, true|false, a)$:- $body.$
   - Or $dynamic\_condition(l_f(\bar{t}), x, true|false, a, I)$:- $body.$
3. For every *State Constraint*:
   - Replace $l(\bar{t}) = x$ **invalidates** $l(\bar{t}) = y$ **if** *body*
   - With $validity\_constraint(l(\bar{t}), x, l_f(\bar{t}), y)$:- $body.$
   - Or $transition\_constraint(l(\bar{t}), x, l_f(\bar{t}), y, I)$:- $body.$
4. For every *Fluent Dependency*:
   - Replace $l_{f0}(\bar{t}) = x$ **defines** $l_{f1}(\bar{t}) = y$ **if** *body*
   - With $holds(l_{f1in}(\bar{t}), y, I)$:- $holds(l_{f0}(\bar{t}), x, I),\ body.$
   - Or $holds(l_{f1d}(\bar{t}), true, I)$:- $holds(l_{f0}(\bar{t}), x, I),\ body.$
5. For every *Concurrency Condition*:
   - Replace **impossible** $a_0, ..., a_k$
   - With :- $occurs(a_0, I), ..., occurs(a_k, I).$

Note that statements of AL_d can be encoded via time dependant or time independent statements. The body of a time dependant statement cannot contain any actions and the body of a time independent statement cannot contain any fluents or actions. For example, dynamic action effects are time dependant, meaning that the current state has a direct impact on the effect of executing the action. Whereas, fixed action effects are time independent, meaning that the current state has no impact on the effect of the action, therefore whenever the action is executed the result will always be the same.

Action effects describe the robot's expectations for the direct effect of each available action at every step in the plan. Action effects do not concern themselves with whether an action is actually possible or not. An action effect says that if an action a is executed in a state satisfying *body* then $l_{in}(\bar{t}) = x$ must hold true is the resulting state.

Executability conditions, as before, describe conditions under which particular actions are cannot be executed. An executability condition says that for any state which satisfies *body* and $l_f(\bar{t}) = x$ holds the specified Boolean truth value, the action a cannot be executed. Concurrency conditions define sets of actions which cannot be executed simultaneously.

State constraints now only describe physically invalid states and physically impossible state transitions. If $l(\bar{t})$ is a defined fluent or a static, x must be a Boolean value. A state constraint says that for any state which satisfies *body* and $l(\bar{t}) = x$ holds true, $l_f(\bar{t}) = y$ must be false. Note that state constraints are actually split distinctly into static and fluent state constraints because the predicates which describe the truth values for statics and fluents are different.

Fluent dependencies now describe the indirect effects of actions and the relationships between fluents. Fluent dependencies describe the fluent dependency graph as defined on pages 170 and 171 of [Gelfond and Kahl 2014], which has not been described in this report due to time and space constraints. Unfortunately, there is not currently a new predicate for fluent dependencies. A predicate similar to those for the other statements of AL_d for fluent dependencies would serve to allow the fluent dependency graph to be more easily explained, but attempts to do this have thus far not worked.

7.3 Architecture Design

The proposed architecture is split into six distinct *modules*. This idea takes advantage of Clingo's *program parts* described in section 3.1.15 of [Gebser et al. 2019b]. These modules allow the Clingo program to be partitioned into distinct individual pieces which may be selectively grounded and solved. This can be used to implement *multi-shot solving* and integrate observations into the program read and write to files or re-ground and re-solve the entire program. The proposed architecture makes use of the following six modules:

1. *System Description* (SD) - This module includes the AL_d system description possibly translated from SPARC.
2. *Recorded History* (RH) - This module includes a set of statements defining observations made by the robot during plan execution, of the form shown in section 5.4, and the *reality check axioms*.
3. *Commonsense Knowledge Module* (CKM) - This module deals with default knowledge and resolves a complete consistent initial model of the domain.
4. *Diagnostics and Sensing Module* (DSM) - This module reasons about the possible occurrence of exogenous actions to explain observations that contradict beliefs and plans goal-critical sensor measurements.
5. *Minimal Planning Module* (MPM) - This module computes minimal, complete and correct plans, such that all goals are satisfied in the minimum possible number of steps.
6. *Optimisation Module* (OPM) - This module preforms optimisation of cost and reward functions.

Figure 1 shows a graphical model of the architecture. Control and information flow between modules is indicated by the arrows. Each of these modules, with the exception of the system description and history modules, includes one or more CR-rules. These CR-rules are prioritised such that each module takes control of the solving process in the listed order. This process works as described by algorithm 1 as follows:

Algorithm 1: Architecture Control Flow

```
Translate system description from SPARC;
while Goals not completed do
    if Observations contradict beliefs or No actions yet executed then
        Resolve a consistent system model based on initial defaults, exogenous actions and history of observations;
        Compute the minimal complete plan that satisfies all goals according to the rules in the system description;
        Compute the optimal plan based on domain specific cost and reward functions;
    end
    Execute next control action in the plan and take required observations;
end
```

It is important to note that the system description is only ever translated from SPARC once. The translation process returns a set of non-ground rules which Clingo can use as a "template" for grounding. Clingo can thusly expand the ground program of the system description by incrementally increasing the planing horizon until the goals are satisfied.

