Hierarchical Abstraction Planning and Diagnostics for Human-Robot Collaboration

PHD THESIS

Oliver Michael Kamperis
School of Computer Science
University of Birmingham
United Kingdom
oxk312@student.bham.ac.uk
Primary Supervisor: Marco Castellani
Secondary Supervisor: Yongjing Wang
Nominal Supervisor: David Parker

June 23, 2020

ABSTRACT

Human-Robot Collaboration (HRC) is an emerging and challenging frontier in robotics involving many interdisciplinary research fields such as classical robotics, psychology and artificial intelligence. HRC has high commercial interest in a variety of applications, particularly those involving repetitive or highly physically intensive tasks, such as re-manufacturing and construction. This thesis focuses of the development of domain-independent HRC planning paradigms robust to the numerous control degrading properties of the real-world. These primarily include the prevalence of non-deterministic factors such as the unpredictable actions of humans and presence of incomplete domain knowledge. This combined with the need for efficiency and safety, makes planning in real-world HRC domains more complex than in highly structured classical domains such as factories and warehouses.

We present a novel Answer Set Programming (ASP) based reasoning system for HRC which performs centralised planning and diagnostics over a hierarchy of abstraction spaces for discrete planning domains with multiple heterogeneous agents. Our system inherently supports reasoning over planning domain descriptions with any arbitrary number of desired abstraction levels. It generates plans consisting of finite deterministic sequences of actions at each available abstraction level by passing the effects of actions planned at the previous abstraction as sub-goals to the next. The resulting set of plans represent hierarchically decomposed sequences of actions containing relational descriptions defining which actions planned at the lower-levels achieve the effects of higher-level actions. At any abstraction level below the most abstract, the planner can generate a partial plan by splitting the planning problem into smaller sub-problems inferred from the plan at the previous abstraction.

The use of ASP guarantees that the planner will always return and the minimal plan will always be found if the planning problem is satisfiable. The ability for ASP to precisely capture defeasible reasoning further allows the system to perform a variety of complex diagnostic reasoning processes. These include default reasoning, a form of commonsense reasoning, used to generate assumptions about the initial domain state when complete knowledge is unavailable, and abductive reasoning to hypothesise possible exogenous actions which can explain observations that contradict expectations.

We aim to demonstrate that our system is applicable to a range of planning problems, particularly those involving complex ordering constraints over execution of actions. The reasoning system has been implemented, and experiments will be run to validate the overall approach and theoretical model.

Contents

1 Introduction

There exist many exciting and prosperous potential applications for new emerging generations of intelligent autonomous robots in the modern world. Robots may soon be deployed into human populated spaces, including our homes and work-places, to allow the automation of almost all laborious and dangerous tasks whose complexity usually reserves them only for humans [Thrun 2004]. The extent of these robots' capabilities grants them great commercial viability and consumer desire, which has already been indicated even by the simple robotic vacuum cleaner [Forlizzi and DiSalvo 2006].

Human-Robot Collaboration (HRC) is the research field relating to the operation of autonomous robots within human populated spaces. HRC usually involves joint cooperative effort between humans and robots towards some shared goal, but can often involve sole operation from robots towards a specific goal with the only human role being supervisory. There exists a diverse range of application domains in which HRC is highly desirable due to the potential to automate (or partially automate) many complex and physically intensive tasks. These include construction sites, re-manufacturing facilities, and residential estates. This however requires robots to be capable of operating efficiently, reliably and safely in close proximity to humans in highly dynamic and non-deterministic real-world environments.

Historically, autonomous robots have been primarily used in highly structured application domains, such as in the manufacturing industry, where the sequence of operations to be performed can be pre-determined by human experts and fixed long before execution commences. The use of robots in such applications generally leads to increased efficiency and reduced operating costs but a reduction in flexibility. In contrast, the unpredictable and highly dynamic nature of real-world domains does not permit the use of fixed action sequences and requires robots to be able to autonomously plan on-line and without the assistance of human experts. Generating and executing plans in the real-world however requires robots to be capable of many complex reasoning processes. Real-world domains do not provide the same guarantees of the safe and productive operation of robots which classical domains are specifically designed to ensure. Particularly, the presence of exogenous actors, such as humans, whose behaviour is usually very difficult to predict, will cause frequent changes to the state unbeknownst to the robots. This is true even when humans are actively assisting robots in a cooperative team. Robots are therefore rarely able to obtain accurate projections of what they will encounter in the future, and any past observations are highly transient, causing existing domain knowledge to become rapidly subject to entropy. It is therefore likely that observations obtained during plan execution will often contradict the robots' expectations determined during plan generation. This may result in plan failure. This volatility in the domain state makes generating complete concrete plans a highly futile endeavour. Furthermore, for any planning problem of non-trivial complexity, it is not feasible to pre-generate even partial plans that account for all possible contingencies.

In order for robots to plan and act in real-world HRC domains they must be able to rapidly construct plans on-line and in the presence of great uncertainty. If plan failure occurs, the robots must re-generate or repair existing plans at run-time, usually with restrictions on available time. Unfortunately, planning problems in the real-world are usually so complex that they become computationally intractable to solve in time commensurate with the need of most practical applications. There are two techniques that a robot can use to overcome this; (1) plan in an abstract representation of the domain reducing overall planning time by ignoring low-relevancy details of the original problem, (2) compute a partial plan into a short future horizon reducing time spent pre-generating plans before execution. The development of robust planning and diagnostics paradigms that have such capabilities however continues to remain an unsolved problem.

This is further complicated by the diversity in the possible practical applications of robotics in the real-world. Problem representations used between application domains usually differ substantially. For example, the strategies a robot might use for a collaborative disassembly task will be very different from those used to inspect a power-line or house-keep in the residential estate. Each application may have a bespoke model used to represent the problem and unique method for finding a solution. It is therefore difficult to develop an universal domain-independent system and modelling language to which the problem representations for any application domain can be transcribed and solved.

2 Thesis Contributions

This thesis aims to develop a domain-independent reasoning system for planning, diagnostics and inference over a hierarchy of abstraction spaces for HRC. Our reasoning system is built in Answer Set Programming (ASP), a declarative non-monotonic logical programming paradigm supporting advanced Knowledge Representation and Reasoning (KRR) capabilities, including defeasible reasoning processes such as default and abductive reasoning [Gelfond and Kahl 2014]. The reasoning capabilities of ASP allow the direct combination of planning and diagnostics into a single unified system. This is advantageous because planning and diagnostics are highly related problems in real-world HRC applications. The use of abstraction for planning in Artificial Intelligence (AI) [Tenenberg 1988] and human problem solving [Timpf et al. 1992] has been studied extensively. These concepts are brought to ASP for the first time to support highly expressive abstraction based reasoning capabilities including hierarchical planning, partial planning and plan repair.

We present a theoretical model and corresponding ASP encoding that inherently supports the ability to reason over any arbitrary number of desired abstractions. This allows the system to generate hierarchical plans consisting of finite sequences of discrete actions over all available levels of abstraction. Partial plans can be generated at any abstraction below the most abstract by splitting the planning problem into sub-problems inferred from the previous abstraction. Planning can be performed with multiple heterogeneous agents, including determining actions that must be performed by human collaborators. The planner is guaranteed to return, and if the planning problem is satisfiable, the resulting plan will be cardinality minimal. When observations obtained during execution contradict expectations determined during planning, the system enters diagnostics to resolve this contradiction and re-generate or repair its previous plan.

The proposed reasoning system builds primarily upon the original work for KRR for intelligent agents in ASP [Gelfond and Kahl 2014]. This provided a simple but effective encoding which solved discrete classical planning and diagnostics problems with greater elaboration tolerance when compared to other popular classical planning paradigms such as STRIPS [Fikes and Nilsson 1971]. We also build upon recent work that expanded this to support refinement based descriptions of a single robot's beliefs at only two levels of abstraction in ASP [Sridharan et al. 2019]. We draw upon many concepts from previous research in hierarchical and abstraction based planning in AI [Sacerdoti 1974, Tenenberg 1988, Knoblock 1990], and research on the use of abstraction by humans when solving complex problems [Newell et al. 1972, Timpf et al. 1992]. Further, we employ concepts that have been demonstrated in abstraction based planning paradigms such as ABSTRIPS [Knoblock et al. 1991, Knoblock 1992] and Hierarchical Task Networks (HTN) [Erol et al. 1994, Erol et al. 1995], but which have not been previously supported by any existing ASP system.

It is important to note that the proposed reasoning system is fully centralised. It performs planning and diagnostics for all available agents in any given domain. Actions are sequentially served to each agent and discretised observations for a minimum of the effects and preconditions of each of the agent's actions are demanded in return. This ensures that the ordering constraints over the execution of the planned actions are properly adhered to whilst allowing the system to remain entirely agnostic to the methods with which its plans are executed at the concrete level or the actual continuous real-world amount of time required to execute them. We aim to demonstrate that this allows the system to be hybridised with a large variety of application specific systems for executing its plans at the concrete level. Further, we hope to show that, due to its foundations in non-monotonic logic and KRR, ASP is very capable of dealing with the large amounts of uncertain information flow that must be handled in complex planning domains such as in construction sites.

2.1 Current Progress

This section describes the novel contributions we have implemented thus far and are the focus of our current work. For most of these contributions the theoretical model has not been fully mathematically defined but has been tested experimentally. There has been significant prior work on ASP based KRR and planning techniques which we mention only briefly here. In Section 5.2 we review this past work in detail and identify flaws which this thesis seeks to solve.

1. **Extension of ASP based planning to support abstraction planning**: Our reasoning system incrementally generates plans at multiple levels of abstraction by passing the effects of actions planned at the previous abstraction as sub-goals to the next. The resulting set of plans represent hierarchically decomposed sequences of actions containing relational descriptions defining which actions planned at the lower-levels achieve the effects of higher-level actions. The initial state knowledge and final desired goal state are provided at the lowest abstraction and are automatically abstracted to all higher levels. This technique ensures plan conformance between abstractions and provides minor heuristic guidance to the solver. We further define conditions under which system descriptions of a planning domain ensure there always exists a possible plan at all abstractions.
    - The system provides inherent support for reasoning over an arbitrary number of desired abstraction levels.
    - Planning in abstract space reduces an exponential space to linear, decreasing planning time exponentially.
    - The proposed methods for planning hierarchically are less brittle than those used by HTN planning due to the reduced dependency of designer provided knowledge regarding how to decompose tasks or actions.
2. **Generalisation of ASP based planning to support centralised planning for teams of multiple agents with heterogeneous capabilities and concurrent actions**: Our reasoning system deals with planning amoungst diverse teams of robots with differing functional abilities and determines actions that must be performed by humans. Planning with multiple agents in ASP has been demonstrated before but has previously required significant application specific knowledge for concurrency [Balduccini et al. 2014, Saribatur et al. 2019].
    - The system provides inherent support for centralised planning in arbitrarily complex multi-agent problems, including those with human participants, in a highly elaboration tolerant and domain-independent manner.
    - The only required application specific knowledge is the specification of each of the agents' capabilities.
    - A set of general conditions are used to restrict the occurrence of concurrent actions to ensure plan validity.

3. **Extension of ASP based diagnostics**: Our reasoning system exposes the effects and preconditions of each of its planned actions to allow the necessary plan-critical observations to be determined externally to the system. Obtained observations are then automatically abstracted to the higher abstractions. When observations contradict the robots' expectations for the precondition of an action the system enters diagnostics to resolve this contradiction. It first attempts to generate exceptions to initial state defaults, a type of commonsense knowledge used to make assumptions about the initial domain state, if this cannot resolve the contradiction it will hypothesise possible exogenous actions which may have lead to the contradiction. When observations contradict the robots' expectations for the effect of an action, the system assumes that the action's effect failed and the action itself is abnormal. Designer provided knowledge can be used to infer the cause of this failure.
    - The system does away with the need for sensing actions and any additional designer provided rules used for abstracting observations which were necessary in past work [Sridharan et al. 2019].
    - It further does away with the need to represent the robots' intentions [Rocio Gomez and Riley 2018] by instead using selective grounding to switch the solver between planning or diagnostic mode.
    - We redefine the notion of a non-Boolean fluent[1] by including a new *continuity axiom* which ensures that all fluents have exactly one value at any given time and all other possible values are false. This axiom ensures that the state representation is always valid and allows more compact modelling of planning domains due to the reduction in the number of necessary domain-dependent rules to ensure validity.
    - We leverage the capabilities of the modern ASP solver Clingo [Gebser et al. 2014] to encode prioritised optimisation statements to allow specification of a preference hierarchy over initial state defaults.

2.2 Future Contributions

The following is a list of contributions we plan to make in the future in order of priority. Particularly, achieving partial planning is a primary goal of this thesis as it is a central motivator for the use of abstraction. Whilst abstraction planning in ASP is novel, it is does not provide much practical benefit without the ability to perform partial planning because a concrete plan is always necessary to begin execution. Partial planning has the potential to massively reduce the time spent pre-planning before execution can commence, making the use of ASP in HRC far more realistically feasible.

1. **Enable the computation of partial plans at any abstraction level below the most abstract level**: In past work, ASP based planners were limited because they must compute full models and thus had to generate a complete plan. For large and complex domains this would often result in plan computation times that were not commensurate with the needs of their application domains. The proposed reasoning system aims to remove this limitation. At all but the highest abstraction level the planner can compute a partial plan in which only a sub-set of the sub-goals from the previous abstraction are committed to the lower levels. This splits the planning problem into smaller sub-problems that are far more computationally tractable to solve at run time.
    - A method is required for deciding the quantity of sub-goals to commit to the next abstraction for planning. In other words, we need some way to choose the size of the sub-problem to compute at each abstraction.
    - A method is also required for limiting the problem space to only that which is most relevant to the given planning sub-problem, this may be inferred from the sub-goals from the plan at the previous abstraction.
2. **Enable plan-repair when plans are invalidated by observations that contradict the robots' beliefs**: Another important motivator behind the use of abstraction is the idea that sufficiently abstract plans become very unlikely to fail. In contrast, because the dynamic non-deterministic nature of real-world HRC domains makes the state highly volatile, concrete plans are likely to fail very often. It is computationally expensive to re-compute an entire concrete plan during execution when unexpected observations are obtained. It is therefore desirable to instead repair an existing plan by re-using valid elements where possible.
    - This requires defining some theoretical model to determine which parts of the ground-level plan are invalidated by a given unexpected observation and require repair, and which parts can be re-used.
    - The hierarchical natural of generated plans may in theory be used to determine whether, and which parts of, abstract plans are also invalidated given an invalidated sub-set of a ground plan.
3. **Provide support for oversubscription planning**: A planning paradigm in which the highest utility sub-set of goals are to be achieved within the available resource and time constraints. This has not been demonstrated before because the solver must compute a model that satisfies all goals.
    - Requires defining conditions for allowing goal rejection and ensuring goal conformance over abstractions.
    - Requires a model for specifying priorities over available goals to optimise goal utility.

[1] The term *fluent* is the name given to a dynamic property of a domain which may change with time, such as object locations.

2.3 Evaluative Domains and Experimental Testing

One of the primary motivations behind this thesis is the desire to achieve domain-independency. It is therefore of great importance to evaluate the overall approach and theoretical model on a variety of application domains. The following listed domains are our proposed evaluative domains. We review more related work on each domain in Section 5.1.

