Journal of Artificial Intelligence Research 65 (2019) 1-94 Submitted 10/2018; published 04/2019

REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics

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

Michael Gelfond MICHAEL.GELFOND@TTU.EDU
Department of Computer Science
Texas Tech University, USA

Shiqi Zhang SZHANG@CS.BINGHAMTON.EDU
Department of Computer Science
SUNY Binghamton, USA

Jeremy Wyatt JLW@CS.BHAM.AC.UK
School of Computer Science
University of Birmingham, UK

Abstract

This article describes REBA, a knowledge representation and reasoning architecture for robots that is based on tightly-coupled transition diagrams of the domain at two different levels of granularity. An action language is extended to support non-boolean fluents and non-deterministic causal laws, and used to describe the domain's transition diagrams, with the fine-resolution transition diagram being defined as a refinement of the coarse-resolution transition diagram. The coarse-resolution system description, and a history that includes prioritized defaults, are translated into an Answer Set Prolog (ASP) program. For any given goal, inference in the ASP program provides a plan of abstract actions. To implement each such abstract action, the robot automatically zooms to the part of the fine-resolution transition diagram relevant to this action. The zoomed fine-resolution system description, and a probabilistic representation of the uncertainty in sensing and actuation, are used to construct a partially observable Markov decision process (POMDP). The policy obtained by solving the POMDP is invoked repeatedly to implement the abstract action as a sequence of concrete actions. The coarse-resolution observations corresponding to the fine-resolution outcomes of executing these concrete actions are added to the coarse-resolution history and used for subsequent coarse-resolution reasoning. The architecture thus combines the complementary strengths of declarative programming and probabilistic graphical models to represent and reason with non-monotonic logic-based and probabilistic descriptions of uncertainty and incomplete domain knowledge. In addition, we describe a general methodology for the design of software components of a robot based on these knowledge representation and reasoning tools, and provide a path for proving correctness of these components. The architecture is evaluated in simulation and on a mobile robot finding and moving objects to desired locations in indoor domains, to show that the architecture supports reasoning with violation of defaults, noisy observations and unreliable actions, in complex domains.

1. Introduction

Robots are increasingly being used to assist humans in homes, offices, warehouses, and other complex domains. To truly assist humans in such domains, robots need to be re-taskable and robust. We consider a robot to be re-taskable if its reasoning system enables it to achieve a wide range of goals in a wide range of environments. We consider a robot to be robust if it is able to cope with unreliable sensing, unreliable actions, changes in the environment, and the existence of atypical environments, by representing and reasoning with different description of knowledge and uncertainty. While there have been many attempts, satisfying these desiderata remains an open research problem.

Robotics and artificial intelligence researchers have developed many approaches for robot reasoning, drawing on ideas from two very different classes of systems for knowledge representation and reasoning, which are based on logic and probability theory respectively. Systems based on logic incorporate compositionally structured commonsense knowledge about objects and relations, and support powerful generalization of reasoning to new situations. Systems based on probability reason optimally (or near optimally) about the effects of numerically quantifiable uncertainty in sensing and actuation. There have been many attempts to combine the benefits of these two classes of systems, including work on joint (i.e., logic-based and probabilistic) representations of state and action, and algorithms for planning and decision-making in such formalisms. These approaches provide significant expressive power, but they also impose a significant computational burden. More efficient (and often approximate) reasoning algorithms for such unified probabilistic-logical paradigms are being developed. However, practical robot systems that combine abstract task-level reasoning with probabilistic reasoning and action execution, link, rather than unify, their logic-based and probabilistic representations. Such an approach is pursued primarily because roboticists often need to trade expressivity or correctness guarantees for computational speed. Also, a unified representation is not necessary for all the reasoning tasks that have to be performed by the robot. As a result, information close to the sensorimotor level is often represented probabilistically to quantitatively model and reason about the uncertainty in sensing and actuation, with the robot's beliefs including statements such as "the robotics book is on the shelf with probability 0.9". At the same time, logic-based systems are used to reason with (more) abstract commonsense knowledge, which may not necessarily be natural or easy to represent probabilistically. This knowledge may include hierarchically organized information about object sorts (e.g., "a cookbook is a book"), and default information that holds in all but a few exceptional situations (e.g., "books are typically found in the library"). The probabilistic reasoning system will periodically commit particular claims about the world being true, with some residual uncertainty, to the logical reasoning system, which then reasons about those claims as if they were true. There are thus representations of different expressive strengths within an architecture, and proper transfer of control and information between the corresponding reasoning systems is essential for reliable and efficient reasoning.

The existing work in architectures for robot reasoning has some key limitations. First, many of these systems are driven by the demands of robot systems engineering, and there is little formalization of the different representations, reasoning methods, or the links between them, in the corresponding architectures. Second, many systems employ a logical language that is indefeasible, e.g., first order predicate logic, and incorrect commitments can lead to irrecoverable failures, or reason with a purely probabilistic representation that does not make full use of the available knowledge. Our proposed architecture seeks to address these limitations. It represents and reasons about the world, and the robot's knowledge of it, at two granularities. A fine-resolution description of the do-

main, close to the data obtained from the robot's sensors and actuators, is reasoned about probabilistically, while a coarse-resolution description of the domain, including commonsense knowledge, is reasoned about using non-monotonic logic. While we do not use a unified logical-probabilistic representation, our architecture establishes and precisely defines a tight coupling between the representations at the two granularities, enabling the robot to represent and efficiently reason about commonsense knowledge, what the robot knows (or does not know), and how actions change the robot's knowledge. The interplay between the two types of knowledge and the corresponding reasoning methods is viewed as a conversation between, and the (physical and mental) actions of, a *logician* and a *statistician*. Consider, for instance, the following exchange:

Logician: *the goal is to find the robotics book. I do not know where it is, but I know that books are typically in the library and I am in the library. We should first look for the robotics book in the library*.

Logician → Statistician: *look for the robotics book in the library. You only need to reason about the robotics book and the library*.

Statistician: *In my representation of the world, the library is a set of grid cells. I shall determine how to locate the book probabilistically in these cells considering the probabilities of movement failures and error in visual processing*.

Statistician: *I visually searched for the robotics book in the grid cells of the library, but did not find the book. Although there is a small probability that I missed the book, I am prepared to commit that the robotics book is not in the library*.

Statistician → Logician: *here are my observations from searching the library; the robotics book is not in the library*.

Logician: *the robotics book was not found in the library either because it was not there, or because it was moved to another location. The next default location for books is the bookshelf in the lab. We should go look there next*.

and so on...
where the representations used by the logician and the statistician, and the transfer of control and information between them, is coordinated by a *controller*. This imaginary exchange illustrates the following key features of our approach:

- Reasoning about the states of the domain, and the effects of actions, happens at different levels of granularity, e.g., the logician reasons about rooms, whereas the statistician reasons about grid cells in those rooms.
- For any given goal, the logician computes a plan of abstract actions, and each abstract action is executed probabilistically as a sequence of concrete actions planned by the statistician.
- The effects of the coarse-resolution (logician's) actions are non-deterministic, but the effects of the statistician's fine-resolution actions, and thus the corresponding beliefs, have probabilistic measures of uncertainty associated with them.
- The coarse-resolution knowledge base (of the logician) may include knowledge of things that are irrelevant to the current goal. Probabilistic reasoning at fine resolution (by the statistician) only considers things deemed relevant to the current coarse-resolution transition.

- Fine-resolution probabilistic reasoning about observations and actions updates probabilistic beliefs, and highly likely statements (e.g., probability > 0.9) are considered as being completely certain for subsequent coarse-resolution reasoning by the logician.

1.1 Technical Contributions

The design of our architecture, REBA, is based on tightly-coupled transition diagrams at two levels of granularity. A coarse-resolution description includes commonsense knowledge, and the fine-resolution transition diagram is defined as a *refinement* of the coarse-resolution transition diagram. For any given goal, non-monotonic logical reasoning with the coarse-resolution system description and the system's recorded history, results in a sequence of *abstract* actions. Each such abstract action is implemented as a sequence of *concrete* actions by *zooming* to a part of the fine-resolution transition diagram relevant to this abstract action, and probabilistically modeling the non-determinism in action outcomes. The technical contributions of REBA are summarized below.

Action language extensions. An action language is a formalism used to model action effects, and many action languages have been developed and used in robotics, e.g., STRIPS, PDDL (Ghallab, Nau, & Traverso, 2004), BC (Lee, Lifschitz, & Yang, 2013), and $\mathcal{AL}_d$ (Gelfond & Inclezan, 2013). We extend $\mathcal{AL}_d$ in two ways to make it more expressive. First, we allow fluents (domain attributes that can change) that are non-Boolean, which allows us to compactly model a wider range of situations. Second, we allow non-deterministic causal laws, which captures the non-deterministic effects of the robot's actions, not only in probabilistic but also qualitative terms. This extended version of $\mathcal{AL}_d$ is used to describe the coarse-resolution and fine-resolution transition diagrams of the proposed architecture.

Defaults, histories and explanations. Our architecture makes three contributions related to reasoning with default knowledge and histories. First, we expand the notion of the history of a dynamic domain, which typically includes a record of actions executed and observations obtained (by the robot), to support the representation of (prioritized) default information. We can, for instance, say that a textbook is typically found in the library and, if it is not there, it is typically found in the auxiliary library. Second, we define the notion of a model of a history with defaults in the initial state, enabling the robot to reason with such defaults. Third, we limit reasoning with such expanded histories to the coarse resolution, and enable the robot to efficiently (a) use default knowledge to compute plans to achieve the desired goal; and (b) reason with the history to generate explanations for unexpected observations. For instance, in the absence of knowledge about the locations of a specific object, the robot constructs a plan using the object's default location to speed up search. Also, the robot builds a revised model of the history to explain subsequent observations that contradict expectations based on initial assumptions.

Tightly-coupled transition diagrams. The next set of contributions are related to establishing and precisely defining the relationship between different models of the domain used by the robot. In other words, these contributions precisely define the tight coupling between the transition diagrams at two resolutions. First, we provide a formal definition of one transition diagram being a *refinement* of another, and use this definition to formalize the notion of the coarse-resolution transition diagram being refined to obtain the fine-resolution transition diagram. This definition is obtained in two steps—we first define a notion of *weak refinement* that does not consider the robot's ability to observes the values of domain fluents, and then introduce a *theory of observations* to define a

notion of *strong refinement* that includes the robot's ability to observe the values of fluents. The fact that both transition diagrams are described in the same language facilitates their construction and this formalization. A coarse-resolution state is, for instance, magnified to provide multiple states at the fine-resolution—the corresponding ability to reason about space at two different resolutions is central for scaling to larger environments. We find two resolutions to be practically sufficient for many robot tasks, and leave extensions to other resolutions as an open problem. Second, we define *randomization* of a fine-resolution transition diagram, replacing deterministic causal laws by non-deterministic ones. Third, we formally define and automate *zooming* to a part of the fine-resolution transition diagram relevant to any given coarse-resolution transition. This zooming allows the robot, while implementing any given abstract action, to avoid considering parts of the fine-resolution transition diagram irrelevant to this action. For example, if a robot is moving a cup between two neighboring rooms, the fine-resolution states do not consider grid cells except those in the two rooms and do not consider object parts other than the parts of the cup being moved.

Dynamic generation of probabilistic representations. The next set of innovations connect the contributions described so far to quantitative models of action and observation uncertainty. First, we use a semi-supervised algorithm, the randomized fine-resolution transition diagram, prior knowledge (if any), and experimental trials, to collect statistics and compute probabilities of fine-resolution action outcomes and observations. Second, we provide an algorithm that, for any given abstract action, uses these computed probabilities and the zoomed fine-resolution description to automatically construct the data structures for probabilistic reasoning. This construction uses the axioms encoded in the zoomed fine-resolution description to automatically eliminate impossible states, observations, and transitions from further consideration, thus significantly limiting the computational requirements of probabilistic reasoning. Third, based on the coupling between transition diagrams at the two resolutions, the outcomes of probabilistic reasoning update the coarse-resolution history for subsequent reasoning.

Methodology and architecture. The final set of contributions are related to the overall architecture. First, for the design of the software components of robots that are re-taskable and robust, we articulate a methodology that is general, provides a path for proving correctness of these components, and enables us to predict the robot's behavior. Second, the proposed knowledge representation and reasoning architecture combines the representation and reasoning methods from action languages, declarative programming, probabilistic state estimation and probabilistic planning, to support reliable and efficient operation. The domain representation for logical reasoning is translated into a program in SPARC (Balai, Gelfond, & Zhang, 2013), an extension of CR-Prolog, and the representation for probabilistic reasoning is translated into a partially observable Markov decision process (POMDP) (Kaelbling, Littman, & Cassandra, 1998). CR-Prolog (Balduccini & Gelfond, 2003b) and SPARC incorporate consistency-restoring rules in Answer Set Prolog (ASP)—in this article, the terms ASP, CR-Prolog and SPARC are often used interchangeably—and have a close relationship with our action language, allowing us to reason efficiently with hierarchically organized knowledge and default knowledge, and to pose state estimation, planning, and explanation generation within a single framework. Also, using an efficient approximate solver to reason with POMDPs supports a principled and quantifiable trade-off between accuracy and computational efficiency in the presence of uncertainty, and provides a near-optimal solution under certain conditions (Kaelbling et al., 1998; Ong, Png, Hsu, & Lee, 2010). Third, our architecture avoids exact, inefficient probabilistic reasoning over the entire fine-resolution representation, while still tightly coupling the

reasoning at different resolutions. This intentional separation of non-monotonic logical reasoning and probabilistic reasoning is at the heart of the representational elegance, reliability and inferential efficiency provided by our architecture.

The proposed architecture is evaluated in simulation and on a physical robot finding and moving target objects to desired locations in indoor domains. We show that the architecture enables a robot to reason with violation of defaults, noisy observations, and unreliable actions, in more complex domains, e.g., with more rooms and objects, than was possible before.

1.2 Structure of the Article

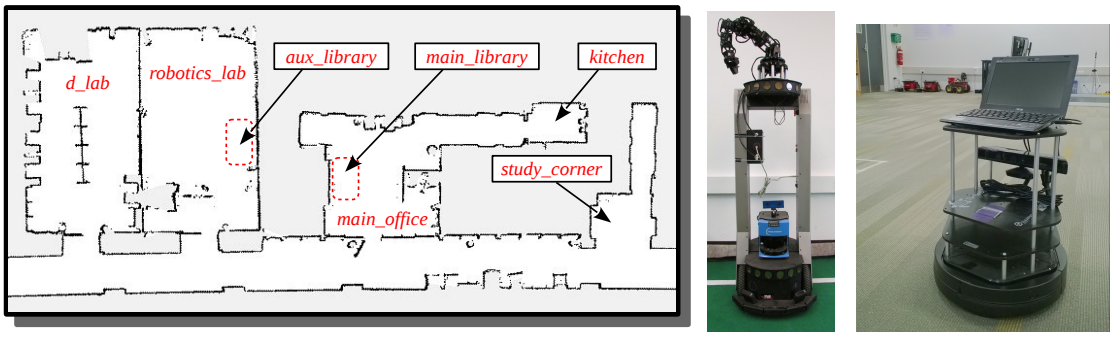
The remainder of the article is organized as follows. Section 2 introduces a domain used as an illustrative example throughout the article, and Section 3 discusses related work in knowledge representation and reasoning for robots. Section 4 presents the methodology associated with the proposed architecture, and Section 5 introduces definitions of basic notions used to build mathematical models of the domain. Section 5.1 describes the action language used to describe the architecture's coarse-resolution and fine-resolution transition diagrams. Section 5.2 introduces histories with initial state defaults as an additional type of record, describes models of system histories, and reduces planning with the coarse-resolution domain representation to computing the answer set of the corresponding ASP program. The logician's domain representation based on these definitions is provided in Section 6. Next, Section 7 describes the (a) refinement of the coarse-resolution transition diagram to obtain the fine-resolution transition diagram, including the introduction of a theory of observations; (b) randomization of the fine-resolution system description; (c) collection of statistics to compute the probability of action outcomes, and (d) zoom operation that identifies the part of the randomized system description relevant to the execution of any given abstract action. Next, Section 8 describes how a POMDP is constructed and solved to obtain a policy that implements the abstract action as a sequence of concrete actions. The overall control loop of the architecture is described in Section 9. Section 10 describes the experimental results in simulation and on mobile robots, followed by conclusions in Section 11. In what follows, we refer to the functions and abstract actions of the coarse-resolution transition diagram using H as the subscript or superscript. The concrete functions and actions of the fine-resolution transition diagram are referred to using L as the subscript or superscript.

2. Illustrative Example: Office Domain

The following domain (with some variants) will be used as an illustrative example throughout the article.

Example 1. [Office domain] Consider a robot that is assigned the goal of moving specific objects to specific places in an office domain. This domain contains:

- The sorts: $place$, $thing$, $robot$, and $object$, with $object$ and $robot$ being subsorts of $thing$. Sorts $textbook$ and cup are subsorts of the sort $object$. Sort names and constants are written in lower-case, while variable names are in uppercase.
- Four specific places: $office$, $main\_library$, $aux\_library$, and $kitchen$. We assume that these places are accessible without the need to navigate any corridors, and that the doors between these places are always open.



(a) Domain map (b) Peoplebot (c) Turtlebot

Figure 1: (a) Subset of the map of an entire floor of a building—specific places are labeled as shown, and used in the goals assigned to the robot; (b)-(c) the "Peoplebot" and "Turtlebot" robot platforms used in the experimental trials.

- Instances of the subsorts of the sort $object$, and an instance of sort $robot$, called rob_1; we do not consider other robots, but any such robots are assumed to have similar sensing and actuation capabilities.

♦

As an extension of this illustrative example that will be used in the experimental trials on physical robots, consider the robot shown in Figure 1(b) operating in an office building whose map is shown in Figure 1(a). Assume that the robot can (a) build and revise the domain map based on laser range finder data; (b) visually recognize objects of interest; and (c) execute actuation commands, although neither the information extracted from sensor inputs nor the action execution is completely reliable. Next, assume that the robot is in the study corner and is given the goal of fetching the robotics textbook. Since the robot knows that books are typically found in the main library, ASP-based reasoning provides a plan of abstract actions that require the robot to go to the main library, pick up the book and bring it back. For the first abstract action, i.e., for moving to the main library, the robot can focus on just the relevant part of the fine-resolution representation, e.g., the cells through which the robot must pass, but not the robotics book that is irrelevant at this stage of reasoning. It then creates and solves a POMDP for this movement sub-task, and executes a sequence of concrete movement actions until it believes that it has reached the main library with high probability. This information is used to reason at the coarse resolution, prompting the robot to execute the next abstract action to pick up the robotics book. Now, assume that the robot is unable to pick up the robotics book because it fails to find the book in the main library despite a thorough search. This observation violates what the robot expects to see based on default knowledge, but the robot explains this by understanding that the book was not in the main library to begin with, and creates a plan to go to the auxiliary library, the second most likely location for textbooks. In this case, assume that the robot finds the book and completes the task. REBA enables such robot behavior.

3. Related Work

We motivate the contributions of our architecture by discussing related work. We first discuss some work on knowledge representation and reasoning in robotics, followed by work on action languages, refinement and zooming. We then discuss some existing frameworks for hybrid reasoning with logical and probabilistic representations, including general frameworks based on unified representations.

Knowledge Representation and Reasoning. There are many recent examples of researchers using probabilistic graphical models such as POMDPs to formulate tasks such as planning, sensing, navigation, and interaction on robots (Bai et al., 2014; Göbelbecker et al., 2011; Hoey et al., 2010; Rosenthal et al., 2011).These formulations, by themselves, are not well-suited for reasoning with commonsense knowledge, i.e., for key desired capabilities in robotics such as default reasoning and non-monotonic logical reasoning. In parallel, research in classical planning and logic programming has provided many algorithms for knowledge representation and reasoning, which have been used on mobile robots. These algorithms typically require a significant amount of prior knowledge of the domain, the agent's capabilities, and the preconditions and effects of the actions. Many of these algorithms are based on first-order logic, and do not support capabilities such as non-monotonic logical reasoning, default reasoning, and the ability to merge new, unreliable information with the current beliefs in a knowledge base. Other logic-based formalisms address some of these limitations. This includes, for instance, theories of reasoning about action and change, as well as Answer Set Prolog (ASP), a non-monotonic logic programming paradigm, which is well-suited for representing and reasoning with commonsense knowledge (Baral, 2003; Gelfond & Kahl, 2014). An international research community has developed around ASP, with applications in cognitive robotics (Erdem & Patoglu, 2012, 2018) and other non-robotics domains. For instance, ASP has been used for planning and diagnostics by a team of heterogeneous, simulated or physical robots operating as housekeepers (Erdem, Aker, & Patoglu, 2012) or in toy factory settings (Saribatur, Patoglu, & Erdem, 2019), and for representation of domain knowledge learned through natural language processing by robots interacting with humans (Chen, Xie, Ji, & Sui, 2012). ASP-based architectures have also been used for the control of unmanned aerial vehicles in dynamic indoor environments (Balduccini, Regli, & Nguyen, 2014). More recent research has removed the need to solve ASP programs entirely anew when the problem specification changes. As a result, new information can expand existing programs, and ground rules and conflict information can be reused to support interactive theory exploration (Gebser, Janhunen, Jost, Kaminski, & Schaub, 2015). However, ASP, by itself, does not support probabilistic models of uncertainty, whereas a lot of information available to robots is represented probabilistically to quantitatively model the uncertainty in sensor input processing and actuation.

Action Languages. Many approaches for reasoning about actions and change in robotics and artificial intelligence (AI) are based on action languages, which are formal models of parts of natural language used for describing transition diagrams. The syntax and intuitive semantics of action languages are much easier for system designers to understand than those of lower level declarative languages such as ASP, Plog (Baral, Gelfond, & Rushton, 2009) and situation calculus (Reiter, 2014). Even designers without any prior expertise in action languages are able to learn their use without any knowledge of the formal semantics. There are many different action languages such as STRIPS, PDDL (Ghallab et al., 2004), BC (Lee et al., 2013), $\mathcal{AL}_d$ (Gelfond & Inclezan, 2013), and $\mathcal{ALM}$ (Inclezan & Gelfond, 2016), which have been used for different applications (Brenner

& Nebel, 2009; Erdem, Gelfond, & Leone, 2016; Khandelwal, Yang, Leonetti, Lifschitz, & Stone, 2014). In robotics applications, we often need to represent and reason with recursive state constraints, non-boolean fluents and non-deterministic causal laws. We chose to expand $\mathcal{AL}_d$ to address these requirements because it already supports the main construct needed for our goal—recursive state constraints. One other option was to use action language $\mathcal{BC}$, which also supports such constraints, but it contains some constructs that are not easy to use given our knowledge representation requirements. For instance, in addition to basic fluents of $\mathcal{AL}_d$, $\mathcal{BC}$ allows fluents whose behavior is described by arbitrary defaults. This certainly gives more power to a knowledge engineer, but it may make the task of representing knowledge more difficult. This is especially true when such defaults interfere with each other and have a substantial number of exceptions. The use of $\mathcal{AL}_d$ allows us to keep this key component of our architecture reasonably simple. In our approach we do assign default values to fluents but it is only done in the initial state and is dealt with by suitably expanding the original notion of history. Although there is no reasoning system for the new version of $\mathcal{AL}_d$, an independent group of researchers have developed (and are in the process of releasing) software to automate the translation between a description in $\mathcal{AL}_d$ and the corresponding ASP description. In this article, however, we describe the steps for this translation and apply them manually.

Refinement and Zooming. Refinement of models or action theories has been researched in different fields. In the field of software engineering and programming languages, there are approaches for type and model refinement (Freeman & Pfenning, 1991; Lovas, 2010; Lovas & Pfenning, 2010; Mellies & Zeilberger, 2015). These approaches do not consider the theories of actions and change that are important for robot domains. More recent work in AI has examined the refinement of action theories of agents, which are represented using situation calculus (Banihashemi, Giacomo, & Lesperance, 2017, 2018). This work assumes the existence of a bisimulation relation between the action theories for a given refinement mapping between the theories, which often does not hold for robotics domains. They also do not support key capabilities that are needed in robotics such as: (i) reasoning with commonsense knowledge; (ii) automatic construction and use of probabilistic models of sensing and actuation; and (iii) automatic zooming to the relevant part of the refined description. Although we do not describe it here, it is possible to introduce simplifying assumptions and a mapping that reduces our approach to one that is similar to work discussed above. In parallel, there has been work on combining discrete and continuous planning at different resolutions in the context of motion planning in robotics. For instance, one approach uses classical planners based on first-order propositional logic for planning discrete abstract movement actions, and implements each abstract action using continuous planners such as rapidly exploring random trees (RRTs) (Srivastava, Riano, Russell, & Abbeel, 2013). This approach was extended to enable a search for suitable (continuous space) instantiations of pose references in the abstract plan, and to communicate relevant geometric information to the abstract (task) planner using logical predicates (Srivastava et al., 2014). However, these approaches do not support non-monotonic logical reasoning with commonsense knowledge or non-deterministic action outcomes. They also do not formally define refinement or zooming for states, actions and observations.

Hybrid Reasoning. Although our architecture does not include a unified representation for logical and probabilistic reasoning, this is a related fundamental problem in robotics and AI. Many principled frameworks have been developed to address this problem over the previous few decades. For instance, a Markov logic network (MLN) combines probabilistic graphical models and first order logic, assigning weights to logic formulas (Richardson & Domingos, 2006). Bayesian Logic

(BLOG) relaxes the unique name constraint of first-order probabilistic languages to provide a compact representation of distributions over varying sets of objects (Milch et al., 2006). Probabilistic Logic (ProbLog) programming annotates facts in logic programs with probabilities and supports efficient inference and learning using weighted Boolean formulas (Fierens, Broeck, Renkens, Shterionov, Gutmann, Thon, Janssens, & Raedt, 2015; Raedt & Kimmig, 2015). Other examples include independent choice logic (Poole, 2000), PRISM (Gorlin, Ramakrishnan, & Smolka, 2012), probabilistic first-order logic (Halpern, 2003), first-order relational POMDPs (Juba, 2016; Sanner & Kersting, 2010), and a system (Plog) that assigns probabilities to different possible worlds represented as answer sets of ASP programs (Baral et al., 2009; Lee & Wang, 2016; Lee & Yang, 2017). Despite the development of these sophisticated frameworks, combining logical and probabilistic reasoning continues to be an open problem, especially in the context of robots collaborating with humans in complex domains. Algorithms based on first-order logic do not support non-monotonic logical reasoning, and do not provide the desired expressiveness for capabilities such as default reasoning—it is not always possible or meaningful to express degrees of belief and uncertainty quantitatively, e.g., by attaching probabilities to logic statements. Other algorithms based on logic programming do not support one or more of the desired capabilities such as reasoning about relations as in causal Bayesian networks; incremental addition of probabilistic information; reasoning with large probabilistic components; or dynamic addition of variables to represent open worlds.