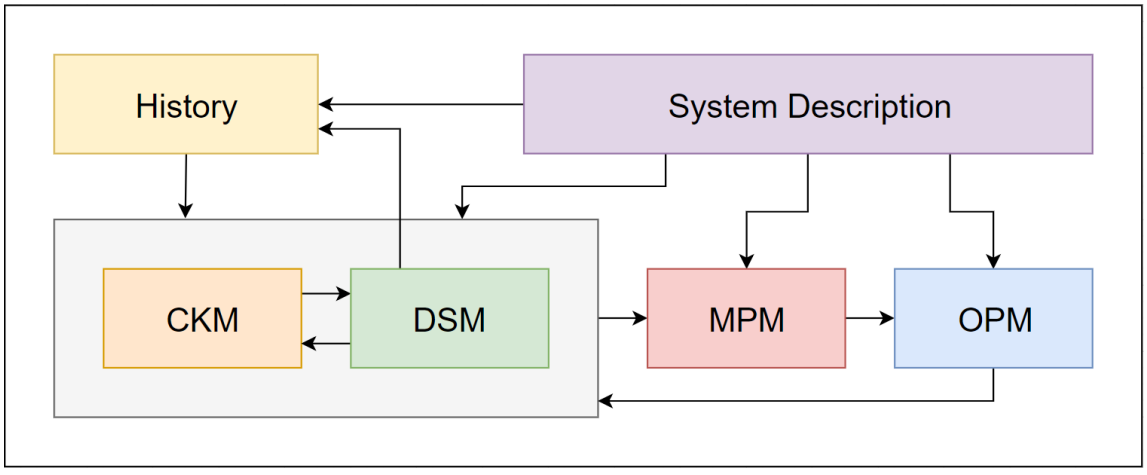

Figure 1: Architecture Graphical Model

8 The Robotic Hotel Proof-Of-Concept Domain

The proposed proof-of-concept domain is a hotel staffed by robot assistants. In such a hotel there may be many robots with heterogeneous abilities operating, each of which will be assigned any number of a large variety of different possible tasks. These tasks include; moving luggage into guests' rooms, bringing guests room service, replacing missing items from rooms, cooking and cleaning. There will also be many humans collaborators in the hotel, such as; staff members, with different levels of authority over the robots, and of course many human guests.

A hospitality domain is greatly departed from traditional robot domains, such as those in the manufacturing industry, in which the domain can be designed around the incorporation of robots, maximising safety and productivity. Robots deployed in such domains can be enclosed in safety cages, equipped with complete and correct domain knowledge, and employees are likely to have a good understanding of the robots' functionality. Hotels on the other hand have none such guarantees. They are meant for human comfort and should not be designed around the inclusion of robots, the hotel should be designed for the needs and desires of the guests and the robot should therefore be designed around the domain. The robot will also have to collaborate with staff members with a good understanding of the robot's behaviour, and with guests who will mostly have very little understanding. Furthermore, the range and complexity of queries posed by such individuals may vary significantly between them. A robot may also have to apply restrictions to various aspects of its knowledge base, based on the role of the person querying it, for example the robot should not divulge sensitive information between guests but should do so if asked by a hotel staff member. Robots must therefore be capable of working around these complexities with high reliability and interacting effectively with a wide variety of human collaborators with differing needs from the robots.

8.1 Justification for the Domain Choice

The robotic hotel domain presents an ideal environment for testing explanation generation systems both in simulation and in the real world. The most pivotal reasons for the domain choice are summarised as:

- It presents many different difficult challenges for robot planning and diagnostics which generalise well to other similar domains of great interest.
- It is highly demanding for explanation generation due to the large variety of possible questions that may be demanded at vastly different levels of abstraction by many people.
- It's structure is simple and yet highly expandable, allowing it's dynamics to be easily understood and encoded in simulation both for a small and a very large hotel.
- It is a greatly desirable application of robots from a human quality of life perspective and is a commercially lucrative application of robotic assistants to preform relatively simple but tedious tasks systematically.
- Hotels are already rigorously reviewed, if robots are to be implemented into real-world hotels a large amount of evaluation data may be obtained regarding the effectiveness of explanations.