- **Service Robots in Hospitality**: The use of service robots in applications such as healthcare [Roy et al. 2000], tourism [Thrun et al. 1999], and hospitality [Pinillos et al. 2016] has been studied extensively. Service robots are one of the most ubiquitous type of robot and are likely to be prevalent on almost all civilian HRC domains due their potential to automate almost all laborious tasks whose complexity usually reserves them only for humans. In service robotics the main challenge is to adapt to the diverse needs of the many differing individuals they will encounter. We aim to use a hospitality domain staffed by service robot as our initial example domain.
- **Human-Robot Collaborative Disassembly (HRCD)**: In re-manufacturing, HRCD is the field which deals with the semi-autonomous disassembly of End-Of-Life (EOL) products ready for reuse, repair or recycling [Liu et al. 2019]. Automation of disassembly tasks is attractive from both economical and environmental perspectives due to increased efficiency, lower operating costs and reduced waste. However, the diversity of available products and uncertainty in the quality of their components makes complete automation difficult. For example, human assistance may be required when dexterous manipulation is necessary to remove a part. Disassembly Sequence Planning (DSP) is the process of computing the sequence in which components of EOL products should be removed. DSP has received considerable research interest [Zhou et al. 2019] but remains an open problem. We aim to demonstrate that our reasoning system can solve complex sequence planning problems in HRCD. This is problem to which ASP based reasoning approaches has not before been applied.
- **Human-Robot Collaborative Construction (HRCC)**: The construction industry is the seminal field of HRC and has seen the greatest commercial interest to date [Bauer et al. 2008]. The ability for robots to efficiently perform the physically intensive tasks that are so prevalent on construction sites allows for significant increases in productivity and safety. There has been some recent success in the development of automated construction equipment and robotics, although skilled labourers are still required to complete complex tasks such as wiring [Saidi et al. 2016]. Despite this, robots are only rarely used on modern construction sites. This is primarily because construction sites are very poorly structured, and synchronising the large teams of highly specialised robots and unreliable human participants is very challenging. We aim to demonstrate in simulation that our reasoning system can support collaborative planning with heterogeneous robots in complex construction sites.
- **The Blocks World**: The blocks world is classic domain used for testing planning paradigms. This domain will be developed during the writing up period for a online appendix as a tutorial for the proposed system.

In will be rewarding to experimentally test the proposed reasoning system on these domains to obtain statistical data which we can use to validate the overall approach. Initially, we plan to perform testing entirely within simulation, but time permitting we may extend testing to real-world robots. The following is a list of possible experimental tests:

- Determine whether, and the extent to which, the sub-goals used to ensure conformance of plans between different abstraction levels in the hierarchy provide any significant reduction in planning time.
- The correlation between quantity of domain rules/entity constants and planning time at various abstractions.
- Determine how average planning times change between single-robot and multi-robot planning problems, as well as those with human participants, on planning problems of various complexity.
- How different possible methods of specifying the cost functions, such as those proposed by [Lee et al. 2013], over actions effect the trade-off between plan computation time and the quality of the resulting plans.
- Determine average time saved before plan execution can commence through the use of partial planning.
- Determine whether, and the extent to which, plan repair provides any significant reduction in time spent re-planning when plans are invalidated by observations that contradict expectations.

2.4 Possible Extension Tasks

There exist a number of possible extensions to the current proposed contributions. These are given below in no particular order. This is not an exhaustive list and it is likely that only a sub-set of these extensions may be realised.

- Implement a decentralised reasoning system coupled with the current one to enable network-aware reasoning.
- Support searching for objects, reasoning about object existence, and inferring class types from object attributes.
- Support mapping of domains with unknown structure, and discovery and revisions to the static domain state.

3 Changes since the Thesis Proposal

The focus and contributions of this thesis have clearly changed significantly since the thesis proposal. I no longer plan to have explainability as the main focus of this thesis and it is now very unlikely to feature in it. The change comes primarily due to the change of supervisors and my experience my previous supervisor which has made me lose most of my enthusiasm for the subject. My previous supervisor and I disagreed on a large variety of things related to the original thesis contributions including the main direction of the work, the major open problems to be addressed by the thesis, or the way these should be tackled. We reached an impasse whereby we could no longer form a mutual decision regarding our plan of future work. It was clear to me at this point that the previous proposal was no longer workable. My previous supervisor is also heavily invested in the work in explainability so I felt to best to let him pursue his own path on that.

I have moved the direction of my work towards that which I find much more interesting and am far more motivated to actively pursue. As described above, this work involves extending previous ASP based systems for planning and diagnostics to inherently support reasoning over an arbitrary number abstractions of a state space for planning problems in HRC applications. Since the project is now of my own design my progress has accelerated significantly and I look forward to applying my reasoning system to HRCD and HRCC applications to which ASP based reasoning techniques have not before been applied. I believe the new contributions I am proposing are substantially more novel than the those presented in the original thesis proposal, as they overcome more open problems in robotics. The nature of the proposed contributions also make their application to real world robots more realistically feasible, and the experimental analysis of the effectiveness of the overall approach easier and more objective to evaluate.

4 Plan of Future Work

The current plan of future work up until the end of year 3 of this thesis is represented in Figure 1. At present, a significant amount of work has already been done towards implementing an initial prototype of the proposed reasoning system. However, much of the theory has not been mathematically defined. After the RSMG 4 meeting we will therefore work primarily towards developing the theoretical model of the work completed thus far. As the theory begins to develop we intend to begin writing it up into a publishable paper whose primary contribution will be the intrinsic support for abstraction based reasoning in ASP. This paper will demonstrate either the office domain presented in Section 7 or a hospitality domain containing service robots. This will be presented in the RSMG 5 meeting in October 2020.

Following the RSMG 5 we will begin work towards developing an ASP compatible model defining and solving HRCD sequence planning problems. It is not yet clear what the complexity of the problem we will tackle will be, exactly how this problem will be encoded, and whether experiments will occur on real-world robots or in simulation. Hence we have allocated significant time towards development of this domain. We will evaluate the effectiveness of our new approach against previous approaches which used methods such as the Bees algorithm [Laili et al. 2019, Xu et al. 2020]. This work will be written up into a second publishable paper and presented in the RSMG 6 meeting in April 2021.

Following the RSMG 6 we will work towards developing the theory for the contributions given in Section 2.2. We plan to develop a construction domain to demonstrate the most powerful contributions of this thesis, partial planning, plan repair, and over subscription planning. HRCC is the most complex and ambitious of the proposed domains but its high commercial interest motivates the application of our novel approach to this important field. This work will be written up into a third publishable paper and presented in the RSMG 7 meeting in November 2021.

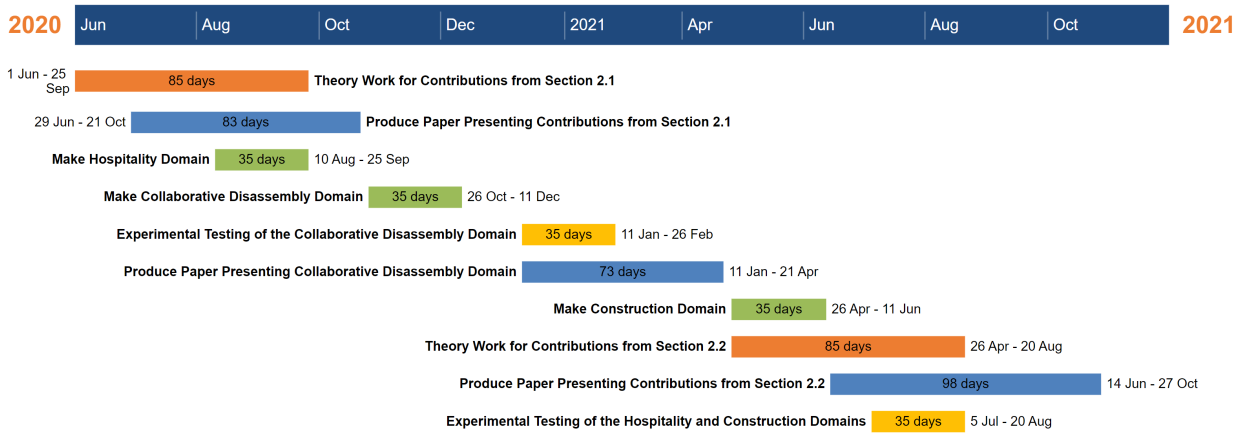

Figure 1: Plan of future work up until the end of year 3.

5 Literature Review

This section reviews literature related to the core topics of this thesis. The review is not exhaustive and but the principal papers are highlighted to overview the main points of interest. First, in Section 5.1 we discuss HRC and literature related to our chosen evaluative domains as listed in Section 2.3. Then, in Section 5.2 we review past work and progress towards planning with ASP. Finally, in Section 5.3 hierarchical abstraction based planning paradigms are reviewed.

5.1 Human-Robot Collaboration

In this thesis we refer to HRC as the act of humans and robots working together as part of a team, usually in close proximity to one another [Bauer et al. 2008]. HRC does not refer only to human-robot cooperation, which is strictly the act of humans labouring jointly with robots to actively provide physical assistance in a shared task. Cooperation is a team based effort towards common goals, in which humans are seen as agents that are available to perform actions towards the accomplishment of these goals [Cohen and Levesque 1991]. In HRC the human effort may however be entirely supervisory, in which case the human is seen as a commander who assigns goals and is not an agent available for performing actions. HRC is also not the same as Human-Robot Interaction (HRI), which is a much more general term, which includes HRC, but refers to any form of interaction between humans and robots [Grosz 1996, Thrun 2004].

Planning in HRC is about coordinating and synchronising the actions of the available agents towards achieving a set of desired goals. Most commonly, the collaborating humans are mostly unavailable or unwilling to assist in completion of the goal, and are instead a diverse group of commanding humans with varying needs and desires from the robots. This situation is prevalent in service robotics domains such as residential estates, hospitality or healthcare. When humans are actively cooperatively with robots, the robots can plan what tasks must be completed by the participating humans rather than themselves. This situation is most common in well structured domains such as manufacturing facilities. In other cases, the robots may often need the assistance of humans but will not be able to predict the intentions/contributions of the humans towards the goal and will need to cope with the uncertainty in their actions. This situation will occur often in unstructured domains such as construction sites were delegating tasks to available humans is less reliable.

5.1.1 Service Robots

Service robotics have been defined as robots that "perform useful service to human activities in an every-day environment" [Kawamura et al. 1996]. Arguably the seminal application for service robotics is in the assistance of the elderly in nursing homes and healthcare centres [Pineau et al. 2003, Roy et al. 2000]. This seems motivated by the desire to allow the elderly to age with decency, particularly when they become less capable of caring for themselves. The currently most prevalent form of service robot is however the common household robotic vacuum cleaner, such as the Roomba [Forlizzi and DiSalvo 2006]. The capabilities of domestic service robots such as the Roomba are currently very limited. The human desire to automate laborious tasks however motivates the development of more capable service robots to be introduced into our homes and workplaces [Thrun 2004]. The main challenges impeding this goal are in the diversity of each domain, the variance in the needs of inhabiting humans [Roy et al. 2000], the need to ensure human safety [Traver et al. 2000], and the fact that human populated spaces are usually not "robot-inclusive" [Tan et al. 2016].

The use of robotics in hospitality has seen a particularly large amount of interest [Ivanov et al. 2019]. Hospitality domains, such as hotels and cruse liners, are a commercially lucrative application of robotics [Ivanov and Webster 2017], with some emphasising that hospitality companies are bound to eventually deliver the majority of their services through robots [Ivanov 2019]. The opening of the worlds first robotic hotel "Henn-na hotel" in 2015 [Osawa et al. 2017] would seem to support this claim. Long term assessments of HRI between guests and service robots deployed in hotels and how they impact upon the comfort of guests have already been made [Pinillos et al. 2016, Rodriguez-Lizundia et al. 2015, Zalama et al. 2014]. Trust has been identified as the primary prerequisite for successful HRC in hospitality and that performance, operating proximity and physical appearance are the main factors affecting trust [Simon et al. 2019].

5.1.2 Human-Robot Collaborative Disassembly

Re-manufacturing is the field associated with the recovery, reuse and recycling of EOL products and is an important process towards improving economic and environmental sustainability [Liu et al. 2019]. Disassembly is the first step of re-manufacturing in which products are reduced to their component parts [Zhou et al. 2019]. There has been significant work towards automating disassembly tasks to reduce the dependency on costly and low efficiency human labour. Unfortunately, automating disassembly tasks is challenging because there are large variances in the frequency and quantity of each type of returned product, as well as significant uncertainty in the condition of their component parts [Huang et al. 2020]. Human-Robot Collaborative Disassembly (HRCD) attempts to overcome these problems through a semi-autonomous approach which sits in a middle-ground in terms of cost, efficiency and flexibility [Djuric et al. 2016].

In manufacturing, robot assisted disassembly tasks have historically been primarily statically scheduled. This means the order of procedures (or actions) to be executed by each robot is pre-determined and fixed before operation. However, it is common for such schedules to fail because component degradation can often make parts impossible to remove [Colledani and Battaïa 2016, Abdullah et al. 2015]. Disassembly Sequence Planning (DSP) is the field of research which deals with dynamically planning the sequence in which components of a EOL product should be removed [Zhou et al. 2019]. DSP is challenging because products often have interlocking parts which means that many of the possible disassembly sequences are not valid, finding a valid sequence requires identifying sub-assemblies that can be removed to break the interlock [Lan et al. 2018]. Numerous approaches from AI have been applied to solving DSP problems, including Particle Swarm Optimisation (PSO) [Yeh 2012, Zhang and Zhang 2009], Ant Colony Optimisation (ACO) [Agrawal and Tiwari 2008, Zhang et al. 2007], and various genetic algorithms [Hui et al. 2008, Kongar and Gupta 2006]. However, non-monotonic KRR techniques such as ASP have not before been applied to DSP.

5.1.3 Human-Robot Collaborative Construction

The construction industry is one of the largest in the world, accounting for 10-20% of GNP in the majority of countries [Elattar 2008]. Mechanisation and automation of construction sites is desirable due the ability for machines to perform physically difficult and dangerous tasks at speed and with high precision [Kosuge et al. 2000]. Historically however, it has been very challenging to use fully autonomous robotic teams in construction sites because robots are poor at "information-intensive" tasks such as inspection and planning [Everett and Slocum 1994]. As such, must current autonomous robots used in construction sites are decentralised and highly specialised, with individual teams of human expert operators supervising each robot [Devadass et al. 2019]. This technique, called Human-Robot Collaborative Construction (HRCC), is effective and has lead to significant increases in productivity in construction sites.

The lack of interoperability between different HRCC teams (or individual robots) does however make centralised planning and scheduling of construction jobs challenging [Saidi et al. 2016]. Autonomous robots need to be aware of the current and predicted future states of the site to plan effectively. Unfortunately, this information may be known by human workers, but is rarely known by the robots, leading to the high dependency of robots on their human operators. The lack of reliable knowledge exchange, management, and monitoring tools for construction sites is the most major issue limiting HRCC. KRR techniques such as ASP may however be well suited to overcoming these issues.

5.2 Planning with Answer Set Programming

In broad terms, the task of planning requires computing a sequence of actions, which if executed in order will transition the initial state to some desired goal state [Ghallab et al. 2004]. There exist many methods for reasoning about actions and change in robotics to compute such plans. Many of which are based upon first-order logics which no not support defeasible reasoning processes. ASP overcomes by supporting non-monotonic logical reasoning processes including abductive and default reasoning, a form of commonsense reasoning [Baral 2003, Lifschitz 2008]. ASP based KRR grants four primary benefits for planning; (1) elaboration tolerance of the modelling language, (2) guaranteed to always return, (3) always produces provably optimal plans, (4) ability to directly combine planning with diagnostics.