Hybrid Reasoning in Robotics. Many algorithms and architectures have been designed based on the understanding that robots interacting with the environment through sensors and actuators need both logical and probabilistic reasoning capabilities. For instance, architectures have been developed to support hierarchical representation of knowledge and axioms in first-order logic, and probabilistic processing of perceptual information (Laird, 2008; Langley & Choi, 2006; Talamadupula, Benton, Kambhampati, Schermerhorn, & Scheutz, 2010), while deterministic and probabilistic algorithms have been combined (as stated earlier) for task and motion planning on robots (Kaelbling & Lozano-Perez, 2013). Another approach for behavior control of a robot included semantic maps and commonsense knowledge in a probabilistic relational representation, and used a continual planner to switch between decision-theoretic and classical planning procedures based on degrees of belief (Hanheide et al., 2011). More recent work extended this approach to use a three-layered organization of knowledge (instance, default and diagnostic), with knowledge at the higher level modifying that at the lower levels. A three-layered architecture (competence layer, belief layer and deliberative layer) was then used for distributed control of information flow, combining first-order logic and probabilistic reasoning for open world planning (Hanheide et al., 2017). The performance of such architectures can be sensitive to the choice of threshold for switching between the different planning approaches, and the use of first order logic in these architectures limits expressiveness and the use of commonsense knowledge. Declarative programming has also been combined with continuous-time planners for decision making in teams of simulated or physical robots operating in scenarios that mimic housekeeping or manufacturing in a toy factory (Saribatur, Erdem, & Patoglu, 2014; Saribatur et al., 2019). These architectures do not provide a tight coupling between the deterministic and probabilistic components, e.g., through refinement and zooming or a unified representation. This lack of coupling has a negative effect on the computational efficiency, reliability and the ability to fully utilize the available information. More recent work has combined a probabilistic extension of ASP with POMDPs for commonsense inference and probabilistic planning in the context of human-robot dialog (Zhang & Stone, 2015), used a probabilistic extension of ASP to determine some model parameters of POMDPs (Zhang, Khandelwal, & Stone, 2017), and used

an ASP-based architecture to learn action costs on a robot (Khandelwal et al., 2014). ASP-based reasoning has also been combined with reinforcement learning (RL), e.g., to enable an RL agent to only explore relevant actions (Leonetti, Iocchi, & Stone, 2016), or to compute a sequence of symbolic actions that guides a hierarchical MDP controller computing actions for interacting with the environment (Yang, Lyu, Liu, & Gustafson, 2018). However, these architectures do not establish or formally define the coupling between the different representations included in the architecture or the corresponding reasoning methods.

Our Initial Work. The authors of this article have developed other architectures that support some of the knowledge representation and reasoning capabilities of REBA. Early work included an architecture that coupled probabilistic planning based on a hierarchy of POMDPs (Zhang, Sridharan, & Washington, 2013) with ASP-based inference. The domain knowledge used for non-monotonic logical inference in this architecture was incomplete and included default knowledge, but it did not include a model of action effects (Zhang, Sridharan, & Bao, 2012). In other work, ASP-based inference provided priors for POMDP state estimation, and observations and historical data from comparable domains were considered for early termination of the execution of action sequences unlikely to achieve the desired goal (Zhang, Sridharan, & Wyatt, 2015). The initial version of the architecture described in this article focused on the concept of step-wise refinement for knowledge representation and reasoning on robots (Sridharan & Gelfond, 2016; Zhang, Sridharan, Gelfond, & Wyatt, 2014). These papers introduced the idea of representing and reasoning with tightly-coupled transition diagrams at two different levels of granularity. More recent work built on this idea to establish a tight coupling between ASP-based reasoning, active learning, and relational RL, enabling an agent to interactively and cumulatively learn previously unknown actions, related domain axioms, and action capabilities (Sridharan & Meadows, 2018, 2017b). In this article, we formalize and establish the properties of such a coupling, present a general methodology for the design of software components of robots, provide a path for establishing correctness of these components, and describe detailed experimental results in simulation and on physical robot platforms.

4. Design Methodology

REBA is based on a design methodology. A designer following this methodology will:

1. Provide a coarse-resolution description of the robot's domain in action language $\mathcal{AL}_d$ together with the description of the initial state.
2. Provide the necessary domain-specific information for, and construct and examine correctness of, the fine-resolution refinement of the coarse-resolution description.
3. Provide domain-specific information and randomize the fine-resolution description of the domain to capture the non-determinism in action execution.
4. Run experiments and collect statistics to compute probabilities of the outcomes of actions and the reliability of observations.
5. Provide these components, together with any desired goal, to a reasoning system that directs the robot towards achieving this goal.

The reasoning system implements an action loop that can be viewed as an interplay between a logician and statistician (Section 1 and Section 9). In this article, the reasoning system uses ASP-based

non-monotonic logical reasoning, POMDP-based probabilistic reasoning, models and descriptions constructed during the design phase, and records of action execution and observations obtained from the robot. The following sections describe components of the architecture, design methodology steps, and the reasoning system. We first define some basic notions, specifically action description and domain history, which are needed to build mathematical models of the domain.

5. Action Language and Histories

This section first describes extensions to action language $\mathcal{AL}_d$ to support non-boolean fluents and non-deterministic causal laws (Section 5.1). Next, Section 5.2 expands the notion of the history of a dynamic domain to include initial state defaults, defines models of such histories, and describes how these models can be computed. Section 5.3 describes how these models can be used for reasoning. The subsequent sections describe the use of these models (of action description and history) to provide the coarse-resolution description of the domain, and to build more refined fine-resolution models of the domain.

5.1 $\mathcal{AL}_d$ with Non-Boolean Functions and Non-Determinism

Action languages are formal models of parts of natural language used for describing transition diagrams. In this article, we extend action language $\mathcal{AL}_d$ (Gelfond & Inclezan, 2009, 2013; Gelfond & Kahl, 2014) (we preserve the old name for simplicity) to allow functions (fluents and statics) with non-boolean values, and non-deterministic causal laws.

5.1.1 SYNTAX AND INFORMAL SEMANTICS OF $\mathcal{AL}_d$

The description of the syntax of $\mathcal{AL}_d$ will require some preliminary definitions.

Sorted Signature: By *sorted signature* we mean a tuple:

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

where $\mathcal{C}$ and $\mathcal{F}$ are sets of strings, over some fixed alphabet, which are used to name "user-defined" *sorts* and *functions* respectively. $\mathcal{S}$ is a *sort hierarchy*, a directed acyclic graph whose nodes are labeled by sort names from $\mathcal{C}$. A link $\langle c_1, c_2 \rangle$ of $\mathcal{S}$ indicates that c_1 is a subsort of c_2. A pair $\langle \mathcal{C}, \mathcal{S} \rangle$ will occasionally be referred to as an *ontology*. Each function symbol $f \in \mathcal{F}$ is assigned a non-negative integer n (called f's arity), sorts $c_0, \ldots, c_n$ for its parameters, and sort c for its values. We refer to $c_0 \times c_1 \cdots \times c_n$ as the *domain* of f, written as $dom(f)$, and to c as the *range* of f, written as $range(f)$. If $n > 0$ we use the standard mathematical notation $f : c_0 \times \cdots \times c_n \rightarrow c$ for this assignment. We refer to a vector $c_0, \ldots, c_n, c$ as the *signature* of f. For functions of arity 0 (called *object constants*), the notation turns into $f : c$. We say that $o : c$ is *compatible* with sort c' if $\mathcal{S}$ contains a path from c to c'. A sort denoted by a sort name c is the collection $\{o_1, \ldots, o_n\}$ of all object constants compatible with c; this will be written as $c = \{o_1, \ldots, o_n\}$.

In addition to all these "user-defined" sorts and functions, sorted signatures often contain standard arithmetic symbols such as $0, 1, 2, \ldots$ of sort N of natural numbers, and relations and functions such as $\geq$ and $+$, which are interpreted in the usual way.

Terms of a sorted signature Σ are constructed from variables and function symbols as follows:

- A variable is a term.

- An object constant $o : c$ is a term of sort c.
- If $f : c_0 \times \cdots \times c_n \rightarrow c$ where $n > 0$ is a function symbol and o_i is a variable or a constant compatible with sort c_i for $0 \leq i \leq n$, then $f(o_0, \ldots, o_n)$ is a term of sort c.

Atoms of Σ are either of the form:

$$f(\bar{x}) = y$$

where y and elements of $\bar{x}$ are variables or properly-typed object constants, or they are standard arithmetic atoms formed by $\geq$, $>$, etc. If f is boolean, we use the standard notation $f(\bar{x})$ and $\neg f(\bar{x})$. *Literals* are expressions of the form $f(\bar{x}) = y$ and $f(\bar{x}) \neq y$. Furthermore, terms and literals not containing variables are called *ground*.

Action Signature: Signatures used by action languages are often referred to as *action signatures*. They are sorted signatures with some special features that include various classifications of functions from $\mathcal{F}$ and the requirements for inclusion of a number of special sorts and functions. In what follows, we describe the special features of the action signatures that we use in this article.

To distinguish between actions and attributes of the domain, $\mathcal{F}$ is divided into two disjoint parts: $\mathcal{A}$ and $\mathcal{DA}$. Functions from $\mathcal{A}$ are always boolean. Terms formed by function symbols from $\mathcal{A}$ and $\mathcal{DA}$ will be referred to as *actions* and *domain attributes* respectively. $\mathcal{DA}$ is further partitioned into $\mathcal{DA}_s$ and $\mathcal{DA}_f$. Terms formed by functions from $\mathcal{DA}_s$ are referred to as *statics*, and denote domain attributes whose truth values cannot be changed by actions (e.g., locations of walls and doors). Terms formed by functions from $\mathcal{DA}_f$ are referred to as *fluents*. $\mathcal{DA}_f$ is further divided into $\mathcal{DA}_{bf}$ and $\mathcal{DA}_{df}$. Terms formed by symbols from $\mathcal{DA}_{bf}$ are called *basic fluents* and those formed by symbols from $\mathcal{DA}_{df}$ are called *defined fluents*. The defined fluents are always boolean—they do not obey laws of inertia, and are defined in terms of other fluents. Basic fluents, on the other hand, obey laws of inertia (thus often called *inertial fluents* in the knowledge representation literature) and are directly changed by actions. Distinction between basic fluents and defined fluents, as introduced in (Gelfond & Inclezan, 2013), was the key difference between the previous version of $\mathcal{AL}_d$ and its predecessor $\mathcal{AL}$.

The new version of $\mathcal{AL}_d$ described in this article introduces an additional partition of basic fluents into *basic physical fluents* ($\mathcal{DA}_{pbf}$) describing physical attributes of the domain, and *basic knowledge fluent* ($\mathcal{DA}_{kbf}$) describing the agent's knowledge. There is a similar partition of $\mathcal{A}$ into *physical actions* ($\mathcal{A}_p$) that can change the physical state of the world (i.e., the value of physical fluents), and *knowledge producing actions* that are only capable of changing the agent's knowledge (i.e., the value of knowledge fluents). Since robots observe their world through sensors, we also introduce *observable fluents* ($\mathcal{DA}_{obsf}$) to represent the fluents whose values can be checked by the robot by processing sensor inputs, or inferred based on the values of other fluents. The set $\mathcal{DA}_{obsf}$ can be divided into two parts: the set $\mathcal{DA}_{dobsf}$ of *directly observable fluents*, i.e. fluents whose values can be observed directly through sensors, and the set $\mathcal{DA}_{indobsf}$ of *indirectly observable fluents* i.e., fluents whose values are not observed directly but are (instead) inferred from the values of other directly or indirectly observed fluents. For instance, in Example 1 (Section 2), the robot in any given grid cell can directly observe if a cup is in that grid cell. The observation of the cup in a particular cell can be used to infer the room location of the cup. Our classification of functions is also expanded to literals of the language. Specifically, if f is static then $f(\bar{x}) = y$ is a static literal, and if f is a basic fluent then $f(\bar{x}) = y$ is a basic fluent literal.

In addition to the classifications of functions, action signatures considered in this article also include a collection of special sorts like $robot$, $place$, etc., and fluents intrinsic to reasoning about observations. We will refer to the latter as *observation related fluents*. A typical example is a collection of defined fluents:

$$observable_f : robot \times dom(f) \times range(f) \rightarrow boolean \tag{1}$$

where f is an observable function. These fluents are used to specify domain-specific conditions under which a particular robot can observe particular values of particular observable fluents. For instance, in the domain in Example 1, we may need to say that robot rob_1 can only observe the place location of an object if it is also in the same place:

$$observable_{loc}(rob_1, O, Pl) \ \textbf{if} \ loc(rob_1) = Pl$$

For readability, we will slightly abuse the notation and write the above statements as:

$$observable : robot \times obs\_fluent \times value \rightarrow boolean$$

where obs_fluent stands for "observable fluent", and:

$$observable(rob_1, loc(O), Pl) \ \textbf{if} \ loc(rob_1) = Pl$$

In Section 7.1.2, we describe the use of these (and other such) observation-related fluents for describing a theory of observations. Then, in Section 10.1, we describe the processing of inputs from sensors to observe the values of fluents.

Statements of $\mathcal{AL}_d$: Action language $\mathcal{AL}_d$ allows five types of statements: *deterministic causal laws*, *non-deterministic causal laws*, *state constraints*, *definitions*, and *executability conditions*. With the exception of non-deterministic causal law (Statement 3), these statements are built from ground literals.

- Deterministic causal laws are of the form:

$$a \ \textbf{causes} \ f(\bar{x}) = y \ \textbf{if} \ body \tag{2}$$

  where a is an action literal, f is a basic fluent literal, and $body$ is a collection of fluent and static literals. If a is formed by a knowledge producing action, f must be a knowledge fluent. Intuitively, Statement 2 says that if a is executed in a state satisfying $body$, the value of f in any resulting state would be y. Non-deterministic causal laws are of the form:

$$a \ \textbf{causes} \ f(\bar{x}) \ = \ \{Y : p(Y)\} \ \textbf{if} \ body \tag{3}$$

  where p is a unary boolean function symbol from $\mathcal{DA}$, or:

$$a \ \textbf{causes} \ f(\bar{x}) \ : \ sort\_name \ \textbf{if} \ body \tag{4}$$

  Statement 3 says that if a is executed in a state satisfying $body$, f may take on any value from the set $\{Y \ : \ p(Y)\} \cap range(f)$ in the resulting state. Statement 4 says that f may take any value from $\{sort\_name \cap range(f)\}$. If the $body$ of a causal law is empty, the **if** part of the statement is omitted. Note that these axioms are formed from terms and literals

that are ground, and (possibly) from the expression $\{Y : p(Y)\}$ that is sometimes referred to as a *set term*. Occurrences of Y in a set term are called *bound*. A statement of $\mathcal{AL}_d$ is *ground* if every variable occurring in it is bound. Even though the syntax of $\mathcal{AL}_d$ only allows ground sentences, we often remove this limitation in practice. For instance, in the context of Example 1, we may have the deterministic causal law:

$$move(R, Pl) \ \textbf{causes} \ loc(R) = Pl$$

which says that for every robot R moving to place Pl will end up in Pl. In action languages, each such statement is usually understood as shorthand for a collection of its ground instances, i.e., statements obtained by replacing its variables by object constants of the corresponding sorts. We use a modified version of this approach in which only non-bound variables are eliminated in this way.

- State constraints are of the form:

$$f(\bar{x}) = y \ \textbf{if} \ body \tag{5}$$

where f is a basic fluent or static. The state constraint says that $f(\bar{x}) = y$ must be true in every state satisfying $body$. For instance, the constraint:

$$loc(Ob) = Pl \ \textbf{if} \ loc(R) = Pl, \ \ in\_hand(R, Ob)$$

guarantees that the object grasped by a robot shares the robot's location.

- The definition of the value of a defined fluent f on $\bar{x}$ is a collection of statements of the form:

$$f(\bar{x}) \ \textbf{if} \ body \tag{6}$$

where $f(\bar{x})$ is true if it follows from the truth of at least one of its defining statements. Otherwise, $f(\bar{x})$ is false.

- Executability conditions are statements of the form:

$$\textbf{impossible} \ a_0, \ldots, a_k \ \textbf{if} \ body \tag{7}$$

which implies that in a state satisfying $body$, actions $a_0, \ldots a_k$ cannot occur simultaneously. For instance, the following executability condition:

$$\textbf{impossible} \ \ move(R, Pl) \ \ \textbf{if} \ \ loc(R) = Pl$$

implies that a robot cannot move to a location if it is already there; and

$$\textbf{impossible} \ \ grasp(R_1, Th), grasp(R_2, Th) \ \ \textbf{if} \ \ R_1 \neq R_2$$

prohibits two robots from simultaneously grasping the same thing.

We can now define the notion of a system description.

Definition 1. [System Description]
A *system description* of $\mathcal{AL}_d$ is a collection of $\mathcal{AL}_d$ statements over some action signature Σ. ▲

Next, we discuss the formal semantics of $\mathcal{AL}_d$.

5.1.2 FORMAL SEMANTICS OF $\mathcal{AL}_d$

The semantics of system description $\mathcal{D}$ of the new $\mathcal{AL}_d$ is similar to that of the old one. In fact, the old language can be viewed as the subset of $\mathcal{AL}_d$ in which all functions are boolean, causal laws are deterministic, and no distinction is made between physical and knowledge related actions and fluents. The semantics of $\mathcal{D}$ is given by a transition diagram $\tau(\mathcal{D})$ whose nodes correspond to possible states of the system. The diagram contains an arc $\langle \sigma_1, a, \sigma_2 \rangle$ if, after the execution of action a in state σ_1, the system may move into state σ_2. We define the states and transitions of $\tau(\mathcal{D})$ in terms of answer sets of logic programs, as described below—see (Gelfond & Inclezan, 2009; Gelfond & Kahl, 2014) for more details.

In what follows, unless otherwise stated, by "atom" and "term" we refer to "ground atom" and "ground term" respectively. Recall that an *interpretation* of the signature of $\mathcal{D}$ is an assignment of a value to each term $f(\bar{x})$ in the signature. An interpretation can be represented by the collection of atoms of the form $f(\bar{x}) = y$, where y is the value of $f(\bar{x})$. For any interpretation σ, let σ^{nd} be the collection of atoms of σ formed by basic fluents and statics—nd stands for *non-defined*. Let $\Pi_c(\mathcal{D})$, where c stands for constraints, denote the logic program defined as:

1. For every state constraint (Statement 5) and definition (Statement 6), program $\Pi_c(\mathcal{D})$ contains:

$$f(\bar{x}) = y \ \leftarrow body \tag{8}$$

2. For every defined fluent f, $\Pi_c(\mathcal{D})$ contains the closed world assumption (CWA):

$$\neg f(\bar{x}) \leftarrow \text{ not } f(\bar{x}) \tag{9}$$

   where, unlike classical negation "$\neg\, a$" that implies "a is believed to be false", *default negation* "$not\ a$" only implies that "a is not believed to be true", i.e., a can be true, false or just unknown.

We can now define states of $\tau(\mathcal{D})$.

Definition 2. [State of $\tau(\mathcal{D})$]
An interpretation σ is a **state** of the transition diagram $\tau(\mathcal{D})$ if it is the unique answer set of program $\Pi_c(\mathcal{D}) \cup \sigma^{nd}$. ▲

To illustrate the need for the uniqueness condition, consider the following example. Let system description $\mathcal{D}_s$ from (Gelfond & Kahl, 2014) with two defined fluents f and g defined by mutually recursive laws:

$$g \ \textbf{if} \ \neg f$$
$$f \ \textbf{if} \ \neg g$$

For this system description, the program $\Pi_c(\mathcal{D}_s)$ consists of the following statements:

$$g \ \leftarrow \ \neg f$$
$$f \ \leftarrow \ \neg g$$
$$\neg g \ \leftarrow \ not\ g$$
$$\neg f \ \leftarrow \ not\ f$$

and $\sigma^{nd} = \emptyset$ because all the fluents of $\mathcal{D}_s$ are defined. $\Pi_c(\mathcal{D}_s) \cup \sigma^{nd}$ has two answer sets $\{f, \neg g\}$ and $\{g, \neg f\}$; based on Definition 2, the transition diagram $\tau(\mathcal{D}_s)$ has no states. This outcome is expected because the mutually recursive laws are not strong enough to uniquely define f and g.

Next, we define a sufficient condition for guaranteeing that the defined fluents of a system description are uniquely defined by the system's statics and basic fluents. To do so, we introduce some terminology from (Gelfond & Kahl, 2014). A system description $\mathcal{D}$ is said to be *well-founded* if for any complete and consistent set of fluent literals and statics σ satisfying the state constraints of $\mathcal{D}$, program $\Pi_c(\mathcal{D}) \cup \sigma^{nd}$ has an unique answer set. Next, the *fluent dependency graph* of $\mathcal{D}$ is a directed graph such that:

- its vertices are arbitrary domain literals.
- it has an edge:
  - from l to l' if l is formed by a static or a basic fluent, and $\mathcal{D}$ contains a state constraint with the head l and the body containing l';
  - from f to l' if f is a defined fluent, and $\mathcal{D}$ contains a state constraint with the head f and body containing l' and not f; and
  - from $\neg f$ to f for every defined fluent f.

Also, a fluent dependency graph is said to be *weakly acyclic* if it does not contain paths from defined fluents to their negations. A system description with a weakly acyclic fluent dependency graph is also said to be weakly acyclic. Although well-foundedness is not easy to check, it is easy to check (and automate the checking of) weak acyclicity, and Proposition 8.4.1 in (Gelfond & Kahl, 2014) establishes weak acyclicity as a sufficient condition for well-foundedness (Gelfond & Inclezan, 2013). It is easy to show that all system descriptions discussed in this article are well-founded, a fact that we will use later in this article, e.g., in Proposition 1 in Section 5.2 and Proposition 2 in Section 5.3.

Next, to define the transition relation of $\tau(\mathcal{D})$, we first describe the logic programming encoding $\Pi(\mathcal{D})$ of $\mathcal{D}$. This encoding $\Pi(\mathcal{D})$ consists of the encoding of the signature of $\mathcal{D}$ and rules obtained from statements of $\mathcal{D}$, as described below.

Definition 3. [Logic programming encoding of $\mathcal{D}$]

- **Encoding of the signature:** we start with the encoding $sig(\mathcal{D})$ of signature of $\mathcal{D}$.
  - For each sort c, $sig(\mathcal{D})$ contains: $sort\_name(c)$.
  - For each subsort link $\langle c_1, c_2 \rangle$ of the hierarchy of sorts, $sig(\mathcal{D})$ contains: $s\_link(c_1, c_2)$.
  - For each constant $x : c$ from the signature of $\mathcal{D}$, $sig(\mathcal{D})$ contains: $m\_link(x, c)$.
  - For every static $g(\bar{x})$ of $\mathcal{D}$, $sig(\mathcal{D})$ contains: $static(g(\bar{x}))$.
  - For every basic fluent $f(\bar{x})$, $sig(\mathcal{D})$ contains: $fluent(basic, f(\bar{x}))$.
  - For every defined fluent $f(\bar{x})$, $sig(\mathcal{D})$ contains: $fluent(defined, f(\bar{x}))$.
  - For every observable fluent $f(\bar{x})$, $sig(\mathcal{D})$ contains: $obs\_fluent(f(\bar{x}))$.
  - For every directly observable fluent $f(\bar{x})$, $sig(\mathcal{D})$ contains: $dir\_obs\_fluent(f(\bar{x}))$.

- For every indirectly observable fluent $f(\bar{x})$, $sig(\mathcal{D})$ contains: $indir\_obs\_fluent(f(\bar{x}))$.
- For every action a of $\mathcal{D}$, $sig(\mathcal{D})$ contains: $action(a)$.

We also need axioms describing the hierarchy of basic sorts:

$$\begin{aligned} &subsort(C_1, C_2) \leftarrow s\_link(C_1, C_2) \\ &subsort(C_1, C_2) \leftarrow s\_link(C_1, C), \ subsort(C, C_2) \\ &member(X, C) \leftarrow m\_link(X, C) \\ &member(X, C_1) \leftarrow m\_link(X, C_0), \ subsort(C_0, C_1) \end{aligned} \tag{10}$$

- **Encoding of statements of $\mathcal{D}$:** To define transitions of our diagram we need two time-steps that stand for the beginning and the end of a transition. We would like, however, to later use the rules of our program to describe longer chains of events. To make this possible we introduce a symbolic constant n and allow time-steps of the program to range over $[0, max\_step]$. This is expressed by statement:

$$step(0..max\_step)$$

For defining transitions we set max_step to 1:

$$\#const \ max\_step = 1$$

We also need a relation $val(f(x_1, \ldots, x_n), y, i)$, which states that the value of $f(x_1, \ldots, x_n)$ at step i is y; and relation $occurs(a, i)$, which states that action a occurred at step i. We then encode statements of $\mathcal{D}$ as follows:

- For every deterministic causal law (Statement 2), $\Pi(\mathcal{D})$ contains a rule:

$$val(f(\bar{x}), y, I+1) \leftarrow val(body, I), \ occurs(a, I), \ I < n \tag{11}$$

  where $val(body, I)$ is obtained by replacing every literal of the form $f_m(\bar{x}_m) = z$ from $body$ by the literal $val(f_m(\bar{x}_m), z, I)$.
- For every non-deterministic causal law (described in Statements 3-4), where the range of f is $\{y_1, \ldots, y_k\}$, $\Pi(\mathcal{D})$ contains a rule:

$$\begin{aligned} val(f(\bar{x}), y_1, I+1) \ \textbf{or} \ \ldots \textbf{or} \ val(f(\bar{x}), y_k, I+1) \leftarrow & val(body, I), \\ & occurs(a, I), \ I < n \end{aligned} \tag{12}$$

  For Statement 3, to encode that due to this action, $f(\bar{x})$ only takes a value that satisfies property p, $\Pi(\mathcal{D})$ contains a constraint:

$$\leftarrow val(f(\bar{x}), Y, I+1), \ \text{not} \ val(p(Y), true, I) \tag{13}$$

  For Statement 4, we need a similar constraint:

$$\leftarrow val(f(\bar{x}), Y, I+1), \ \text{not} \ member(Y, sort\_name) \tag{14}$$

  The next two axioms guarantee that in the case of Statement 3, action a is not executed in a state in which property p is not satisfied:

$$\begin{aligned} satisfied(p, I) &\leftarrow val(p(Y), true, I) \\ \neg occurs(a, I) &\leftarrow \ \text{not} \ satisfied(p, I) \end{aligned} \tag{15}$$

– For every state constraint and definition (Statements 5, 6), $\Pi(\mathcal{D})$ contains:

$$val(f(\bar{x}), y, I) \leftarrow val(body, I) \tag{16}$$

– $\Pi(\mathcal{D})$ contains the CWA for defined fluents:

$$val(F, false, I) \leftarrow fluent(defined, F),\ \text{not}\ val(F, true, I) \tag{17}$$

– For every executability condition (Statement 7), $\Pi(\mathcal{D})$ contains:

$$\neg occurs(a_0, I)\ \mathbf{or}\ \ldots\ \mathbf{or}\ \neg occurs(a_k, I) \leftarrow val(body, I),\ I < n \tag{18}$$

– For every static $g(\bar{x})$, $\Pi(\mathcal{D})$ contains:

$$g(\bar{x}) = y \tag{19}$$

– $\Pi(\mathcal{D})$ contains the Inertia Axiom:

$$\begin{aligned} val(F, Y, I+1) \leftarrow & fluent(basic, F), \\ & val(F, Y, I),\ not\ \neg val(F, Y, I+1),\ I < n \end{aligned} \tag{20}$$

– $\Pi(\mathcal{D})$ contains CWA for actions:

$$\neg occurs(A, I) \leftarrow not\ occurs(A, I),\ I < n \tag{21}$$

– Finally, we need the rule:

$$\neg val(F, Y_1, I) \leftarrow val(F, Y_2, I),\ Y_1 \neq Y_2 \tag{22}$$

which says that a fluent can only have one value at each time step.

This completes the construction of encoding $\Pi(\mathcal{D})$ of system description $\mathcal{D}$. Later we will consider a version of $\mathcal{D}$ in which time step max_step is set to some positive number k. We denote such a program by $\Pi^k(\mathcal{D})$. ▲

Recall that the axioms described above are shorthand for the set of ground instances obtained by replacing variables by ground terms from the corresponding sorts. We now formally define a transition of the transition diagram $\tau(\mathcal{D})$.