8.2 Recent Work on Robotics In Hospitality

There has been significant research regarding the use of robotics in hospitality [Ivanov et al. 2019]. This includes long term assessments of HRI between guests and service robots deployed in hotel domains and how they impact the comfort of guests [Pinillos et al. 2016, Rodriguez-Lizundia et al. 2015, Zalama et al. 2014]. Hotels have been identified as a complex, yet highly commercially viable application of robots [Ivanov and Webster 2017a]. Some researchers have emphasised that hospitality companies are bound to eventually adopt robot-delivered services [Ivanov 2019]. Indeed it can be seen that significant commercial interest exists for robotics in hospitality, with the worlds first robotic hotel "Henn-na hotel" opening in 2015 in Japan [Osawa et al. 2017], with multiple other similar hotels planned to open in 2019. However, the Henn-na hotel focuses on the idea of creating a "unique experience" for the consumers in which the existence of the robot serves mostly as a novelty to attract more guests, although claims a reduction in operating costs by employing the use of robots over large numbers of human workers.

A recent study [Ivanov and Webster 2017b] has made suggestions to hospitality companies to make hotels more "robot inclusive" (the extent to which the design of the domain supports reliable operation of robots [Tan et al. 2016]) to make deployment of robots easier and more financially viable. They describe how less robot inclusive domains require more intelligent robots and vice versa, using hospitality as an example. However, creation of sufficiently robot inclusive environments requires a bespoke design for each such domain which will not generalise to other, even very similar, domains and will cause major issues for engineers and owners resulting from greater design complexity and build costs.

9 Conclusions

This report has summarised the importance for explainable agency for human-robot collaboration and the motivations behind this thesis. It then reviewed and summarised techniques for KRR in robotics, pointing out that there is a distinct need to produce robotic KRR techniques that can elegantly and efficiently represent and reason with both logical and probabilistic representations of incomplete domain knowledge and uncertainty. The fundamental capabilities of an explainable agent and the nature of explanation were then identified from prior literature, before current explainable robotics systems were reviewed and their corresponding capabilities discussed. The goals and contributions of the thesis were then summarised based on lessons learnt from literature reviewed in previous sections, identifying the fundamental open areas for work in the field. The details of two areas of particular interest, action languages and ASP, were then covered, before moving onto the work done so far towards an explainable reasoning and learning architecture for robots.

This report has outlined a high-level description of the current implementation of the ASP based logical KRR architecture. If the proposed architecture is successful it will preform better than previous ASP based explanation generation systems with regards to the first functional ability of an explainable agent described in section 1 and the first element of explainable agency described in section 3.1. However, the resulting answer sets of the current implementation must presently be interpreted "manually" and can only be properly understood by the designer.

Although there exists many varied techniques for KRR in robotics, ASP's ability to represent knowledge in a format that intrinsically supports explanation seems to put it a level above the rest from the perspective of this thesis. This, coupled with the versatility of ASP, the efficiency of modern SAT solvers, and the interfaces they provide to allow them to be easily incorporated into combined systems, makes ASP based KRR systems the most promising technique, at least for high-level planning and reasoning, from the perspective of explainable agency.

10 Plan for Future Work

The architecture currently has some difficulties dealing with the conflict between explaining observations that contradict beliefs with exceptions to initial defaults or the occurrence of exogenous actions. This is because, whilst the current system for specifying initial defaults is elegant and powerful, it makes reasoning about exogenous actions more difficult due to the complexity of specifying the preference hierarchies between the conflicting sets of CR-rules that deal with each of these. This will require some significant work, but is aimed to be ready before the thesis proposal.

The current architecture does not currently support probabilistic reasoning or incorporate the concepts of refinement and zoom introduced in prior work [Zhang et al. 2015, Sridharan et al. 2015, Sridharan and Meadows 2019]. This technique is powerful for many reasons; (1) it substantially reduces the computational complexity of planning with, (2) it allows mixing of the complementary capabilities of both logical and probabilistic planning by separating them distinctly but still allowing a smooth automatic transition between them, (3) it intrinsically supports the ability to represent, reason with and revision different descriptions of incomplete domain knowledge and probabilistic representations of uncertainty at multiple levels of abstraction. An explanation generation system employing this idea with an ASP based high-level logical reasoner inherently supports the ability to extract explanations at multiple levels of abstraction. This is a ability which has been identified in many previous works as being highly critical for explainable agents. This thesis will work towards introducing this concept into the proposed architecture in tome for the thesis proposal.

It is also important that explainable agents be capable of reasoning about goals and intentions, and allow explanations to be provided justifying why to chose to adopt, revise or abandon particular goals [Dannenhauer et al. 2018]. There already exists work towards representing and reasoning about intentional actions on an ASP based intelligent agent [Rocio Gomez and Riley 2018]. This technique will be incorporated into the architecture before the thesis proposal.