In ASP, planning domains are represented as a *program* whose stable models, called *answer sets*, represent possible plans that satisfy the given planning problem [Lifschitz 2002, Gelfond and Kahl 2014]. The program consists of a set of rules (or axioms) defining, amoungst other things, the effects and preconditions of the agents' actions and constraints on the domain state. It is a type of planning by satisfiability, where a satisfiable plan must consist of a sequence of actions and corresponding domain states at each step must simultaneously satisfy all rules of the program. There is no universal theoretical model for planning in ASP because ASP it a general problem solving tool, as such, planning domain descriptions are usually encoded into an intermediate language which is then transcribed to a corresponding ASP encoding. For a general overview of ASP see Appendix A or for a comprehensive overview see [Gebser et al. 2012].

5.2.1 Past Work

ASP has been applied to a variety of different application domains involving planning amoungst multiple robots in human populated areas [Esra Erdem 2016]. This includes a system for planning with a team of two collaborative housekeeping robots [Aker et al. 2012] which incorporated commonsense knowledge automatically obtained from ConceptNet [Speer et al. 2017] to determine the most likely location of objects in a given domain. They integrated ASP with hybrid reasoning approaches by using external function calls to probabilistic motion planners to perform feasibility checks of locomotive plans in continuous space and estimation of the duration of actions. This work however did not include human collaborators and it is unclear whether their system supports heterogeneous robots or how they optimise plans over multiple robots. Their later work [Saribatur et al. 2019] did however extended this to provide a more concrete system for cooperative planning with heterogeneous robots in a cognitive toy factory.

In other work [Zhang et al. 2015], ASP planning was combined with the probabilistic sequential decision making algorithm Partially Observable Markov Decision Process (POMDP) [Thrun et al. 2005] for visually localising and reasoning about the existence of objects in an office domain. ASP planning domain descriptions where written in action language AL_d, which was developed specifically as a high level notation for ASP programs [Gelfond and Inclezan 2013]. Their system used a CR-Prolog solver [Balai et al. 2013], an extension of ASP including consistency restoring rules [Balduccini and Gelfond 2003], to allow the robot to reason about exogenous changes in the domain state, but not simultaneously with commonsense domain knowledge. Other work attempted to overcome this limitation to allowing the system to switch between commonsense or abductive reasoning modes [Rocio Gomez and Riley 2018]. There has further been work towards automatically learning previously unknown AL_d planning domain rules in ASP with relational reinforcement learning techniques [Sridharan and Meadows 2016, Sridharan and Meadows 2018].

Later work used the more expressive action language BC [Lee et al. 2013] to encode ASP planning domain descriptions [Chen et al. 2016] for a service robot operating in a dynamic human populated domain. Action language BC was similarly used for a multi-robot platform for HRI [Khandelwal et al. 2017] aimed at allowing service robots to understand human intentions during interaction. Differing approaches have combined P-Log [Baral et al. 2009], a probabilistic extension of ASP, with POMDPs, as applied to dialog with a mobile robot [Zhang and Stone 2015]. Which was later expanded into the Interleaved Commonsense Reasoning and Probabilistic Planning system [Zhang et al. 2017, Zhang and Stone 2017] which 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. A system called ROSoClingo [Andres et al. 2013] has also been developed which integrates the ASP solver Clingo [Gebser et al. 2014] into the popular Robot Operating System (ROS) [Quigley et al. 2009]. This was demonstrated in a planning problem with a mobile robot delivering items in an office space [Andres et al. 2015]. This work brought ASP to a general robotics framework but did not deal with diagnostics or multi-robot problems.

5.2.2 Limitations

Despite the significant work on ASP based planning paradigms, there does not exists a precise theoretical model or grounded system which fully handles the complexity of HRC. There is no general theory for planning with multiple robots or ensuring that the occurrence of concurrent actions do not lead to an invalid state, with past work requiring significant designer provided domain knowledge to ensure this. This seems caused by the variety of action languages in use, differences in their expressiveness, and a lack of consensus about how to encode planning problems in ASP. Whilst cooperative planning has been demonstrated between multiple robots, it has not be demonstrated with human participants, a more challenging problem because there is no guarantee that humans will perform the desired actions. In past work, humans were always seen as exogenous actors, to enable HRC in ASP we need to define a theory for planning with human participants as agents available to execute actions but be robust to the uncertainty in their behaviour.

One of the most major limitations of ASP based planning is caused by the inherent nature of an answer set that requires the solver to find a full model. This means that all goals must be achievable, otherwise the program is unsatisfiable, and the planning problem cannot be solved. This has two important implications:

1. **Only complete plans can be generated**: Intuitively, the ASP solver cannot yield a model until it has found a complete answer set. This means that if there is a limit on the available time to compute a complete plan, the solver cannot return a partial plan. It is also not possible to expose its current progress towards computing the plan to allow the robots to begin operation and return to computing the remainder later. Further, even if a model is found, the search space must be exhausted to guarantee minimality of the action cost of the plan.
2. **Only classical planning can be performed**: All goals must be simultaneously satisfied in the state at the maximum planning horizon. Where the maximum planning horizon must be specified prior to making the solver call. This means that *oversubscription* planning cannot be performed. In oversubscription planning, the objective is to satisfy the highest utility sub-set of goals within the available resource and time constraints. This primarily demands achieving the greatest number of goals within the available planning horizon.

5.3 Hierarchical and Abstraction based Planning

The use of abstraction has been studied extensively and is widely accepted as an advantageous technique for a variety of reasoning processes used by humans and in AI [Newell et al. 1972]. Humans often employ abstraction extensively when solving complex problems such as planning actions and providing explanations [Giunchiglia and Walsh 1992]. Through abstraction one is able to focus on only that which is most relevant to the problem at the current time. Details of a problem which are not currently pertinent can be neglected partially or entirely to reduce its complexity or assumptions can be made to cope with uncertainty or unknowns [Timpf et al. 1992]. This technique allows one to find an abstract problem solution quickly and easily, which holds whilst our assumptions do and the details remain irrelevant.

Real-world robotics planning problems are difficult because the search space is usually too large that it becomes computationally intractable to solve in time commensurate with the needs of most practical applications. It is however well known that the use of abstraction can reduce an exponential search space into a linear one, reducing planning times exponentially [Knoblock 1990]. In abstraction planning, the agents are usually given descriptions of there domain at at least two different levels of abstraction. The problem is first solved in an abstract space and the resulting solution is then used to guide the generation of a solution in the concrete space. The abstract solution can further be used to split the original problem into independent sub-problems, allowing the computation of partial plans [Knoblock et al. 1991].

A simple example of the use of abstraction by a planning agent may involve the generation of an abstract plan consisting of a sequence of way-points between which to travel to reach a desired destination. Initially it does not need to decide exactly how to traverse between each given way-point. The assumption the agent makes is that there exists at least one traversable path between each of these way-points. When the agent proceeds to execute this plan, it will then continuously generate a more concrete plan in a short horizon into the future, updating this plan as it obtains more knowledge about its environment. Abstraction based planning also proves useful because sufficiently abstract plans become very unlikely to fail. In this example, the way-point based plan only fails if our only assumption is untrue. Therefore, even if it becomes necessary to revise the concrete plan, revisions to the abstract plan are rarely necessary.

A large variety of hierarchical and abstraction based planners already exist [Ghallab et al. 2016]. Hierarchical Task Network (HTN) planning is the branch of AI planning which deals with hierarchies [Georgievski and Aiello 2015]. In HTN planning, tasks and actions are given to the planner in a hierarchy of different refinements, equivalent to abstractions. Domain-dependent methods act as descriptions of possible strategies for accomplishing abstract tasks via the completion of ground actions. HTN planning does not compute plans over multiple abstractions and whilst execution traces can be represented as a decomposition tree, the final plan is still a single sequence of individual actions at the ground level. See Section 5.3.1 for a brief description of HTN planning. ABSTRIPS, an extension of the classical planning language STRIPS [Fikes and Nilsson 1971], which provides support for abstraction, is another well known paradigm for abstraction planning [Knoblock 1992, Giunchiglia 1999]. In abstraction planners such as ABSTRIPS, the planner is instead provided with a hierarchy of multiple complete domain descriptions at different abstractions [Sacerdoti 1974]. In ABSTRIPS, there are only two abstractions which differ only in the preconditions of their operators. ABSTRIPS first finds a solution to the planning problem in the abstract space which is then used to generate a "skeleton" plan at the ground level. The planner is then invoked recursively to solve the individual sub-problems defined by this skeleton. The resulting set of plans contain within them the decomposition trees for all abstract actions and the necessary relations to re-construct them. This is however some debate over the effectiveness of abstraction planning because the heuristic effect provided by the abstract solution is often insignificant [Hoffmann et al. 2006]. The primary motivator behind abstraction planning is thus that of partial planning. Abstraction planning of this nature is more complex than HTN planning and is also the foundation of our own planner, so we detail the semantics later in Section 6.2.

5.3.1 Hierarchical Task Network Planning

Hierarchical Task Networks (HTNs) are a planning paradigm in which the objective is not to achieve some goals, but rather to preform some set of tasks [Erol et al. 1994, Ghallab et al. 2004]. A HTN contains primitive operators similar to those used in STRIPS planning but also a set of *methods* which may be used to decompose tasks into a sequence of sub-tasks. The central idea is that a task is recursively decomposed into smaller and smaller sub-tasks tasks until only a sequence of primitive, directly executable, tasks remain. HTN planning is considerably more expressive than classical planning because classical planning cannot express undecidable planning problems [Ghallab et al. 2004]. However, a common criticism of HTN planning is its brittleness caused by the critical reliance on their methods being complete and correct. The following provides a very brief description of the methodology behind HTN planning:

- A HTN planning problem is defined by the tuple $P = \langle s_0, W_0, D \rangle$,
- A HTN domain is defined by the tuple $D = \langle O, M \rangle$ and,
- A task network by the tuple $W = \langle U, C \rangle$.

Where O is a set of operators (primitive and non-primitive tasks), M is a set of methods, U is a set of task nodes, C is a set of ordering constraints over M, s_0 is the initial state and W_0 is the initial task network. The output is a task network containing only a sequence of primitive directly executable tasks. This is obtained by recursively decomposing the tasks in W_0 using methods from M. Each method $m \in M$ accomplishes a single non-primitive task $task(m)$. m contains a set of sub-tasks $subtasks(m)$ and order constraints $constr(m)$ over these tasks. If these sub-tasks are completed in an order which satisfies these constraints then $task(m)$ is complete. Intuitively, the final task network is obtained from the initial by replacing tasks from W_0 with sub-tasks from methods in M until only primitive tasks remain. The full implementation of HTN planning allows arbitrary ordering constrains over task networks including precedence (equivalent to edges of a task network) and before constraints (task preconditions) [Ghallab et al. 2004].

6 The Proposed Reasoning System

The proposed reasoning system overcomes many of the shortcomings of previous ASP based systems detailed in Section 5.2. The most central contribution is that of enabling the system with the integral ability to reason over any arbitrary number of abstraction spaces. Its planning domain modelling capabilities are significantly more expressive than recent work upon which it is founded [Sridharan et al. 2019] and the original work which introduced the concept of planning in ASP [Gelfond and Kahl 2014]. This has required us to substantially modify the original syntax used to model planning domains. In previous work action languages were used as the sole modelling tool for planning domains and the ASP system was implemented such as to enforce the semantics of that language.

We have instead modelled the planning domain in terms of logic rules and statements which can be encoded directly into ASP. Despite this, it is still possible to translate action languages such as AL_d or BC into the proposed system by representing planning domain descriptions over multiple abstractions as a hierarchy of individual action language domain descriptions. However, this is technique is both crude and poorly defined. In the future, we will need to redefine or extend an existing action language to precisely support the reasoning capabilities of our system. We do make use of AL_d to write domain rules for our examples given in Section 7 due to the simplicity of the language.

6.1 System Architecture

The proposed reasoning system is architectured to be modular in nature. It is split into six individual modules based on their distinct functional role within the system. This concept takes advantage of the ASP solver Clingo's program parts described in section 3.1.15 of [Gebser et al. 2019]. These program parts allow the ASP program to be partitioned into individual pieces which may be selectively grounded and solved. This allows the reasoning "mode" of the system to be easily switched between planning and diagnostics. The reasoning system is split into the following six modules:

1. **ASP Controller**: This is the "master" that governs the control flow between all other modules.
2. **Planning Domain Description**: This module includes the application specific domain description.
3. **Recorded History**: This module includes a set of statements defining the observations made by the robot.
4. **Base Module**: This module includes a set of general rules used by all solver calls and system descriptions.
5. **Planning Module**: This module computes minimal and preferable plans, such that a maximal number of goals are satisfied in the minimum possible number of steps and preforms optimisation of cost functions.
6. **Diagnostics Module**: This module solves diagnostics problems through a number of defeasible reasoning processes, including default and abductive reasoning, to explain observations that contradict expectations.

Figure 2 shows the dependencies between these modules and the control flow of the reasoning system. The high-level informal Algorithm 1 describes the basic execution loop of the reasoning system when generating and executing plans. Figure 3 displays a rough concept drawings of how the system may handle centralised reasoning and knowledge transfer for teams of multiple agents and between different types of human actors.

Algorithm 1: High-Level Architecture Execution Loop

```
Invoke Diagnostics - Resolve a consistent model of the initial state;
Invoke Planner - Compute an initial minimal and complete plan that satisfies all goals for all abstractions;
if ASP solver returned UNSAT then
    return Program inconsistent or goals unsatisfiable
end
while Goals not completed do
    Commit observations for the effects of previous actions and preconditions of next actions to history;
    if Observations contradict beliefs then
        Invoke Diagnostics - Find a consistent model of the recorded history to explain contradictions;
        Invoke Planner - Compute a plan that satisfies a maximal number of possible goals;
        if ASP solver returned UNSAT then
            return Failure
        end
    end
    Execute next actions in the plan and commit corresponding records to history;
end
return Success
```

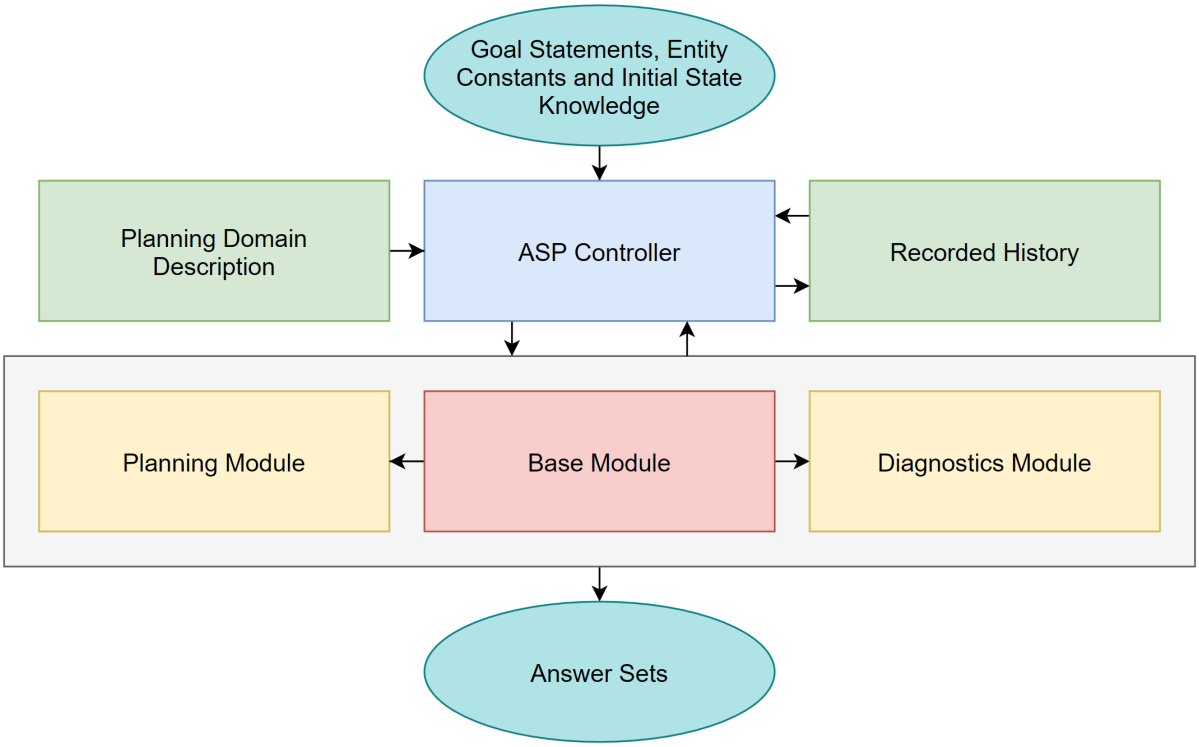

Figure 2: Diagram of the reasoning system. It takes as input a set of goal statements, object constants, and initial state knowledge obtained from various sources. Outputs are answer sets containing abstract plans. The ASP controller manages the control flow between various components, including; loading and grounding the domain descriptions, maintaining the recorded history of observations, and invoking the relevant domain-independent modules to solve planning and diagnostics problems. The domain-independent modules within the grey box include the planning, diagnostics and base modules. The base module is always invoked during solver calls along with either the planning module or diagnostics module, but never both simultaneously, to solve planning or diagnostics problems respectively.