Definition 4. [Transition of $\tau(\mathcal{D})$]
Let a be a non-empty collection of actions, and σ_0 and σ_1 be states of the transition diagram $\tau(\mathcal{D})$ defined by system description $\mathcal{D}$. To describe a transition $\langle \sigma_0, a, \sigma_1 \rangle$, we construct a program $\Pi(\mathcal{D}, \sigma_0, a)$ comprising:

- Logic programming encoding $\Pi(\mathcal{D})$ of system description $\mathcal{D}$, as described above.
- The encoding $val(\sigma_0, 0)$ of initial state σ_0:

$$\begin{aligned} val(\sigma_0, 0) =_{def}\ & \{val(f(\bar{x}), y, 0) : (f(\bar{x}) = y) \in \sigma_0\},\ \text{where } f \text{ is a fluent } \cup \\ & \{f(\bar{x}) = y : (f(\bar{x}) = y) \in \sigma_0\}\ \text{where } f \text{ is a static} \end{aligned}$$

- Encoding $occurs(a, 0)$ of set of actions a:

$$occurs(a, 0) =_{def} \{occurs(a_i, 0) : a_i \in a\}$$

In other words, the program $\Pi(\mathcal{D}, \sigma_0, a)$ includes our description of the system's laws, the initial state, and the actions that occur in it:

$$\Pi(\mathcal{D}, \sigma_0, a) =_{def} \Pi(\mathcal{D}) \cup val(\sigma_0, 0) \cup occurs(a, 0)$$

A state-action-state triple $\langle \sigma_0, a, \sigma_1 \rangle$ is a **transition** of $\tau(\mathcal{D})$ iff $\Pi(\mathcal{D}, \sigma_0, a)$ has an answer set AS such that $\sigma_1 = \{f(\bar{x}) = y : val(f(\bar{x}), y, 1) \in AS\}$. The answer sets of $\Pi(\mathcal{D}, \sigma_0, a)$ thus determine the states the system can move into after executing of a in σ_0.

▲

5.2 Histories with Defaults

In action languages, domain knowledge is typically represented by a system description containing general knowledge about the domain and the agent's abilities, and the domain's *recorded history* containing information pertinent to the agent's activities in the domain. This history $\mathcal{H}$ typically contains the agent's observations of the value of domain attributes, and the occurrences of actions, as recorded by statements of the form:

$$obs(rob_1, f(\bar{x}), y, true, i) \tag{23a}$$
$$obs(rob_1, f(\bar{x}), y, false, i) \tag{23b}$$

and

$$hpd(a, i) \tag{24}$$

where f is an observable fluent, y is a possible value of this fluent, a is an action, and i is a time-step. Statements 23(a-b) say that a particular fluent was observed to have (or not have) a particular value at time-step i by robot rob_1, and Statement 24 says that action a happened at time-step i. For instance, $obs(rob_1, loc(tb_1), office, true, 0)$ denotes the observation of textbook tb_1 in the $office$ by robot rob_1, and $hpd(move(rob_1, kitchen), 1)$ is the record of successful execution of rob_1's move to the kitchen at time step 1. Note that the standard representation of obs does not include the $robot$ as the first argument—we include it to emphasize that observations are obtained by specific robots. Also, for convenience, we write $obs(rob_1, f(\bar{x}), y, true, i)$ as $obs(rob_1, f(\bar{x}) = y, i)$, and $obs(rob_1, f(\bar{x}), y, false, i)$ as $obs(rob_1, f(\bar{x}) \neq y, i)$. In addition, the notion of observations at the coarse resolution is different from that of observations obtained from sensor inputs, which are modeled at the fine resolution; the former is based on the latter, as described in Section 7.1. Furthermore, there is a subtle difference between relation $occurs$ used in logic programming encoding of system descriptions and relation hpd. Statement $occurs(a, i)$ may denote an actual occurrence of action a at i as well as a hypothetical action (e.g., in a plan computed for a specific goal), whereas $hpd(a, i)$ indicates that a was actually executed at i. For a discussion on the need for such a distinction between hpd and $occurs$, please see Section 10.5 in (Gelfond & Kahl, 2014).

We say that n is the *current step* of history $\mathcal{H}$ if $n - 1$ is the maximum time step occurring in statements of the form $hpd(a, i)$ in $\mathcal{H}$. If no such statement exists, the current step of $\mathcal{H}$ is 0. The

recorded history thus defines a collection of paths in the transition diagram that, from the standpoint of the agent, can be interpreted as the system's possible pasts. The precise formalization of this is given by the notion of a *model* of the recorded history. The definition of such a model for histories consisting of Statements 23 and 24 can be found in Section 10.1 in (Gelfond & Kahl, 2014).

In our work, we extend the syntax and semantics of recorded histories to support a more convenient description of the domain's initial state. In addition to the statements above, we introduce an additional type of historical record:

$$\textbf{initial default } d(\bar{x}) : f(\bar{x}) = y \textbf{ if } body(d) \tag{25}$$

and:

$$\textbf{prefer}(d_1, d_2) \tag{26}$$

where f is a basic fluent and the ds are the names of defaults. Statements 25 and 26 refer to the initial state of the system. Statement 25 is a default named d stating that in any initial state satisfying $body(d)$, the default value of $f(\bar{x})$ is y. Statement 26 defines an anti-symmetric and transitive preference relation between defaults, stating that if the simultaneous application of defaults d_1 and d_2 leads to a contradiction, then d_1 is preferred to d_2.

The addition of defaults makes the task of defining models substantially more challenging. Before providing a formal semantics of a recorded history with defaults (i.e., before defining models of such histories), we illustrate the intended meaning of these statement with an example.

Example 2. [Example of initial state defaults]
Consider the following statements about the locations of textbooks in the initial state in our illustrative domain. *Textbooks are typically in the main library. If a textbook is not there, it is typically in the auxiliary library. If a textbook is checked out, it is usually in the office.* These defaults can be represented as:

$$\textbf{initial default } d_1(X) \ : \ loc(X) = main\_library \textbf{ if } textbook(X) \tag{27}$$

$$\textbf{initial default } d_2(X) \ : \ loc(X) = aux\_library \textbf{ if } textbook(X) \tag{28}$$

$$\textbf{initial default } d_3(X) \ : \ loc(X) = office \textbf{ if } textbook(X) \tag{29}$$

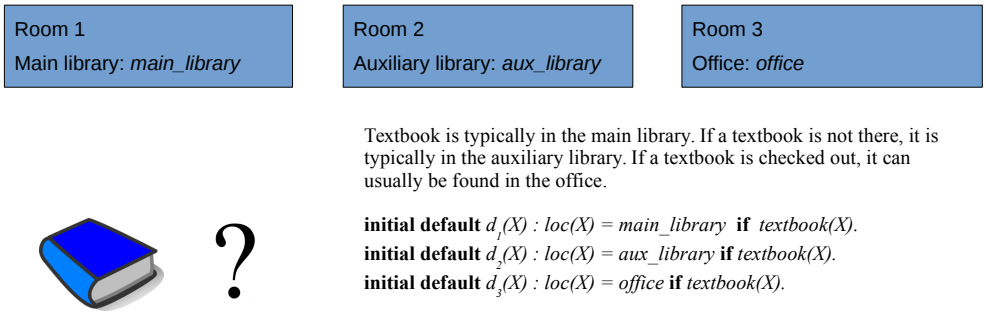
$$\begin{array}{l} \textbf{prefer}(d_1(X), d_2(X)) \\ \textbf{prefer}(d_2(X), d_3(X)) \end{array} \tag{30}$$

where the fluent $\{loc : thing \rightarrow place\}$, as before, represents the place where a particular thing is located. A history $\mathcal{H}_a$ with the above statements will entail $val(loc(tb_1) = main\_library, true, 0)$ for textbook tb_1 using default $d_1(tb_1)$. The other two defaults (Statements 28, 29) are disabled (i.e., not used) due to Statement 30 and the transitivity of the $prefer$ relation. A history $\mathcal{H}_b$ that adds $obs(rob_1, loc(tb_1) \neq main\_library, 0)$ as an observation to $\mathcal{H}_a$ renders default $d_1(tb_1)$ (see Statement 27) inapplicable. Now the second default (i.e., Statement 28) is enabled and entails $val(loc(tb_1) = aux\_library, true, 0)$. A history $\mathcal{H}_c$ that adds observation $obs(rob_1, loc(tb_1) \neq$

$aux\_library, 0)$ to $\mathcal{H}_b$ should entail $val(loc(tb_1) = office, true, 0)$. In all these histories, the defaults were defeated by initial observations and by higher priority defaults.

Now, consider the addition of observation $obs(rob_1, loc(tb_1) \neq main\_library, 1)$ to $\mathcal{H}_a$ to obtain history $\mathcal{H}_d$. This observation is different because it defeats default $d_1(tb_1)$, but forces the agent to reason back in time. If the default's conclusion, $loc(tb_1) = main\_library$, were true in the initial state, it would also be true at step 1 (by inertia), which would contradict the observation. Default $d_2(tb_1)$ will be used to conclude that textbook tb_1 is initially in the $aux\_library$; the inertia axiom will propagate this information to entail $val(loc(tb_1) = aux\_library, true, 1)$. For more information on indirect exceptions to defaults and their formalization see Section 5.5 in (Gelfond & Kahl, 2014).

Figure 2 illustrates the beliefs corresponding to these four histories—the column labeled "CR rule outcome" and the row labeled "$\mathcal{H}_e$" are explained later in this section. Please see `example2.sp` at `https://github.com/mhnsrdhrn/refine-arch` for the complete program formalizing this reasoning in SPARC. ⧫



	Observation(s)	CR-rule outcome(s)	Applicable default	Believed location of tb_1
H_a	∅	∅	d_1	main_library
H_b	obs(rob$_1$, loc(tb$_1$) ≠ main_library, 0)	ab(d$_1$)	d_2	aux_library
H_c	obs(rob$_1$, loc(tb$_1$) ≠ main_library, 0) obs(rob$_1$, loc(tb$_1$) ≠ aux_library, 0)	ab(d$_1$), ab(d$_2$)	d_3	office
H_d	obs(rob$_1$, loc(tb$_1$) ≠ main_library, 1)	ab(d$_1$)	d_2	aux_library
H_e	obs(rob$_1$, loc(tb$_1$) ≠ main_library, 1) obs(rob$_1$, loc(tb$_1$) ≠ aux_library, 1) obs(rob$_1$, loc(tb$_1$) ≠ office, 1)	ab(d$_1$), ab(d$_2$), ab(d$_3$)	none	kitchen

Figure 2: Illustration of the beliefs of a robot corresponding to the histories with the same initial state defaults, as described in Example 2 and Example 3.

To better understand the definition of histories with defaults, recall the definition of a model for histories not containing defaults. In this case, a model of $\mathcal{H}^n$ is a path $M = \langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_n, a_n \rangle$ of $\tau(\mathcal{D})$ such that:

- M *satisfies* every $obs(rob_1, f(\bar{x}) = y, i) \in \mathcal{H}^n$, i.e., for every such observation, we also have that $(f(\bar{x}) = y) \in \sigma_i$.
- $a_i = \{e : hpd(e, i) \in \mathcal{H}^n\}$.

In the presence of defaults, however, these conditions, though necessary, are not sufficient. Consider, for instance, history $\mathcal{H}_a$ from Example 2. Since it contains no actions or observations, these conditions are satisfied by any path $M = \langle \sigma_0 \rangle$. However, M is a model of $\mathcal{H}_a$ only if σ_0 contains $loc(tb1) = main\_library$. In general, to define the initial states of models of $\mathcal{H}^n$, we need to understand reasoning in the presence of defaults, along with their direct and indirect exceptions. The situation is similar to, but potentially more complex than, the definition of transitions of $\tau(\mathcal{D})$. To define the models of $\mathcal{H}^n$, we thus pursue an approach similar to that used to define the transitions of $\tau(\mathcal{D})$. Specifically, we define models of $\mathcal{H}^n$ in terms of answer sets of the logic program $\Pi(\mathcal{D}, \mathcal{H})$ that axiomatizes the agent's knowledge. However, due to the presence of indirect exceptions, our language of choice will be CR-Prolog, an extension of ASP well-suited for representing and reasoning with such knowledge. The syntax and semantics of CR-Prolog can be found in Appendix A.1. For more information about CR-Prolog, and its use for reasoning with defaults and their exceptions, please see (Gelfond & Kahl, 2014). We begin by defining the program encoding both $\mathcal{D}$ and $\mathcal{H}$.

Definition 5. [Program $\Pi(\mathcal{D}, \mathcal{H})$]
Program $\Pi(\mathcal{D}, \mathcal{H})$, which encodes the system description $\mathcal{D}$ and history $\mathcal{H}$ of the domain, is obtained by changing the value of constant n in $\Pi(\mathcal{D})$ from 1 to the current step of $\mathcal{H}$ and adding to the resulting program:

- Observations and actions, i.e., Statements 23 and 24, from $\mathcal{H}$.
- Encoding of each default, i.e., for every default such as Statement 25 from $\mathcal{H}$, we add:

$$val(f(\bar{x}), y, 0) \leftarrow val(body(d(\bar{x})), 0), \text{ not } ab(d(\bar{x})) \tag{31}$$

$$ab(d(\bar{x})) \stackrel{+}{\leftarrow} val(body(d(\bar{x})), 0) \tag{32}$$

  where Statement 31 is a simplified version of the standard CR-Prolog (or ASP) encoding of a default, and the relation $ab(d)$, read as *default d is abnormal*, holds when default d is not applicable. Statement 32 is a *consistency restoring* (CR) rule, which says that to restore consistency of the program one may refrain from applying default d. It is an axiom in CR-Prolog used to allow indirect exceptions to defaults—it is not used unless assuming $f(\bar{x}) = y$ leads to a contradiction.

- Encoding of preference relations. If there are two or more defaults with preference relations, e.g., Statements 27-30, we first add the following:

$$ab(D_2) \leftarrow prefer(D_1, D_2), val(body(D_1), 0), \text{not } ab(D_1) \tag{33}$$

where D_1 and D_2 are defaults. Then, we add the following:

$$prefer(D_1, D_3) \leftarrow prefer(D_1, D_2), \; prefer(D_2, D_3) \tag{34}$$

$$\neg prefer(D, D) \tag{35}$$

Statement 33 prevents the applicability of a default if another (preferred) default is applicable. The other two axioms (Statements 34, 35) express transitivity and anti-symmetry of the preference relation.

- Rules for initial observations, i.e., for every basic fluent f and its possible value y:

$$val(f(\bar{x}), y, 0) \leftarrow obs(rob_1, f(\bar{x}) = y, 0) \tag{36a}$$

$$\neg val(f(\bar{x}), y, 0) \leftarrow obs(rob_1, f(\bar{x}) \neq y, 0) \tag{36b}$$

These axioms say that the initial observations are correct. Among other things they may be used to defeat the defaults of $\mathcal{H}$.

- Assignment of initial values to basic fluents that have not been defined by other means. Specifically, the initial value of a basic fluent not defined by a default is selected from the fluent's range. To do so, for every initial state default (of the form of Statement 25) from $\mathcal{H}$:

$$defined\_by\_default(f(\bar{x})) \leftarrow val(body(d(\bar{x})), 0), \; \text{not } ab(d(\bar{x})) \tag{37}$$

Then, for every basic fluent f:

$$val(f(\bar{x}), y_1, 0) \text{ or } \ldots \text{ or } val(f(\bar{x}), y_n, 0) \leftarrow \text{ not } defined\_by\_default(f(\bar{x})) \tag{38}$$

where $\{y_1, \ldots, y_n\}$ are elements in the range of $f(\bar{x})$ that do not occur in the head of any initial state default of $\mathcal{H}$.

- A reality check (Balduccini & Gelfond, 2003a):

$$\leftarrow val(F, Y_1, I), \; obs(rob_1, F = Y_2, I), \; Y_1 \neq Y_2 \tag{39}$$

which says that the value of a fluent predicted by our program shall not differ from its observed value, or (equivalently) that a robot cannot simultaneously observe a fluent to have a particular value and believe that the fluent has a different value. It does not, however, directly force everything observed to hold true—there is a subtle difference between these two interpretations, as discussed further in Section 10.5 in (Gelfond & Kahl, 2014).

- And a rule:

$$occurs(A, I) \leftarrow hpd(A, I) \tag{40}$$

which establishes the relation between relation hpd of the language of recorded histories and relation $occurs$ used in the program. Recall that $occurs$ denotes both actual and hypothetical occurrences of actions, whereas hpd indicates an actual occurrence.

This completes construction of the program. ▲

We will also need the following terminology in the discussion below. Let $\mathcal{H}$ be a history of $\mathcal{D}$ and AS be an answer set of $\Pi(\mathcal{D}, \mathcal{H})$. We say that a sequence $M = \langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_n, a_n \rangle$ such that $\forall i \in [0, n]$:

- $\sigma_i = \{f = y : val(f, y, i) \in AS\}$,
- $a_i = \{e : hpd(e, i) \in AS\}$.

is *induced* by AS. Now we are ready to define semantics of $\mathcal{H}$.

Definition 6. [Model]
A sequence $\langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_n, a_n \rangle$ induced by an answer set AS of $\Pi(\mathcal{D}, \mathcal{H})$ is called a *model* of $\mathcal{H}$ if it is a path of transition diagram $\tau(\mathcal{D})$. ▲

Definition 7. [Entailment]
A literal l is true at step i of a path $M = \langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_n, a_n \rangle$ of $\tau(\mathcal{D})$ if $l \in \sigma_i$. We say that l is *entailed* by a history $\mathcal{H}$ of $\mathcal{D}$ if l is true in all models of $\mathcal{H}$. ▲

The following proposition shows that for well-founded system descriptions this definition can be simplified.

Proposition 1. [Answer sets of $\Pi(\mathcal{D}, \mathcal{H})$ and paths of $\tau(\mathcal{D})$]
If $\mathcal{D}$ is a well-founded system description and $\mathcal{H}$ is its recorded history, then every sequence induced by an answer set of $\Pi(\mathcal{D}, \mathcal{H})$ is a model of $\mathcal{H}$. ★

The proof of this proposition is in Appendix A.2. This proposition guarantees that for a well-founded system description $\mathcal{D}$, computing a model of a history $\mathcal{H}$ can be reduced to just computing the answer set of the program $\Pi(\mathcal{D}, \mathcal{H})$, i.e., we do not need to check if such an answer set is a path of the transition diagram $\tau(\mathcal{D})$. Next, we look at some examples of histories with defaults.

Example 3. [Example 2 revisited]
Let us revisit the histories described in Example 2 and show how models of system descriptions from this example can be computed using our axiomatization $\Pi(\mathcal{D}, \mathcal{H})$ of models of a recorded history. We see that models of $\mathcal{H}_a$ are of the form $\langle \sigma_0 \rangle$ where σ_0 is a state of the system containing $\{loc(tb_1) = main\_library\}$. Since $textbook(tb_1)$ is a static relation, it is true in every state of the system. The axiom encoding default d_1 (Statement 31) is not blocked by a CR rule (Statement 32) or a preference rule (Statement 33), and the program entails $val(loc(tb_1), main\_library, 0)$. Thus, $\{loc(tb_1) = main\_library\} \in \sigma_0$.

Now consider history $\mathcal{H}_b$ containing $obs(rob_1, loc(tb_1) \neq main\_library, 0)$. Based on rules for initial observations (Statement 36) we have $\neg val(loc(tb_1), main\_library, 0)$ which contradicts the first default. The corresponding CR-rule (Statement 32) restores consistency by assuming $ab(d_1)$, making default d_1 inapplicable. Default $d_2(tb_1)$, which used to be blocked by a preference rule (i.e., $prefer(d_1(tb_1), d_2(tb_1))$), becomes unblocked and we conclude that $val(loc(tb_1), aux\_library, 0)$. Models of $\mathcal{H}_b$ are states of $\tau(\mathcal{D})$ that contain $\{loc(tb_1) = aux\_library\}$. The models of $\mathcal{H}_c$ in Example 2 are computed in a similar manner.

Recall that the last history, $\mathcal{H}_d$, is slightly different. The current step of $\mathcal{H}_d$ is 1 and its models are of the form $\langle \sigma_0, a, \sigma_1 \rangle$. It contains $obs(rob_1, loc(tb_1) \neq main\_library, 1)$. Since $\Pi(\mathcal{D}, \mathcal{H}_d)$ has no

rules with an action in the head, $a = \{\ \}$. Based on default d_1, $\{loc(tb_1) = main\_library\}$ should belong to state σ_0. However, if this were true, $\{loc(tb_1) = main\_library\}$ would belong to σ_1 by inertia, which contradicts the observation and the reality check axiom creates an inconsistency. This inconsistency is resolved by the corresponding CR-rule (Statement 32) by assuming $ab(d_1)$ in the initial state, i.e., at time 0. Default d_2 is activated and the reasoner infers $\{loc(tb_1) = aux\_library\}$ at time step 0 and (by inertia) at time step 1.

To illustrate the use of axioms governing the initial value of a basic fluent not defined by a default (Statements 37 and 38), consider history $\mathcal{H}_e$ in which observations at step 1 establish that textbook tb_1 is not in any of the default locations. An argument similar to that used for $\mathcal{H}_d$ would allow the reasoner to conclude $ab(d_1(tb_1))$, $ab(d_2(tb_1))$, and $ab(d_3(tb_1))$, and $defined\_by\_default(loc(tb_1))$ can not be derived. Statement 38 is now used to allow a choice between the four locations that form the range of the $loc()$ function. The first three are eliminated by observations at step 1 and we thus conclude $val(loc(tb_1), kitchen, 0)$, i.e., $\{loc(tb_1) = kitchen\} \in \sigma_1$. Note that if the domain included other available locations, we would have additional models of history $\mathcal{H}_e$. ♦

Example 4. [Examples of models of history]
As further examples of models of history, consider a system description $\mathcal{D}_a$ with basic boolean fluents f and g (and no actions), and a history $\mathcal{H}_a$ consisting of:

$$\textbf{initial default } \neg g \textbf{ if } f$$

The paths of this history consist of states without any transitions. Using axiom in Statement 38, we see that $\{f, \neg g\}$, $\{\neg f, g\}$, and $\{\neg f, \neg g\}$ are models of $\langle \mathcal{D}_a, \mathcal{H}_a \rangle$ and $\sigma = \{f, g\}$ is not. The latter is not surprising since even though σ may be physically possible, the agent, relying on the default, will not consider σ to be compatible with the default since the history gives no evidence that the default should be violated. If, however, the agent were to record an observation $obs(rob_1, g, 0)$, the only states compatible with the resulting history $\mathcal{H}_b$ would be $\{f, g\}$ and $\{\neg f, g\}$.)

Next, we expand our system description $\mathcal{D}_a$ by a basic fluent h and a state constraint:

$$h \textbf{ if } \neg g$$

In this case, to compute models of a history $\mathcal{H}_c$ of a system $\mathcal{D}_b$, where $\mathcal{H}_c$ consists of the default in $\mathcal{H}_a$ and an observation $obs(rob_1, \neg h, 0)$, we need CR rules. The models are $\{f, \neg h, g\}$ and $\{\neg f, \neg h, g\}$.

Next, consider a system description $\mathcal{D}_c$ with basic fluents f, g, and h, the initial-state default, and an action a with the following causal law:

$$a \textbf{ causes } h \textbf{ if } \neg g$$

and a history $\mathcal{H}_d$ consisting of $obs(rob_1, f, 0)$, $hpd(a, 0)$. We then have $\langle \{f, \neg g, h\}, a, \{f, \neg g, h\} \rangle$ and $\langle \{f, \neg g, \neg h\}, a, \{f, \neg g, h\} \rangle$ as the two models of $\mathcal{H}_d$. Finally, history $\mathcal{H}_e$ obtained by adding $obs(rob_1, \neg h, 1)$ to $\mathcal{H}_d$ has a single model $\langle \{f, g, \neg h\}, a, \{f, g, h\} \rangle$. The new observation is an indirect exception to the initial default, which is resolved by the corresponding CR rule. ♦

5.3 Reasoning

The main reasoning task of an agent with a high level deterministic system description $\mathcal{D}$ and history $\mathcal{H}$ is to find a plan (i.e., a sequence of actions[1]) that would allow it to achieve goal G. We assume that the length of this sequence is limited by some number h referred to as the planning horizon. This is a generalization of a classical planning problem in which the history consists of a collection of atoms which serves as a complete description of the initial state. If history $\mathcal{H}$ has exactly one model, the situation is not very different from classical planning. The agent believes that the system is currently in some unique state σ_n—this state can be found using Proposition 1 that reduces the task of computing the model of $\mathcal{H}$ to computing the answer set of $\Pi(\mathcal{D}, \mathcal{H})$. Finding a plan is thus equivalent to solving a classical planning problem $\mathcal{P}_c$, i.e., finding a sequence of actions of length not exceeding h, which leads the agent from an initial state σ to a state satisfying G. The ASP-based solution of this planning problem can be traced back to work described in (Dimopoulos, Koehler, & Nebel, 1997; Subrahmanian & Zaniolo, 1995). Also see program $plan(\mathcal{P}_c, h)$ and Proposition 9.1.1 in Section 9.1 of (Gelfond & Kahl, 2014), which establish the relationship between answer sets of this program and solutions of $\mathcal{P}_c$, and can be used to find a plan to achieve the desired goal. A more subtle situation arises when $\mathcal{H}$ has multiple models. Since there are now multiple possible current states, we can either search for a *possible plan*, i.e. a plan leading to G from at least one of the possible current states, or for a *conformant* plan, i.e., a plan that can achieve G independent of the current state. In this article, we only focus on the first option[2].

Definition 8. [Planning Problem]
We define a *planning problem* $\mathcal{P}$ as a tuple $(\mathcal{D}, \mathcal{H}, h, G)$ consisting of system description $\mathcal{D}$, history $\mathcal{H}$, planning horizon h and a goal G. A sequence $\langle a_0, \ldots, a_{k-1} \rangle$ is called a *solution* of $\mathcal{P}$ if there is a state σ such that:

- σ is the current state of some model M of $\mathcal{H}$; and
- $\langle a_0, \ldots, a_{k-1} \rangle$ is a solution of classical planning problem $\mathcal{P}_c = (\mathcal{D}, \sigma, G)$ with horizon h.

▲

To find a solution of $\mathcal{P}$ we consider:

- CR-Prolog program $Diag =_{def} \Pi^n(\mathcal{D}, \mathcal{H})$ with maximum time step n where n is the current step of $\mathcal{H}$.
- ASP program $Classical\_plan$ consisting of:
  1. $\Pi^{[n..n+h]}(\mathcal{D})$ obtained from $\Pi(\mathcal{D})$ by setting max_step to $n + h$ and sort $step$ to (n, max_step).
  2. Encoding of the goal $f(\bar{x}) = y$ by the rule:

$$goal(I) \leftarrow val(f(\bar{x}), y, I)$$

1. For simplicity we only consider sequential plans in which only one action occurs at a time. The approach can be easily modified to allow actions to be performed in parallel.
2. An ASP-based approach to finding conformant plans can be found in (Tu, Son, Gelfond, & Morales, 2011).

3. Simple planning module, PM, obtained from that in Section 9.1 of (Gelfond & Kahl, 2014) (see statements on page 194) by letting time step variable I range between n and $n + h$.

- Diagnoses Preserving Constraint (DPC):

$$\leftarrow Y = count\{X : ab(X)\}, Y > m$$

where m is the size of abductive support of $Diag$[3]. For any program Π, if Π^{reg} is the set of all regular rules of Π and $\alpha(R)$ is the set of regular rules obtained by replacing $\stackrel{+}{\leftarrow}$ by $\leftarrow$ in each CR rule in R, a cardinality-minimal set of CR rules such that $\Pi(R) =_{def} \Pi^{reg} \cup \alpha(R)$ is consistent, is called an abductive support of Π[4].

We reduce the task of finding the solutions of the planning problem $\mathcal{P}$ to:

1. Computing the size, m, of an abductive support of $Diag$.
2. Computing answer sets of CR-Prolog program:

$$Plan = Diag \cup Classical\_plan \cup \{DPC\}$$

Based on Proposition 1, the first sub-task of finding the abductive support of $Diag$ can be accomplished by computing a model of:

$$Diag \cup \{size(Y) \leftarrow count\{X : ab(X)\} = Y\}$$

and displaying atom $size(m)$ from this model. The second sub-task (and overall task) of reducing planning to computing answer sets is based on the following proposition that is analogous to Proposition 9.1.1 in Section 9.1 of (Gelfond & Kahl, 2014).