The current architecture does not preform any processing of the resulting answer sets of the program. To present intuitive and understandable answers to queries the architecture will thusly need to parse the answer sets according to a query to construct a relevant answer. Therefore, this will be the final piece of work that will be accomplished before the thesis proposal. Providing at least a basic system for producing answers in some constrained natural language that will allow the complexity of the answer to be modified along the fundamental axes of explanations outlined in section 3.2.

References

[Adadi and Berrada 2018] Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160.

[Aker et al. 2012] Aker, E., Patoglu, V., and Erdem, E. (2012). Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics. *IFAC Proceedings Volumes*, 45(22):77–83.

[Alviano et al. 2013] Alviano, M., Dodaro, C., Faber, W., Leone, N., and Ricca, F. (2013). Wasp: a native asp solver based on constraint learning.

[Andersen and Hooker 1994] Andersen, K. A. and Hooker, J. N. (1994). Bayesian logic. *Decis. Support Syst.*, 11(2):191–210.

[Andres et al. 2013] Andres, B., Obermeier, P., Sabuncu, O., Schaub, T., and Rajaratnam, D. (2013). Rosoclingo: A ros package for asp-based robot control. *CoRR*, abs/1307.7398.

[Babb 2015] Babb, J. (2015). Action language bc +.

[Balai 2018] Balai, E. (2018). *SPARC Manual*. Texas Tech University, 2500 Broadway Lubbock TX 79409 USA.

[Balai 2019] Balai, E. (2019). Sparc. `https://github.com/iensen/sparc`.

[Balai et al. 2013a] Balai, E., Gelfond, M., and Zhang, Y. (2013a). Sparc - sorted asp with consistency restoring rules. *CoRR*, abs/1301.1386.

[Balai et al. 2013b] Balai, E., Gelfond, M., and Zhang, Y. (2013b). Towards answer set programming with sorts. In *LPNMR*.

[Balai et al. 2019] Balai, E., Gelfond, M., and Zhang, Y. (2019). P-log: refinement and a new coherency condition. *Annals of Mathematics and Artificial Intelligence*, pages 1–44.

[Balduccini 2007] Balduccini, M. (2007). cr-models: An inference engine for cr-prolog. In *LPNMR*, pages 18–30.

[Balduccini and Gelfond 2003] Balduccini, M. and Gelfond, M. (2003). Logic programs with consistency-restoring rules.

[Balduccini et al. 2014] Balduccini, M., Regli, W. C., and Nguyen, D. N. (2014). An asp-based architecture for autonomous uavs in dynamic environments: Progress report. *CoRR*, abs/1405.1124.

[Baral 2003] Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving with Answer Sets*. Cambridge University Press.

[Borgo et al. 2018] Borgo, R., Cashmore, M., and Magazzeni, D. (2018). Towards providing explanations for ai planner decisions. *CoRR*, abs/1810.06338.

[Calimeri et al. 2005] Calimeri, F., Fabery, W., Leone, N., and Perri, S. (2005). Declarative and computational properties of logic programs with aggregates. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 406–411, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Cashmore et al. 2015] Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtós, N., and Carreras, M. (2015). Rosplan: Planning in the robot operating system. *Proceedings International Conference on Automated Planning and Scheduling, ICAPS*, 2015:333–341.

[Colaco and Sridharan 2015] Colaco, Z. and Sridharan, M. (2015). What happened and why? a mixed architecture for planning and explanation generation in robotics. In *In Australasian Conference on Robotics and Automation (ACRA)*.

[Dannenhauer et al. 2018] Dannenhauer, D., Floyd, M. W., Magazzeni, D., and Aha, D. W. (2018). Explaining rebel behavior in goal reasoning agents.

[DARPA 2016] DARPA (2016). Broad agency announcement: Explainable artificial intelligence (xai). `https://www.darpa.mil/attachments/DARPA-BAA-16-53.pdf`.

[Davis et al. 1962] Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.

[Dell'Armi et al. 2003] Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., and Pfeifer, G. (2003). Aggregate functions in dlv. In *Answer Set Programming*.

[Dodaro et al. 2018] Dodaro, C., Galatà, G., Maratea, M., and Porro, I. (2018). An overview of asp applications in the health-care domain. In *RiCeRcA@AI*IA*.

[Dodaro et al. 2016] Dodaro, C., Gasteiger, P., Leone, N., Musitsch, B., Ricca, F., and Shchekotykhin, K. M. (2016). Combining answer set programming and domain heuristics for solving hard industrial problems (application paper). *CoRR*, abs/1608.00730.

[Doran et al. 2017] Doran, D., Schulz, S., and Besold, T. R. (2017). What does explainable AI really mean? A new conceptualization of perspectives. *CoRR*, abs/1710.00794.