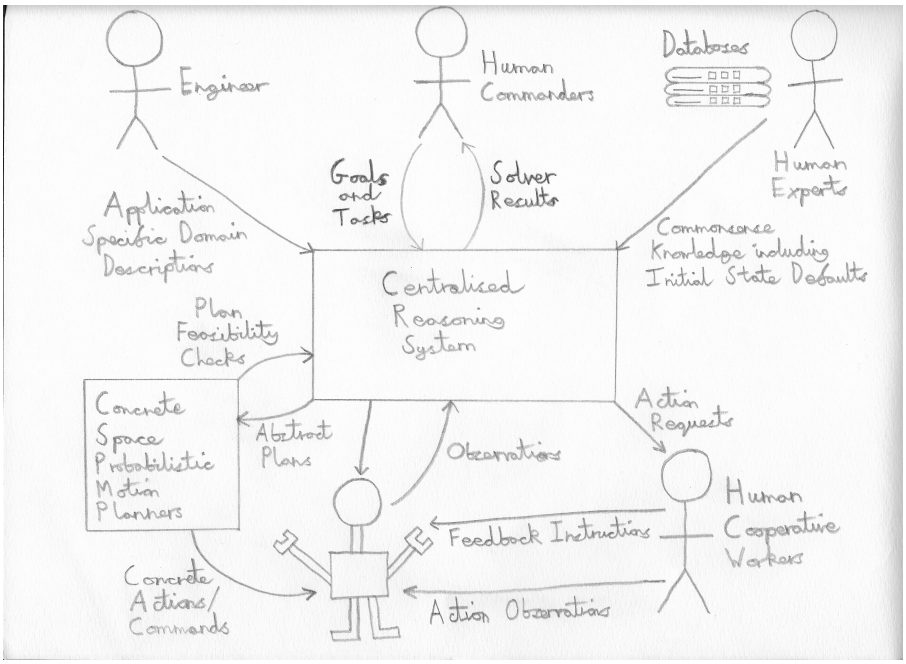

Figure 3: A rough concept drawing of the system-to-agent networking diagram displaying the various knowledge sources and information transfer between different possible types of human and robotics actors in the planning domain.

6.2 Domain Descriptions over Multiple Abstractions

A planning domain description is a formal specification of a dynamic planning domain which defines its *state transition diagram*. A state transition diagram is a directed graph whose nodes correspond to unique domains states and whose arcs are labelled by sets of actions whose effects change the state between those connected by that arc. A sequence of state transitions through such a diagram is called a *trajectory* and is defined by a sequence of sets of actions (i.e. a plan) which label the arcs of the path traced by the trajectory. Solving a planning problem involves finding a trajectory through a state transition diagram whose path transitions the state from the initial to the goal. A planning domain description represents all its dynamic and static properties, the features present within it, the actions of the available planning agents, and the rules that define them. We informally define a planning domain description as follows:

Definition 6.1 (Planning Domain Description) *A sorted hierarchy of collections of application domain specific logic rules defining the dynamic behaviour of the given domain at each desired level of abstraction. An additional hierarchy of collections of abstraction rules define the mapping between each adjacent pair of abstraction levels. This provides a description of a dynamic planning domain over a abstraction hierarchy of strongly linked state transition diagrams.*

More formally, a planning domain description DD^N over abstraction levels 1 to $N \in \mathbb{N}$, where N is referred to as the *maximum abstraction level*, is defined by the three tuple:

$$DD^N = \langle DR^{1,N}, \Sigma^{1,N}, M^{1,N-1} \rangle \tag{1}$$

Where $DR^{1,N}$ is a abstraction hierarchy of collections of domain specific logic rules $DR^l \; \forall \, l \in [1, N]$ defining the transition diagrams $\tau(DR^l, \Sigma^{l,N}, \Psi)$ restricted by the sort definitions $\Sigma^{l,N}$ over the same range, where Ψ are the available entity constants, and $M^{1,N-1}$ is the mapping between all adjacent pairs of abstraction levels in the hierarchy defined by $DR^{1,N}$ such that there is a mapping $M^l \; \forall \, l \in [1, N-1]$ defining the mapping between levels l and $l+1$.

The collection of domain rules DR^1 at abstraction level 1 are said to to be at the *ground level* and must always be present in all domain descriptions. In general, DR^1 should be designed first as the engineer should be careful to ensure that the ground level is sufficiently expressive to handle the planning problem specific to that application. The ground level is then seen as the *original problem* from which abstractions are created through a number of simplifications of which we discuss below. Each abstraction of the ground description is assigned a unique number $l \in \mathbb{N}$ where $l \geq 1$. The abstraction levels in the hierarchy must be contiguous such that $DR^i \in DR^{1,N} \; \forall \, i \in \mathbb{N}$ where $1 \leq i < N$. See Figure 4 for an abstract visual representation of a planning domain description defined over multiple abstraction spaces.

The sort definitions $\Sigma^{i,N}$ define the class hierarchy of tangible entities which may exist in a domain, its static and dynamic properties, and the actions of the available types of planning agent, as defined by the five tuple:

$$\Sigma^{i,N} = \langle C, H^{i,N}, S^{i,N}, F^{i,N}, A^{i,N} \rangle \tag{2}$$

Where C is a set of domain specific class names, $H^{i,N}$ is a hierarchy of directed acyclic graphs representing the class hierarchy at each available abstraction level $H^l \; \forall \, l \in [i, N-1]$ whose nodes are labelled by class names from C and whose arcs $\langle c_0, c_1 \rangle$ indicate that c_0 is a sub-sort of c_1, and $S^{i,N}, F^{i,N}, A^{i,N}$ are sets of function symbols containing *statics*, *fluents* and *actions* respectively. Statics and fluents are *domain properties*. The value of fluents are dynamic and can be changed by actions. The value of statics cannot be changed by actions. Fluent domain properties are partitioned into *inertial fluents* and *defined fluents*. Inertial fluents obey the law of inertia such that their value will not change without cause but can be changed directly or indirectly by actions. Defined fluents cannot be changed directly by actions and are instead defined in terms of other fluents, but therefore may be changed indirectly by actions.

The generation of the hierarchy of class hierarchies $H^{i,N}$ over different abstractions is fully automatic. Unlike abstractions of the state space, which are defined in an upwards manner, i.e. abstract spaces are defined in terms of ground spaces, the class hierarchies are generated in a downwards manner. The generation of the hierarchies can be thought of as the discovery of class types from C and their respective entity instances from Ψ as the details of the ground level planning problem are progressively uncovered as the abstraction level is incrementally reduced. The class hierarchy H^l at the lower abstraction level inherits all instances from the class hierarchy H^{l+1} from the previous abstraction level whose types are not overridden by newly discovery classes. The mathematical theory defining this process is complex and not currently precisely defined so we do not elaborate further upon this in this report.

There are a number of simplification processes an engineer can use to obtain an abstract model of a planning domain. Where we use the term model it refer to a collection $\{DR^l, \Sigma^l, M^{l-1}\}$ of the domain rules and sort definitions at an abstraction level l, and the abstraction rules from $l-1$ to l. These simplification process are listed below:

1. **Reduced model**: A reduced model of a domain description is obtained via a constraint deletion process. The actions, fluents and statics, as well as their arguments are preserved. Most domain rules are copied directly across to the reduced model but enabling conditions prohibiting the execution of actions may be ignored. As the simplification process only allows the deletion of a sub-set of domain rules, the state space in both models is identical. This means that the mapping between the reduced model and the original model is direct and one-to-one. This also provides a very well defined way to reason about how solutions to the reduced problem can be used to guide the search for solutions to the ground problem. Reduced models have a important property known as the *upward solution property* which means that any solution to the ground problem is also a solution to the reduced problem because it satisfies all of the constraints of the reduced problem plus those that were deleted to obtain it [Tenenberg 1988]. See Example 7.3 for a simple example of a model reduction.
2. **Refined model**: A refined model of a domain description is obtain via a class type deletion process. The actions, fluents and statics are preserved, but the sorts of their arguments may change. All domain rules remain identical to the original model. In this case, the abstract model is generated by automatically pruning away branches of the class hierarchy whose abstraction level is less than that of the refined model. This achieves a model in which unwanted detailed entities of a domain are abstracted away. The semantics of model refinement are complex in terms of constructing the class hierarchy because top-level super classes can never been pruned away otherwise the abstract model because redefined. The mapping between the reduced model and the original model is direct but is many-to-one. Abstraction rules are therefore necessary to define the relation to the simplified state space of the refined model. See Example 7.5 for a simple example of a model refinement.
3. **Redefined model**: A redefined model of a domain description is intuitively obtained by redefining the nature of the planning problem. The designer may change all elements of the planning problem in the redefined model. However, in this case no part of the redefinition process is be automatic and there is no general direct mapping between between an original model and its redefinition. This makes a redefined model much more complex to obtain and the designer must be careful to ensure that the mapping between models is correct and complete. A redefined model will usually function as some bespoke abstraction of the application specific task to be completed. For HRCD a redefined model could be used to identify sub-assemblies of a product to be separated prior to complete disassembly. See Example 7.4 for a simple example of a model redefinition.

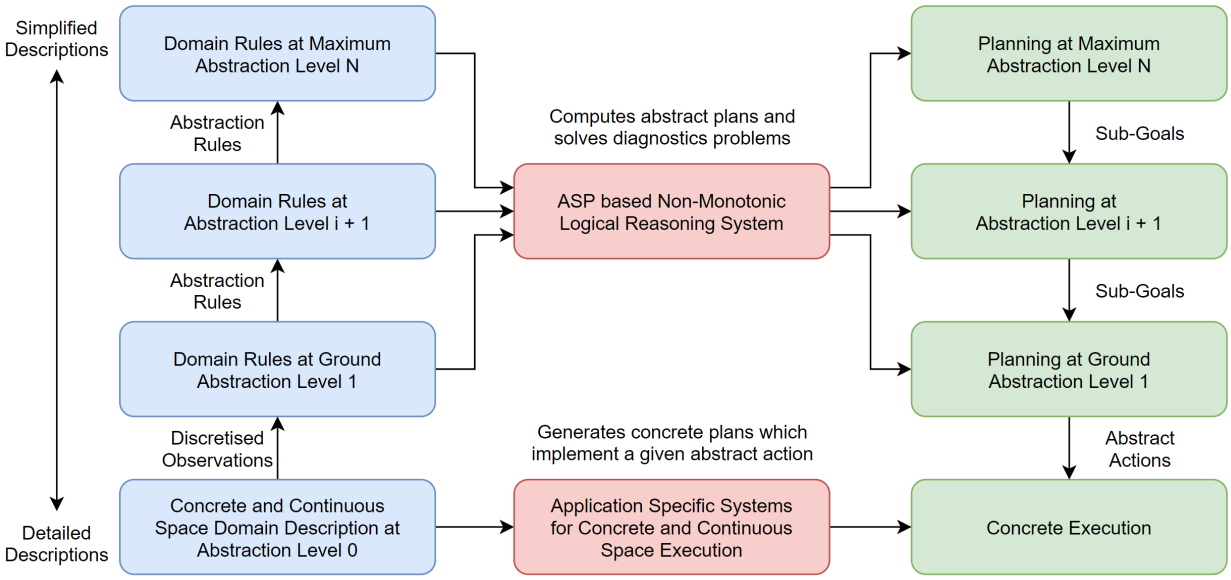

Figure 4: A representation of planning domain descriptions defined over multiple abstractions. A collection of domain rules at each abstraction level defines the nature of the planning problem at that level. Abstraction rules map the state space between adjacent levels in an upwards manner and sub-goals ensure plan conformance in a downwards manner.

6.3 The Planning Module

The role of the planning module is to solve planning problems. Solving the planning problem demands finding a possible trajectory from the initial state to the goal state within the state transition diagram for each available abstraction level defined by a given planning domain description. Before we can define a planning problem we need some additional definitions and notation for the notion of state and actions as represented by the proposed reasoning system.

A *term* is a *constant*, *variable*, *arithmetic expression* or *function*. Constants and variables are defined by strings over some fixed alphabet whose leading letters are lowercase and uppercase respectively. Variables can only occur within arithmetic expressions or functions and are used as shorthand for a collection of *ground instances* of the same statement obtained by replacing the variables by constants of the same sort. A *function symbol* is an statement of the form $f^l(\bar{t})$, where f is a string over some fixed alphabet defining the function's *name* and $\bar{t}$ is a vector of terms of length n defining its *parameters* where n is called its *arity*. A *function literal* is an expression of the form $f^l(\bar{t}) = v$ or its classical negation $f^l(\bar{t}) \neq v$, where v is a term defining its assigned *value*. Each function symbol $f^l(\bar{t})$ is assigned a set of sorts $\{c_0, ...c_{n-1}\}$ called its *domain* given as $domain(f^l(\bar{t}), i)\ \forall\ i \in [0, n)$ which define the sorts of each of its parameters, and a sort c_v called its *range* given as $range(f^l(\bar{t}))$ which defines the sort of its assignable values. A *domain literal* is a function literal defining a domain property with an assigned value $f^l(\bar{t}) = v$ or its classical negation $f(\bar{t}) \neq v$. Domain literals formed by fluents are *fluent domain literals*, otherwise they are *static domain literals*.