Proposition 2. [Reducing planning to computing answer sets]
Let $\mathcal{P} = (\mathcal{D}, \mathcal{H}, h, G)$ be a planning problem with a well-founded, deterministic system description $\mathcal{D}$. A sequence $\langle a_0, \ldots, a_{k-1}\rangle$ where $k < h$ is a solution of $\mathcal{P}$ iff there is an answer set A of $Plan$ such that:

1. For any $n < i \leq n + k$, $occurs(a_i, i - 1) \in A$,
2. A contains no other atoms of the form $occur(*, i)$[5] with $i \geq n$.

★

The proof of this proposition is provided in Appendix B. Similar to classical planning, it is possible to find plans for our planning problem that contain irrelevant, unnecessary actions. We can avoid this problem by asking the planner to search for plans of increasing length, starting with plans of length 1, until a plan is found (Gebser, Kaminski, Kaufmann, & Schaub, 2014). There are other ways to find minimum-length plans, but we do not discuss them here.

3. Here *count* is an aggregate function. For semantics of ASP with aggregates, see for instance (Gelfond & Zhang, 2014).
4. Although a program may have multiple abductive support, they all have the same size due to the minimality requirement.
5. The "*" denotes a wild-card character.

6. Logician's Domain Representation

We are now ready for the first step of our design methodology (see Section 4), which is to provide a coarse-resolution description of the robot's domain in $\mathcal{AL}_d$ along with a description of the initial state—we re-state this step as specifying the transition diagram of the logician.

1. Specify the transition diagram, τ_H, which will be used by the logician for coarse-resolution reasoning, including planning and diagnostics.

This step is accomplished by providing the signature and $\mathcal{AL}_d$ axioms of system description $\mathcal{D}_H$ defining this diagram. We will use standard techniques for representing knowledge in action languages, e.g., (Gelfond & Kahl, 2014). We illustrate this process by describing the domain representation for the office domain introduced in Example 1.

Example 5. [Logician's domain representation]
The system description $\mathcal{D}_H$ of the domain in Example 1 consists of a sorted signature (Σ_H) and axioms describing the transition diagram τ_H. Σ_H defines the names of objects and functions available for use by the logician. Building on the description in Example 1, Σ_H has an ontology of sorts, i.e., sorts such as $place$, $thing$, $robot$, and $object$, which are arranged hierarchically, e.g., $object$ and $robot$ are subsorts of $thing$, and $textbook$ and cup are subsorts of $object$. The statics include a relation $next\_to : place \times place \rightarrow boolean$, which holds iff two places are next to each other. This domain has two basic fluents that are subject to the laws of inertia: $loc : thing \rightarrow place$, $in\_hand : robot \times object \rightarrow boolean$. For instance, the $loc(Th) = Pl$ if thing Th is located at place Pl, and the value of $in\_hand(R, Ob)$ is $true$ if robot R is holding object Ob. In this domain, the basic fluents are observable.

The domain has three actions: $move(robot, place)$, $grasp(robot, object)$, and $putdown(robot, object)$. The domain dynamics are defined using axioms that consist of causal laws such as:

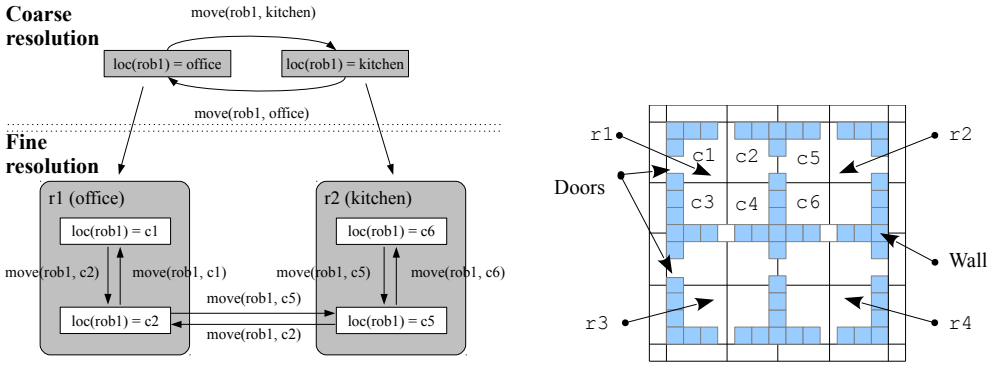
$$move(R, Pl) \ \textbf{causes} \ loc(R) = Pl \tag{41a}$$
$$grasp(R, Ob) \ \textbf{causes} \ in\_hand(R, Ob) \tag{41b}$$
$$putdown(R, Ob) \ \textbf{causes} \ \neg in\_hand(R, Ob) \tag{41c}$$

state constraints such as:

$$loc(Ob) = Pl \ \textbf{if} \ loc(R) = Pl, \ \ in\_hand(R, Ob) \tag{42a}$$
$$next\_to(P1, P2) \ \textbf{if} \ next\_to(P2, P1) \tag{42b}$$

and executability conditions such as:

$$\textbf{impossible} \ move(R, Pl) \ \textbf{if} \ loc(R) = Pl \tag{43a}$$
$$\textbf{impossible} \ move(R, Pl_2) \ \textbf{if} \ loc(R) = Pl_1, \ \neg next\_to(Pl_1, Pl_2) \tag{43b}$$
$$\textbf{impossible} \ A_1, \ A_2 \ \textbf{if} \ A_1 \neq A_2 \tag{43c}$$
$$\textbf{impossible} \ grasp(R, Ob) \ \textbf{if} \ loc(R) = Pl_1, \ loc(Ob) = Pl_2, \ Pl_1 \neq Pl_2 \tag{43d}$$
$$\textbf{impossible} \ grasp(R, Ob) \ \textbf{if} \ in\_hand(R, Ob) \tag{43e}$$
$$\textbf{impossible} \ putdown(R, Ob) \ \textbf{if} \ \neg in\_hand(R, Ob) \tag{43f}$$



(a) Some state transitions (b) Different resolutions

Figure 3: (a) Illustration of state transitions for specific $move$ actions in our illustrative (office) domain, viewed at coarse resolution and at fine resolution; and (b) A closer look at specific places brings into focus the corresponding rooms and grid cells in those rooms.

The part of Σ_H described so far, the sort hierarchy and the signatures of functions, is unlikely to undergo substantial changes for any given domain. However, the last step in the constructions of Σ_H is likely to undergo more frequent revisions—it populates the sorts of the hierarchy with specific objects; e.g $robot = \{rob_1\}$, $place = \{r_1, \ldots, r_n\}$ where rs are rooms, $textbook = \{tb_1, \ldots\ tb_m\}$, $kitchenware = \{cup_1, cup_2, plate_1, plate_2\}$ etc. Ground instances of the axioms are obtained by replacing variables by ground terms from the corresponding sorts.

The transition diagram τ_H described by $\mathcal{D}_H$ is too large to depict in a picture. The top part of Figure 3(a) shows the transitions of τ_H corresponding to a $move$ between two places. The only fluent shown there is the location of the robot rob_1—the values of other fluents remain unchanged and are not shown. The actions of this transition diagram τ_H of the logician, as described above, are assumed to be deterministic. Also, the values of coarse-resolution fluents are assumed to be known at each step. These assumptions allow the robot to do fast, tentative planning and diagnostics necessary for achieving its assigned goals.

The domain representation described above should ideally be tested extensively. This can be done by including various recorded histories of the domain, which may include histories with prioritized defaults (Example 2), and using the resulting programs to solve various reasoning tasks. ♦

The logician's model of the world thus consists of the system description $\mathcal{D}_H$ (Example 5), and recorded history $\mathcal{H}$ of initial state defaults (Example 2), actions, and observations. The logician achieves any given goal by first translating the model (of the world) to an ASP program $\Pi(\mathcal{D}_H, \mathcal{H})$, as described in Sections 5.1, 5.2, and expanding it to include the definition of goal and suitable axioms, as described at the end of Section 5.3. For planning and diagnostics, this program is passed to an ASP solver—we use SPARC, which expands CR-Prolog and provides explicit constructs to specify objects, relations, and their sorts (Balai et al., 2013). Please see `example4.sp` at `https://github.com/mhnsrdhrn/refine-arch` for the SPARC version of the complete program. The solver returns the answer set of this program. Atoms of the form:

$$occurs(action, step)$$

belonging to this answer set, e.g., $occurs(a_1, 1), \ldots, occurs(a_n, n)$, represent the shortest plan, i.e., the shortest sequence of abstract actions for achieving the logician's goal. Prior research results in the theory of action languages and ASP ensure that the plan is provably correct (Gelfond & Kahl, 2014). In a similar manner, suitable atoms in the answer set can be used for diagnostics, e.g., to explain unexpected observations by triggering suitable CR rules.

7. Refinement, Zoom and Randomization

For any given goal, each abstract action in the plan created by the logician by reasoning with the coarse-resolution domain representation is implemented as a sequence of concrete actions by the statistician. To do so, the robot probabilistically reasons about the part of the fine-resolution transition diagram relevant to the abstract action to be executed. This section defines refinement, randomization, and the zoom operation, which are necessary to build the fine-resolution models for such probabilistic reasoning, along with the corresponding steps of the design methodology. In doing so, we formally define the relationship, and establish the tight coupling, between the transition diagrams at the two resolutions.

7.1 Refinement

Although the representation of a domain used by a logician specifies fluents with observable values and assumes that all of its actions are executable, the robot may not be able to directly make some of these observations or directly execute some of these actions. For instance, a robot may not have the physical capability to directly observe if it is located in a given room, or to move in a single step from one room to another. We refer to such actions that cannot be executed directly and fluents that cannot be observed directly as *abstract*; actions that can be executed and fluents that can be observed directly are, on the other hand, referred to as *concrete*. The second step of the design methodology (see Section 4) requires the designer to refine the coarse-resolution transition diagram τ_H of the domain by including information needed to execute the abstract actions suggested by a logician, and to observe values of relevant abstract statics and fluents. This new transition diagram τ_L defined by system description $\mathcal{D}_L$, is called the *refinement* of τ_H. Its construction may be imagined as the designer taking a closer look at the domain through a magnifying lens. Looking at objects of a sort s of Σ_H at such finer resolution may lead to the discovery of parts of these objects and their attributes previously abstracted out by the designer. Instead of being a single entity, a room may be revealed to be a collection of cells with some of them located next to each other, a cup may be revealed to have parts such as handle and base, etc. If such a discovery happens, the sort s and its objects will be said to have been *magnified*, and the newly discovered parts are called the *components* of the corresponding objects. In a similar manner, a function $f : s_1, \ldots, s_n \rightarrow s_0$ from Σ_H is affected by the increased resolution or magnified if:

- It is an abstract action or fluent and hence can not be executed or observed directly by robots; and
- At least one of $s_0, \ldots, s_n$ is magnified.

In the signature Σ_L of the fine-resolution model τ_L, the newly discovered components of objects from a sort s of Σ_H form a new sort s^*, which is called the *fine-resolution counterpart* of s. For instance, in our illustrative example domain (Example 1 and Example 5), $place^*$, which is a collection

of grid cells $\{c_1, \ldots, c_n\}$, is the fine-resolution counterpart of $place$, which is a collection of rooms, and $object^*$ may be the collection of parts of cups. A vector $\bar{s}^*$ is a fine-resolution counterpart of $\bar{s}$ with respect to magnified sorts $s_{i_1}, \ldots, s_{i_k}$ (with $k > 0$) if it is obtained by replacing $s_{i_1}, \ldots, s_{i_k}$ by $s^*_{i_1}, \ldots, s^*_{i_k}$. Every element $\bar{x}$ of $\bar{s}^*$ is obtained from the unique[6] element $\bar{u}$ of $\bar{s}$ by replacing $u_{i_1}, \ldots, u_{i_k}$ from $s_{i_1}, \ldots, s_{i_k}$ by their components. We say that $\bar{u}$ is the *generator* of $\bar{x}$ in $\bar{s}^*$ and $\bar{x}$ is a fine-resolution counterpart of $\bar{u}$. A function f^* with signature $\bar{s}^*$ is called the fine-resolution counterpart of a magnified function f with respect to $\langle s_{i_1}, \ldots, s_{i_k} \rangle$ if for every $\langle u_1, \ldots, u_n, v \rangle \in \bar{s}$, $f(u_1, \ldots, u_n) = v$ iff there is a fine-resolution counterpart $\langle x_1, \ldots, x_n, y \rangle \in \bar{s}^*$ of $\langle u_1, \ldots, u_n, v \rangle$ such that $f^*(x_1, \ldots, x_n) = y$. For instance, fluents $loc^* : thing \rightarrow place^*$, $loc^* : object^* \rightarrow place^*$, and $loc^* : object^* \rightarrow place$ are fine-resolution counterparts of loc with respect to $\langle place \rangle$, $\langle object, place \rangle$ and $\langle object \rangle$ respectively; and action $move^* : robot \times place^* \rightarrow boolean$ is the fine-resolution counterpart of $move : robot \times place \rightarrow boolean$ with respect to $place$. In many interesting domains, some fine-resolution counterparts can be used to execute or observe magnified functions of Σ_H, e.g., an abstract action of moving to a neighbouring room can be executed by a series of moves to neighbouring cells. We describe other such examples later in this section.

We now define the refinement of a transition diagram in two steps. We first define a notion of *weak refinement* that does not consider the robot's ability to observe the values of domain fluents. We then introduce our *theory of observations*, and define a notion of *strong refinement* (or simply refinement) that includes the robot's ability to observe the values of domain fluents.

7.1.1 WEAK REFINEMENT

We introduce some terminology used in the definition below. Let signature Σ_1 be a subsignature of signature Σ_2 and let σ_1 and σ_2 be interpretations over these signatures. We say that σ_2 is an *extension* of σ_1 if $\sigma_2|_{\Sigma_1} = \sigma_1$[7].

Definition 9. [Weak refinement of τ_H]
A transition diagram τ_L over Σ_L is called a *weak refinement* of τ_H if:

1. For every state $\sigma^\diamond$ of τ_L, the collection $\sigma^\diamond|_{\Sigma_H}$ of atoms of $\sigma^\diamond$ formed by symbols from Σ_H is a state of τ_H.
2. For every state σ of τ_H, there is a state $\sigma^\diamond$ of τ_L such that $\sigma^\diamond$ is an extension of σ.
3. For every transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle$ of τ_H, if $\sigma_1^\diamond$ and $\sigma_2^\diamond$ are extensions of σ_1 and σ_2 respectively, then there is a path P in τ_L from $\sigma_1^\diamond$ to $\sigma_2^\diamond$ such that:
   - actions of P are concrete, i.e., directly executable by robots; and
   - P is *pertinent* to T, i.e., all states of P are extensions of σ_1 or σ_2.

▲

We are now ready to construct the fine-resolution system description $\mathcal{D}_{L,nobs}$ corresponding to the coarse-resolution system description $\mathcal{D}_H$ for our running example (Example 5). This construction does not consider the robot's ability to observe the values of domain fluents (hence the subscript "nobs"). We start with the case in which the only magnified sort in Σ_H is $place$. The

6. For simplicity we assume that no object can be a component of two different objects.
7. As usual $f|_B$ where f is a function with domain A and $B \subset A$ denotes the restriction of f on B.

signature Σ_L will thus contain three fine-resolution counterparts of functions from Σ_H: (i) basic fluent $loc^* : thing \rightarrow place^*$; (ii) action $move^* : robot \rightarrow place^*$; and (iii) defined static $next\_to^* : place^* \times place^* \rightarrow boolean$. We assume that loc^* and $next\_to^*$ are directly observable and $move^*$ is executable by the robot. These functions ensure indirect observability of loc and $next\_to$ and indirect executability of $move$. Although this construction is domain dependent, the approach is applicable to other domains.

2. Constructing the fine-resolution system description $\mathcal{D}_L$ corresponding to the coarse-resolution system description $\mathcal{D}_H$.
(a) Constructing $\mathcal{D}_{L,nobs}$.

To construct signature $\Sigma_{L,nobs}$:

1. Preserve all elements of signature Σ_H.

   In our running example, this includes sorts $thing$, $place$, $robot$, cup etc, object constants rob_1, $kitchen$, $office$, cup_1, tb_1 etc, static $next\_to(place, place)$, fluents $loc : thing \rightarrow place$ and $in\_hand : robot \times place \rightarrow boolean$, and actions $move(robot, place)$, $grasp(robot, object)$ and $putdown(robot, object)$.

2. Introduce a new sort s^* for every sort s of Σ_H that is magnified by the increase in resolution, with s^* comprising the components of elements of s. Add s^* to the sort hierarchy as a sibling of s. Also, for each abstract function f magnified by the increase in resolution, introduce appropriate fine-resolution counterparts that support the execution or observation of f at the fine-resolution.

   In our example, we introduce the sort $place^*$ as the fine-resolution counterpart of $place$, and object constants $c_1, \ldots, c_n$ of sort $place^*$ that are grid cells; no new sort is introduced for the sort $object$. Also, $\Sigma_{L,nobs}$ includes new static relation $next\_to^*(place^*, place^*)$, new fluent $loc^* : thing \rightarrow place^*$, and new action $move^*(robot, place^*)$, but the signature does not include any new symbols corresponding to in_hand or $grasp$.

3. Introduce static relations $component(O^*, O)$, which hold iff object O^* of sort s^* is a component of magnified object O of sort s of Σ_H. *These relations are domain dependent and need to be provided by the designer.*

   Continuing with our running example, we introduce the static relation:

   $$component : place^* \times place \rightarrow boolean$$

   where $component(c, r)$ is true iff cell c is part of room r.

Next, to construct the axioms of $\mathcal{D}_{L,nobs}$:

1. For each axiom of $\mathcal{D}_H$, if it contains any abstract functions, replace them by their fine-resolution counterparts and make these functions' variables range over appropriate sorts required by these fine-resolution counterparts.

In our running example, occurrences of the functions $next\_to(place, place)$, $loc : thing \rightarrow place$, and $move(robot, place)$ in the axioms of $\mathcal{D}_H$ are replaced by $next\_to^*(place^*, place^*)$, $loc : thing \rightarrow place^*$, and $move^*(robot, place^*)$ respectively. At the same time, the functions $in\_hand(robot, object)$, $grasp(robot, object)$, and $putdown(robot, object)$ remain unchanged. This results in $\mathcal{D}_{L,nobs}$ having causal laws:

$$move^*(R, C) \ \textbf{causes} \ loc^*(R) = C \tag{44a}$$
$$grasp(R, O) \ \textbf{causes} \ in\_hand(R, O) \tag{44b}$$
$$putdown(R, O) \ \textbf{causes} \ \neg in\_hand(R, O) \tag{44c}$$

state constraints:

$$loc^*(O) = C \ \textbf{if} \ loc^*(R) = C, \ in\_hand(R, O) \tag{45a}$$
$$next\_to^*(C_2, C_1) \ \textbf{if} \ next\_to^*(C_1, C_2) \tag{45b}$$

and executability conditions such as:

$$\textbf{impossible} \ move^*(R, C) \ \textbf{if} \ loc^*(R) = C \tag{46a}$$
$$\textbf{impossible} \ move^*(R, C_2) \ \textbf{if} \ loc^*(R) = C_1, \ \neg next\_to^*(C_1, C_2) \tag{46b}$$
$$\textbf{impossible} \ grasp(R, O) \ \textbf{if} \ loc^*(R) = C_1, \ loc^*(O) = C_2, \ C_1 \neq C_2 \tag{46c}$$
$$\textbf{impossible} \ putdown(R, O) \ \textbf{if} \ \neg in\_hand(R, O) \tag{46d}$$

where C, C_1, and C_2 are grid cells.

2. Introduce *bridge axioms*, i.e., axioms relating the coarse-resolution functions and their fine-resolution counterparts. These axioms have the form:

$$f(X_1, \ldots, X_m) = Y \ \textbf{if} \ component(C_1, X_1), \ \ldots \ , component(C_m, X_m), \ component(C, Y), \ f^*(C_1, \ldots, C_m) = C \tag{47}$$

In our running example, we have:

$$loc(Th) = P \ \textbf{if} \ component(C, P), \ loc^*(Th) = C \tag{48a}$$
$$next\_to(P_1, P_2) \ \textbf{if} \ component(C_1, P_1), \ component(C_2, P_2), \ next\_to^*(C_1, C_2) \tag{48b}$$

These axioms are domain dependent and need to be provided by the designer.

This completes the construction of $\mathcal{D}_{L,nobs}$ for our running example.

To illustrate the robot's reasoning with $\mathcal{D}_{L,nobs}$, consider a fine-resolution state δ_1 in which the robot is in cell c_2 of the $office$, i.e., $(loc^*(rob_1) = c_2) \in \delta_1$. If δ_1 is a fine-resolution counterpart of a coarse-resolution state σ_1, then $(loc(rob_1) = office) \in \sigma_1$ because the bridge axiom in Statement 48(a) infers $loc(rob_1) = office$ from $loc^*(rob_1) = c_2$. Next, consider the robot's move from δ_1, with book tb_1 in its hand, to a cell c_5 in the $kitchen$. If δ_2 is the resultant fine-resolution state, $(loc^*(rob_1) = c_5) \in \delta_2$ based on Statement 44(a), and $(loc^*(tb_1) = c_5) \in \delta_2$ based on Statement 45(a). Now, if δ_2 is a fine-resolution counterpart of a coarse-resolution state σ_2, then based on the bridge axiom in Statement 48(a), $(loc(rob_1) = kitchen) \in \sigma_2$ and $(loc(tb_1) = kitchen) \in \sigma_2$.

The following proposition says that $\mathcal{D}_{L,nobs}$ as constructed above is a weak refinement of $\mathcal{D}_H$.

Proposition 3. [Weak Refinement]
Let $\mathcal{D}_H$ and $\mathcal{D}_{L,nobs}$ be the coarse-resolution and fine-resolution system descriptions from our running example. Then $\tau_{L,nobs}$ is a weak refinement of τ_H. ★

The proof of this proposition is in Appendix C. Although the statement of the proposition and its proof are provided here for our example domain, this approach can also be used to construct $\mathcal{D}_{L,nobs}$ and establish weak refinement in many other robotics domains.

7.1.2 THEORY OF OBSERVATIONS

The definition of weak refinement does not take into account the robot's ability to observe the values of fluents in the domain. This ability to observe plays an important role in updating beliefs and monitoring the results of action execution in the fine-resolution. As stated earlier, observability of fluents and executability of actions are specified in the coarse resolution description. The actual observations are obtained when the robot interacts with the domain through knowledge-producing actions. This interaction only happens in the fine resolution in our architecture and it is thus modeled during refinement. Recall that abstract fluents and statics are indirectly observable and the concrete fluents and statics are directly observable. In our running example, the observation of a thing being in a room can be inferred by checking if the thing can be observed in some cell of this room. Also, the robot has to monitor its movement between a series of neighboring cells when it attempts to execute the abstract action of moving to a neighbouring room. In this section, we introduce a *Theory of Observations* that supports this ability. This theory is used in conjunction with any given system description $\mathcal{D}_{L,nobs}$ as follows.

1. Expand $\Sigma_{L,nobs}$:

   - For every directly observable function f, include the actions:

     $$test_f : robot \times dom(f) \times range(f) \rightarrow boolean$$

     For $y \in range(f)$, this action checks if the value of f is y in a given state. For readability, we will sometimes abuse notation and write this action as $test(R, F, Y)$.

     In our example domain, we include an action such as $test_{loc^*}(rob_1, O, C)$ for the robot to check if the location of an object O is a particular cell C.

   - For every (directly or indirectly) observable function f, include the basic knowledge fluent:

     $$observed_f : robot \times dom(f) \times range(f) \rightarrow \{true, false, undet\}$$

     where the outcome *undet* stands for "undetermined". For every $x \in dom(f)$ and $y \in range(f)$, the value of $observed_f(rob_1, x, y)$ is the result of the most recent execution of $test_f(rob_1, x, y)$. Initially, the value is set to *undet*. After $test_f(rob_1, x, y)$ is executed at least once, the value becomes (and remains) boolean. It is *true* if the most recent test returned *true* and *false* otherwise.

     In our example domain, we have basic knowledge fluents such as:

     $$observed_{loc^*}(rob_1, O, C)$$
     $$observed_{loc}(rob_1, O, P)$$

- For every indirectly observable function f, introduce an observation-related, domain-dependent defined fluent, as described in Statement 1:

$$observable_f : robot \times dom(f) \times range(f) \rightarrow boolean$$

Also, for every directly observable domain function f, introduce observation-related, domain-dependent defined fluents:

$$can\_be\_observed_f : robot \times dom(f) \times range(f) \rightarrow boolean \tag{49}$$

These fluents will be used to describe conditions for the observability of the corresponding functions. *These domain dependent fluents need to be defined by the designer.*

2. Expand axioms of $\mathcal{D}_{L,nobs}$ by axioms that model the robot's ability to observe.

- Introduce causal laws to describe the effects of $test_{f^*}(R, \bar{X}, Y)$, i.e., the knowledge-producing action, on the fine-resolution basic fluent f^*:

$$test_{f^*}(R, \bar{X}, Y) \ \textbf{causes} \ observed_{f^*}(R, \bar{X}, Y) = true \ \textbf{if} \ f^*(\bar{X}) = Y \tag{50}$$
$$test_{f^*}(R, \bar{X}, Y) \ \textbf{causes} \ observed_{f^*}(R, \bar{X}, Y) = false \ \textbf{if} \ f^*(\bar{X}) = Y_1, \ Y_1 \neq Y$$

Also introduce the executability condition:

$$\textbf{impossible} \ test_{f^*}(R, \bar{X}, Y) \ \textbf{if} \ \neg can\_be\_observed_{f^*}(R, \bar{X}, Y) \tag{51}$$

where $\bar{X}$ represents the domain of f^*.

In our running example, if robot rob_1 located in cell c checks the presence or absence of an object o, $observed_{loc^*}(rob_1, o, c)$ will be *true* iff o is in c during testing; it will be *false* iff o is not in c. These values will be preserved by inertia axioms until the state is observed to have changed when the same cell is tested again. If the robot has not yet tested a cell c for an object o, the value of $observed_{loc^*}(rob_1, o, c)$ remains *undet*.

- Introduce axioms for domain-dependent defined fluents describing the ability of the robot to sense the values of directly and indirectly observable functions.

In our running example, an object's room location is observable by a robot only when the robot and the object are in the same room:

$$observable_{loc}(rob_1, O, Pl) \ \textbf{if} \ loc(rob_1) = Pl \tag{52}$$

Also, the robot in a particular cell can test the presence (or absence) of an object in that cell, and it can always test whether it has an object in its grasp. We encode this knowledge as:

$$can\_be\_observed_{loc^*}(rob_1, Th, C) \ \textbf{if} \ loc^*(rob_1) = C \tag{53a}$$
$$can\_be\_observed_{in\_hand}(rob_1, Th, true) \tag{53b}$$

We use different fluents ($observable$ and $can\_be\_observed$) to serve a similar purpose because the conditions under which a particular value of a particular function can be observed may be significantly different at the coarse-resolution and fine-resolution.

- Introduce axioms for indirect observation of functions. First, we introduce a defined fluent for each indirectly observable function f:

$$may\_be\_true_f : robot \times dom(f) \times range(f) \rightarrow boolean$$

which holds true if the value of $f(x)$, where $x \in dom(f)$ may be discovered to be $y \in range(f)$.
The axioms for indirect observation are then given by:

$$observed_f(R, \bar{X}, Y) = true \ \textbf{if} \ observed_{f^*}(R, \bar{X}^*, C) = true, \ component(X_1^*, X_1), \ldots, \ component(X_m^*, X_m), \ component(C, Y) \tag{54a}$$

$$may\_be\_true_f(R, \bar{X}, Y) \ \textbf{if} \ observed_f(R, \bar{X}, Y) = true \tag{54b}$$

$$may\_be\_true_f(R, \bar{X}, Y) \ \textbf{if} \ observed_f(R, \bar{X}, Y) = undet, \ component(C, Y), \ observed_{f^*}(R, \bar{X}^*, C) = undet \tag{54c}$$

$$observed_f(R, \bar{X}, Y) = false \ \textbf{if} \ \neg may\_be\_true_f(R, \bar{X}, Y) \tag{54d}$$

$$observed_f(R, \bar{X}, Y_1) = false \ \textbf{if} \ observed_f(R, \bar{X}, Y_2), \ Y_1 \neq Y_2 \tag{54e}$$

which implies that a coarse-resolution function is observed to have a particular value if any of its fine-resolution counterparts is observed to be true, and that the coarse-resolution function may be observed to have a particular value as long as it is possible that at least one of its fine-resolution counterparts may be observed to be true.