[Dovier et al. 2015] Dovier, A., Formisano, A., Pontelli, E., and Vella, F. (2015). Parallel execution of the asp computation - an investigation on gpus. *CEUR Workshop Proceedings*, 1433.

[Downs et al. 1994] Downs, J., Reichgelt, H., and Shadbolt, N. (1994). Automatic derivation of world update schemes. pages 237–242.

[Došilović et al. 2018] Došilović, F., Brcic, M., and Hlupic, N. (2018). Explainable artificial intelligence: A survey.

[Eiter and Faber 2000] Eiter, T. and Faber, W. R. (2000). Declarative problem-solving using the dlv system.

[Eiter et al. 1998] Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1998). The kr system dlv: progress report, comparisons and benchmarks. In *KR 1998*.

[Erdem et al. 2011] Erdem, E., Erdogan, H., and Öztok, U. (2011). Bioquery-asp: Querying biomedical ontologies using answer set programming. In *RuleML America*.

[Erdem and Patoglu 2012] Erdem, E. and Patoglu, V. (2012). Applications of action languages in cognitive robotics. pages 229–246.

[Erdem and Patoglu 2018] Erdem, E. and Patoglu, V. (2018). Applications of asp in robotics. *KI - Künstliche Intelligenz*, 32:143–149.

[Esra Erdem 2016] Esra Erdem, Michael Gelfond, N. L. (2016). Applications of answer set programming.

[EU 2017a] EU (2017a). Civil law rules on robotics - european parliament resolution of 16 february 2017 with recommendations to the commission on civil law rules on robotics (2015/2103(inl)). `http://www.europarl.europa.eu/sides/getDoc.do?pubRef=-//EP//NONSGML+TA+P8-TA-2017-0051+0+DOC+PDF+V0//EN`.

[EU 2017b] EU (2017b). European parliament committee on legal affairs, "report with recommendations to the commission on civil law rules on robotics" (2015/2103(inl). `http://www.europarl.europa.eu/sides/getDoc.do?pubRef=-//EP//NONSGML+REPORT+A8-2017-0005+0+DOC+PDF+V0//EN`.

[Falkner et al. 2018] Falkner, A., Friedrich, G., Schekotihin, K., Taupe, R., and Teppan, E. C. (2018). Industrial applications of answer set programming. *KI - Kunstliche Intelligenz*, 32(2):165–176.

[Fikes and Nilsson 1971] Fikes, R. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

[Fox et al. 2017] Fox, M., Long, D., and Magazzeni, D. (2017). Explainable planning.

[Francesco Calimeri 2012] Francesco Calimeri, Wolfgang Faber, M. G. G. I. R. K. T. K. N. L. F. R. T. S. (2012). Asp-core-2: Input language format.

[Gebser et al. 2010] Gebser, M., Guziolowski, C., Ivanchev, M., Schaub, T., Siegel, A., Thiele, S., and Veber, P. (2010). Repair and prediction (under inconsistency) in large biological networks with answer set programming.

[Gebser et al. 2015a] Gebser, M., Harrison, A., Kaminski, R., Lifschitz, V., and Schaub, T. (2015a). Abstract gringo. *Theory and Practice of Logic Programming*, 15(4-5):449–463.

[Gebser et al. 2015b] Gebser, M., Janhunen, T., Jost, H., Kaminski, R., and Schaub, T. (2015b). Asp solving for expanding universes. In *LPNMR*.

[Gebser et al. 2019a] Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S., and Wanko, P. (2019a). Clingo python module version 5.3.0. `https://potassco.org/clingo/python-api/current/clingo.html`.

[Gebser et al. 2019b] Gebser, M., Kaminski, R., Kaufmann, B., Lindauer, M., Ostrowski, M., Romero, J., Schaub, T., Thiele, S., and Wanko, P. (2019b). *Potassco User Guide*. University of Potsdam, Am Neuen Palais 10, 14469 Potsdam, Germany, 2nd edition.

[Gebser et al. 2008] Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Thiele, S. (2008). Engineering an incremental asp solver. pages 190–205.

[Gebser et al. 2015c] Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., and Schaub, T. (2015c). Progress in clasp series 3. In *LPNMR*.

[Gebser et al. 2014] Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2014). Clingo = asp + control: Extended report. *CoRR*, abs/1405.3694.

[Gebser et al. 2019c] Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2019c). Multi-shot asp solving with clingo. *TPLP*, 19:27–82.

[Gebser et al. 2011a] Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T., and Schnor, B. (2011a). Cluster-based asp solving with claspar. volume 6645, pages 364–369.

[Gebser et al. 2011b] Gebser, M., Kaminski, R., König, A., and Schaub, T. (2011b). Advances in gringo series 3. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR'11, pages 345–351, Berlin, Heidelberg. Springer-Verlag.