- Let σ_i^l be the domain state at abstraction level l at time step i representing the expected positive or negated values of all fluent domain properties which is defined by a set of fluent domain literals of the form:
  - **holds**$(f^l(\bar{t}) = v, i)$
  - **holds**$(f^l(\bar{t}) \neq v, i)$
- Let ω^l be the specific goal state at abstraction level l representing the final desired positive or negated values of fluent domain properties which is defined by a set of fluent domain literals of the form:
  - **goal**$(f^l(\bar{t}) = v)$
  - **goal**$(f^l(\bar{t}) \neq v)$
- Let α_i^l be the set of actions planned at abstraction level l at time step i which is a collection of atoms of the following form, which define whether or not an action a^l was planned by a robot r:
  - **occurs**(r, a^l, i)
  - $\neg$**occurs**(r, a^l, i)
- Let $\pi_{j,k}^l$ be a complete or partial plan at abstraction level l between steps $j \leq i < k$ which is a ordered sequence of sets of planned actions $\alpha_i^l\ \forall\ i \in [j, k)$ and the expected resulting domain states $\sigma_i^l\ \forall\ i \in [j, k]$.

The abstraction level of a fluent domain literal is defined by the abstraction level of its value, if it is a boolean fluent then its abstraction level is defined by the abstraction level of its last argument. The abstraction level of an action defines the abstraction level of the fluents it can act upon. In other words, actions can only effect the state at their own abstraction level. All arguments of an action must also have the same abstraction level.

We can now define what it means to solve a planning problem in the context of the proposed reasoning system:

Definition 6.2 (Planning Problem) *A planning problem consists of a planning domain description over multiple levels of abstraction $DD^{1,N}$. From the planning domain description a unique state transition diagram can be obtained for each available abstraction level $\tau(DR^l, \Sigma^{l,N}, \Psi)\ \forall\ l \in [1, N]$. At each level of abstraction a plan $\pi_{0,k}^l$, where $k \in \mathbb{N}$ is the maximum planning horizon, must be generated in the transition diagram $\tau(DR^l, \Sigma^{l,N}, \Psi)$ from the initial state σ_0^l to the goal state ω^l which must conform to the sub-goals ρ^{l+1} generated from $\pi_{0,k}^{l+1}$ according to the mapping M^l.*

More formally, a planning problem $PP_{s,k}^{1,N}$ over abstraction levels 1 to $N \in \mathbb{N}$ is defined by the seven tuple:

$$PP_{s,k}^{1,N} = \langle DR^{1,N}, \Sigma^{1,N}, M^{1,N-1}, \Psi, \sigma_0^1, \omega^1, \Gamma_{s-1}^1 \rangle \tag{3}$$

And the planning problem $PP_{s,k}^l$ at each distinct abstraction level can be expressed by the eight tuple:

$$PP_{s,k}^l = \langle DR^l, \Sigma^{l,N}, M^l, \Psi, \sigma_0^l, \omega^l, \rho^{l+1}, \Gamma_{s-1}^l \rangle \tag{4}$$

Where s is the current step which is 0 if plan execution has not yet been started, ρ^{l+1} are the sub-goals generated from $\pi_{0,k}^{l+1}$ which contains n sets of sub-goals $sub\_goals(\pi_{0,k}^{l+1}, i)\ \forall\ i \in [0, k)$ where $n = |\pi_{0,k}^{l+1}|$ and Γ_{s-1} is the recorded history, see following section. Solving $PP_{s,k}^{1,N}$ involves solving all the problems $PP_{s,k}^l\ \forall\ l \in [1, N]$ in descending order, passing the effects of actions planned at the previous level as sub-goals to the next. Due to time and space constraints we do not elaborate on the logic rules or the algorithm used to perform this process.

6.4 The Diagnostics Module

The role of the diagnostics module is to solve diagnostics problems. Such situations arise when observations committed to the recorded history during execution contradict expectations determined during planning. These contradictions can be solved either by retracting invalidated assumptions made the initial domain state or by hypothesising the occurrence of possible exogenous actions which may have lead to the contradiction. As standard, the system will attempt to solve diagnostics problems by retracting assumptions before hypothesising exogenous actions. It is however possible to specify a preference which will swap this order on a fluent by fluent basis. We define a diagnostics problem as follows:

Definition 6.3 (Diagnostics Problem) *A diagnostics problem consists of the recorded history of a dynamic planning domain such that its latest collection of observations contains at least one contradiction. A contradiction is a pair of statements defining the observed and expected value of the same fluent whose values are different. An explanation of a contradiction is a collection of unobserved past occurrences of exogenous actions and exceptions to initial state defaults that may account for the unexpected observations. When two observations made at different time steps contradict each other and no actions made by the robots caused this change, only exogenous actions can resolve this contradiction.*

State observations are made to check the effects and preconditions of planned actions. Action execution observations are made when a robot executes an action or when a robot observes either a human cooperator or an exogenous actor perform an action. These observations are committed to the recorded history which we define as follows:

Definition 6.4 (Recorded History) *The recorded history Γ_{s-1} of a planning problem up to the current time step s consists of a set of observations made by the robots during plan execution. These are defined by a collection of the following atoms, where r is the name of a robot, a^l is an action at abstraction level l, and i a time step:*

1. *Domain State Observation:*
   - ***observed***$(r, f^l(\bar{t}) = v, i)$ *- fluent $f^l(\bar{t})$ was observed by robot r to have a value v at step $i < s$,*
   - ***observed***$(r, f^l(\bar{t}) \neq v, i)$ *- fluent $f^l(\bar{t})$ was observed by robot r to not have a value v at step $i < s$.*
2. *Action Execution Observation:*
   - ***executed***(r, a^1, i) *- Robot r executed action a^1 at step $i < s$,*
   - ***happened***(r, e, a^1, i) *- Robot r observed non-robotic actor e perform actions a^1 at step $i < s$.*

The recorded history defines a collection of possible trajectories of a state transition diagram up to the current time step which can be interpreted as the domain's possible past states, as defined by the following:

Definition 6.5 (Model of Recorded History) *A trajectory $\langle \sigma_0^l, \alpha_0^l, \sigma_1^l, ...\alpha_{s-1}^l, \sigma_s^l \rangle$ of the state transition diagram $\tau(DR^l, \Sigma^{l,N}, \Psi)$ from a plan $\pi_{0,s}^l$ is a model of the recorded history Γ_{s-1}^l if:*

1. ***executed***$(r, a^l, i) \in \Gamma_{n-1}$ *then* ***occurs***$(r, a^l, i) \in \alpha_i^l$,
2. ***happened***$(r, e, a^l, i) \in \Gamma_{n-1}$ *then* ***occurs***$(r, a^l, i) \in \alpha_i^l$,
3. ***observed***$(r, f^l(\bar{t}) = v, i) \in \Gamma_{n-1}$ *then* ***holds***$(f^l(\bar{t}) = v, i) \in \sigma_i^l$,
4. ***observed***$(r, f^l(\bar{t}) \neq v, i) \in \Gamma_{n-1}$ *then* ***holds***$(f^l(\bar{t}) = v, i) \notin \sigma_i^l$.

If Γ_{s-1}^l has at least one model then it is said to be consistent with $\pi_{0,s}^l$, otherwise it is inconsistent.

In the proposed reasoning system the state representation is always complete and it is assumed that observations are deterministic. As such, it is simple to determine whether Γ_{s-1}^l has model by finding the set of contradictions:

Definition 6.6 (Contradicting Domain State Observation) *A contradiction is defined by an observation whose value logically contradicts the value that the robots expected to observe. The set of contradictions κ_i^l between recorded history Γ_i^l and the expected state σ_i^l is a set of fluent domain literals $f^l(\bar{t}) = v$ where either of the following is true:*

1. ***observed***$(r, f^l(\bar{t}) = v, i) \in \Gamma_i^l$ *and* ***holds***$(f^l(\bar{t}) = v, i) \notin \sigma_i^l$
2. ***observed***$(r, f^l(\bar{t}) \neq v, i) \in \Gamma_i^l$ *and* ***holds***$(f^l(\bar{t}) = v, i) \in \sigma_i^l$

If any of the sets κ_i^l where $0 \leq i < s$ is non-empty and $\sigma_i^l \in \pi_{0,s}^l$ then Γ_{s-1}^l is inconsistent with $\pi_{0,s}^l$.

If Γ_{s-1}^l is inconsistent with $\pi_{0,s}^l$ when the current plan is invalid and the system must enter diagnostics. The task of the diagnostics model is then to explain all of the contradictions in κ_{s-1}^l before re-planning can occur.

7 Examples

This section provides examples for simple planning problems which demonstrate the capabilities of the proposed reasoning system. For each example we use action language AL_d to encode domain rules using super script notation define the abstraction level of a given statement. For a brief description of AL_d see Appendix B or for a comprehensive description see [Gelfond and Kahl 2014]. All following example plans have been generated by a software implementation of the reasoning system available at [Kamperis 2020] which contains domain files for each example.

The following examples are based in a simple domain consisting of an office room, with a connected hall leading to a main library and auxiliary library. This is shown in Figure 5 where the arrows indicate connectedness. We use the logical relation $connected^l(Place_1^l, Place_2^l)$ to refer to these connections in domain rules. The planning problems consist of a simple fetch and return objective requiring the robot to obtain a specific object and deliver it to a specific location or person. To achieve this, the robot must travel to the location of the object, or search for the object if its location is unknown, collect and stow the object if necessary, travel to the goal location, and deliver it accordingly.

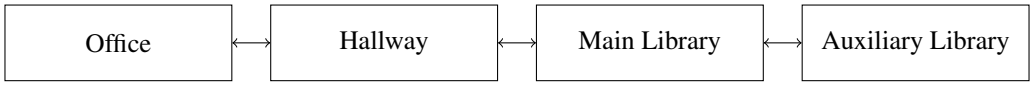

Figure 5: Static structure of the office example domain.

7.1 Planning at a Single Level of Abstraction

Our first examples serve as a foundation for later examples by demonstrating planning problems at a single abstraction.

Example 7.1 (Example 1) *This example contains one robot and one book, named r_0 and b_0 respectively. In the initial state, the robot starts in the office and has been given explicit correct knowledge that the book is located in the main library. We assume that no other entities or actors exist in the domain. The goal state is for the book to be located in the office and not be grasped by the robot. The robot has two types of actions available to it at abstraction level x:*

1. *Locomotive action $move^x(Place)$ allows the robot to change location between connected rooms,*
2. *Manipulation actions $grasp^x(Object)$ and $release^x(Object)$ allow the robot to grasp or release objects.*

And there are two fluents necessary to represent the dynamic state of the domain at abstraction level x to define:

1. *Locations of objects: $loc^x(Object) = Place$,*
2. *Objects a robot is grasping: $grasping^x(Robot) = Object$.*

The robot's actions are defined by the following domain rules of AL_d, where capitalised terms are variables which are shorthand for the ground rules obtained by replacing all variables with constants of the sort with the same name.

- **Causal Laws**:
  1. Moving causes a location change
  $$Robot : move^x(Place) \textbf{ causes } loc^x(Robot) = Place$$
  2. Grasping an Object causes it to be grasped
  $$Robot : grasp^x(Object) \textbf{ causes } grasping^x(Robot) = Object$$
  3. Releasing an Object causes it to no longer be grasped
  $$Robot : release^x(Object) \textbf{ causes } grasping^x(Robot) = nothing$$
- **Executability Conditions**:
  1. A Robot cannot move to a location that is not connected to its current location
  $$\textbf{impossible } Robot : move^x(Place_1) \textbf{ if } loc^x(Robot) = Place_2,\ \neg connected^x(Place_1, Place_2)$$
  2. A Robot cannot grasp an Object that does not share its current location
  $$\textbf{impossible } Robot : grasp^x(Object) \textbf{ if } loc^x(Object) = Place,\ loc^x(Robot) \neq Place$$
  3. A Robot cannot release an Object that it is not currently grasping
  $$\textbf{impossible } Robot : release^x(Object) \textbf{ if } grasping^x(Robot) = Object$$

- **State Constraints**:
  1. A grasped object shares the location of the robot that is grasping it

$$loc^x(Object) = Place \textbf{ if } grasping^x(Robot) = Object,\ loc^x(Robot) = Place$$

The plan shown in Figure 6 is a possible plan which satisfies this planning problem. It is the minimal plan, consisting of six actions, four locomotive and two manipulation. Informally, the plan reads as follows, robot moves to the main library via the hall, grasps the book, returns to the office via the hall, and releases the book.

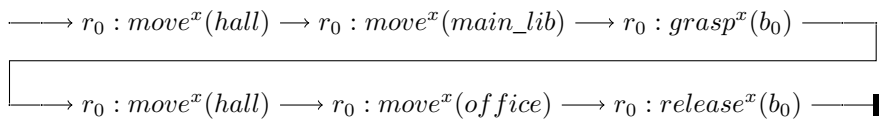

Figure 6: Plan for Example 1 at Abstraction Level x - A fetch and return objective for a single robot.

Example 7.2 (Example 2) *This example extends our previous by introducing more complexity to the planning problem. The robot is now no longer allowed to arbitrarily grasp objects. Instead we introduce the concept of manipulator arms and the robot must now reason about how to use its arms in order to grasp objects. A new constant for a manipulator arm arm_0 is added along with a new logical relation $comp\_of(Object^i, Component^j) \mid i \geq j$ to relate objects to their components. Manipulator arm arm_0 is a component of r_0 expressed as $comp\_of(r_0, arm_0)$. The initial and goal states remain the same, as do our original assumptions. The robot now has three types of actions available to it:*

1. *Locomotive action $move^x(Place)$ allows the robot to change location between connected rooms.*
2. *Manipulation actions $grasp^x(Arm, Object)$ and $release^x(Arm, Object)$ allow the robot to grasp or release objects with a manipulator arm, maintaining the possibility that a robot may have more than one arm.*
3. *Configuration action $config^x(Arm, C) \ : \ C \in \{extend, retract\}$ allow the robot to extend/retract arms.*

And three fluents are necessary to represent the dynamic state of the domain at abstraction level x to define:

1. *Locations of objects: $loc^x(Object) = Place$,*
2. *Objects a robot is grasping: $grasping^x(Robot, Arm) = Object : comp\_of(Robot, Arm)$,*
3. *Configuration state of a robot's components: $state^x(Robot, Component) = C : C \in \{extend, retract\}$.*

These actions are defined by the following domain rules, where the first causal law, executability condition and only state constraint are omitted as they are identical to the previous example.

- **Causal Laws**:
  2. Grasping an object causes it to be grasped

$$Robot : grasp^x(Arm, Object) \textbf{ causes } grasping^x(Robot, Arm) = Object$$

  3. Releasing an object causes it to no longer be grasped

$$Robot : release^x(Arm, Object) \textbf{ causes } grasping^x(Robot, Arm) = nothing$$

  4. Configuring a manipulator arm causes it to change configuration state accordingly

$$Robot : config^x(Arm, C) \textbf{ causes } state^x(Robot, Arm) = C$$

- **Executability Conditions**:
  2. A Robot cannot move if any of its manipulator arms are extended

$$\textbf{impossible } Robot : move^x(Place) \textbf{ if } state^x(Robot, Arm) = extend, comp\_of(Robot, Arm)$$

  3. A Robot cannot grasp an object with a manipulator arm if that object does not share its current location

$$\textbf{impossible } Robot : grasp^x(Arm, Object) \textbf{ if } loc^x(Object) = Place, loc^x(Robot) \neq Place$$

  4. A Robot cannot grasp or release an object with a manipulator arm that is currently retracted

$$\textbf{impossible } Robot : grasp^x(Arm, Object) \textbf{ if } state^x(Robot, Arm) = retract$$
$$\textbf{impossible } Robot : release^x(Arm, Object) \textbf{ if } state^x(Robot, Arm) = retracted$$

- **State Constraints**:
  1. If an object has any components then those components share the locations of its parent

$$loc^x(Component) = Place \textbf{ if } loc^x(Object) = Place,\ comp\_of(Object, Component)$$

The plan shown in Figure 7 is a possible plan which satisfies this planning problem. It is again the minimal plan, consisting of nine actions, four locomotive, two manipulation and two additional configuration actions as compared to the previous example. The robot moves to the main library via the hall, extends its manipulator arm to grasp the book, retracts the manipulator arm, returns to the office, and finally extends its manipulator arm to release the book.