In our example domain, observing an object in a cell in a room implies that the object is indirectly observed to be in the room:

$$observed_{loc}(R, O, P) = true \ \textbf{if} \ observed_{loc^*}(R, O, C) = true, component(C, P)$$

Example 6 includes other examples of the use of such axioms for indirect observations.

7.1.3 STRONG REFINEMENT

We are now ready to define a notion of strong refinement that takes into account the theory of observations. We do so by expanding Definition 9 of weak refinement as follows.

Definition 10. [Strong refinement of τ_H]
A transition diagram τ_L over Σ_L is called a *strong refinement* of τ_H if:

1. For every state $\sigma^\diamond$ of τ_L, the collection $\sigma^\diamond|_{\Sigma_H}$ of atoms of $\sigma^\diamond$ formed by symbols from Σ_H is a state of τ_H.

2. For every state σ of τ_H, there is a state $\sigma^\diamond$ of τ_L such that $\sigma^\diamond$ is an extension of σ.

3. For every transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle$ of τ_H, if $\sigma_1^\diamond$ is an extension of σ_1, then for every observable fluent f such that $observable_f(rob_1, \bar{x}, y) \in \sigma_2$, there is a path P in τ_L from $\sigma_1^\diamond$ to an extension $\sigma_2^\diamond$ of σ_2 such that:

   - P is *pertinent* to T, i.e., all states of P are extensions of σ_1 or σ_2;

- actions of P are concrete, i.e., directly executable by robots; and
- $observed_f(rob_1, x, y) = true \in \sigma_2^\diamond$ iff $(f(x) = y) \in \sigma_2^\diamond$, and $observed_f(rob_1, x, y) = false \in \sigma_2^\diamond$ iff $(f(x) = y_1) \in \sigma_2^\diamond$ and $y_1 \neq y$.

▲

We are now ready to complete the second step of the design methodology, i.e., constructing the fine-resolution system description that considers the robot's ability to observe the values of domain fluents. We do so by expanding $\mathcal{D}_{L,nobs}$ to include the theory of observations.

2. Constructing the fine-resolution system description $\mathcal{D}_L$ that is the refinement of the coarse-resolution system description $\mathcal{D}_H$. (b) Constructing $\mathcal{D}_L$ with theory of observations.

Specifically, the system description $\mathcal{D}_L$ is obtained by:

1. Augmenting the signature $\Sigma_{L,nobs}$ of system description $\mathcal{D}_{L,nobs}$ with the actions (e.g., $test$) and fluents (e.g., $can\_be\_observed$) of the theory of observations.
2. Augmenting the axioms of $\mathcal{D}_{L,nobs}$ with the axioms needed to represent the robot's ability to observe, i.e., Statements 50, 51, and 54, and axioms for domain-dependent, observation-related defined fluents.

Next, consider the relationship between $\mathcal{D}_{L,nobs}$, a well-founded system description, and $\mathcal{D}_L$, its extension by the theory of observations, which is used in the proof of the following proposition. If $\tau_{L,nobs}$ and τ_L are the transition diagrams defined by these system descriptions, then:

- The states of $\tau_{L,nobs}$ and τ_L differ mainly in the knowledge functions $observed_f$ for directly or indirectly observable fluents.
- For every transition $\langle \sigma_1^\diamond, test_f(rob_1, y), \sigma_2^\diamond \rangle$ of τ_L:
  - Physical fluents are the same in $\sigma_1^\diamond$ and $\sigma_2^\diamond$.
  - $observed_f(rob_1, \bar{x}, y) = true \in \sigma_2^\diamond$ iff $(f(\bar{x}) = y) \in \sigma_2^\diamond$.
  - $observed_f(rob_1, \bar{x}, y) = false \in \sigma_2^\diamond$ iff $(f(\bar{x}) = y_1) \in \sigma_2^\diamond$ and $y_1 \neq y$.

Finally, the following proposition says that $\mathcal{D}_L$ as constructed above is a strong refinement of $\mathcal{D}_H$.

Proposition 4. [Strong Refinement]
Let $\mathcal{D}_H$ and $\mathcal{D}_L$ be the coarse-resolution and fine-resolution system descriptions from our running example. Then τ_L is a strong refinement of τ_H. ★

The proof of this proposition is in Appendix D. Please see `refined.sp` at `https://github.com/mhnsrdhrn/refine-arch` for the ASP program (in SPARC format) describing the refined signature and refined axioms for our illustrative example, along with additional axioms that support planning to achieve particular goals.

Example 6. [Expanded example of refinement]
Let $\mathcal{D}_H$ be as in Example 5, and let its refinement $\mathcal{D}_L$ be as described above in Sections 7.1.1– 7.1.3. The key difference is that, in addition to the cells of rooms, the increase in resolution has also led to the discovery of component of cups such as $handle$ and $base$. To construct a refinement $\mathcal{D}_L^e$ of $\mathcal{D}_H$ suitable for this expanded domain, we expand the signature of $\mathcal{D}_L$ by a new sort cup^* and add it to the sort hierarchy of $\mathcal{D}_H$ as a sibling of sort cup. Now the sort $object$ has three children: cup, cup^* and $textbook$. Similar to $\mathcal{D}_L$, we will need the sort $place^*$ and object constants of specific sorts such as:

$$textbook = \{tb_1, tb_2\}$$
$$cup = \{cup_1\}$$
$$cup^* = \{cup\_base_1, cup\_handle_1\}$$

Similar to $\mathcal{D}_L$, we will need the function loc^*, and we need new instances of the component relation:

$$component(cup\_base_1, cup_1)$$
$$component(cup\_handle_1, cup_1)$$
$$\ldots$$

To construct $\mathcal{D}_L^e$, we consider actions that can no longer be executed directly, and then consider fluents that can no longer be observed directly.

In our example, actions $grasp$ and $putdown$ are no longer directly executable on cups, but are executable on the components of cups. To support indirect execution of these actions on cups, we introduce new executable actions $grasp^*(robot, cup^*)$ and $putdown^*(robot, cup^*)$ for grasping and putting down a cup's handle and base. System description $\mathcal{D}_L^e$ will inherit from $\mathcal{D}_L$ the axioms for $next\_to^*$, $move^*$ and loc^*, i.e., Statements 44(a), 45(a-b), 46(a-b), and 48(a-b). Ground instances of the axiom describing the effects of $grasp$ for objects other than cups and their parts will remain as in $\mathcal{D}_H$; this can be written as:

$$grasp(R, O) \ \textbf{causes} \ in\_hand(R, O) \ \textbf{if} \ O \notin cup, O \notin cup^* \tag{55}$$

A new axiom is needed to describe the effects of grasping parts of cups:

$$grasp^*(R, O) \ \textbf{causes} \ in\_hand(R, O) \ \textbf{if} \ O \in cup^* \tag{56}$$

Executability conditions for $grasp$ and $grasp^*$ are handled in a similar manner. In addition to Statement 46(c) of $\mathcal{D}_L$:

$$\textbf{impossible} \ grasp(R, O) \ \textbf{if} \ loc^*(R) = C_1, \ loc^*(O) = C_2, \ C_1 \neq C_2$$

we will need an additional axiom for $grasp^*$:

$$\textbf{impossible} \ grasp^*(R, O) \ \textbf{if} \ loc^*(R) = C_1, \ loc^*(O) = C_2, \ C_1 \neq C_2 \tag{57}$$

Similar axioms are also introduced for actions $putdown$ and $putdown^*$. Finally, we will need axioms describing newly discovered relationships between objects and their parts:

$$in\_hand(R, O) = in\_hand(R, OPart) \textbf{ if } component(OPart, O) \tag{58a}$$
$$(loc^*(O) = C) = (loc^*(OPart) = C) \textbf{ if } component(OPart, O) \tag{58b}$$

where the equality is shorthand for two statements[8]. To illustrate reasoning with D_L^e consider initial situation in which rob_1 and cup_1 are in a cell c_5 of $kitchen$. Suppose rob_1 grasps the cup's handle, i.e., $grasp^*(rob_1, cup\_handle_1)$ is executed, and moves to location c_2 of $office$, i.e., executes $move^*(rob_1, c_2)$. Both actions are clearly executable based on Statement 57 and Figure 3(b). By Statement 56, after the execution of $grasp^*$, the handle will be in the robot's hand, i.e., $in\_hand(rob_1, cup\_handle_1)$. Based on Statement 44(a), executing action $move^*$ will result in $loc^*(rob_1) = c_2$. Based on Statement 45(a), we conclude that $loc^*(cup\_handle_1) = c_2$. Then, based on Statement 58(b), we have $loc^*(cup_1) = c_2$ and thus, by Statement 48(a), we have $loc(cup_1) = office$.

Next, we examine the effect of the robot being able to directly observe neither the location of a cup nor if it is holding a cup, but being able to do so for a cup's parts and for textbooks. This requires us to introduce fine-resolution counterparts $loc^* : cup^* \rightarrow place^*$ and $in\_hand^*(robot, cup^*)$ of loc and in_hand respectively for cups, and change the related axioms. Statement 55 is not related to cups and remains unchanged. Statement 56 will, on the other hand, be replaced by the axiom:

$$grasp^*(R, O) \textbf{ causes } in\_hand^*(R, O) \textbf{ if } O \in cup^* \tag{59}$$

The bridge axioms in Statement 58(a-b) will be replaced by bridge axioms:

$$in\_hand(R, Cup) \textbf{ if } in\_hand^*(R, Part),\ component(Part, Cup) \tag{60a}$$
$$loc^*(O) = C \textbf{ if } loc^*(OPart) = C,\ component(OPart, O) \tag{60b}$$

defining in_hand and loc for cups in terms of its fine-resolution counterparts. Next, we introduce actions $test_{loc^*}$ and $test_{in\_hand^*}$ to check the cell location of a cup's part and to check whether a part of a cup is in the robot's hand:

$$test_{loc^*}(R, OPart, C) \textbf{ causes } observed_{loc^*}(R, OPart, C) = true \textbf{ if } loc^*(OPart) = C$$
$$test_{loc^*}(R, OPart, C) \textbf{ causes } observed_{loc^*}(R, OPart, C) = false \textbf{ if } loc^*(OPart) = C_1, C_1 \neq C$$
$$test_{in\_hand^*}(R, OPart, V) \textbf{ causes } observed_{in\_hand^*}(R, OPart, V) = true \textbf{ if } in\_hand^*(R, OPart) = V$$
$$test_{in\_hand^*}(R, OPart, V) \textbf{ causes } observed_{in\_hand^*}(R, OPart, V) = false \textbf{ if } in\_hand^*(R, OPart) = V_1,\ V_1 \neq V$$

We also replace Statements 53(a-b) about the observation-related, domain-dependent defined fluents with the following statements:

$$can\_be\_observed_{loc^*}(R, Th, C) \textbf{ if } loc^*(R) = C,\ Th \notin cup \tag{61a}$$
$$can\_be\_observed_{in\_hand}(R, O, V) \textbf{ if } O \notin cup, \notin cup^* \tag{61b}$$
$$can\_be\_observed_{in\_hand^*}(R, O, V) \textbf{ if } O \in cup^* \tag{61c}$$

8. $f(x) = g(x)$ **if** *body* is shorthand for $f(x) = y$ **if** *body*, $g(x) = Y$ and $g(x) = y$ **if** *body*, $f(x) = Y$.

which imply that the robot can no longer directly observe the location of a cup or whether a cup is in its hand; it can do so for parts of cups. Reasoning similar to that used in the context of $grasp^*$ above can be used to show that if the robot grasps a cup's handle and moves to a cell c_2 of the $office$, the robot, the cup, and the cup's handle will be in the $office$. If needed, test actions and the theory of observations can be used to observe that the robot is holding the cup, and to observe the locations of other things in the domain.

Next, consider the inclusion of an additional action $fill(robot, cup)$ in $\mathcal{D}_H$. Executing this action causes a cup to be filled—we thus introduce a basic fluent $filled : cup \rightarrow boolean$ in $\mathcal{D}_H$ and the corresponding axioms described below:

$$fill(R, C) \ \textbf{causes} \ filled(C) \tag{62a}$$
$$\textbf{impossible} \ fill(R, C) \ \textbf{if} \ filled(C) \tag{62b}$$
$$\textbf{impossible} \ fill(R, C) \ \textbf{if} \ loc(C) = P_1, \ loc(R) = P_2, \ P_1 \neq P_2 \tag{62c}$$

Here, action $fill$ is directly executable and fluent $filled$ in directly observable. We also include $same\_loc : object \times object \rightarrow boolean$ in $\mathcal{D}_H$, a defined fluent to reason about co-occurrence of objects, with the corresponding axiom:

$$same\_loc(O_1, O_2) \ \textbf{if} \ loc(O_1) = Pl, \ loc(O_2) = Pl \tag{63}$$

which defines when two objects are considered to be in the same place. In the refined system description, the action $fill$ is still directly executable, and the fluent $filled$ is directly observable, for cups. These functions are not defined for parts of cups, e.g., we cannot fill a cup's handle, and we thus do not create their fine-resolution counterparts (e.g., $fill^*$). We do, however, introduce a new function in the signature of the refined system description:

$$same\_loc^*(O_1, O_2), \ O_1, O_2 \notin cup \tag{64}$$

representing parts of cups and/or other objects being in the same grid cell location. Note that elements of sort cup are not included because we cannot directly observe the location of a cup in the fine resolution. We also introduce the following axiom in the refined system description, which corresponds to Statement 63 introduced above:

$$same\_loc^*(O_1, O_2) \ \textbf{if} \ loc^*(O_1) = C, \ loc^*(O_2) = C \tag{65}$$

Finally, we need to introduce a suitable bridge axiom:

$$\begin{aligned} same\_loc(O_1, O_2) \ \textbf{if} \ & loc^*(OPart_1) = C_1, \ loc^*(OPart_2) = C_2, \\ & component(C_1, P), \ component(C_2, P), \\ & component(OPart_1, O_1), \ component(OPart_2, O_2) \end{aligned} \tag{66}$$

Once we have the refined system description, we can reason with it as before. For instance, consider a fine-resolution state in which $loc^*(cup\_handle_1) = c_5$ and $loc^*(tb_1) = c_6$ where c_5 and c_6 are grid cells in the $kitchen$. Based on the bridge axioms, we can infer that $loc(cup_1) = kitchen$, $loc(tb_1) = kitchen$ and $same\_loc(cup_1, tb_1)$. ♦

7.2 Randomization

The system description $\mathcal{D}_L$ of transition diagram τ_L, obtained by refining transition diagram τ_H, is insufficient to implement a coarse-resolution transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle \in \tau_H$. We need to capture the non-determinism in action execution, which is done by the third step of the design methodology (Section 4).

3. Provide domain-specific information and randomize the fine-resolution description of the domain to capture the non-determinism in action execution.

This step of the design methodology models the non-determinism by first creating $\mathcal{D}_{LR}$, the randomized fine-resolution system description. It does so by:

- Replacing the deterministic causal laws of each action in $\mathcal{D}_L$ that has non-determinism in its execution, by non-deterministic ones; and
- Modifying the signature by declaring each affected fluent as a *random fluent*, i.e., define the set of values the fluent can choose from when the action is executed. A defined fluent may be introduced to describe this set of values in terms of other variables.

Note that only causal laws of actions with non-determinism in their execution need to be replaced with non-deterministic ones. For instance, consider a robot moving to a specific cell in the $office$. During this move, the robot can reach the desired cell or one of the neighboring cells. The causal law for the $move$ action in $\mathcal{D}_L$ can therefore be (re)stated as:

$$move^*(R, C_2) \ \textbf{causes} \ loc^*(R) \ = \ \{C : range(loc^*(R), C)\} \tag{67}$$

where the relation $range$ is a defined fluent used to represent the cell the robot currently is in, and the cells next to its current location:

$$range(loc^*(R), C) \ \textbf{if} \ loc^*(R) = C$$
$$range(loc^*(R), C) \ \textbf{if} \ loc^*(R) = C_1, \ next\_to^*(C, C_1)$$

As described by Statement 46(b), the robot can only move to a cell that is $next\_to$ its current location. In general, the fluent affected by the change in the causal law can take one of a set of values that satisfy a given property ($range$ in the current example), as described in Statement 3. In a similar manner, the non-deterministic version of the $test$ action that determines the robot's cell location in the $office$ is:

$$test_{loc^*}(rob_1, rob_1, c_i) \ \textbf{causes} \ observed_{loc^*}(rob_1, rob_1, c_i) = \{true, false\} \ \textbf{if} \ loc^*(rob_1) = c_i$$

which indicates that the result of the $test$ action may not always be as expected, and c_i are cells in the $office$. Similar to refinement, *it is the designer's responsibility to provide domain-specific information needed for randomization*. Furthermore, note that the paths in the randomized transition diagram τ_{LR} match those in τ_L except for the addition of the defined fluents that model the domain-specific information.

Once the randomized system description $\mathcal{D}_{LR}$ has been constructed, we can construct its probabilistic version $\mathcal{D}^P_{LR}$ that consists of:

- System description $\mathcal{D}_{LR}$ that defines a non-deterministic transition diagram τ_{LR} of all the system's trajectories.
- A function P assigning probabilities to each of the transitions.

In addition to the actual states, $\mathcal{D}_{LR}^{P}$ will reason with probability distributions over the states, and consider transitions between one such probabilistic state to another as a result of executing particular actions. We obtain the probabilities needed to construct $\mathcal{D}_{LR}^{P}$, we experimentally collect statistics of action outcomes and the reliability of observations as described below.

Collecting statistics: Running experiments to collect statistics that are used to compute the probabilities of action outcomes and the reliability of observations, corresponds to the fourth step of the design methodology (see Section 4).

4. Run experiments, collect statistics, and compute probabilities of action outcomes and reliability of observations.

Specifically, we need to compute the:

- Causal probabilities for the outcomes of physical actions; and
- Probabilities for the outcomes of the knowledge-producing actions, i.e., a quantitative model for the observations being correct.

This collection of statistics is *typically a one-time process performed in an initial training phase*, although it is also possible to do this incrementally over time. Also, the statistics are computed separately for each basic fluent in $\mathcal{D}_{LR}$. To collect the statistics, we consider the direct effects of one action at a time. In our domain, this corresponds to considering one non-deterministic causal law in $\mathcal{D}_{LR}$ at a time. We sample ground instances of this causal law, e.g., corresponding to different atoms in the causal law. The robot then executes the action corresponding to this sampled instance multiple times, and collects statistics (e.g., counts) of the number of times each possible outcome (i.e., value) is obtained. The robot also collects information about the amount of time taken to execute each such action.

As an example, consider a ground instance of the non-deterministic causal law for $move^*$, considering grid cell locations in a particular room:

$$move^*(rob_1, c_2) \ \textbf{causes} \ loc^*(R) \ = \ \{c_1, c_2, c_3\}$$

where rob_1 in cell c_1 can end up in one of three possible cells when it tries to move to c_2. In ten attempts to move to c_2, assume that rob_1 remains in c_1 in one trial, reaches c_2 in eight trials, and reaches c_3 in one trial. The maximum likelihood estimates of the probabilities of these outcomes are then 0.1, 0.8 and 0.1 respectively—the probability of rob_1 moving to other cells is zero. Similar statistics are collected for other ground instances of this causal law, and averaged to compute the statistics for the fluent loc for rob_1. The same approach is used to collect statistics for other causal laws and fluents, including those related to knowledge actions and basic knowledge fluents. For instance, assume that the collected statistics indicate that testing for the presence of a textbook in a cell requires twice as much computational time (and thus effort) as testing for the presence of a

cup. This information, and the relative accuracy of recognizing textbooks and cups, will be used to determine the relative value of executing the corresponding test actions. The collected statistics are thus used to define the probabilities of two different types of transitions in $\mathcal{D}_{LR}$.

Definition 11. [Learned transition probabilities]
The learned state transition probabilities $P(\delta_x, a, \delta_y)$ are of two types depending on the type of transition between states δ_x and δ_y of $\mathcal{D}_{LR}$:

1. *Physical state transition probability*, where δ_x and δ_y differ in a literal formed of a non-knowledge fluent term, e.g., when we consider the probability of the robot's location changing from $loc^*(rob_1) = c_1$ to $loc^*(rob_1) = c_2$ after executing $move^*(rob_1, c_2)$.

2. *Knowledge state transition probabilities*, where δ_x and δ_y differ in a literal formed of a knowledge fluent term, e.g., when we consider the probability of $observed_{loc^*}(rob_1, cup_1, c_2)$ changing its value from $undet$ in δ_x to $true$ in δ_y.

▲

There are some important caveats about collecting statistics and computing probabilities.

- First, we only consider the direct effects of an action while collecting statistics. Also, since transitions in our domain only differ in a literal formed of one knowledge or non-knowledge fluent term, we explore the change in the value of one basic fluent at a time, e.g., while collecting statistics of a robot's move from one cell to another, we do not consider whether the robot is holding a book in hand. It is a common practice in robotics to compute probabilities of the direct effects of an action, and to consider one causal law at a time. It simplifies the collection of statistics and the computed probabilities can be reused across robots and scenarios. If any action had multiple direct effects, we would consider them together while collecting statistics. However, states can comprise multiple fluents, actions can have multiple direct and indirect effects, and the value of one fluent may constrain the value of other fluents. As discussed later, we ensure that impossible scenarios are not considered, and compute the probabilities of valid states by suitably combining the individual probabilities. For instance, if a robot is holding a book, having the robot and the book in different locations does not constitute a valid state.

- Second, the collection of statistics depends on the availability of relevant ground truth information, e.g., we need the actual location of robot rob_1 after executing $move^*(rob_1, c_2)$. This ground truth information is often provided by an external high-fidelity sensor during the initial training phase, or by a human observer.

- Third, although we do not do so in our experiments, it is possible to use heuristics to model the computational effort, and to update the statistics incrementally over time, e.g., the execution time of a knowledge-producing action can be computed as a function of the size of the input image. If any heuristic functions are to be used, the designer has to make them available to automate subsequent steps of our control loop.

- Fourth, considering all ground instances of one causal law at a time can require a lot of training in complex domains, but this is often unnecessary. For instance, it is often the case

that the statistics of moving from a cell to one of its neighbors is the same for cells in a room and any given robot. In a similar manner, if the robot and an object are (are not) in the same cell, the probability of the robot observing (not observing) the object is often the same for any cell. The designer thus only considers representative samples of the distinct cases to collect statistics, e.g., statistics corresponding to moving between cells will be collected in two different rooms only if these statistics are expected to be different.

There is an extensive literature on estimation of such statistical models for robots. Types of models learned from data in cognitive robotics are sensor models and robot motion models (Thrun, Burgard, & Fox, 2005), motion models for manipulated objects (Kopicki, Zurek, Stolkin, Moerwald, & Wyatt, 2017), and success-failure models for actions (e.g., grasping) (Lu, Chenna, Sundaralingam, & Hermans, 2017). In particular, robot motion models for standard mobile platforms are available for use in the robotics community without need for re-learning. In addition, rigid body physics engines can be used as a source of data for learning (Haidu, Kohlsdorf, & Beetz, 2015). The learned models can be used on physical robots if the simulator has a realistic representation of the robot and the application domain—such a transfer of knowledge learned in simulation to physical robots is an open research problem. In other work, we have explored incremental learning of statistics (Zhang et al., 2015) and domain knowledge (Sridharan & Meadows, 2018), but these topics are beyond the scope of this article.

Even after the desired probabilities of transitions in $\mathcal{D}_{LR}$ are computed, reasoning with $\mathcal{D}_{LR}^{P}$ (as described earlier) will be computationally infeasible for complex domains. Our architecture addresses this problem by automatically zooming to the part of $\mathcal{D}_{LR}$ relevant to each coarse resolution transition T under consideration (as described below), and then reasoning probabilistically over this zoomed system description $\mathcal{D}_{LR}(T)$ using POMDPs (see Section 8.2).

7.3 Zoom

Reasoning probabilistically about the entire randomized fine-resolution system description can become computationally intractable. For any given transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle \in \tau_H$, this intractability could be offset by limiting fine-resolution probabilistic reasoning to the part of transition diagram τ_{LR} whose states are pertinent to T. For instance, for the state transition corresponding to a robot moving from the $office$ to the $kitchen$ in Example 5, i.e., $a^H = move(rob_1, kitchen)$, we could only consider states of τ_{LR} in which the robot's location is a cell in the $office$ or the $kitchen$. However, these states would still contain fluents and actions not relevant to the execution of a^H, e.g., locations of domain objects, and the $grasp$ action. What we need is a fine-resolution transition diagram $\tau_{LR}(T)$ whose states contain no information unrelated to the execution of a^H, while its actions are limited to those which may be useful for such an execution. In the case of $a^H = move(rob_1, kitchen)$, for instance, states of $\tau_{LR}(T)$ should not contain any information about domain objects. In the proposed architecture, the controller constructs such a zoomed fine-resolution system description $\mathcal{D}_{LR}(T)$ in two steps. First, a new action description is constructed by focusing on the transition T, creating a system description $\mathcal{D}_H(T)$ that consists of ground instances of $\mathcal{D}_H$ built from object constants of Σ_H relevant to T. In the second step, the refinement of $\mathcal{D}_H(T)$ is extracted from $\mathcal{D}_{LR}$ to obtain $\mathcal{D}_{LR}(T)$. We first consider the requirements of the zoom operation.

Definition 12. [Requirements of zoom operation]
The following are the requirements the zoom operation should satisfy:

1. Every path in the transition diagram obtained after zooming should correspond to a path in the transition diagram before zooming. In other words, for every path P^z of $\tau_{LR}(T)$ between states $\delta_1^z \subseteq \delta_1$ and $\delta_2^z \subseteq \delta_2$, where δ_1 and δ_2 are refinements of σ_1 and σ_2 respectively, there is a path P between states δ_1 and δ_2 in τ_{LR}.

2. Every path in the transition diagram before zooming should correspond to a path in the zoomed transition diagram. In other words, for every path P of τ_{LR}, formed by actions of $\tau_{LR}(T)$, between states δ_1 and δ_2 that are refinements of σ_1 and σ_2 respectively, there is a path P^z of $\tau_{LR}(T)$ between states $\delta_1^z \subseteq \delta_1$ and $\delta_2^z \subseteq \delta_2$.

3. Paths in transition diagram $\tau_{LR}(T)$ should be of sufficiently high probability for the probabilistic solver to find them.

▲

To construct such a zoomed system description $\mathcal{D}_{LR}(T)$ defining transition diagram $\tau_{LR}(T)$, we begin by defining $relObCon_H(T)$, the collection of object constants of signature Σ_H of $\mathcal{D}_H$ *relevant* to transition T.

Definition 13. [Constants relevant to a transition]
For any given (ground) transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle$ of τ_H, by $relObCon_H(T)$ we denote the minimal set of object constants of signature Σ_H of $\mathcal{D}_H$ closed under the following rules:

1. Object constants occurring in a^H are in $relObCon_H(T)$;

2. If $f(x_1, \ldots, x_n) = y$ belongs to σ_1 or σ_2, but not both, then $x_1, \ldots, x_n, y$ are in $relObCon_H(T)$;

3. If body B of an executability condition of a^H contains an occurrence of a term $f(x_1, \ldots, x_n)$ and $f(x_1, \ldots, x_n) = y \in \sigma_1$ then $x_1, \ldots, x_n, y$ are in $relObCon_H(T)$.

Constants from $relObCon_H(T)$ are said to be *relevant* to T. ▲

In Example 5, consider transition $T = \langle \sigma_1, grasp(rob_1, cup_1), \sigma_2 \rangle$ such that $loc(rob_1) = kitchen$ and $loc(cup_1) = kitchen$ are in σ_1. Then, $relObCon_H(T)$ consists of rob_1 of sort $robot$ and cup_1 of sort $object$ (based on the first rule above), and $kitchen$ of sort $place$ (based on the third rule above and fourth axiom in Statement 43 in Example 5). For more details, see Example 8.