[Gebser et al. 2015d] Gebser, M., Kaminski, R., Obermeier, P., and Schaub, T. (2015d). Ricochet robots reloaded: A case-study in multi-shot asp solving. 9060.

[Gebser et al. 2009] Gebser, M., Kaufmann, B., and Schaub, T. (2009). The conflict-driven answer set solver clasp: Progress report. volume 5753.

[Gebser et al. 2012a] Gebser, M., Kaufmann, B., and Schaub, T. (2012a). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187-188:52–89.

[Gebser et al. 2012b] Gebser, M., Kaufmann, B., and Schaub, T. (2012b). Multi-threaded ASP solving with clasp. *CoRR*, abs/1210.3265.

[Gebser et al. 2018] Gebser, M., Leone, N., Maratea, M., Perri, S., Ricca, F., and Schaub, T. (2018). Evaluation techniques and systems for answer set programming: a survey. In *IJCAI*.

[Gebser et al. 2017] Gebser, M., Maratea, M., and Ricca, F. (2017). The sixth answer set programming competition. *J. Artif. Int. Res.*, 60(1):41–95.

[Gebser et al. 2007] Gebser, M., Schaub, T., and Thiele, S. (2007). Gringo: A new grounder for answer set programming. volume 4483, pages 266–271.

[Gelfond and Inclezan 2009] Gelfond, M. and Inclezan, D. (2009). Yet another modular action language.

[Gelfond and Inclezan 2013] Gelfond, M. and Inclezan, D. (2013). Some properties of system descriptions of ald. *Journal of Applied Non-Classical Logics*, 23:105–120.

[Gelfond and Kahl 2014] Gelfond, M. and Kahl, Y. (2014). *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, New York, NY, USA.

[Gelfond and Lifschitz 1988] Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R., Bowen, and Kenneth, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press.

[Gelfond and Lifschitz 1991] Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385.

[Gelfond and Lifschitz 1998] Gelfond, M. and Lifschitz, V. (1998). Action languages. *Electronic Transactions on Artificial Intelligence*, 3:195–210.

[Giunchiglia et al. 2004] Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., and Turner, H. (2004). Nonmonotonic causal theories. *Artif. Intell.*, 153(1-2):49–104.

[Giunchiglia and Lifschitz 1998] Giunchiglia, E. and Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In *AAAI/IAAI*.

[Grasso et al. 2013] Grasso, G., Leone, N., and Ricca, F. (2013). Answer set programming: Language, applications and development tools. In Faber, W. and Lembo, D., editors, *Web Reasoning and Rule Systems*, pages 19–34, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Hanheide et al. 2017] Hanheide, M., Göbelbecker, M., Horn, G. S., Pronobis, A., Sjöö, K., Aydemir, A., Jensfelt, P., Gretton, C., Dearden, R., Janícek, M., Zender, H., Kruijff, G.-J. M., Hawes, N., and Wyatt, J. L. (2017). Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*, 247:119–150.

[Hastie et al. 2018] Hastie, H. F., Lohan, K. S., Chantler, M. J., Robb, D. A., Ramamoorthy, S., Petrick, R., Vijayakumar, S., and Lane, D. (2018). The ORCA hub: Explainable offshore robotics through intelligent interfaces. *CoRR*, abs/1803.02100.

[Ivanov 2019] Ivanov, S. (2019). Ultimate transformation: How will automation technologies disrupt the travel, tourism and hospitality industries?

[Ivanov et al. 2019] Ivanov, S., Gretzel, U., Berezina, K., Sigala, M., and Webster, C. (2019). Progress on robotics in hospitality and tourism: a review of the literature.

[Ivanov and Webster 2017a] Ivanov, S. and Webster, C. (2017a). Adoption of robots, artificial intelligence and service automation by travel, tourism and hospitality companies – a cost-benefit analysis.

[Ivanov and Webster 2017b] Ivanov, S. and Webster, C. (2017b). Designing robot-friendly hospitality facilities. In *Proceedings of the Scientific Conference, Tourism. Innovations. Strategies*, pages 74–81, Bourgas, Bulgaria.

[Jaumard et al. 2006] Jaumard, B., Fortin, A., Shahriar, I., and Sultana, R. (2006). First order probabilistic logic. pages 341 – 346.

[Kaelbling et al. 1998] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134.

[Kaminski et al. 2017] Kaminski, R., Schaub, T., and Wanko, P. (2017). *A Tutorial on Hybrid Answer Set Solving with clingo*, pages 167–203.

[Keil 2003] Keil, F. C. (2003). Folkscience: Coarse interpretations of a complex reality. *Trends in Cognitive Sciences*, 7(8):368–373.