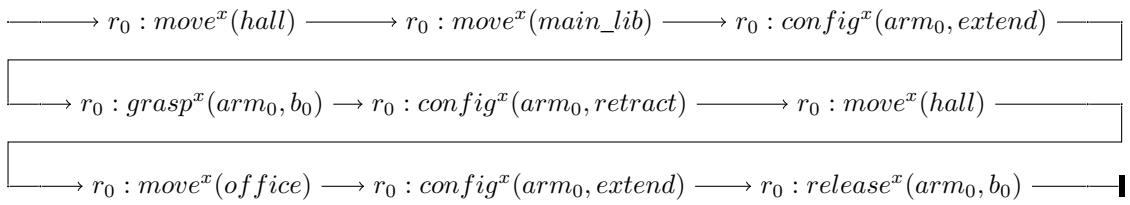

Figure 7: Plan for Example 2 - A fetch and return objective for a single robot with one manipulator arm.

7.2 Plans with Multiple Levels of Abstraction

In the previous examples, planning domains and generated plans were only represented at a single level of abstraction. The following examples demonstrate some of the novel contributions of this thesis which extend the classical approach to planning in ASP to support abstraction planning. This generates multiple plans at different levels of abstraction where higher abstraction level plans represent simpler descriptions of lower abstraction level plans.

Our implementation of abstraction planning is conceptually simple. For each unique abstraction level a plan is generated which must conform to the plan at the previous level, if there is one. Plan conformance between levels is ensured by passing the effects of actions planned at the previous level as sub-goals to the next. These sub-goals must be satisfied by planning actions at the lower level which achieve the same effects as those at the higher in the same order.

Example 7.3 (Example 3) *This example demonstrates how we can add a higher level abstraction to the previous example. In this case, the higher abstraction is a reduced model of the problem in Example 7.2. In a reduced model, the only difference is a deletion of constraints which forbid or penalise the occurrence of certain actions from the original problem. Otherwise, all elements of the original problem are copied to the abstract representation. To achieve this we replace the action* $move^x(Place)$ *with* $travel^{x+1}(Place)$*. These actions are identical in all but their names and their executability conditions. In this case, executability condition 2 is deleted and all other elements of the planning problem are identical to Example 7.2. Note that the action's name usually would not change, we only do so for clarity. The abstraction level of this reduced model is fixed to* $x + 1$ *because it is derived from the model at abstraction level* x*.*

The plan shown in Figure 8 is a possible plan which satisfies this planning problem and is again the minimal plan. As expected, it is very similar to that from Example 7.2. Only the $move^x(Place)$ actions have been replaced by $travel^{x+1}(Place)$ actions, as indicated by the gaps in Figure 8 when compared to Figure 7. Since the $travel^{x+1}(Place)$ action is not subject to the same enabling conditions as $move^x(Place)$, i.e. traveling is not limited to changing location between only those which are directly connected, the length of the resulting plan is shorter.

It is important to note that the plan from Figure 7 is also a satisfiable plan in this reduced model of the domain, but is no longer the minimal plan because the deletion of executability condition 2 no longer necessitates as many locomotive actions. A reduced model of a domain obtained in this manner therefore has the *upward solution property*.

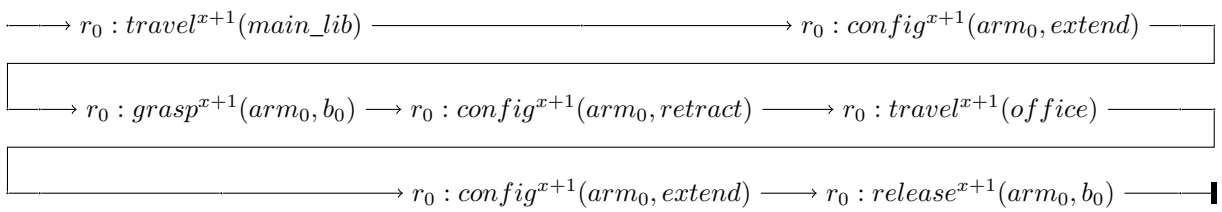

Figure 8: Plan for Example 3 - A reduced fetch and return objective for a single robot with one manipulator arm.

The upward solution property means that any solution to the original ground problem, is also a solution to the abstract problem, since it must, by definition, satisfy all constraints of the abstract problem [Tenenberg 1988]. This is the most simplistic form of abstraction and is extensively used in past work such as in ABSTRIPS [Knoblock 1992]. The advantages of this form of abstraction are that the abstract model can be obtained very easily, requiring no additional designer provided knowledge, and the solutions to the abstract problem are similar in a well defined manner to possible solutions to the ground problem. However, to maintain this property we are limited to constraint deletion only.

As the abstract and ground plans have the upward solution property, and because there is only one fluent domain property related to the goal state, the plans must be *conforming*. This is, the actions in the ground plan achieve at least the same effects as those in the abstract plan, in the same sequence. However, introducing an additional fluent to the goal state, such as an additional book to be fetched, would no longer ensure this conformance as the order of fetching these books would not necessarily be the same between abstractions.

Let us immediately relax the assumption that conformance is guaranteed between abstractions, as this only holds for very simple scenarios. As mentioned previously, to ensure conformance between abstractions we pass the effects of actions planned at the previous abstraction, as sub-goals to the next. These sub-goals must be achieved in the same sequence as the actions from who's effects they were generated from. Each sub-goal is labelled with the name of the robot whose action generated the given sub-goal. There is however no hard constraint that a different robot cannot achieve the sub-goal generated by another robot. Table 1 shows the sub-goals generated from the abstract plan.

Action at Abstraction Level $x+1$	Action Type	Sub-Goal to be Achieved at Abstraction Level x
$r_0 : travel(main\_lib)$	Locomotive	$r_0 : loc(r_0) = main\_lib$
$r_0 : config(arm_0, extend)$	Configuration	$r_0 : configuration(r_0, arm_0) = extend$
$r_0 : grasp(arm_0, b_0)$	Manipulation	$r_0 : grasping(r_0, arm_0) = b_0$
$r_0 : config(arm_0, retract)$	Configuration	$r_0 : state(r_0, arm_0) = retract$
$r_0 : travel(office)$	Locomotive	$r_0 : loc(r_0) = office$
$r_0 : config(arm_0, extend)$	Configuration	$r_0 : state(r_0, arm_0) = extend$
$r_0 : release(arm_0, b_0)$	Manipulation	$r_0 : grasping(r_0, arm_0) = nothing$

Table 1: Sub-Goals Generated from Example 3's Abstract Plan

When planning at the ground level, the effect of the abstract travel action becomes an objective that the robot must achieve by finding one possible path of individual move actions between directly connected locations. A decomposition tree can be used to visually represent how the effects of higher abstraction actions are achieved through the effects of lower abstraction actions in this manner. Although we do not elaborate upon this theory in this report, to ensure that there is a possible plan at all abstractions for all of the robots' available actions at the highest abstraction level, we must ensure that every possible action has a complete decomposition tree. Figure 9 and 10 display the decomposition trees for the two travel actions from the plan in Figure 8. Since the only change between models was the deletion of a constraint on the locomotive action, both locomotive actions planned at the abstract level have a complete decomposition tree to the ground level, and the robot's manipulation and configuration actions planned at the abstract level are trivially achievable through the identical actions at the ground level, we find that indeed the two plans are conforming.

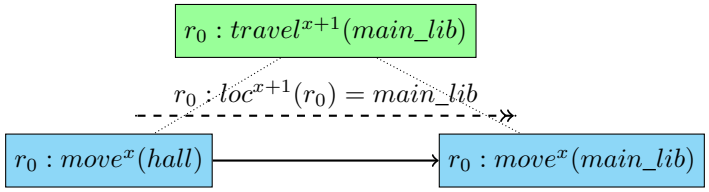

Figure 9: A Decomposition Tree Mapping the $travel^{x+1}(main\_lib)$ Action from Example 7.3 to the corresponding move actions from Example 7.2.

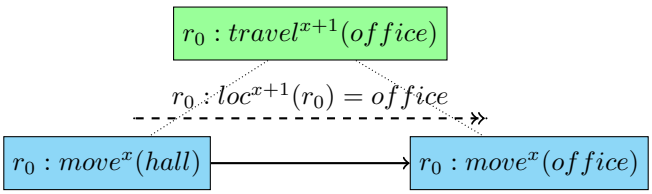

Figure 10: A Decomposition Tree Mapping the $travel^{x+1}(office)$ Action from Example 7.3 to the corresponding move actions from Example 7.2.

Example 7.4 (Example 4) *This example expands our previous to demonstrate how we can achieve a significantly more abstract representation of the ground model defined in Example 7.2 than the reduced model presented in Example 7.3. In this case, the higher abstraction is obtained as a redefinition of the planning problem. In order to obtain this redefinition we simply abstract away the existence of the robots' manipulator arms to return to an encoding that is not dissimilar from the one in Example 7.1. In this case however, we can no longer assume that the robot can only grasp one object at a time, since the robot could have more than one manipulator arm. Since we know that the robot has manipulator arms at the ground level but have chosen to abstract them away, it also does not make much sense for the robot to reason about grasping objects at this higher abstraction. Instead, we can define the planning problem as a logistics task, in which the robot can travel to locations, collect objects, and deposit them to a given location. We do not need to consider how the robot might actually collect or deposit these objects at the ground level, so long as we ensure that the robot does not produce an abstract plan in this redefined model of the original problem, that cannot be achieved in the ground model. The initial and goal states of the problem remain the same, as do our original assumptions. The robot now has three available actions at abstraction level* $x+1$*:*

1. *Locomotive action:* $travel^{x+1}(Place)$ *allows the robot to travel to any location.*
2. *Manipulation actions:* $collect^{x+1}(Object)$ *and* $deposit^{x+1}(Object)$ *allow the robot to collect/deposit objects.*

Three fluents are necessary to represent the dynamic state of the domain at abstraction level $x+1$ *to define:*

1. *The locations of objects:* $loc^{x+1}(Object) = Place$*,*
2. *The objects the robot is carrying:* $carrying^{x+1}(Robot, Object) = B : B \in \{true, false\}$*, in this case the fluent must have a boolean value because a robot could be carrying multiple objects if it has multiple arms,*
3. *The total quantity of objects the robot is carrying:* $total\_carrying^{x+1}(Robot) = N : N \in \mathbb{N}$*.*

And finally we need one additional static:

1. *The carrying capacity of the robot:* $carry\_capacity(Robot) = \#count\ \{\ Arm : comp\_of(Robot, Arm)\ \}$*.*

These are defined by the following domain rules where the second state constraint is omitted as it is identical to the previous examples. We now have a fourth type of rule, the *abstraction rules*. These are used to define the mapping between the state representations at different abstraction levels. The mapping is always upwards, i.e. from a level a to b such that $\sigma^{b,i} \leftarrow \sigma^{a,i}$ where $b \geq a$, in this case $b = x+1$ and $a = x$. This mapping is important because it allows the robot to reason about how to achieve the effects of sub-goals generated from the previous abstraction level, by planning actions at the next abstraction level, by propagating the robots' beliefs to all higher abstractions.

- **Causal Laws**:
  1. Traveling causes a location change
  $$Robot : travel^{x+1}(Place) \textbf{ causes } loc^{x+1}(Robot) = Place$$
  2. Collecting an object causes it to be transported
  $$Robot : collect^{x+1}(Object) \textbf{ causes } carrying^{x+1}(Robot, Object) = true$$
  3. Depositing an object causes it to no longer be transported
  $$Robot : deposit^{x+1}(Object) \textbf{ causes } carrying^{x+1}(Robot, Object) = false$$
- **Executability Conditions**:
  1. A robot cannot collect an Object that does not share its current location
  $$\textbf{impossible } Robot : collect^{x+1}(Object) \textbf{ if } loc^{x+1}(Object) = Place, loc^{x+1}(Robot) \neq Place$$
  2. A robot cannot deposit an Object that it is not currently grasping
  $$\textbf{impossible } Robot : deposit^{x+1}(Object) \textbf{ if } carrying^{x+1}(Robot, Object) = false$$
- **State Constraints**:
  1. Carried objects share the location of the robot that is carrying them
  $$loc^{x+1}(Object) = Place \textbf{ if } carrying^{x+1}(Robot, Object) = true,\ loc^{x+1}(Robot) = Place$$
  3. A robot cannot be carrying more than its carrying capacity
  $$total\_carrying^{x+1}(Robot) \neq X \textbf{ if } carry\_capacity(Robot) = Y, X > Y$$

- **Abstraction Rules**:
  1. If a robot is grasping an object with any of its manipulator arms at abstraction level x then it is also carrying that object at abstraction level $x+1$

$$carrying^{x+1}(Robot, Object) = true \textbf{ if } grasping^{x}(Robot, Arm) = Object$$

The plan shown in Figure 11 is the minimal plan which satisfies this planning problem consisting of just four actions. The robot travels to the main library collects the book, travels back to the office and deposits the book. It is clear that through the use of this redefined model of the planning problem its complexity has been reduced significantly.

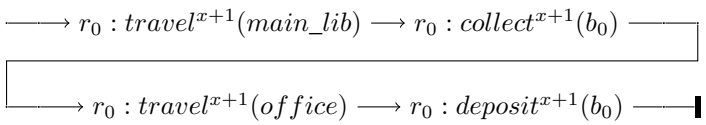

Figure 11: An abstract fetch and return plan for a single robot

Table 2 shows the sub-goals generated from this plan. The two locomotive actions are identical to Example 7.3 and therefore share the same decomposition trees relative to Example 7.2 as given in Figures 9 and 10. The two manipulation actions of this abstract plan are however no longer trivially achievable at the ground level because no ground actions achieve the same effects under the same enabling conditions. The new decomposition trees for both planned manipulation actions from the plan given in Figure 11 are given in Figures 13 and 12. Since all actions from the abstract plan now has a complete decomposition tree from the ground level plan, we know they are conforming.

Action at Abstraction Level $x+1$	Action Type	Sub-Goal to be Achieved at Abstraction Level x
$r_0 : travel(main\_lib)$	Locomotive	$r_0 : loc(r_0) = main\_lib$
$r_0 : collect(b_0)$	Manipulation	$r_0 : carrying(r_0, b_0) = true$
$r_0 : travel(office)$	Locomotive	$r_0 : loc(r_0) = office$
$r_0 : deposit(b_0)$	Manipulation	$r_0 : carrying(r_0, b_0) = false$

Table 2: Sub-Goals Generated from Example 3's Abstract Plan

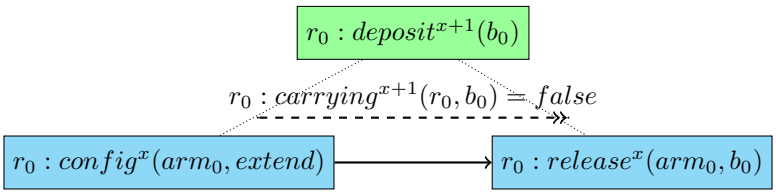

Figure 12: A Decomposition Tree Mapping the $deposit^{x+1}(b_0)$ Action from Example 7.4 to the corresponding actions from Example 7.2.