Now we are ready for the first step of the construction of $\mathcal{D}_{LR}(T)$. Object constants of the signature $\Sigma_H(T)$ of the new system description $\mathcal{D}_H(T)$ are those of $relObCon_H(T)$. Basic sorts of $\Sigma_H(T)$ are non-empty intersections of basic sorts of Σ_H with $relObCon_H(T)$. The domain attributes and actions of $\Sigma_H(T)$ are those of Σ_H restricted to the basic sorts of $\Sigma_H(T)$, and the axioms of $\mathcal{D}_H(T)$ are restrictions of axioms of $\mathcal{D}_H$ to $\Sigma_H(T)$. It is easy to show that the system descriptions $\mathcal{D}_H$ and $\mathcal{D}_H(T)$ satisfy the following requirement—for any transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle$ of transition diagram τ_H corresponding to system description $\mathcal{D}_H$, there exists a transition $\langle \sigma_1(T), a^H, \sigma_2(T) \rangle$ in transition diagram $\tau_H(T)$ corresponding to system description $\mathcal{D}_H(T)$, where $\sigma_1(T)$ and $\sigma_2(T)$ are obtained by restricting σ_1 and σ_2 (respectively) to the signature $\Sigma_H(T)$.

In the second step, the zoomed system description $\mathcal{D}_{LR}(T)$ is constructed by refining the system description $\mathcal{D}_H(T)$. Unlike the description of refinement in Section 7.1, which requires the designer to supply domain-specific information, no additional input is needed from the designer for refining

$\mathcal{D}_H(T)$ and the zoom operation can be automated. We now provide a formal definition of the zoomed system description.

Definition 14. [Zoomed system description]
For a coarse-resolution transition T, system description $\mathcal{D}_{LR}(T)$ with signature $\Sigma_{LR}(T)$ is said to be the *zoomed fine-resolution system description* if:

1. Basic sorts of $\Sigma_{LR}(T)$ are those of $\mathcal{D}_{LR}$ that are components of the basic sorts of $\mathcal{D}_H(T)$.
2. Functions of $\Sigma_{LR}(T)$ are those of $\mathcal{D}_{LR}$ restricted to the basic sorts of $\Sigma_{LR}(T)$.
3. Actions of $\Sigma_{LR}(T)$ are those of $\mathcal{D}_{LR}$ restricted to the basic sorts of $\Sigma_{LR}(T)$.
4. Axioms of $\mathcal{D}_{LR}(T)$ are those of $\mathcal{D}_{LR}$ restricted to the signature $\Sigma_{LR}(T)$.

▲

Consider the transition $T = \langle \sigma_1, move(rob_1, kitchen), \sigma_2 \rangle$ such that $loc(rob_1) = office \in \sigma_1$. The basic sorts of $\Sigma_{LR}(T)$ include $robot_L^z = \{rob_1\}$, $place_L^z = \{office, kitchen\}$ and $place_L^{*z} = \{c_i : c_i \in kitchen \cup office\}$. Functions of $\Sigma_{LR}(T)$ include $loc^*(rob_1)$ taking values from $place_L^{*z}$, $loc(rob_1)$ taking values from $place_L^z$, defined fluent $range(loc^*(rob_1), place_L^{*z})$, the statics $next\_to^*(place_L^{*z}, place_L^{*z})$, $next\_to(place_L^z, place_L^z)$, and properly restricted functions related to testing the values of fluent terms. The actions include $move^*(rob_1, c_i)$ and $test_{loc^*}(rob_1, rob_1, c_i)$, where c_i are individual elements of $place_L^{*z}$. Finally, restricting the axioms of $\mathcal{D}_{LR}$ to the signature $\Sigma_{LR}(T)$ removes causal laws for $grasp$ and put_down, and the state constraint encoded by Statement 42(a) in $\mathcal{D}_{LR}$. Furthermore, in the causal law and executability condition for $move^*$, we only consider cells in the $kitchen$ or the $office$.

Based on Definition 10 and Proposition 4, it is easy to show that the system descriptions $\mathcal{D}_H(T)$ and $\mathcal{D}_{LR}(T)$ satisfy the following requirement—for any transition $\langle \sigma_1(T), a^H, \sigma_2(T) \rangle$ in transition diagram $\tau_H(T)$ of system description $\mathcal{D}_H(T)$, where $\sigma_1(T)$ and $\sigma_2(T)$ are obtained by restricting states σ_1 and σ_2 (respectively) of $\mathcal{D}_H$ to signature $\Sigma_H(T)$, there exists a path in $\tau_{LR}(T)$ between every refinement δ_1^z of $\sigma_1(T)$ and a refinement δ_2^z of $\sigma_2(T)$. We now provide two examples of constructing the zoomed system description. In Example 7, the coarse-resolution action corresponds to a robot grasping a cup. In Example 8, we consider the coarse-resolution action of the robot moving from one room to another, and demonstrate the benefits of zooming when additional functions are included in the system description.

Example 7. [First example of zoom operation]
As an illustrative example of zooming, consider the transition $T = \langle \sigma_1, grasp(rob_1, cup_1), \sigma_2 \rangle$ such that $(loc(rob_1) = kitchen) \in \sigma_1$. Based on Definition 13, $relObCon_H(T)$ consists of rob_1 of sort $robot$ and cup_1 of sort cup, and $kitchen$ of sort $place$—basic sorts of $\Sigma_H(T)$ are intersections of these sorts with those of Σ_H. The domain attributes and actions of $\Sigma_H(T)$ are restricted to these basic sorts, and axioms of $\mathcal{D}_H(T)$ are those of $\mathcal{D}_H$ restricted to $\Sigma_H(T)$. Now, the signature $\Sigma_{LR}(T)$ of the zoomed system description $\mathcal{D}_{LR}(T)$ has the following:

- Basic sorts $robot_L^z = \{rob_1\}$, $place_L^z = \{kitchen\}$, $place_L^{*z} = \{c_i : c_i \in kitchen\}$, and $object_L^z = \{cup_1\}$.

- Functions that include (a) basic non-knowledge fluents $loc^*(robot_L^z)$ and $loc^*(object_L^z)$ that take values from $place_L^{*z}$, $loc(robot_L^z)$ and $loc(object_L^z)$ that take values from $place_L^z$, and fluent term $in\_hand(robot_L^z, object_L^z)$; (b) defined fluent term $range(loc^*(robot_L^z), place_L^{*z})$; (c) statics such as $next\_to^*(place_L^{*z}, place_L^{*z})$ and $next\_to(place_L^z, place_L^z)$; and (d) knowledge fluents restricted to the basic sorts and fluents.

- Actions that include physical actions, e.g., $move^*(robot_L^z, place_L^{*z})$, $grasp(robot_L^z, object_L^z)$, and $putdown(robot_L^z, object_L^z)$; and actions such as $test_{loc^*}(robot_L^z, robot_L^z, place_L^{*z})$ and $test_{loc^*}(robot_L^z, object_L^z, place_L^{*z})$ that are knowledge-producing.

The axioms of $\mathcal{D}_{LR}(T)$ are those of $\mathcal{D}_{LR}$ restricted to the signature $\Sigma_{LR}(T)$. These axioms include:

$move^*(rob_1, c_j)$ **causes** $loc^*(rob_1) = \{C : range(loc^*(rob_1), C)\}$
$grasp(rob_1, cup_1)$ **causes** $in\_hand(rob_1, cup_1) = \{true, false\}$
$test_{loc^*}(rob_1, rob_1, c_j)$ **causes** $observed_{loc^*}(rob_1, rob_1, c_j) = \{true, false\}$ **if** $loc^*(rob_1) = c_j$
$test_{loc^*}(rob_1, cup_1, c_j)$ **causes** $observed_{loc^*}(rob_1, cup_1, c_j) = \{true, false\}$ **if** $loc^*(cup_1) = c_j$
impossible $move^*(rob_1, c_j)$ **if** $loc^*(rob_1) = c_i,\ \neg next\_to^*(c_j, c_i)$
impossible $grasp(rob_1, cup_1)$ **if** $loc^*(rob_1) = c_i,\ loc^*(cup_1) = c_j,\ c_i \neq c_j$

where $range(loc^*(rob_1), C)$ may hold for $C \in \{c_i, c_j, c_k\}$, cells within the range of the robot's current location (c_i) and elements of sort $place_L^{*z}$. The states of $\tau_{LR}(T)$ include atoms such as $loc^*(rob_1) = c_i$ and $loc^*(cup_1) = c_j$, where $c_i, c_j \in place_L^{*z}$, $in\_hand(rob_1, cup_1)$, $next\_to^*(c_i, c_j)$, and $observed_{loc^*}(rob_1, rob_1, c_i) = true$. Actions include $move^*(rob_1, c_i)$, $grasp(rob_1, cup_1)$, $putdown(rob_1, cup_1)$, $test_{loc^*}(rob_1, rob_1, c_i)$ and $test_{loc^*}(rob_1, cup_1, c_i)$. ♦

Example 8. [Second example of zoom operation]
Consider the transition $T = \langle \sigma_1, move(rob_1, kitchen), \sigma_2 \rangle$ such that $loc(rob_1) = office \in \sigma_1$. In addition to the description in Example 5, assume that the coarse-resolution description of the logician includes (a) boolean fluent $broken(robot)$; and (b) fluent $color(robot)$ taking a value from a set of colors—there is also an executability condition:

$$\textbf{impossible}\ move(Rb, Pl)\ \textbf{if}\ broken(Rb)$$

Intuitively, $color(Rb)$ and $broken(Rb)$, where $Rb \neq rob_1$, are not relevant to a^H, but $broken(rob_1)$ is relevant. Specifically, based on Definition 13, $relObCon_H(T)$ consists of rob_1 of sort $robot$, and $\{kitchen, office\}$ of sort $place$—basic sorts of $\Sigma_H(T)$ are intersections of these sorts with those of Σ_H. Similar to Example 7, the domain attributes and actions of signature $\Sigma_H(T)$ are restricted to these basic sorts, and axioms of $\mathcal{D}_H(T)$ are those of $\mathcal{D}_H$ restricted to $\Sigma_H(T)$, e.g., they only include suitably ground instances of Statement 41(a), Statement 42(b), and Statement 43(a-c). The signature $\Sigma_{LR}(T)$ of $\mathcal{D}_{LR}(T)$ has the following:

- The domain's basic sorts $robot_L^z = \{rob_1\}$, $place_L^z = \{office, kitchen\}$ and $place_L^{*z} = \{c_i : c_i \in kitchen \cup office\}$.

- Functions that include (a) fluents $loc(robot_L^z)$ and $loc^*(robot_L^z)$ that take values from $place_L^z$ and $place_L^{*z}$ respectively, and defined fluent term $range(loc^*(robot_L^z), place_L^{*z})$; (b) statics such as $next\_to^*(place_L^{*z}, place_L^{*z})$ and $next\_to(place_L^z, place_L^z)$; (c) fluent $broken(robot_L^z)$; and (d) $observed_{loc^*}(robot_L^z, robot_L^z, place_L^z)$ and other relevant knowledge fluents.

- Actions that include (a) $move^*(robot_L^z, place_L^{*z})$; and (b) $test_{loc^*}(robot_L^z, robot_L^z, place_L^{*z})$.

The axioms of $\mathcal{D}_{LR}(T)$ are those of $\mathcal{D}_{LR}$ restricted to $\Sigma_{LR}(T)$, e.g., they include:

$move^*(rob_1, c_j)$ **causes** $loc^*(rob_1) = \{C : range(loc^*(rob_1), C)\}$
$test_{loc^*}(rob_1, rob_1, c_j)$ **causes** $observed_{loc^*}(rob_1, rob_1, c_j) = \{true, false\}$ **if** $loc^*(rob_1) = c_j$
impossible $move^*(rob_1, c_j)$ **if** $loc^*(rob_1) = c_i,\ \neg next\_to^*(c_j, c_i)$
impossible $move^*(rob_1, c_j)$ **if** $broken(rob_1)$

where $range(loc^*(rob_1), C)$ may hold for $C \in \{c_i, c_j, c_k\}$, which are within the range of the robot's current location (c_i), and are elements of $place_L^{*z}$. Assuming the robot is not broken, each state of $\tau_{LR}(T)$ thus includes an atom of the form $loc^*(rob_1) = c_i$, where c_i is a cell in the $kitchen$ or the $office$, $\neg broken(rob_1)$, direct observations of this atom, e.g., $observed_{loc^*}(rob_1, rob_1, c_i) = true$, and statics such as $next\_to^*(c_i, c_j)$. Particular actions under consideration include $move^*(rob_1, c_i)$ and $test_{loc^*}(rob_1, rob_1, c_i)$.

As an extension to this example, if robot rob_1 is holding textbook tb_1 before executing the action $a^H = move(rob_1, kitchen)$, i.e., $in\_hand(rob_1, tb_1) \in \sigma_1$, then $\Sigma_H(T)$ also includes tb_1 of sort $textbook$, and $\Sigma_{LR}(T)$ includes $object_L^z = \{tb_1\}$. The functions of $\mathcal{D}_{LR}(T)$ include basic fluent $in\_hand(robot_L^z, object_L^z)$ and the corresponding knowledge fluents, and the actions and axioms are suitably restricted. ♦

In Examples 7 and 8, probabilities are assigned to the outcomes of actions based on the statistics collected earlier (see Definition 11 in Section 7.2). For instance, if action $move(rob_1, c_1)$ is executed, the probabilities of the possible outcomes of this action may be:

$$P(loc^*(rob_1) = c_1) = 0.85$$
$$P(loc^*(rob_1) = Cl \mid range(loc^*(rob_1), Cl), Cl \neq c_1) = \frac{0.15}{|Cl|}$$

Similarly, if the robot has to search for a textbook cup_1 once it reaches the $kitchen$, and if a $test$ action is executed to determine the location of a textbook cup_1 in cell c_i in the $kitchen$, the probabilities of the outcomes may be:

$$P\Big(observed_{loc^*}(rob_1, cup_1, c_i) = true \;\Big|\; loc^*(cup_1) = c_i\Big) = 0.9$$
$$P\Big(observed_{loc^*}(rob_1, cup_1, c_i) = false \;\Big|\; loc^*(cup_1) = c_i\Big) = 0.1$$

Also, when the execution of an action changes the value of a fluent that is its indirect consequence, the probabilities are computed by marginalizing the related fluents. For instance, the probability of a cup being in a particular cell is computed by considering the probability of the robot being in the cell and holding the cup, and the probability of the cup being in the cell but not in the robot's grasp.

Given $\mathcal{D}_{LR}(T)$ and the probabilistic information, the robot now has to execute a sequence of concrete actions that implement the desired transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle$. For instance, a robot searching for cup_1 in the $kitchen$ can check cells in the $kitchen$ for cup_1 until either the cell location of cup_1 is determined with high probability (e.g., ≥ 0.9), or all cells are examined without

locating cup_1. In the former case, the probabilistic belief can be elevated to a fully certain statement, and the robot reasons about the action outcome and observations to infer that cup_1 is in the $kitchen$, whereas the robot infers that cup_1 is not in the $kitchen$ in the latter case. Such a probabilistic implementation of an abstract action as a sequence of concrete actions is accomplished by constructing and solving a POMDP, and repeatedly invoking the corresponding policy to choose actions until termination, as described below.

8. POMDP Construction and Probabilistic Execution

In this section, we describe the construction of a POMDP $\mathcal{P}_o(T)$ as a representation of the zoomed system description $D_{LR}(T)$ and the learned probabilities of action outcomes (Section 7.2), and the use of $\mathcal{P}_o(T)$ for the fine-resolution implementation of transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle$ of τ_H. First, Section 8.1 summarizes the use of a POMDP to compute a policy for selecting one or more concrete actions that implement any given abstract action a^H. Section 8.2 then describes the steps of the POMDP construction in more detail.

8.1 POMDP Overview

A POMDP is described by a tuple $\mathcal{P}_o = \langle A^P, S^P, b_0^P, Z^P, T^P, O^P, R^P \rangle$ for specific goal state(s). This formulation of a POMDP builds on the standard formulation (Kaelbling et al., 1998). Since the states and observations of a POMDP are different from the definitions of these terms as used in this article, we begin by introducing some terminology.

We refer to each state represented by the POMDP as a *p-state*. Recall that each state δ_x of the fine-resolution system description $\mathcal{D}_{LR}(T)$ contains atoms formed of statics, non-knowledge fluent terms and knowledge fluent terms. There is a many-to-one correspondence between states of $\mathcal{D}_{LR}(T)$, and the p-states and observations of the POMDP $\mathcal{P}_o(T)$ constructed from $\mathcal{D}_{LR}(T)$. We provide the following definition of this correspondence.

Definition 15. [P-states and observations of POMDP $\mathcal{P}_o(T)$]
Let $\mathcal{P}_o(T)$ be a POMDP constructed from the zoomed fine-resolution system description $\mathcal{D}_{LR}(T)$.

- Each p-state s of $\mathcal{P}_o(T)$ is a projection of states of $\mathcal{D}_{LR}(T)$ on the set of atoms of the form $f(t) = y$, where $f(t)$ is a basic non-knowledge fine-resolution fluent term, or a special p-state called the *terminal p-state*.

- Each observation z of $\mathcal{P}_o(T)$ is a projection of states of $\mathcal{D}_{LR}(T)$ on the set of atoms of basic knowledge fluent terms corresponding to the robot's observation of the possible values of fine-resolution fluent terms such as $observed_{f^*}(robot, x, y) = outcome$, where y is a possible value of the fluent term $f^*(x)$. For simplicity, we use the observation *none* to replace all instances that have $undet$ as the $outcome$.

In other words, the p-states (observations) of $\mathcal{P}_o(T)$ are obtained by dropping the atoms formed of knowledge (non-knowledge) fluent terms and statics in the states of $\mathcal{D}_{LR}(T)$. ▲

We can now define the elements of a POMDP tuple:

- A^P: set of concrete, fine-resolution actions available to the robot.

- S^P: set of p-states to be considered for probabilistic implementation of a^H.
- b_0^P: initial *belief state*, where a belief state is a probability distribution over S^P.
- Z^P: set of observations.
- $T^P : S^P \times A^P \times S^P \rightarrow [0, 1]$, the transition function, which defines the probability of each transition from one p-state to another when particular actions are executed. As described later, impossible state transitions are not included in T^P.
- $O^P : S^P \times A^P \times Z^P \rightarrow [0, 1]$, the observation function, which defines the probability of obtaining particular observations when particular actions are executed in particular p-states. As described later, one valid state-action-observation combinations are included in O^P.
- $R^P : S^P \times A^P \times S^P \rightarrow \Re$, the reward specification, which encodes the relative immediate reward (i.e., numerical value) of taking specific actions in specific p-states.

The p-states are considered to be *partially observable* because they cannot be observed with complete certainty, and the POMDP reasons with probability distributions over the p-states, called *belief states*. In this formulation, the belief state is a sufficient statistic that implicitly captures all the information in the history of observations and actions.

The use of a POMDP has two phases (1) policy computation; and (2) policy execution. The first phase computes *policy* $\pi^P : B^P \rightarrow A^P$ that maps belief states to actions, using an algorithm that maximizes the utility (i.e., expected cumulative discounted reward) over a planning horizon—we use a point-based approximate solver that only computes beliefs at a few samples points in the belief space (Ong et al., 2010). In the second phase, the computed policy is used to repeatedly choose an action in the current belief state, updating the belief state after executing the action and receiving an observation. This belief revision is based on Bayesian updates:

$$b_{t+1}^P(s_{t+1}^P) \propto O^P(s_{t+1}^P, a_{t+1}^P, o_{t+1}^P) \sum_{s_t^P} \{T^P(s_t^P, a_{t+1}^P, s_{t+1}^P) \cdot b_t^P(s_t^P)\} \tag{68}$$

where b_{t+1}^P, s_{t+1}, a_{t+1} and o_{t+1} are the belief state, p-state, action and observation (respectively) at time $t + 1$. Equation 68 says that b_{t+1}^P is proportional to the product of the terms on the right hand side. The belief update continues until policy execution is terminated. In our case, policy execution terminates when doing so has a higher (expected) utility than continuing to execute the policy. This happens when either the belief in a specific p-state is very high (e.g., ≥ 0.8), or none of the p-states have a high probability associated with them after invoking the policy several times—the latter case is interpreted as a failure to execute the coarse-resolution action under consideration.

8.2 POMDP Construction

Next, we describe the construction of POMDP $\mathcal{P}_o(T)$ for the fine-resolution probabilistic implementation of coarse-resolution transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle \in \tau_H$, using $\mathcal{D}_{LR}(T)$ and the statistics collected in the training phase as described in Section 7.2. We illustrate these steps using examples based on the domain described in Example 1, including the example described in Appendix E.

Actions: the set A^P of actions of $\mathcal{P}_o(T)$ consists of concrete actions from the signature of $\mathcal{D}_{LR}(T)$ and new *terminal actions* that terminate policy execution. We use a single terminal action—if A^P is

to include domain-specific terminal actions, it is the designer's responsibility to specify them. For the discussion below, it will be useful to partition A^P into three subsets (1) A_1^P, actions that cause a transition between p-states; (2) A_2^P, knowledge-producing actions for testing the values of fluents; and (3) A_3^P, terminal actions that terminate policy execution. The example in Appendix E includes (a) actions from A_1^P that move the robot to specific cells, e.g., `move-0` and `move-1` cause robot to move to cell 0 and 1 respectively, and the action $grasp(rob_1, tb_1)$; (b) $test_{loc^*}$ actions from A_2^P to check if the robot or target object (tb_1) are in specific cells; and (c) action $finish$ from A_3^P that terminates policy execution.

P-states, observations, and initial belief state: the following steps are used to construct the set of p-states (S^P), set of observations (Z^P), and the initial belief state (b_0^P).

1. Construct ASP program $\Pi_c(\mathcal{D}_{LR}(T)) \cup Q$. Here, $\Pi_c(\mathcal{D}_{LR}(T))$ is constructed as described in Definition 2 (Section 5.1), and Q is a collection of (a) atoms formed by statics; and (b) disjunctions of atoms formed by basic fluent terms. Each disjunction is of the form $\{f(t) = y_1 \vee \ldots \vee f(t) = y_n\}$, where $\{y_1, \ldots, y_n\}$ are possible values of basic fluent term $f(t)$. Observe that if AQ is an answer set of Q, then there is a state δ of $\mathcal{D}_{LR}(T)$ such that $AQ = \delta^{nd}$; also, for every state δ of $\mathcal{D}_{LR}(T)$, there is an answer set AQ of Q such that $AQ = \delta^{nd}$. It can be shown that AS is an answer set of $\Pi_c(\mathcal{D}_{LR}(T)) \cup Q$ iff it is an answer set of $\Pi_c(\mathcal{D}_{LR}(T)) \cup AQ$ where AQ is an answer set of Q. This statement follows from the definition of answer set and the splitting set theorem (Balduccini, 2009).

2. Compute answer set(s) of ASP program $\Pi_c(\mathcal{D}_{LR}(T)) \cup Q$. Based on the observation in Step-1 above, and the well-foundedness of $\mathcal{D}_{LR}(T)$, it is easy to show that each answer set is unique and is a state of $\mathcal{D}_{LR}(T)$.

3. From each answer set, extract all atoms of the form $f(t) = y$, where $f(t)$ is a basic non-knowledge fine-resolution fluent term, to obtain an element of S^P. Basic fluent terms corresponding to a coarse-resolution domain attribute, e.g., room location of the robot, are not represented probabilistically and thus not included in S^P. We refer to such a projection of a state δ of $\mathcal{D}_{LR}(T)$ as the *p-state defined by* δ. Also include in S^P an "absorbing" terminal p-state $absb$ that is reached when a terminal action from A_3^P is executed.

4. From each answer set, extract all atoms such as $directly\_observed(rob_1, f(t), y) = outcome$, which are formed by basic knowledge fluent terms corresponding to the robot sensing a fine-resolution fluent term's value, to obtain elements of Z^P. We refer to such a projection of a state δ of $\mathcal{D}_{LR}(T)$ as an *observation defined by* δ. As described earlier, for simplicity, observation $none$ replaces all instances in Z^P that have $undet$ as the $outcome$.

5. In general, the initial belief state b_0^P is a uniform distribution, i.e., all p-states are considered to be equally likely. This does not prevent the designer from using other priors, but these priors would have to be derived from sources of knowledge external to our architecture.

In the example in Appendix E, abstract action $grasp(rob_1, tb_1)$ has to be executed in the $office$. To do so, the robot has to move and find tb_1 in the $office$. Example 7 above contains the corresponding $\mathcal{D}_{LR}(T)$. Here, Q includes (a) atoms formed by statics, e.g., $next\_to^*(c_1, c_2)$ where c_1 and c_2 are neighboring cells in the $office$; and (b) disjunctions such as $\{loc^*(rob_1) = c_1 \vee \ldots \vee loc^*(rob_1) = c_n\}$ and $\{loc^*(tb_1) = c_1 \vee \ldots \vee loc^*(tb_1) = c_n\}$, where $\{c_1, \ldots, c_n\} \in office$. In Step-3, p-states

such as $\{loc^*(rob_1) = c_1,\ loc^*(tb_1) = c_1,\ \neg in\_hand(rob_1, tb_1)\}$ are extracted from the answer sets. In Step-4, observations such as $observed_{loc^*}(rob_1, rob_1, c_1) = true$ of the location of rob_1, and $observed_{loc^*}(rob_1, tb_1, c_1) = false$ of the location of tb_1 are extracted from the answer sets. Finally, the initial belief state b_0^P is set as a uniform distribution (Step 5).

Transition function and observation function: next, we consider the construction of T^P and O^P from $\mathcal{D}_{LR}(T)$ and the statistics collected in the initial training phase (see Section 7.2).

A transition between p-states of $\mathcal{P}_o(T)$ is defined as $\langle s_i, a, s_j\rangle \in T^P$ iff there is an action $a \in A_1^P$ and a transition $\langle \delta_x, a, \delta_y\rangle$ of $\mathcal{D}_{LR}(T)$ such that s_i and s_j are p-states defined by δ_x and δ_y respectively. The probability of $\langle s_i, a, s_j\rangle \in T^P$ equals that of $\langle \delta_x, a, \delta_y\rangle$. In a similar manner, $\langle s_i, a, z_j\rangle \in O^P$ iff there is an action $a \in A_2^P$ and a transition $\langle \delta_x, a, \delta_y\rangle$ of $\mathcal{D}_{LR}(T)$ such that s_i and z_j are a p-state and an observation defined by δ_x and δ_y respectively. The probability of $\langle s_i, a, z_j\rangle \in O^P$ is that of $\langle \delta_x, a, \delta_y\rangle$.

First, we augment $\mathcal{D}_{LR}(T)$ with causal laws for proper termination:

$$finish \ \textbf{causes} \ absb$$
$$\textbf{impossible} \ A^P \ \textbf{if} \ absb$$

Next, we note that actions in A_1^P cause p-state transitions but provide no observations, while actions in A_2^P do not cause p-state changes but provide observations, and terminal actions in A_3^P cause transition to the absorbing state and provide no observations. To use state of the art POMDP solvers, we need to represent T^P and O^P as a collection of tables, one for each action. Specifically, $T_a^P[s_i, s_j] = p$ iff $\langle s_i, a, s_j\rangle \in T^P$ and its probability is p. Also, $O_a^P[s_i, z_j] = p$ iff $\langle s_i, a, z_j\rangle \in O^P$ and its probability is p. Algorithm 1 describes the construction of T^P and O^P.

Some specific steps of Algorithm 1 are elaborated below.