[Kim and Hinds 2006] Kim, T. and Hinds, P. (2006). Who should i blame? effects of autonomy and transparency on attributions in human-robot interaction. In *ROMAN 2006 - The 15th IEEE International Symposium on Robot and Human Interactive Communication*, pages 80–85.

[Koller and Friedman 2009] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press.

[Langley 2016] Langley, P. (2016). Explainable agency in human-robot interaction.

[Langley 2019] Langley, P. (2019). Explainable, normative, and justified agency.

[Langley et al. 2017] Langley, P., Meadows, B., Sridharan, M., and Choi, D. (2017). Explainable agency for intelligent autonomous systems. In *AAAI*.

[Lee et al. 2013] Lee, J., Lifschitz, V., and Yang, F. (2013). Action language bc: Preliminary report. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

[Lee and Wang 2018] Lee, J. and Wang, J. (2018). A probabilistic extension of action language bc+. *CoRR*, abs/1805.00634.

[Lee 2019] Lee, J. H. (2019). Complementary reinforcement learning towards explainable agents. *CoRR*, abs/1901.00188.

[Leone et al. 2006] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006). The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7:499–562.

[Lifschitz 2008] Lifschitz, V. (2008). What is answer set programming? In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1594–1597.

[Lison 2015] Lison, P. (2015). A hybrid approach to dialogue management based on probabilistic rules. *Comput. Speech Lang.*, 34(1):232–255.

[Marques-Silva and Sakallah 1999] Marques-Silva, J. P. and Sakallah, K. A. (1999). Grasp: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521.

[Meadows et al. 2016] Meadows, B., Sridharan, M., and Colaco, Z. (2016). Towards an explanation generation system for robots: Analysis and recommendations.

[Miller 2019] Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artif. Intell.*, 267:1–38.

[Miller et al. 2017] Miller, T., Howe, P., and Sonenberg, L. (2017). Explainable ai: Beware of inmates running the asylum. *CoRR*, abs/1712.00547.

[Niemelä et al. 2000] Niemelä, I., Simons, P., and Syrjänen, T. (2000). Smodels: A system for answer set programming. *CoRR*, cs.AI/0003033.

[Osawa et al. 2017] Osawa, H., Ema, A., Hattori, H., Akiya, N., Kanzaki, N., Kubo, A., Koyama, T., and Ichise, R. (2017). Analysis of robot hotel: Reconstruction of works with robots. In *26th IEEE International Symposium on Robot and Human Interactive Communication*, pages 219–223.

[Pednault 1989] Pednault, E. P. D. (1989). Adl: exploring the middle ground between strips and the situation calculus. In *KR 1989*.

[Pinillos et al. 2016] Pinillos, R., Marcos, S., Feliz, R., Zalama, E., and Gómez-García-Bermejo, J. (2016). Long-term assessment of a service robot in a hotel environment. *Robotics and Autonomous Systems*, 79:40–57.

[Poole 2000] Poole, D. (2000). Abducing through negation as failure: stable models within the independent choice logic. *J. Log. Program.*, 44:5–35.

[Quigley et al. 2009] Quigley, M., Gerkey, B. P., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system.

[Reiter 1980] Reiter, R. (1980). A logic for default reasoning. *Artif. Intell.*, 13:81–132.

[Ribeiro et al. 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier. In *HLT-NAACL Demos*.

[Richardson and Domingos 2006] Richardson, M. and Domingos, P. (2006). Markov logic networks. *Mach. Learn.*, 62(1-2):107–136.

[Rocio Gomez and Riley 2018] Rocio Gomez, M. S. and Riley, H. (2018). Representing and reasoning with intentional actions on a robot. In *Workshop on Planning and Robotics at International Conference on Automated Planning and Scheduling*, Delft, The Netherlands.

[Rodriguez-Lizundia et al. 2015] Rodriguez-Lizundia, E., Marcos, S., Zalama, E., Gómez-García-Bermejo, J., and Gordaliza, A. (2015). A bellboy robot: Study of the effects of robot behaviour on user engagement and comfort. *Human-Computer Studies*, 82:83–95.

[Sanner and Kersting 2010] Sanner, S. and Kersting, K. (2010). Symbolic dynamic programming for first-order pomdps. volume 2.

[Sequeira et al. 2019] Sequeira, P., Yeh, E., and Gervasio, M. T. (2019). Interestingness elements for explainable reinforcement learning through introspection. In *IUI Workshops*.

[SHAKEY 1972] SHAKEY (1966-1972). Milestone-proposal: Shakey: The world's first mobile, intelligent robot, 1972. `http://ieeemilestones.ethw.org/Milestone-Proposal:Shakey:_The_World%E2%80%99s_First_Mobile,_Intelligent_Robot,_1972`.