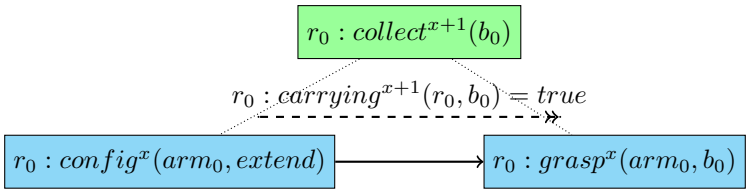

Figure 13: A Decomposition Tree Mapping the $collect^{x+1}(b_0)$ Action from Example 7.4 to the corresponding actions from Example 7.2.

Example 7.5 (Example 5) *This example demonstrates how we can add a more detailed model of the planning domain than that given in example 7.2. In this case, Example 7.2 is a refinement of the model given in this example. The construction of this model can be seen as the uncovering of additional details of the domain which were previously abstracted away. This will also introduce more conditions of the execution of actions. The robots is now aware that the rooms given in Figure 5 are composed of individual cells. To keep things simple each room only has two cells given by* $(n, Room)$ *where* $n \in [0, 1]$*. The class name* $Place$ *now refers to rooms at abstraction level* x *and cells at abstraction*

level $x-1$ and the relation between them is defined by $comp\_of(Room, Cell) : Cell = (n, Room), n \in [0, 1]$. Further, the robot is now aware that its manipulator arms are now composed of a limb and an end effector given as $limb_0$ and end_0 respectively, and whose component relations are defined by $comp\_of(arm_0, limb_0)$ and $comp\_of(arm_0, end_0)$. Limbs have type $Extensible$ and end effectors have type $Grasper$ whereas manipulator arms have both types. The robot now starts in cell 1 of the office and book b_0 now starts in cell 0 of the main library. Additionally, we introduce a table in cell 0 of the office and specify a more precise goal state in which the robot must place the book on the table. The table is of type The robot discovers one additional action at abstraction level $x - 1$ when compared to example 7.2:

4. *Configuration action: $config^{x-1}(Grasper, align(Object))$ allow the robot to align its end effectors.*

And two additional fluents are discovered at abstraction level $x - 1$:

4. *The alignment of end effectors: $state^{x-1}(End) = aligned\_with(Object)$,*
5. *The surface placement of objects: $on^{x-1}(Object) = Surface$.*

These actions are defined by the following domain rules where omitted rules are identical to Example 7.2:

- **Casual Laws**:
  5. Releasing an object whilst it is aligned with a surface causes it to be on that surface

$$Robot : release^{x+1}(Grasper, Object) \textbf{ causes } on^{x-1}(Object) = Surface \textbf{ if}$$
$$state^{x-1}(Grasper) = aligned\_with(Surface)$$

- **Executability Conditions**:
  5. A robot cannot grasp an object with an end effector if it is not aligned with that object

$$\textbf{impossible } Robot : grasp^{x-1}(Grasper, Object) \textbf{ if } state^{x-1}(Grasper) \neq aligned\_with(Object)$$

- **Abstraction Rules**:
  1. If an object is in any cell of a given room then it is also in that room

$$loc^x(Object) = Place^x \textbf{ if } loc^{x-1}(Object) = Place^{x-1}, comp\_of(Place^x, Place^{x-1})$$

  2. If a robot is grasping a component of another object then the robot is also grasping the other object

$$grasping^x(Robot, Grasper^{x-1}) = Object^x \textbf{ if } loc^{x-1}(Robot, Grasper^{x-1}) = Object^{x-1},$$
$$comp\_of(Object^x, Object^{x-1})$$

  3. If a robot is grasping an object with its end effector then it is also grasping it with its manipulator arm

$$grasping^{x-1}(Robot, Grasper^x) = Object^{x-1} \textbf{ if } loc^{x-1}(Robot, Grasper^{x-1}) = Object^{x-1},$$
$$comp\_of(Grasper^x, Grasper^{x-1})$$

The plan shown in Figure 14 is the minimal plan which satisfies this planning problem. It consists of 15 actions in total, significantly more than previous examples. Figures 15, 16, 17 and 18 show the complete decomposition trees for all the actions at level $x + 1$ from Example 7.4 mapped to actions at x from Example 7.2 and $x - 1$ from this example.

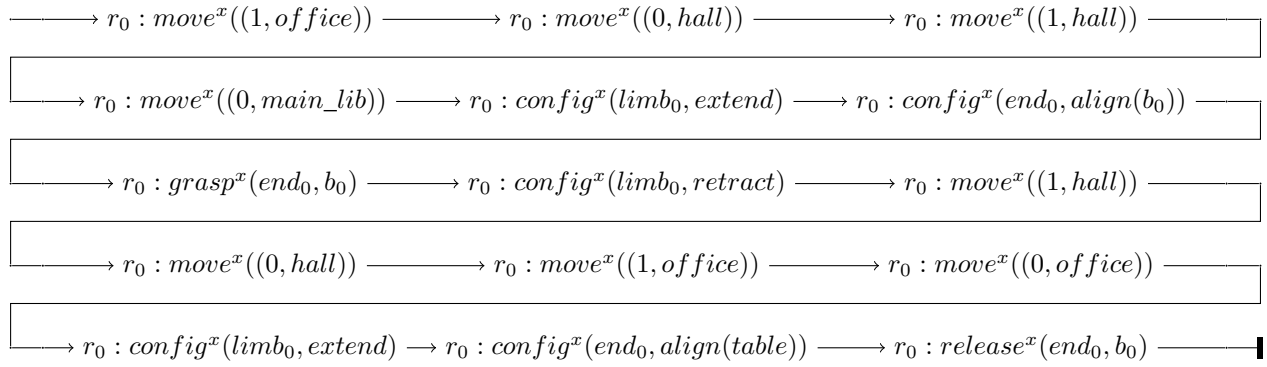

Figure 14: Plan for Example 5 - A fetch and return objective for a single robot with one manipulator arm.

We now have a complete decomposition trees over all available abstraction levels defined throughout our examples. The width of these decomposition trees clearly increases significantly with its depth. In the worst case, the width of the decomposition tree will be exponential in its depth and proportional to 2^{N-1} where N is the maximum abstraction level. This is expected as the search space increases exponentially with a reduction in abstraction [Hoffmann et al. 2006].

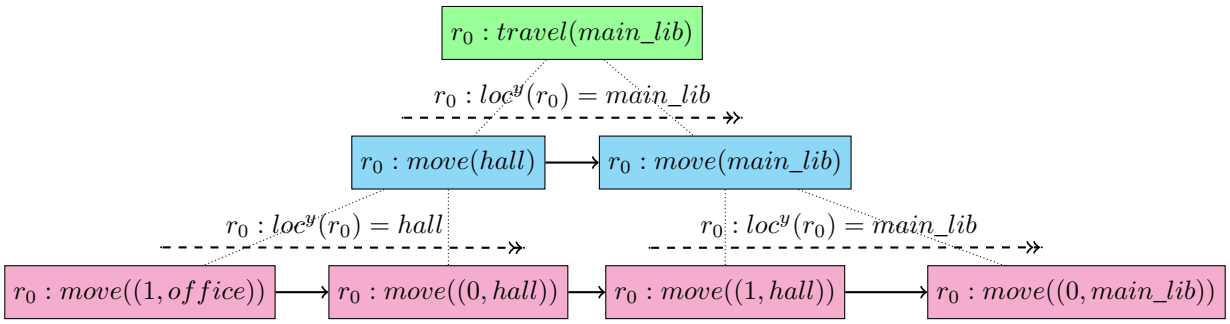

Figure 15: A Decomposition Tree Mapping $travel(main\_lib)$ from Example 7.4 to actions from Examples 7.2 and 7.5.

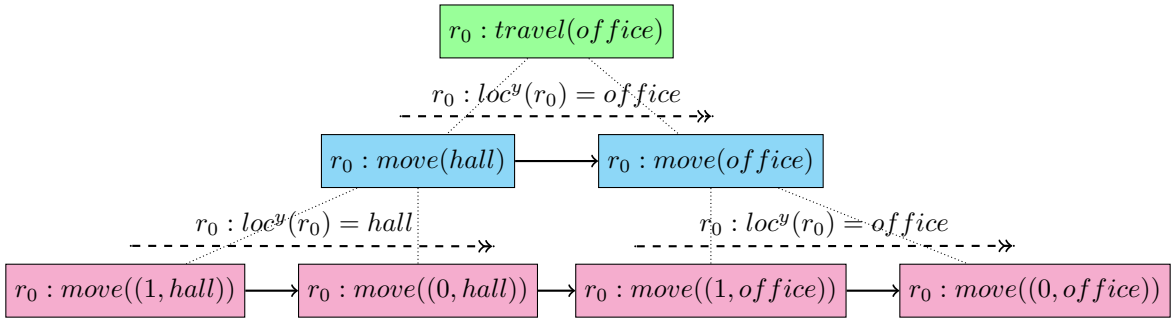

Figure 16: A Decomposition Tree Mapping $travel(office)$ from Example 7.4 to actions from Examples 7.2 and 7.5.

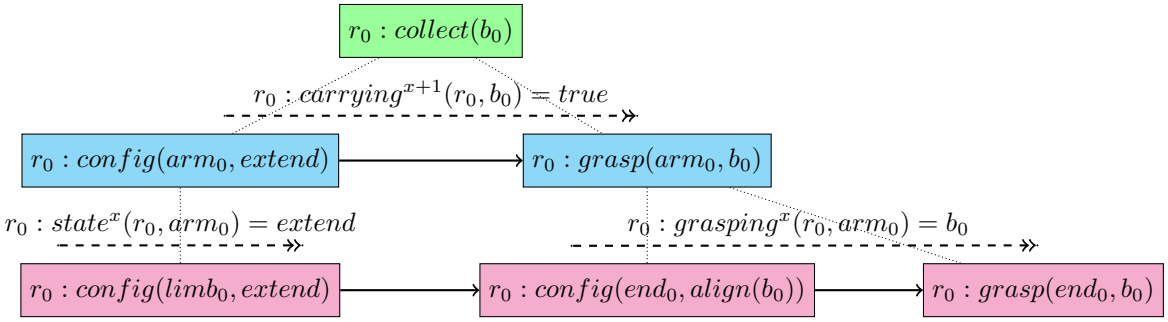

Figure 17: A Decomposition Tree Mapping $collect(b_0)$ from Example 7.4 to actions from Examples 7.2 and 7.5.

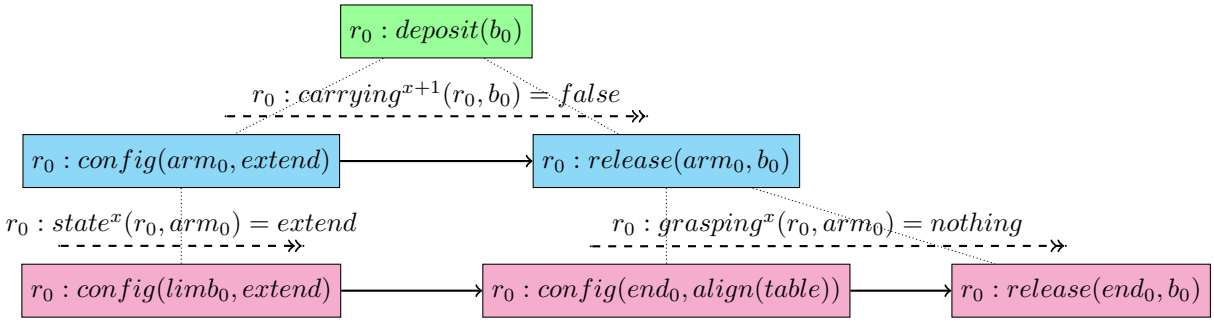

Figure 18: A Decomposition Tree Mapping $deposit(b_0)$ from Example 7.4 to actions from Examples 7.2 and 7.5.

Appendices

A Answer Set Programming

Answer Set Programming (ASP) is now a well known declarative programming paradigm that has established itself as a prominent technique for Knowledge Representation and Reasoning (KRR) in AI. ASP is defined by *stable model semantics* [Gelfond and Lifschitz 1988] and has its roots in non-monotonic logic and logic programming [Gelfond and Lifschitz 1991]. It is closely related to formalisms such as Propositional Satisfiability (SAT), Satisfiability Modulo Theories (SMT), Quantified Boolean Formulas (QBF) and Constraint Programming (CP) [Gebser et al. 2017].

ASP stands as one of the most effective axiomatic logic based approaches to solving combinatorial optimisation problems in artificial intelligence due to its ability to precisely capture deafeasible reasoning, making it effective for representing and reasoning with commonsense knowledge. Its elaboration tolerance makes it particularly effective for highly knowledge intensive problems, such as planning, scheduling and diagnostics [Kaminski et al. 2017]. The successfulness of ASP is generally attributed to the versatility, expressiveness and readability of its modelling language, and the availability of highly efficient solvers [Gebser et al. 2018]. There has been many applications of ASP to date [Erdem and Patoglu 2018, Esra Erdem 2016, Grasso et al. 2013] such as, coordinating Unmanned Aerial Vehicle (UAV) teams [Balduccini et al. 2014], planning systems for housekeeping robots [Aker et al. 2012], and an extension of the Robot Operating System (ROS) [Quigley et al. 2009, Andres et al. 2013]. Other applications include healthcare [Dodaro et al. 2018, Erdem et al. 2011, Gebser et al. 2010] and industry [Falkner et al. 2018].

The central concept of ASP is to represent a problem as a *program*, whose *models*, called *answer sets* correspond to possible solutions to that problem. Where the problem is formulated by a set of logical *axioms* which define its nature. The main component of an ASP program is the *rule*, which consists of a *head* and *body* of the form shown in rule 5.

$$a_0 \mid ... \mid a_m \leftarrow a_{m+1}\ ,\ ...\ ,\ a_n, not\ a_{n+1}\ ,\ ...\ ,\ not\ a_o. \tag{5}$$

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. Atoms a_1 to a_m are called head atoms, a_{m+1} to a_n and $not\ a_{n+1}$ to $not\ a_o$ are called positive and negative body literals, respectively. The logical connective *not* stands for *negation as failure* (also known 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*. The body of a rule 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. An answer set of an ASP program is a set of head atoms which simultaneously satisfy all rules of the program. If an ASP program has at least one answer set it is said to be *satisfiable*, else it is *unsatisfiable*.

There are four fundamental types of rule. A rule with exactly one atom $(m = 0)$ in the head and at least one literal in the body $(n + o \geq 1)$ 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 $(m = 0)$ and no body $(n + o = 0)$ is called a *fact*, all atoms occurring in facts must be in all answer sets of the program. A rule with no head $(m = -1)$ and at least one literal in the body $(n + o \geq 1)$ 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 $(m \geq 1)$ and any number of literals in the body $(n + o \geq 0)$ is called a *disjunctive* rule, disjunctive rules are similar to normal rules, the head is satisfied if exactly 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 $(m \geq 0)$ enclosed in braces, is called a *choice* rule, these are again similar to normal rules, except they are satisfied if any, none or all of the head atoms are in the answer set of the program. Choice rules can be preceded by the aggregate statements such as $\#count$ or $\#sum$ to express *cardinality constraints* and *weight constraints* respectively [Dell'Armi et al. 2003, Calimeri et al. 2005]. For greater details of the input language of ASP the reader should refer to the ASP-core-2 language standard [Francesco Calimeri 2012].

A special type of rule is the *weak constraint*. Weak constraints can be used for cardinality based optimisation of answer sets and encoding of *consistency restoring* (CR-rules) rules [Balduccini and Gelfond 2003]. 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 answer set unsatisfiability. 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. Encoding a CR-rule via a weak constraint is described in algorithm 1 of [Balai et al. 2013]. CR-rules are able to generate indirect exceptions to default knowledge. This is notable for robotics as it allows non-monotonic revision of assumptions made about the initial domain state.