- After initialization, Lines 3–12 of Algorithm 1 handle special cases, e.g., any terminal action will cause a transition to the terminal p-state and provide no observations (Lines 4-5).
- An ASP program of the form $\Pi(\mathcal{D}_{LR}(T), s_i, Disj(A))$ (Lines 14, 17 of Algorithm 1) is defined as $\Pi(\mathcal{D}_{LR}(T)) \cup val(s_i, 0) \cup Disj(A)$. Here, $Disj(A)$ is a disjunction of the form $\{occurs(a_1, 0) \vee \ldots \vee occurs(a_n, 0)\}$, where $\{a_1, \ldots, a_n\} \in A$. Lines 14-16 construct and compute answer sets of such a program to identify all possible p-state transitions as a result of actions in A_1^P. Then, Lines 17-19 construct and compute answer set of such a program to identify possible observations as a result of actions in A_2^P.
- Line 16 extracts a statement of the form $occurs(a_k \in A_1^P, 0)$, and p-state $s_j \in S^P$, from each answer set AS, to obtain p-state transition $\langle s_i, a_k, s_j\rangle$. As stated earlier, a p-state is extracted from an answer set by extracting atoms formed by basic non-knowledge fluent terms.
- Line 19 extracts a statement of the form $occurs(a_j \in A_2^P, 0)$, and observation $z_j \in Z^P$, from each answer set AS, to obtain triple $\langle s_i, a_k, z_j\rangle$. As described earlier, an observation is extracted from an answer set by extracting atoms formed by basic knowledge fluent terms.
- Probabilities of p-state transitions are set (Line 16) based on the corresponding physical state transition probabilities (first type of transition in Definition 11 in Section 7.2). Probabilities of observations are set (Line 19) based on the knowledge state transition probabilities (second type of transition in Definition 11 in Section 7.2).

Algorithm 1: Constructing POMDP transition function T^P and observation function O^P

Input: S^P, A^P, Z^P, $\mathcal{D}_{LR}(T)$; transition probabilities for actions $\in A_1^P$; observation probabilities for actions $\in A_2^P$.
Output: POMDP transition function T^P and observation function O^P.

1 Initialize T^P as $|S^P| \times |S^P|$ identity matrix for each action.
2 Initialize O^P as $|S^P| \times |Z^P|$ matrix of zeros for each action.

`// Handle special cases`
3 **for** *each* $a_j \in A_3^P$ **do**
4 $\quad T^P_{a_j}(*, absb) = 1$
5 $\quad O^P_{a_j}(*, none) = 1$
6 **end**
7 **for** *each action* $a_j \in A_1^P$ **do**
8 $\quad O^P_{a_j}(*, none) = 1$
9 **end**
10 **for** *each* $a_j \in A^P$ **do**
11 $\quad O^P_{a_j}(absb, none) = 1$
12 **end**

`// Handle normal transitions`
13 **for** *each p-state* $s_i \in S^P$ **do**
`// Construct and set probabilities of p-state transitions`
14 Construct ASP program $\Pi(\mathcal{D}_{LR}(T), s_i, Disj(A_1^P))$.
15 Compute answer sets **AS** of ASP program.
16 From each $AS \in \mathbf{AS}$, extract p-state transition $\langle s_i, a_k, s_j \rangle$, and set the probability of $T^P_{a_k}[s_i, s_j]$.
`// Construct and set probabilities of observations`
17 Construct ASP program $\Pi(\mathcal{D}_{LR}(T), s_i, Disj(A_2^P))$.
18 Compute answer sets **AS** of ASP program.
19 From each $AS \in \mathbf{AS}$, extract triple $\langle s_i, a_k, z_j \rangle$, and set value of $O^P_{a_k}[s_i, z_j]$.
20 **end**
21 **return** T^P and O^P.

In the example in Appendix E, a robot in the $office$ has to pick up textbook tb_1 that is believed to be in the $office$. This example assumes that a $move$ action from one cell to a neighboring cell succeeds with probability 0.95—the remaining probability of 0.05 the robot remains in its current cell. It is also assumed that with probability 0.95 the robot observes (does not observe) the textbook when it exists (does not exist) in the cell the robot is currently in. The corresponding T^P and O^P, constructed for this example, are shown in Appendix E.

The correctness of the approach used to extract p-state transitions and observations, in Lines 16, 19 of Algorithm 1, is based on the following propositions.

Proposition 5. [Extracting p-state transitions from answer sets]

Algorithm 2: Construction of POMDP reward function R^P

Input: S^P, A^P, and T^P; statistics regarding accuracy and time taken to execute non-terminal actions.

Output: Reward function R^P.

```
// Consider each possible p-state transition
for each (s, a, s') ∈ S^P × A^P × S^P with T^P(s, a, s') ≠ 0 do
    // Consider terminal actions first
    if a ∈ A_3^P then
        if s' is a goal p-state then
            R^P(s, a, s') = large positive value.
        else
            R^P(s, a, s') = large negative value.
        end
    // Rewards are costs for non-terminal actions
    else
        Set R^P(s, a, s') based on relative computational effort and accuracy.
    end
end
return R^P
```

- If $\langle s_i, a, s_j \rangle \in T^P$ then there is an answer set AS of program $\Pi(\mathcal{D}_{LR}(T), s_i, Disj(A_1^P))$ such that $s_j = \{f(\bar{x}) = y : f(\bar{x}) = y \in AS \text{ and } f \text{ is basic}\}$.
- For every answer set AS of program $\Pi(\mathcal{D}_{LR}(T), s_i, Disj(A_1^P))$ and $s_j = \{f(\bar{x}) = y : f(\bar{x}) = y \in AS \text{ and } f \text{ is basic}\}$, $\langle s_i, a, s_j \rangle \in T^P$.

★

Proposition 6. [Extracting observations from answer sets]

- If $\langle s_i, a, z_j \rangle \in O^P$ then there is an answer set AS of program $\Pi(\mathcal{D}_{LR}(T), s_i, Disj(A_2^P))$ such that $z_j = \{f(\bar{x}) = y : f(\bar{x}) = y \in AS \text{ and } f \text{ is basic}\}$.
- For every answer set AS of program $\Pi(\mathcal{D}_{LR}(T), s_i, Disj(A_1^P))$ and $z_j = \{f(\bar{x}) = y : f(\bar{x}) = y \in AS \text{ and } f \text{ is basic}\}$, $\langle s_i, a, z_j \rangle \in O^P$.

★

Proposition 5 says that a p-state transition is in $\mathcal{P}_o(T)$ *iff* a matching transition is in $\mathcal{D}_{LR}(T)$, and that for any state transition in $\mathcal{D}_{LR}(T)$ a matching p-state transition is in $\mathcal{P}_o(T)$. Proposition 6 makes a similar statement about observations of $\mathcal{P}_o(T)$. These propositions are true by construction, and help establish that every state transition in $\mathcal{D}_{LR}(T)$, the zoomed and randomized fine-resolution system description relevant to the coarse-resolution transition T, is achieved by a sequence of actions (executed) and observations (obtained) in $\mathcal{P}_o(T)$.

Reward specification: the reward function R^P assigns a real-valued reward to each p-state transition, as described in Algorithm 2. Specifically, for any state transition with a non-zero probability in T^P:

1. If it corresponds to a terminal action from A_3^P, the reward is a large positive (negative) value if this action is chosen after (before) achieving the goal p-state.

2. If it corresponds to a non-terminal action, reward is a real-valued cost (i.e., negative reward) of executing the action.

Here, any p-state $s \in S^P$ defined by state δ of $\mathcal{D}_{LR}(T)$ that is a refinement of σ_2 in transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle$ is a goal p-state. In Appendix E, we assign large positive reward (100) for executing $finish$ when textbook tb_1 is in the robot's grasp, and large negative reward (-100) for terminating before tb_1 has been grasped (Lines 3-7, Algorithm 2). We assign a fixed cost (-1) for all other (i.e., non-terminal) actions (Line 9). When necessary, this cost can be a heuristic function of relative computational effort and accuracy, using domain expertise and statistics collected experimentally, e.g., we can set $R^P(*, shape, *) = -1$ and $R^P(*, color, *) = -2$ because statistics indicate that the knowledge-producing action that determines an object's color takes twice as much time as the action that determines the object's shape. Although we do not do so in our example, it is also possible to assign high cost (i.e., large negative reward) to transitions that should be avoided or are dangerous, e.g., actions that take a wheeled robot near a flight of stairs. The reward function, in turn, influences the (a) rate of convergence during policy computation; and (b) accuracy of results during policy execution. Appendix E describes the reward function for a particular example.

Computational complexity and efficiency: Let us consider the complexity of solving POMDPs and or our approach to construct the POMDPs. For exact algorithms (i.e., algorithms that solve POMDPs optimally), the complexity class of infinite-horizon stochastic-transition POMDPs with boolean rewards is known to be *EXPTIME*; for polynomial time-bounded POMDPs, the complexity class improves to *PSPACE* (Littman, 1996). Approximate belief-point approaches, which we employ here, are more efficient. In these the complexity of one backup, i.e., one step of belief update process, across all belief points is given by (Shani, Pineau, & Kaplow, 2013):

$$O(|A^P| \times |Z^P| \times |V^P| \times |S^P|^2 + |B^P| \times |A^P| \times |S^P| \times |Z^P|) \tag{69}$$

where B^P is the set of belief points. This compares favorably with the complexity of one backup across all α-vectors[9] for exact algorithms, which is (Shani et al., 2013):

$$O(|A^P| \times |Z^P| \times |V^P| \times |S^P|^2 + |A^P| \times |S^P| \times |V^P|^{|Z^P|}) \tag{70}$$

where V^P is the set of α-vectors. For more details about the complexity of POMDP solvers, please see (Shani et al., 2013).

Even the (approximate) belief point algorithms are susceptible to problem size, with the best solvers able to tackle problems with a few hundred p-states (i.e., $|S^P| \simeq 100$) if both the transition and observation functions are stochastic, as they are here[10]. Thus, there is an advantage in reducing

9. The α-vectors are hyperplanes computed in belief space—these vectors are used to select the appropriate action to be executed in any given belief state.
10. There are solvers, such as POMCP, which work on very large state spaces, but which have not had demonstrable results on problems that show scaling with both stochastic transitions and observations (Silver & Veness, 2010).

the problem to be tackled by the solver. In a POMDP created from a relational representation, such as employed here, this is particularly critical. In general, if we have m fluents, each with an average of k values, $|S^P| = k^m$. In our approach, domain knowledge and prior information (e.g., defaults encoded in ASP program at coarse-resolution) remove a proportion of atoms formed of fluent literals from consideration during zooming. If we model the remaining proportion of fluent literals as $0 < \beta < 1$, then clearly $|S^P| = k^{\beta m}$. As indicated by Equation 69, this reduction can provide significant computational benefits, especially in more complex domains where many more fluents are likely to be irrelevant to any given transition, e.g., if only two out of a 100 atoms are relevant, $|S^P| = k^{0.02m}$.

For specific tasks such as path planning, it may also be possible to use specific heuristic or probabilistic algorithms that are more computationally efficient than a POMDP. However, POMDPs provide a (a) principled and quantifiable trade-off between accuracy and computational efficiency in the presence of uncertainty in both sensing and actuation; and (b) near-optimal solution if the POMDP's components are modeled correctly. The computational efficiency of POMDPs can also be improved by incorporating hierarchical decompositions, or by dividing the state estimation problem into sub-problems that model actions and observations influencing one fluent independent of those influencing other fluents—we have pursued such options in other work (Zhang et al., 2013). These approaches are not always possible, e.g., when a robot is holding a textbook in hand, the robot's location and the textbook's location are not independent. Instead, in our architecture, we preserve such constraints and construct a POMDP for the relevant part of the domain to significantly reduce the computational complexity of solving the POMDP. Furthermore, many of the POMDPs required for a given domain can be precomputed, solved and reused. For instance, if the robot has constructed a POMDP for locating a textbook in a room, the POMDP for locating a different book (or even a different object) in the same room may only differ in the values of some transition probabilities, observation probabilities, and rewards. This similarity between tasks may not hold in non-stationary domains, in which the elements of the POMDP tuple (e.g., set of p-states) and the collected statistics (e.g., transition probabilities) may need to be revised over time.

Our algorithms for constructing the POMDP $\mathcal{P}_o(T)$ for a specific coarse-resolution transition have two key steps: (1) construction of matrices that represent the functions for transition, observation and reward; and (2) computing answer sets of specific ASP programs to identify valid transitions, observations etc. The first step is polynomial in the size of S^P and Z^P ($|S^P|$ is usually bigger than $|Z^P|$). The second step, which involves grounding the domain attributes and then computing possible answer sets, can (in the worst case) be exponential in the size of (ground) atoms (Gebser, Kaminski, Kaufmann, & Schaub, 2012)[11]. Recall that we only consider object constants relevant to the transition under consideration (see Section 7.3 on zooming). This, in conjunction with the fact that we reuse POMDPs when possible (as described above), makes the construction of $\mathcal{P}_o(T)$ computationally efficient.

Computational error: Although the outcomes of POMDP policy execution are non-deterministic, following an optimal policy produced by an exact POMDP solver is most likely (among all such possible policies) to take the robot to a goal p-state if the following conditions hold:

- The coarse-resolution transition diagram τ_H of the domain has been constructed correctly;

11. In many modern ASP solvers based on SAT algorithms, the exponential factor is a small number greater than 1. We can also use solvers that do incremental grounding (Gebser et al., 2015).

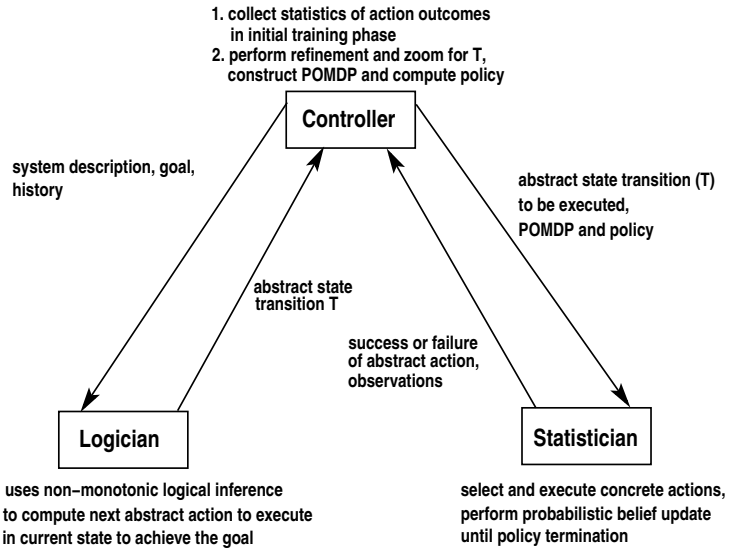


Figure 4: The proposed architecture can be viewed as a *logician* and a *statistician* communicating through a *controller*. The architecture combines the complementary strengths of declarative programming and probabilistic models.

- The statistics collected in the training phase (Section 7.2) correctly model the domain dynamics; and

- The reward function is constructed to suitably reward desired behavior.

This statement is based on existing literature (Kaelbling et al., 1998; Littman, 1996; Sondik, 1971). We use an approximate POMDP solver for computational efficiency, and an exact belief update (Equation 68), which provides a bound on the regret (i.e., loss in value) achieved by following the computed policy in comparison with the optimal policy (Ong et al., 2010). *We can thus only claim that the outcomes of executing of our policy are approximately correct with high probability*. We can also provide a bound on the margin of error (Ong et al., 2010), i.e. the probability that at least one incorrect commitment is made to the history. For instance, if the posterior probability associated with a statement $observed_f(R, X, Y)$ in the fine-resolution representation is p, the probability of error in the corresponding commitment made to the history $\mathcal{H}$ (in the coarse-resolution representation) based on this statement is $(1 - p)$. If a series of statements with probabilities p_i are used to arrive at a conclusion that is committed to $\mathcal{H}$, $(1 - \prod_i p_i)$ is the corresponding probability that at least one erroneous commitment has been made. If a later commitment j is based on a prior belief influenced by previous commitment i then p_j is a conditional probability, conditioned on that previous commitment.

9. Reasoning System and Control Loop of REBA

Recall that the fifth step of the design methodology (see Section 4) is to:

5. Provide the components described above, together with any desired goal, to a reasoning system that directs the robot towards achieving this goal.

Algorithm 3: Control loop of REBA

Input: coarse-resolution system description $\mathcal{D}_H$ and history $\mathcal{H}$; its randomized refinement $\mathcal{D}_{LR}$; coarse-resolution description of goal G; coarse-resolution initial state σ.

Output: Return success/true, indicating that with high probability the robot is in a state satisfying G; return failure/false, indicating that with high probability achieving G is impossible.

```
1  Function Main()
2      done := false
3      currState := σ
4      while ¬ done do
5          Logician extracts a plan of abstract actions a_1^H, ..., a_n^H for G from the answer set of
           Plan := Π^n(D_H, H) ∪ Classical_plan ∪ DPC (see Proposition 2)
6          if no plan found then
7              return failure
8          end
9          done := Implement(a_1^H, ..., a_n^H, currState, H)
10     end
11     return success
   // Function below implements plan of abstract actions; records
      successful execution of action and observations in H and
      sets currState to current state of the system; returns
      true/success if entire sequence is implemented, otherwise
      returns false/failure
12 Function Implement(a_1^H, ..., a_n^H, currState, H)
13     i := 1
14     executable := true
15     while (i ≤ n) ∧ executable do
16         currAction := a_i^H
17         executable := Implement(currState, currAction, H)
18         if executable then
19             i := i+1
20         end
21     end
22     return executable
```

Algorithm 3 describes the reasoning system and overall control loop of REBA, our architecture for building intelligent robots. For this description, we (once again) view a robot as consisting of a logician and a statistician, who communicate through a controller, as described in Section 1 and shown in Figure 4. For any given goal G, the logician takes as input the system description $\mathcal{D}_H$ that corresponds to a coarse-resolution transition diagram τ_H, recorded history $\mathcal{H}$ with initial state

```
   // Implement particular coarse-resolution action at
   fine-resolution; return failure/false if action cannot be
   executed in current state, or if the fine-resolution execution
   terminates without implementing this action; otherwise, update
   coarse-resolution state currState and history H, and return
   success/true
```
23 **Function** *Implement*($currState, currAction, \mathcal{H}$)
24 Controller extracts $T = \langle currState, currAction, \sigma' \rangle$ from answer set of $\Pi(\mathcal{D}_H, currAction, currState)$

```
   // Exit if the coarse-resolution action can no longer be
   executed
```
25 **if** *no answer set* **then**
26 **return** failure
27 **end**

```
   // Zoom, construct and solve POMDP relevant to T
```
28 Controller zooms to $\mathcal{D}_{LR}(T)$, the part of $\mathcal{D}_{LR}$ relevant to transition T and constructs a POMDP $\mathcal{P}_o(T)$.
29 Controller solves $\mathcal{P}_o(T)$ to compute a policy.

```
   // Execute POMDP policy until termination
```
30 executed := false
31 **while** $\neg$ *executed* **do**
32 Statistician selects action using policy, executes action, obtains observation, and updates belief state
33 **if** *terminal action executed* **then**
34 Statistician communicates observations to the controller
35 executed := true
36 **end**
37 **end**

```
   // Fine-resolution inference, update H and currState
```
38 Controller performs fine-resolution inference, adds coarse-resolution outcomes and observations to $\mathcal{H}$
39 $currState :=$ current coarse-resolution state
40 **if** $currState \neq \sigma'$ **then**
41 **return** failure
42 **return** success

defaults (see Example 2), and the current coarse-resolution state σ. Diagnostics and planning to achieve G are reduced to computing answer sets of the corresponding CR-Prolog program $Plan = \Pi^n(\mathcal{D}_H, \mathcal{H}) \cup Classical\_plan \cup DPC$ (Line 5, also see Proposition 2). If no such answer set is found, the control loop terminates reporting failure (Lines 6-8). If a plan exists, each coarse-resolution action $a_i^H, i \in [1, n]$ in the plan is implemented one after the other until either one of the

actions can no longer be executed, or the entire sequence is implemented successfully (Lines 12-22, Algorithm 3).

To implement a given coarse-resolution action a_i^H in current state $currState$, the controller first checks whether the corresponding transition is feasible (Line 24). If not, the implementation of this action, and thus the entire coarse-resolution plan, is terminated early (Lines 25-27). If the transition is feasible, the controller zooms to the relevant part of the randomized, fine-resolution system description $\mathcal{D}_{LR}(T)$, constructs the corresponding POMDP $\mathcal{P}_o(T)$, and solves it to obtain a policy (Lines 28-29). The statistician repeatedly invokes this policy to select an action, execute the action, obtain an observation, and update the belief state, until a terminal action is executed (Lines 31-37). The action outcomes are communicated to the controller, which performs fine-resolution inference, updates coarse-resolution history $\mathcal{H}$, and updates $currState$ to be the current coarse-resolution state. Note that the fine-resolution implementation of a coarse-resolution action succeeds iff the desired coarse-resolution transition is achieved (Lines 40-42).

Notice that executing Algorithm 3 involves:

1. Applying the planning and diagnostics algorithm from Section 5.3 for planning with τ_H and $\mathcal{H}$;
2. For any given coarse-resolution transition T, automatically constructing $\mathcal{D}_{LR}(T)$ by zooming, as described in Section 7; and
3. Constructing a POMDP from $\mathcal{D}_{LR}(T)$, solving it, and using the corresponding policy to execute a sequence of fine-resolution actions implementing T until termination, as discussed in Section 8.

It is not difficult to show that the algorithm satisfies the specifications. Consider the algorithm's behavior when it receives the appropriate input and there is a state satisfying the assigned goal. In this case, when the control loop is completed, with high probability (or equivalently, with a low margin of error) the robot will be in a state satisfying the goal. Also, if the goal cannot be achieved, the robot will (with high probability) report failure in achieving this goal. The control loop thus results in *correct* behavior of the robot.

The execution of fine-resolution actions based on probabilistic models of uncertainty in perception and actuation (e.g., Line 32, Algorithm 3) is supported by probabilistic state estimation algorithms that process inputs from sensors and actuators. For instance, the robot builds a map of the domain and estimates its position in the map using a *Particle Filter* algorithm for Simultaneous Localization and Mapping (SLAM) (Thrun et al., 2005). This algorithm represents the true underlying probability distribution over the possible states using samples drawn from a proposal distribution. Samples more likely to represent the true state, determined based on the degree of match between the expected and actual sensor observations of domain landmarks, are assigned higher (relative) weights and re-sampled to incrementally converge to the true distribution. Implementations of the particle filtering algorithm are used widely in the robotics literature to track multiple hypotheses of system state. A similar algorithm is used to estimate the pose of the robot's arm. On the physical robot, other algorithms used to process specific sensor inputs. For instance, we use existing implementations of algorithms to process camera images, which are the primary source of information to identify specific domain objects. The robot also uses an existing implementation of a SLAM algorithm to build a domain map and localize itself in the map. These algorithms are summarized in Section 10, when we discuss experiments on physical robots.

10. Experimental Setup and Results

This section describes the experimental setup and results of evaluating REBA's capabilities.

10.1 Experimental Setup

The proposed architecture was evaluated in simulation and on a physical robot. As stated in Section 8, statistics of action execution, e.g., observed outcomes of all actions and computation time for knowledge producing actions, are collected in an initial training phase. These statistics are used by the controller to compute the relative utility of different actions, and the probabilities of obtaining different action outcomes and observations. The simulator uses these statistics to simulate the robot's movement and perception. In addition, the simulator represents objects using probabilistic functions of features extracted from images, with the corresponding models being acquired in an initial training phase—see (Zhang et al., 2013) for more details about such models.

In each experimental trial, the robot's goal was to find and move specific objects to specific places—the robot's location, the target object, and locations of domain objects were chosen randomly. An action sequence extracted from an answer set of the ASP program provides a plan comprising abstract actions, each of which is executed probabilistically. Our refinement-based architecture "REBA" was compared with: (1) POMDP-1, which constructs a POMDP from the fine-resolution description (and computed statistics), computes the policy, and uses this policy to implement the desired abstract action; and (2) POMDP-2, which revises POMDP-1 by assigning specific probability values to default statements to bias the initial belief. The performance measures were: (a) *success*, the fraction (or %) of trials in which the robot achieved the assigned goals; (b) *planning time*, the time taken to compute a plan to achieve the assigned goal; and (c) the average *number of actions* that were executed to achieve the desired goal. We experimentally evaluate the following three key hypotheses:

H1 REBA simplifies design in comparison with architectures based on purely probabilistic reasoning and increases confidence in the correctness of the robot's behavior;

H2 REBA achieves the assigned goals more reliably and efficiently than POMDP-1; and

H3 Our representation for defaults improves reliability and efficiency in comparison with not using defaults or assigning specific probability values to defaults.

We examine the first hypothesis qualitatively in the context of some execution traces grounded in the illustrative domain described in Example 1 (Section 10.2). We then discuss the quantitative results corresponding to the experimental evaluation of the other two hypotheses in simulation and on physical robots (Section 10.3).

10.2 Execution Traces

The following (example) execution traces illustrate some of the key capabilities of the proposed architecture.

Execution Example 1. *[Planning with default knowledge]*
Consider the scenario in which a robot is assisting with a meeting in the $office$, i.e., $loc(rob_1, office)$, and is assigned a goal state that contains:

$$loc(cup_1, office)$$

where the robot's goal is to move coffee cup cup_1 to the $office$.

- The plan of abstract actions, as created by the logician, is:

$$move(rob_1, kitchen), \;\; grasp(rob_1, cup_1)$$
$$move(rob_1, office), \;\; putdown(rob_1, cup_1)$$

  Note that this plan uses initial state default knowledge that $kitchenware$ are usually found in the $kitchen$. Each abstract action in this plan is executed by computing and executing a sequence of concrete actions.

- To implement $move(rob_1, kitchen)$, the controller constructs $\mathcal{D}_{LR}(T)$ by zooming to the part of $\mathcal{D}_{LR}$ relevant to this action. For instance, only cells in the $kitchen$ and the $office$ are possible locations of rob_1, and $move$ is the only action that can change the physical state, in the fine-resolution representation.

- $\mathcal{D}_{LR}(T)$ is used to construct and solve a POMDP to obtain an action selection policy, which is provided to the statistician. The statistician repeatedly invokes this policy to select actions (until a terminal action is selected) that are executed by the robot. In the context of Figure 3(b), assume that the robot moved from cell $c_1 \in office$ to $c_5 \in kitchen$ (through cell $c_2 \in office$) with high probability.

- The direct observation from the POMDP, $observed_{loc^*}(rob_1, rob_1, c_5) = true$, is used by the controller for inference in $\mathcal{D}_{LR}(T)$ and $\mathcal{D}_L$, e.g., to produce $observed_{loc}(rob_1, rob_1, kitchen)$. The controller adds this information to the coarse-resolution history $\mathcal{H}$ of the logician, e.g., $obs(rob_1, loc(rob_1) = kitchen, 1)$. Since the first abstract action has had the expected outcome, the logician sends the next abstract action in the plan, $grasp(rob_1, cup_1)$ to the controller for implementation.

- A similar sequence of steps is performed for each abstract action in the plan, e.g., to grasp cup_1, the robot locates the coffee cup in the $kitchen$ and then picks it up. Subsequent actions cause rob_1 to move cup_1 to the $office$, and put cup_1 down to achieve the assigned goal.

Execution Example 2. *[Planning with unexpected failure]*
Consider the scenario in which a robot in the $office$ is assigned the goal of fetching textbook tb_1, i.e., the initial state includes $loc(rob_1, office)$, and the goal state includes:

$$loc(tb_1, office)$$

The coarse-resolution $\mathcal{D}_H$ and history $\mathcal{H}$, along with the goal, are passed on to the logician.

- The plan of abstract actions, as created by the logician, is:

$$move(rob_1, main\_library), \;\; grasp(rob_1, tb_1)$$
$$move(rob_1, office), \;\; putdown(rob_1, tb_1)$$

  This plan uses default knowledge, i.e., that textbooks are typically in the $main\_library$ (Statement 27). Each abstract action in this plan is executed by computing a POMDP policy that is invoked to execute a sequence of concrete actions.

- Assume that $loc(rob_1, main\_library)$, i.e., that the robot is in the $main\_library$ after successfully executing the first abstract action. To execute the $grasp(rob_1, tb_1)$ action, the controller constructs $\mathcal{D}_{LR}(T)$ by zooming to the part of $\mathcal{D}_{LR}$ relevant to this action. For instance, only cells in the $main\_library$ are possible locations of rob_1 and tb_1 in the fine-resolution representation.