[Shani et al. 2013] Shani, G., Pineau, J., and Kaplow, R. (2013). A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51.

[Sheh 2017a] Sheh, R. (2017a). Different xai for different hri. In *AAAI Fall Symposium Series*.

[Sheh 2017b] Sheh, R. (2017b). "why did you do that?" explainable intelligent robots.

[Simon et al. 2019] Simon, O., Neuberg, M., and Tissawi, A. (2019). Robotics in hospitality: Exploring factors facilitating trust building in human-robot frontline teams through lego® serious play®.

[Simons et al. 2002] Simons, P., Niemelá, I., and Soininen, T. (2002). Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1-2):181–234.

[Sreedharan et al. 2017] Sreedharan, S., Chakraborti, T., and Kambhampati, S. (2017). Balancing explicability and explanation in human-aware planning. *CoRR*, abs/1708.00543.

[Sridharan 2016] Sridharan, M. (2016). Towards an architecture for representation, reasoning and learning in human-robot collaboration. In *AAAI Spring Symposium on Enabling Computing Research in Socially Intelligent Human-Robot Interaction*, Stanford, USA.

[Sridharan and Gelfond 2016] Sridharan, M. and Gelfond, M. (2016). Using knowledge representation and reasoning tools in the design of robots. In *KnowProS@IJCAI*.

[Sridharan et al. 2015] Sridharan, M., Gelfond, M., Zhang, S., and Wyatt, J. L. (2015). A refinement-based architecture for knowledge representation and reasoning in robotics. *CoRR*.

[Sridharan and Meadows 2016] Sridharan, M. and Meadows, B. (2016). Should i do that? using relational reinforcement learning and declarative programming to discover domain axioms. *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 252–259.

[Sridharan and Meadows 2018] Sridharan, M. and Meadows, B. (2018). Knowledge representation and interactive learning of domain knowledge for human-robot interaction. In *Workshop on Integrated Planning, Acting and Execution*, ICAPS, Delft, The Netherlands.

[Sridharan and Meadows 2019] Sridharan, M. and Meadows, B. (2019). A theory of explanations for human-robot collaboration.

[Syrjänen 1998] Syrjänen, T. (1998). Implementation of local grounding for logic programs with stable model semantics.

[Syrjänen 2001] Syrjänen, T. (2001). Omega-restricted logic programs. pages 267–279.

[Tan et al. 2016] Tan, N., Mohan, R. E., and Watanabe, A. (2016). Toward a framework for robot-inclusive environments. *Automation in Construction*, 69:68–78.

[Thrun et al. 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[van der Waa et al. 2018] van der Waa, J., van Diggelen, J., van den Bosch, K., and Neerincx, M. A. (2018). Contrastive explanations for reinforcement learning in terms of expected consequences. *CoRR*, abs/1807.08706.

[Wachter et al. 2017a] Wachter, S., Mittelstadt, B., and Floridi, L. (2017a). Transparent, explainable, and accountable ai for robotics. *Science Robotics*, 2(6).

[Wachter et al. 2017b] Wachter, S., Mittelstadt, B., and Floridi, L. (2017b). Why a right to explanation of automated decision-making does not exist in the general data protection regulation. *International Data Privacy Law, 2017*.

[XAI 2017] XAI (2017). *IJCAI-17 Workshop on Explainable AI (XAI)*, volume 1, Melbourne, Australia.

[Zalama et al. 2014] Zalama, E., Gómez-García-Bermejo, J., Marcos, S., Dominguez, S., Feliz, R., Pinillos, R., and López, J. (2014). Sacarino, a service robot in a hotel environment. *Advances in Intelligent Systems and Computing*, 253:3–14.

[Zhang et al. 2017a] Zhang, S., Khandelwal, P., and Stone, P. (2017a). Dynamically constructed (po)mdps for adaptive robot planning. In *AAAI*.

[Zhang et al. 2014] Zhang, S., Sridharan, M., Gelfond, M., and Wyatt, J. (2014). Towards an architecture for knowledge representation and reasoning in robotics. In Beetz, M., Johnston, B., and Williams, M.-A., editors, *Social Robotics*, pages 400–410, Cham. Springer International Publishing.

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

[Zhang and Stone 2015] Zhang, S. and Stone, P. (2015). Corpp: Commonsense reasoning and probabilistic planning, as applied to dialog with a mobile robot. In *AAAI Spring Symposia*.

[Zhang and Stone 2017] Zhang, S. and Stone, P. (2017). Integrated commonsense reasoning and probabilistic planning.

[Zhang et al. 2017b] Zhang, Y. L., Sreedharan, S., Kulkarni, A., Chakraborti, T., Zhuo, H., and Kambhampati, S. (2017b). Plan explicability and predictability for robot task planning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1313–1320.