B Action Language

We employ action language AL_d [Gelfond and Inclezan 2013] to encode system descriptions for each of our example planning problems. Each abstraction level can be modelled as a distinct system description of AL_d with additional bridging axioms used to define the relationships between abstractions. The standard notation of AL_d is minorly modified to represent this. We refer to these bridging axioms as *abstraction rules*. Abstraction rules are integral to the proposed method of abstraction planning because they define the relationship between the robot's beliefs at each abstraction and thus the achievements of the effects of higher level actions when planning at lower levels. The translation from AL_d to the proposed reasoning system is not always simple, but the clarity of the modelling language is however attractive for the purposes of example. Furthermore, since AL_d is very well defined under answer sets semantics, we make extensive use of its mathematical syntax as a foundation for our own. However, it is important to note that our choice of action language does not imply that the proposed reasoning system is limited to representing only AL_d system descriptions.

B.1 Transition Diagrams of AL_d

Action languages are formal models of parts of natural language used to describe *transition diagrams* of dynamic domains [Gelfond and Lifschitz 1998]. A transition diagram is a directed graph whose nodes correspond to possible domains states and whose arcs are defined by actions which can change the state between those connected by that arc. A set of statements of AL_d defines a system description SD which itself defines a transition diagram $\tau(SD)$. A single state transition of $\tau(SD)$ of AL_d can be described by $\langle \sigma_0, \{a_1, ..., a_k\}, \sigma_1 \rangle$. Where $\{a_1, ..., a_k\}$ is a non-empty 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. A path $\langle \sigma_0, a_0, \sigma_1, ..., a_{n-1}, \sigma_n \rangle$ of $\tau(SD)$ represents a *trajectory* from the initial state σ_0 to the goal state σ_n. A transition diagram $\tau(SD)$ contains all trajectories of the system SD. Solving a planning problem of a system SD requires finding some trajectory of $\tau(SD)$.

B.2 System Descriptions of AL_d

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

1. *Causal Laws* define the direct effects of executing actions. Statements of the following form say that if an action a^l is executed in a state satisfying *body* then the value of $f^l(\bar{t})$ is x in the resulting state.

$$a^l \textbf{ causes } f^l(\bar{t}) = x \textbf{ if } body$$

2. *State Constraints* define the relationships between different fluent domain literals and prohibit domain states which are physically impossible. Statements of the following form say that the value of $f^l(\bar{t})$ must be x for every state that satisfies *body*. No negation of a defined fluent may occur in the head of a state constraint.

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

3. *Executability Conditions* define states under which particular actions are cannot be executed. Statements of the following form say that actions $a^l_0, ..., a^l_k$ cannot be simultaneously executed in a state which satisfies *body*.

$$\textbf{impossible } a^l_0, ..., a^l_k \textbf{ if } body$$

Where a^l is an action, $f^l(\bar{t})$ is an fluent domain literal, at abstraction level l, and the *body* of a rule is formed by a set $f^l_0(\bar{t_0}) = y_0, ..., f^l_m(\bar{t_m}) = y_m$ of domain literals of size $m \geq -1$. If $m = -1$ then the keyword *if* can be omitted.

B.3 Sorted Signatures in AL_d

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

$$\Sigma = \langle C, F, S \rangle \tag{6}$$

Where C is a set of sorts and F is a set of function symbols both defined by strings over a fixed alphabet, and S is a directed acyclic graph representing a sort hierarchy. Function symbols from F contain *statics*, *fluents* and *actions*. Statics and fluents are *domain properties*. The value of fluents can be changed by actions and the value of statics cannot change. Fluent domain properties are partitioned into *inertial fluents* and *defined fluents*. Inertial fluents obey the law of inertia and are changed directly by actions. Defined fluents can't be changed by actions and are defined by other fluents.

References

[Abdullah et al. 2015] Abdullah, N., Jafar, F. A., and Maslan, M. N. (2015). Analysis on factors impeding the disassembly process with consideration on automated disassembly planning. *Procedia Manufacturing*, 2:191–195.

[Agrawal and Tiwari 2008] Agrawal, S. and Tiwari, M. (2008). A collaborative ant colony algorithm to stochastic mixed-model u-shaped disassembly line balancing and sequencing problem. *International journal of production research*, 46(6):1405–1429.

[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.

[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.

[Andres et al. 2015] Andres, B., Rajaratnam, D., Sabuncu, O., and Schaub, T. (2015). Integrating asp into ros for reasoning in robots. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 69–82. Springer.

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

[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.

[Baral et al. 2009] Baral, C., Gelfond, M., and Rushton, N. (2009). Probabilistic reasoning with answer sets. *Theory Pract. Log. Program.*, 9(1):57–144.

[Bauer et al. 2008] Bauer, A., Wollherr, D., and Buss, M. (2008). Human-robot collaboration: A survey. *International Journal of Humanoid Robotics*, 5(01):47–66.

[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.

[Chen et al. 2016] Chen, K., Yang, F., and Chen, X. (2016). Planning with task-oriented knowledge acquisition for a service robot. In *IJCAI*, pages 812–818.

[Cohen and Levesque 1991] Cohen, P. R. and Levesque, H. J. (1991). Teamwork. *Nous*, 25(4):487–512.

[Colledani and Battaïa 2016] Colledani, M. and Battaïa, O. (2016). A decision support system to manage the quality of end-of-life products in disassembly systems. *CIRP Annals*, 65(1):41–44.

[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*.

[Devadass et al. 2019] Devadass, P., Stumm, S., and Brell-Cokcan, S. (2019). Adaptive haptically informed assembly with mobile robots in unstructured environments. In *Proceedings of the 36th International Symposium on Automation and Robotics in Construction (ISARC)*, volume 36, pages 469–476. International Association for Automation and Robotics in Construction (IAARC).

[Djuric et al. 2016] Djuric, A. M., Urbanic, R., and Rickli, J. (2016). A framework for collaborative robot (cobot) integration in advanced manufacturing systems. *SAE International Journal of Materials and Manufacturing*, 9(2):457–464.

[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*.

[Elattar 2008] Elattar, S. (2008). Automation and robotics in construction: opportunities and challenges. *Emirates journal for engineering research*, 13(2):21–26.

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

[Erol et al. 1994] Erol, K., Hendler, J., and Nau, D. S. (1994). Htn planning: Complexity and expressivity. In *AAAI*, volume 94, pages 1123–1128.

[Erol et al. 1995] Erol, K., Hendler, J. A., and Nau, D. S. (1995). Semantics for hierarchical task-network planning. Technical report, MARYLAND UNIV COLLEGE PARK INST FOR SYSTEMS RESEARCH.

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

[Everett and Slocum 1994] Everett, J. G. and Slocum, A. H. (1994). Automation and robotics opportunities: construction versus manufacturing. *Journal of construction engineering and management*, 120(2):443–452.

[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.

[Forlizzi and DiSalvo 2006] Forlizzi, J. and DiSalvo, C. (2006). Service robots in the domestic environment: a study of the roomba vacuum in the home. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 258–265.

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

[Gebser et al. 2012] Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2012). *Answer Set Solving in Practice*. Morgan & Claypool Publishers.

[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. 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.

[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.

[Georgievski and Aiello 2015] Georgievski, I. and Aiello, M. (2015). Htn planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156.

[Ghallab et al. 2004] Ghallab, M., Nau, D., and Traverso, P. (2004). *Automated Planning: theory and practice*. Elsevier.

[Ghallab et al. 2016] Ghallab, M., Nau, D., and Traverso, P. (2016). *Automated planning and acting*. Cambridge University Press.

[Giunchiglia 1999] Giunchiglia, F. (1999). Using abstrips abstractions–where do we stand? *Artificial Intelligence Review*, 13(3):201–213.

[Giunchiglia and Walsh 1992] Giunchiglia, F. and Walsh, T. (1992). A theory of abstraction. *Artificial intelligence*, 57(2-3):323–389.

[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.

[Grosz 1996] Grosz, B. J. (1996). Collaborative systems (aaai-94 presidential address). *AI magazine*, 17(2):67–67.

[Hoffmann et al. 2006] Hoffmann, J., Sabharwal, A., and Domshlak, C. (2006). Friends or foes? an ai planning perspective on abstraction and search. In *ICAPS*, pages 294–303.

[Huang et al. 2020] Huang, J., Pham, D. T., Wang, Y., Qu, M., Ji, C., Su, S., Xu, W., Liu, Q., and Zhou, Z. (2020). A case study in human-robot collaboration in the disassembly of press-fitted components. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 234(3):654–664.

[Hui et al. 2008] Hui, W., Dong, X., and Guanghong, D. (2008). A genetic algorithm for product disassembly sequence planning. *Neurocomputing*, 71(13-15):2720–2726.

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

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

[Kamperis 2020] Kamperis, O. M. (2020). An asp based abstraction planner. `https://github.com/OllieKampo/Abstraction_Reasoning`.

[Kawamura et al. 1996] Kawamura, K., Pack, R. T., Bishay, M., and Iskarous, M. (1996). Design philosophy for service robots. *Robotics and Autonomous Systems*, 18(1-2):109–116.

[Khandelwal et al. 2017] Khandelwal, P., Zhang, S., Sinapov, J., Leonetti, M., Thomason, J., Yang, F., Gori, I., Svetlik, M., Khante, P., Lifschitz, V., et al. (2017). Bwibots: A platform for bridging the gap between ai and human–robot interaction research. *The International Journal of Robotics Research*, 36(5-7):635–659.

[Knoblock 1990] Knoblock, C. A. (1990). A theory of abstraction for hierarchical planning. In *Change of Representation and Inductive Bias*, pages 81–104. Springer.

[Knoblock 1992] Knoblock, C. A. (1992). An analysis of abstrips. In *Artificial Intelligence Planning Systems*, pages 126–135. Elsevier.

[Knoblock et al. 1991] Knoblock, C. A., Tenenberg, J. D., and Yang, Q. (1991). Characterizing abstraction hierarchies for planning. In *AAAI*, pages 692–697.

[Kongar and Gupta 2006] Kongar, E. and Gupta, S. M. (2006). Disassembly sequencing using genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 30(5-6):497–506.

[Kosuge et al. 2000] Kosuge, K., Sato, M., and Kazamura, N. (2000). Mobile robot helper. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 1, pages 583–588. IEEE.

[Laili et al. 2019] Laili, Y., Tao, F., Pham, D. T., Wang, Y., and Zhang, L. (2019). Robotic disassembly re-planning using a two-pointer detection strategy and a super-fast bees algorithm. *Robotics and Computer-Integrated Manufacturing*, 59:130–142.

[Lan et al. 2018] Lan, F., Wang, Y., Pham, D. T., Liu, J., Huang, J., Ji, C., Su, S., Xu, W., Liu, Q., and Zhou, Z. (2018). Interlocking problem in automatic disassembly planning and two solutions. In *International Conference on Informatics in Control, Automation and Robotics*, pages 193–213. Springer.

[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)*.

[Lifschitz 2002] Lifschitz, V. (2002). Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54.

[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.

[Liu et al. 2019] Liu, Q., Liu, Z., Xu, W., Tang, Q., Zhou, Z., and Pham, D. T. (2019). Human-robot collaboration in disassembly for sustainable manufacturing. *International Journal of Production Research*, 57(12):4027–4044.

[Newell et al. 1972] Newell, A., Simon, H. A., et al. (1972). *Human problem solving*, volume 104. Prentice-Hall Englewood Cliffs, NJ.

[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.

[Pineau et al. 2003] Pineau, J., Montemerlo, M., Pollack, M., Roy, N., and Thrun, S. (2003). Towards robotic assistants in nursing homes: Challenges and results. *Robotics and autonomous systems*, 42(3-4):271–281.

[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.

[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.

[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.

[Roy et al. 2000] Roy, N., Baltus, G., Fox, D., Gemperle, F., Goetz, J., Hirsch, T., Margaritis, D., Montemerlo, M., Pineau, J., Schulte, J., et al. (2000). Towards personal service robots for the elderly. In *Workshop on Interactive Robots and Entertainment (WIRE 2000)*, volume 25, page 184.

[Sacerdoti 1974] Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial intelligence*, 5(2):115–135.

[Saidi et al. 2016] Saidi, K. S., Bock, T., and Georgoulas, C. (2016). Robotics in construction. In *Springer handbook of robotics*, pages 1493–1520. Springer.

[Saribatur et al. 2019] Saribatur, Z. G., Patoglu, V., and Erdem, E. (2019). Finding optimal feasible global plans for multiple teams of heterogeneous robots using hybrid reasoning: an application to cognitive factories. *Autonomous Robots*, 43(1):213–238.

[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®.

[Speer et al. 2017] Speer, R., Chin, J., and Havasi, C. (2017). Conceptnet 5.5: An open multilingual graph of general knowledge.

[Sridharan et al. 2019] Sridharan, M., Gelfond, M., Zhang, S., and Wyatt, J. (2019). Reba: A refinement-based architecture for knowledge representation and reasoning in robotics. *Journal of Artificial Intelligence Research*, 65:87–180.

[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.

[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.

[Tenenberg 1988] Tenenberg, J. (1988). Abstraction in planning. Technical report, University of Rochester, Department of computer science.

[Thrun 2004] Thrun, S. (2004). Toward a framework for human-robot interaction. *Human-Computer Interaction*, 19(1):9–24.

[Thrun et al. 1999] Thrun, S., Bennewitz, M., Burgard, W., Cremers, A. B., Dellaert, F., Fox, D., Hähnel, D., Rosenberg, C., Roy, N., Schulte, J., et al. (1999). Minerva: A tour-guide robot that learns. In *Annual Conference on Artificial Intelligence*, pages 14–26. Springer.

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

[Timpf et al. 1992] Timpf, S., Volta, G. S., Pollock, D. W., and Egenhofer, M. J. (1992). A conceptual model of wayfinding using multiple levels of abstraction. In *Theories and methods of spatio-temporal reasoning in geographic space*, pages 348–367. Springer.

[Traver et al. 2000] Traver, V. J., Del Pobil, A. P., and Pérez-Francisco, M. (2000). Making service robots human-safe. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*, volume 1, pages 696–701. IEEE.

[Xu et al. 2020] Xu, W., Tang, Q., Liu, J., Liu, Z., Zhou, Z., and Pham, D. T. (2020). Disassembly sequence planning using discrete bees algorithm for human-robot collaboration in remanufacturing. *Robotics and Computer-Integrated Manufacturing*, 62:101860.

[Yeh 2012] Yeh, W.-C. (2012). Simplified swarm optimization in disassembly sequencing problems with learning effects. *Computers & Operations Research*, 39(9):2168–2177.

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

[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 and Zhang 2009] Zhang, X. and Zhang, S. (2009). Product disassembly sequence planning based on particle swarm optimization algorithm. *Computer Integrated Manufacturing Systems*, 15(3):508–514.

[Zhang et al. 2007] Zhang, X.-h., Li, S.-q., Wang, J.-f., and Li, X. (2007). Single object selective disassembly sequence planning based on ant colony algorithm. *COMPUTER INTEGRATED MANUFACTURING SYSTEMS-BEIJING-*, 13(6):1109.

[Zhou et al. 2019] Zhou, Z., Liu, J., Pham, D. T., Xu, W., Ramirez, F. J., Ji, C., and Liu, Q. (2019). Disassembly sequence planning: recent developments and future trends. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 233(5):1450–1471.