- $\mathcal{D}_{LR}(T)$ is used to construct and solve a POMDP to obtain an action selection policy, which is provided to the statistician. The statistician repeatedly invokes this policy to select actions (until a terminal action is selected) that are executed by the robot. In the context of Figure 3(b), if r_2 is the $main\_library$, the robot may move to and search for tb_1 in each cell in r_2, starting from its current location.

- The robot unfortunately does not find tb_1 in any cell of the $main\_library$ in the second step. These observations from the POMDP, i.e., $observed_{loc^*}(rob_1, tb_1, c_i) = false$ for each $c_i \in main\_library$, are used by the controller for inference in $\mathcal{D}_{LR}(T)$ and $\mathcal{D}_L$. This inference produces observations such as $observed_{loc}(rob_1, tb_1, main\_library) = false$, which (in turn) results in suitable statements being added by the controller to the coarse-resolution history $\mathcal{H}$, e.g., $obs(rob_1, loc(tb_1) \neq main\_library, 2)$.

- The inconsistency caused by the observation is resolved by the logician using a CR rule, and the new plan is created based on the second initial state default that a textbook not in the $main\_library$ is typically in the $aux\_library$ (Statement 28):

$$move(rob_1, aux\_library),\ \ grasp(rob_1, tb_1)$$
$$move(rob_1, office),\ \ putdown(rob_1, tb_1)$$

- This time, the robot is able to successfully execute each abstract action in the plan, i.e., it is able to move to the $aux\_library$, find tb_1 and grasp it, move back to the $office$, and put tb_1 down to achieve the assigned goal.

Both these examples illustrate key advantages provided by the formal definitions, e.g., of the different system descriptions and the tight coupling between them, which are part of our architecture:

1. Once the designer has provided the domain-specific information, e.g., for refinement or for computing the probabilities of action outcomes, planning, diagnostics, and execution of a plan computed for any given goal can be automated.

2. Attention is automatically directed to the relevant knowledge at the appropriate resolution. For instance, reasoning by the logician (statistician) is restricted to a coarse-resolution (zoomed fine-resolution) system description. It is thus easier to understand, and to fix errors in, the observed behavior, in comparison with architectures that consider all the available knowledge or only support probabilistic reasoning.

3. There is smooth transfer of control and relevant knowledge between components of the architecture, and confidence in the correctness of the robot's behavior. Also, the proposed methodology supports the use of this architecture on different robots in different domains, e.g., Section 10.3 describes the use of this architecture on robots in two different indoor domains.

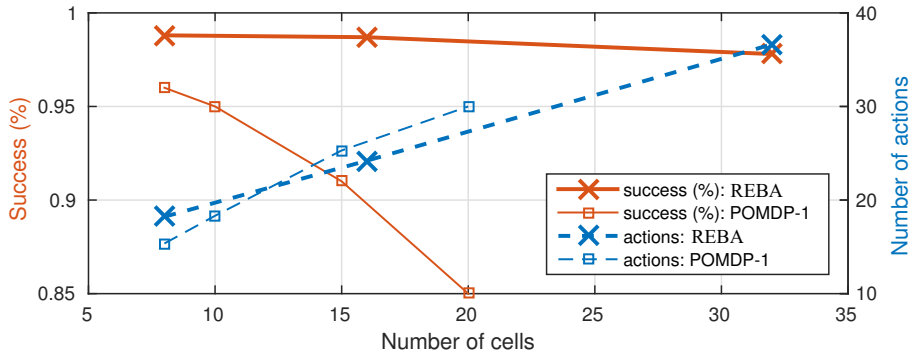


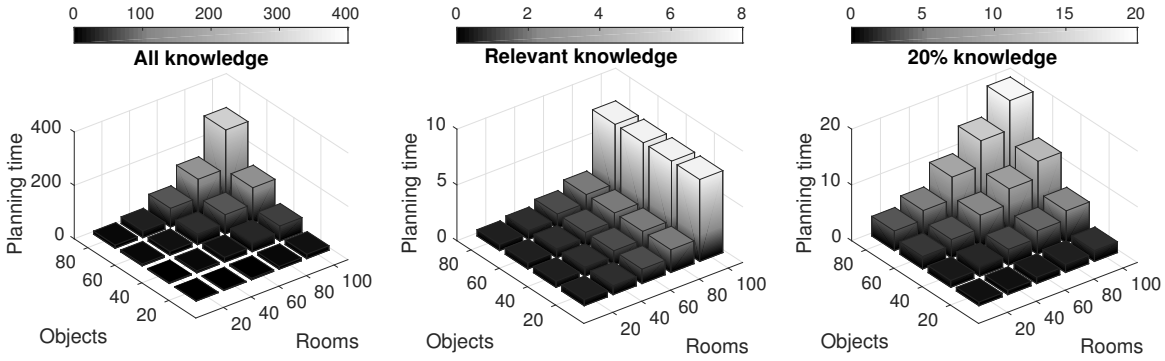
Figure 5: Ability to successfully achieve the assigned goal, and the number of actions executed before termination, as a function of the number of cells in the domain. REBA significantly increases accuracy and reduces the number of actions executed, in comparison with POMDP-1, as the number of cells in the domain increases.

Next, we describe the experimental evaluation of hypotheses H2 and H3 in simulation and on a mobile robot.

10.3 Experimental Results

To evaluate hypothesis H2, we first compared REBA with POMDP-1 in a set of trials in which the robot's initial position is known but the position of the object to be moved is unknown. The solver used in POMDP-1 was evaluated with different fixed amounts of time for computing action policies. Figure 5 summarizes the results; each point is the average of 1000 trials, and we set (for ease of interpretation) each room to have four cells. The brown-colored plots in Figure 5 represent the ability to successfully achieve the assigned goal (y-axis on the left), as a function of the number of cells in the domain. The blue-colored plots show the number of actions executed before termination. For the plots corresponding to POMDP-1, the number of actions the robot is allowed to execute before it has to terminate is set to 50. We note that REBA significantly improves the robot's ability to achieve the assigned goal in comparison with POMDP-1. As the number of cells (i.e., size of the domain) increases, it becomes computationally difficult to generate good policies with POMDP-1. The robot needs a greater number of actions to achieve the goal and there is a loss in accuracy if the limit on the number of actions the robot can execute before termination is reduced. While using POMDP-1, any incorrect observations (e.g., incorrect sightings of objects) significantly impacts the ability to complete the trials. REBA, on the other hand, directs the robot's attention to relevant regions of the domain (e.g., specific rooms), and it is thus able to recover from errors and operate efficiently.

Next, we evaluated the time taken by REBA to generate a plan as the size of the domain increases. We characterize domain size based on the number of rooms and the number of objects in the domain. We conducted three sets of experiments in which the robot reasons with: (1) all available knowledge of domain objects and rooms; (2) only knowledge relevant to the assigned goal—e.g., if the robot knows an object's default location, it need not reason about other objects and rooms in the domain to locate this object; and (3) relevant knowledge and knowledge of an



(a) Using all knowledge (b) Using relevant knowledge (c) Using some knowledge

Figure 6: Planning time as a function of the number of rooms and the number of objects in the domain—REBA only uses relevant knowledge for reasoning, and is thus able to scale to larger number of rooms and objects.

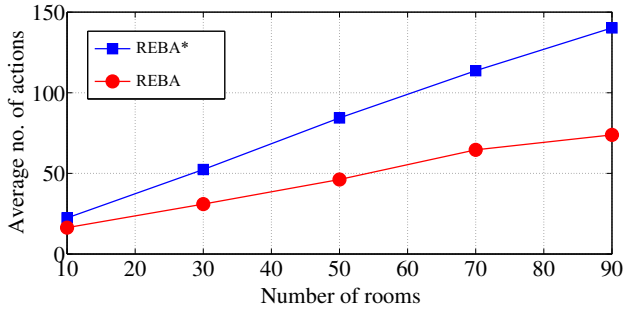


Figure 7: Effect of using default knowledge—principled representation of defaults significantly reduces the number of actions (and thus time) for achieving assigned goal.

additional 20% of randomly selected domain objects and rooms. Figures 6(a)-6(c) summarize these results. We observe that using just the knowledge relevant to the goal to be accomplished significantly reduces the planning time. REBA supports the identification of such knowledge based on the refinement and zooming operations described in Section 7. As a result, robots equipped with REBA will be able to generate appropriate plans for domains with a large number of rooms and objects. Furthermore, if we only use a probabilistic approach (POMDP-1), it soon becomes computationally intractable to generate a plan for domains with many objects and rooms. These results are not shown in Figure 6, but they are documented in prior papers evaluating just the probabilistic component of the proposed architecture (Sridharan, Wyatt, & Dearden, 2010; Zhang et al., 2013).

To evaluate hypothesis H3, i.e., to evaluate the effect of our representation and use of default knowledge on reliability and computational efficiency of decision making, we first conducted trials in which REBA was compared with REBA*, a version that does not include any default knowledge, e.g., when the robot is asked to fetch a textbook, there is no prior knowledge regarding the location of textbooks, and the robot explores the closest location first. Figure 7 summarizes the average

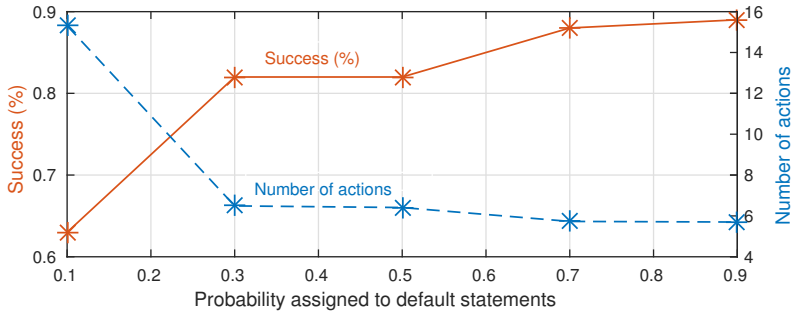


Figure 8: Ability to achieve goals, and number of actions executed, using only POMDPs, when different probability values are assigned to default statements and the ground truth locations of objects perfectly matches the default locations. The number of actions decreases and success (%) increases as the probability value increases.

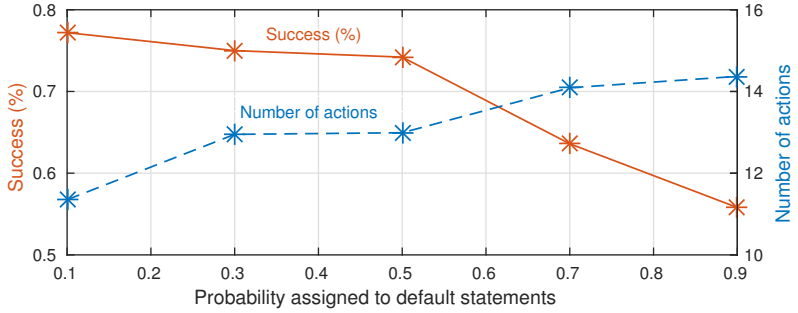


Figure 9: Ability to achieve goals, and number of actions executed, using only POMDPs, when different probability values are assigned to default statements and the ground truth locations of objects never matches the default locations. The number of actions increases and success (%) decreases as the probability value increases.

number of actions executed per trial as a function of the number of rooms in the domain—each sample point in this figure is the average of 10000 trials. The goal in each trial is (as before) to move a specific object to a specific place. We observe that our (proposed) representation and use of default knowledge significantly reduces the number of actions (and thus time) required to achieve the assigned goal.

Next REBA was compared with POMDP-2, a version of POMDP-1 that assigns specific probability values to default knowledge (e.g., "textbooks are in the library with probability 0.9") and suitably revises the initial belief state. The goal (once again) was to find and move objects to specific locations, and we measured the ability to successfully achieve the assigned goal and the number of actions executed before termination. Figures 8-9 summarize the corresponding results under two extreme cases representing a perfect match (mismatch) between the default locations and ground truth locations of objects. In Figure 8, the ground truth locations of target objects (unknown to the robot) match the default locations of the objects, i.e., there are no exceptions to the default statements. We observe that as the probability assigned to the default statement increases, the number of

actions executed by the robot decreases and the fraction of trials completed successfully increases. However, for larger values along the x-axis, the difference in the robot's performance for two different values of the probability (assigned to defaults) is not that significant. In Figure 8, the ground truth locations of the target objects never match the default locations of the objects, i.e., unknown to the robot, all trials correspond to exceptions to the default knowledge. In this case, the robot executes many more actions before termination and succeeds in a smaller fraction of trials as the probability value assigned to default statements increases. We also repeated these experimental trials after varying the extent to which the ground truth locations of objects matched their default locations. We noticed that when the probability assigned to default statements accurately reflects the ground truth, the number of trials in which the robot successfully achieves the goal increases and approaches the performance obtained with REBA. However, recall that computing the probabilities of default statements accurately takes a lot of time and effort. Also, these probabilities may change over time and the robot's ability to achieve the assigned goals may be sensitive to these changes, making it difficult to predict the robot's behavior with confidence. In addition, it is all the more challenging to accurately represent and efficiently use probabilistic information about prioritized defaults (e.g., Example 2). In general, we observed that the effect of assigning a probability value to defaults is arbitrary depending on factors such as (a) the numerical value chosen; and (b) the degree of match between ground truth and the default information. For instance, *if a large probability is assigned to the default that books are typically in the library, but the book the robot has to move is an exception to the default (e.g., a cookbook), it takes significantly longer for POMDP-2 to recover from the initial belief.* REBA, on the other hand, supports elegant representation of, and reasoning with, defaults and exceptions to these defaults.

Robot Experiments: In addition to the trials in simulated domains, we implemented and evaluated REBA with POMDP-1 on physical robots using the Robot Operating System (ROS). We conducted experimental trials with two robot platforms (see Figure 1) in variants of the domain described in Example 1. Visual object recognition is based on learned object models that consist of appearance-based and contextual visual cues (Li & Sridharan, 2013). Since, in each trial, the robot's initial location and the target object(s) are chosen randomly, it is difficult to compute a meaningful estimate of variance, and statistical significance is established through paired trials. In each paired trial, for each approach being compared (e.g., REBA or POMDP-1), the target object(s), the robot's initial location, and the location of domain objects are the same, and the robot has the same initial domain knowledge.

First, we conducted 50 trials on two floors of our Computer Science department building. This domain includes places in addition to those included in our illustrative example, e.g., Figure 1(a) shows a subset of the domain map of the third floor of the building, and Figure 1(b) shows the *Peoplebot* wheeled robot platform used in these trials. The robot is equipped with a stereo camera, laser range finder, microphone, speaker, and a laptop running Ubuntu Linux that performs all the processing. The domain maps are learned and revised by the robot using laser range finder data and the existing ROS implementation of a SLAM algorithm (Dissanayake, Newman, & Clark, 2001). This robot has a manipulator arm that can be moved to reachable 3D locations relative to the robot. However, since robot manipulation is not a focus of this work, once the robot is next to the desired object, it extends its gripper and asks for the object to be placed in it. For experimental trials on the third floor, we considered 15 rooms, which includes faculty offices, research labs, common areas and a corridor. To make it feasible to use POMDP-1 in such large domains, we used our prior

work on a hierarchical decomposition of POMDPs for visual sensing and information processing that supports automatic belief propagation across the levels of the hierarchy and model generation in each level of the hierarchy (Sridharan et al., 2010; Zhang et al., 2013). The experiments included paired trials, e.g., over 15 trials (each), POMDP-1 takes 1.64 as much time as REBA (on average) to move specific objects to specific places. For these paired trials, this 39% reduction in execution time provided by REBA is statistically significant: *p-value* $= 0.0023$ at the 95% significance level.

Consider a trial in which the robot's objective is to bring a particular textbook to the *study_corner*. The robot uses default knowledge to create a plan of abstract actions that causes the robot to move to and search for the textbook in the *main_library*. When the robot does not find this textbook in the *main_library* after searching using a suitable POMDP policy, replanning by the logician causes the robot to investigate the *aux_library*. The robot finds the desired textbook in the *aux_library* and moves it to the target location. A video of such an experimental trial can be viewed online at `http://youtu.be/8zL4R8te6wg`

To explore the applicability of REBA in different domains, we also conducted 40 experimental trials using the *Turtlebot* wheeled robot platform in Figure 1(c) in a variant of the illustrative domain in Example 1. This domain had three rooms in the Electrical Engineering department building arranged to mimic a robot operating as a robot butler, with additional objects (e.g., tables, chairs, food items etc). The robot was equipped with a Kinect (RGB-D) sensor, a laser range finder, and a laptop running Ubuntu Linux that performs all the processing. As before, the robot used the ROS implementation of a SLAM algorithm, and a hierarchical decomposition of POMDPs for POMDP-1. This robot did not have a manipulator arm—once it reached a location next to the location of the desired object, it asks for the object to be placed on it. The experiments included paired trials, e.g., in 15 paired trials, POMDP-1 takes 2.3 as much time as REBA (on average) to move specific objects to specific places—this reduction in execution time by REBA is statistically significant at the 95% significance level.

Consider a trial in which the robot's goal was to fetch a bag of crisps for a human. The robot uses default knowledge about the location of the bag of crisps (e.g., that they are usually in the $kitchen$), to create a plan of abstract actions to achieve this goal. Execution of this plan causes the robot to first move to the $kitchen$ and search for the bag of crisps. The robot finds the bag of crisps, asks for the bag to be placed on it (since it has no manipulator), and moves back to $table1$ in $lab1$ (the location of the human who wanted the crisps), only to be told that it has brought a bag of chocolates instead. The robot diagnoses the cause for this unexpected observation as human error (i.e., that it was given incorrect bag in the $kitchen$ by a human). The robot then computes and executes a plan that has it go back and fetch the correct bag (of crisps) this time. A video of this trial can be viewed online at `https://vimeo.com/136990534`

11. Conclusions and Future Work

This article described a refinement-based knowledge representation and reasoning architecture (REBA) that combines the complementary strengths of declarative programming and probabilistic graphical models. The architecture is based on tightly-coupled transition diagrams that represent domain knowledge, and the robot's abilities and beliefs, at two levels of granularity. The architecture makes the following contributions:

- Action language $\mathcal{AL}_d$ is extended to support non-Boolean fluents and non-deterministic causal laws, and is used to describe the coarse-resolution and fine-resolution transition diagrams.

- The notion of a history of a dynamic domain is extended to include default knowledge in the initial state, and a model of this history is defined. These definitions are used to define a notion of explanation of unexpected observations, and to provide an algorithm for coarse-resolution planning and diagnostics that translates history into a program of CR-Prolog, computes answer set of this program, and extracts plan and explanation (if needed) from the answer set.

- A formal definition is provided of one transition diagram being a weak refinement of another transition diagram, and a fine-resolution diagram is defined as a weak refinement of the domain's coarse-resolution transition diagram.

- A theory of observations is introduced and a formal definition is provided of one transition diagram being a strong refinement of another transition diagram. This theory of observations is combined with the weakly refined fine-resolution transition diagram to obtain a fine-resolution transition diagram that is a strong refinement of the coarse-resolution transition diagram.

- The randomization of the fine-resolution transition diagram is defined, and an approach is described for experimental collection of statistics. These statistics are used to compute the probabilities of action outcomes and observations at the fine-resolution.

- A formal definition is provided for zooming to a part of the randomized fine-resolution diagram relevant to any given coarse-resolution (abstract) transition. This definition is used to automate the zoom operation and implement each abstract action in the coarse-resolution plan.

- An algorithm is provided for automatically constructing data structures appropriate for the fine-resolution probabilistic implementation of any given abstract action. This algorithm uses probabilistic models of the uncertainty in sensing and actuation, and the zoomed part of the fine-resolution transition diagram. The outcomes of the fine-resolution execution update the coarse-resolution history for subsequent reasoning.

- Finally, and possibly one of the major contributions, is a general methodology for the design of software components of robots that are re-taskable and robust. This design methodology is based on Dijkstra's view of step-wise refinement of the specification of a program.

In this article, the domain representation for coarse-resolution non-monotonic logical reasoning is translated to a CR-Prolog program, and the representation for probabilistic reasoning is translated to a POMDP. The key advantages of using REBA are:

- It substantially simplifies the design process and increases confidence in the correctness of the robot's behavior. In particular:

  - Step-wise refinement leads to clear separation of concerns and supports early testing of the different components of the architecture.
  - The formal (i.e., mathematical) descriptions of the different components, and of the flow of control and information between the components, helps characterize the robot's behavior accurately and prove correctness of the algorithms.

  - The domain-independent representations of part of the commonsense knowledge used by the robot, e.g., theory of observations, weak refinement and strong refinement, allow for the reuse of these representations on other robots and application domains.
  - There is a single framework for inference, planning, diagnostics, and for a quantifiable trade off between accuracy and computational efficiency in the presence of probabilistic models of uncertainty in sensing and actuation.

- It significantly improves the computational efficiency and reliability of planning and execution of the robot's actions. In particular:

  - The robot reasons reliably and efficiently with hierarchically-organized knowledge and beliefs.
  - Experimental results in simulation and on physical robots in different domains indicate the ability to reason at the sensorimotor level and the cognitive level with violation of defaults, noisy observations and unreliable actions.
  - The tight coupling between representation and reasoning at different resolutions, established by formally defining concepts such as refinement and zooming, supports precise reasoning while still demonstrating the potential to scale to complex domains.

REBA opens up many directions for further research, some of which relax the constraints imposed in the design of our current architecture. First, we will further explore the tight coupling between the transition diagrams, and between logical and probabilistic reasoning, in dynamic domains. We have, for instance, explored different resolutions for reasoning probabilistically (Colaco & Sridharan, 2015), and investigated the inference, planning and diagnostics capabilities of architectures that reason at different resolutions (Zhang et al., 2015). However, we have so far not explored non-stationary domains, a limiting constraint that we seek to relax in future work. Second, our architecture has so far focused on a single robot, although we have instantiated the architecture in different domains. Another direction of further research is to extend the architecture to enable collaboration between a team of robots working towards a shared goal. It is theoretically possible to use our current architecture on multiple robots, but it will open up challenging questions and choices regarding communication (between robots) and propagation of beliefs held by different members of the team. Third, REBA has focused on representation and reasoning with incomplete knowledge, but a robot collaborating with humans in a dynamic domain also needs to be able to revise and augment its existing knowledge. Preliminary work in this direction, e.g., based on exploiting the complementary strengths of relational reinforcement learning, active (inductive) learning, and reasoning with commonsense knowledge, has provided promising results (Sridharan & Meadows, 2017a, 2017b, 2018), and we seek to further explore this direction of work in the future. The long-term objective is to better understand the coupling between non-monotonic logical reasoning and probabilistic reasoning, and to use this understanding to develop architectures that enable robots to assist humans in complex domains.

Acknowledgments

This work was supported in part by the U.S. Office of Naval Research Science of Autonomy Awards N00014-13-1-0766 (Mohan Sridharan, Shiqi Zhang) and N00014-17-1-2434 (Mohan Sridharan),

the Asian Office of Aerospace Research and Development award FA2386-16-1-4071 (Mohan Sridharan), and the EC-funded Strands project FP7-IST-600623 (Jeremy Wyatt). Opinions and conclusions in this article are those of the authors.

Appendix A. CR-Prolog and Proof of Proposition 1

In this section, we describe the syntax and semantics of CR-Prolog, and prove Proposition 1.

A.1 Syntax and Semantics of CR-Prolog

A program of CR-Prolog consists of regular rules of ASP and rules of the form:

$$l \stackrel{+}{\leftarrow} body \tag{71}$$

referred to as consistency-restoring (CR) rules. Intuitively, Statement 71 says that if the reasoner is in a state in which program believes the rule's body, it may possibly believe the rule's head; however, this possibility may be used only if there is no way to obtain a consistent set of beliefs by using only the regular rules of the program. This intuition is formalized by the following terminology.

Let R be a set of CR-rules. By $\alpha(R)$ we denote the set of regular ASP rules obtained by replacing $\stackrel{+}{\leftarrow}$ by $\leftarrow$ in rules of R. Let Π be a program of CR-Prolog. By Π^{reg} we denote the set of all regular rules of Π. A cardinality-minimal set of CR-rules of Π such that ASP program $\Pi(R) =_{def} \Pi^{reg} \cup \alpha(R)$ is consistent, i.e. has an answer set, is called an abductive support of Π. A is an answer set of Π if it is an answer set of program $\Pi(R)$ for some abductive support R of Π. Note that this is a special case of a more general definition from (Gelfond & Kahl, 2014) where minimality of R is determined by an arbitrary preference relation. Finally, recall that if σ is a state of a transition diagram, σ^{nd} is obtained from σ by removing all atoms formed by defined fluents.

A.2 Proof of Proposition 1

Proposition 1 in Section 5.2 states that:

If $\mathcal{D}$ is a well-founded system description and $\mathcal{H}$ is its recorded history, then every sequence induced by an answer set of $\Pi(\mathcal{D}, \mathcal{H})$ is a model of $\mathcal{H}$.

To prove this proposition, we begin with some notation. Let σ be a collection of literals, $\alpha_k = \langle a_0, \ldots, a_k \rangle$ be a (possibly empty) sequence of actions, $occurs(\alpha_k) = \{occurs(a_i, i) : 0 \leq i \leq k\}$ and $\Pi(\mathcal{D}, \sigma, \alpha_k) =_{def} \Pi^{k+1}(\mathcal{D}) \cup val(\sigma, 0) \cup occurs(\alpha_k)$.

Lemma 1. Let A be an answer set of $\Pi(\mathcal{D}, \mathcal{H})$. Then there exists a state σ_0 of $\tau(\mathcal{D})$ and a sequence of actions $\alpha_k = \langle a_0, \ldots, a_k \rangle$ such that the set B obtained from A by removing literals formed by obs, hpd, $prefer$, ab and $defined\_by\_default$ is an answer set of $\Pi(\mathcal{D}, \sigma_0, \alpha_k)$. ■

Proof of Lemma 1. Let A be an answer set of $\Pi =_{def} \Pi(\mathcal{D}, \mathcal{H})$. By the CR-Prolog definition of answer sets:

(1) A is an answer set of an ASP program $\Pi(R) = \Pi^{reg} \cup \alpha(R)$ for some abductive support R of Π.

Clearly, $\alpha(R)$ is (a possibly empty) collection of rules of the form: $ab(d(\bar{x})) \leftarrow val(body(d(\bar{x})), 0)$. We will prove the existence of σ_0 and α_k by construction. Let:

(2) $\sigma_0 = \{f(\bar{x}) = y : val(f(\bar{x}), y, 0) \in A$ or $f(\bar{x}) = y \in A\}$

We will show that σ_0 is a state of $\tau(\mathcal{D})$, i.e., that:

(a) σ_0 is an interpretation, i.e. for every $f(\bar{x})$ there is unique y such that $f(\bar{x}) = y \in \sigma_0$; and

(b) σ_0 is the unique answer set of program $\Pi_c(\mathcal{D}) \cup \sigma_0^{nd}$.

To show (a) consider first an arbitrary basic fluent $f(\bar{x})$. Based on (2), for every y, $f(\bar{x}) = y \in \sigma_0$ iff $val(f(\bar{x}), y, 0) \in A$. Hence, we need to show that there is y such that $val(f(\bar{x}), y, 0) \in A$. There are two cases depending on whether or not the body of Statement 37 is satisfied by A. In the former case the existence of y such that $val(f, y, 0) \in A$ is guaranteed by Statement 31; otherwise it follows from Statement 38. If $f(\bar{x})$ is static, we have that there is y such that $f(\bar{x}) = y \in A$ by Statement 19 of $\Pi(\mathcal{D})$. If $f(\bar{x})$ is a defined fluent, its boolean value is included in A by the axioms for the defined fluents (Statement 17). Uniqueness of the value assigned to $f(\bar{x})$ follows from Statement 22 and consistency of A.

To show (b) it suffices to notice that since A satisfies rules such as Statement 16 in $\Pi(R)$, σ_0 satisfies rules such as Statement 8 in $\Pi_c(\mathcal{D})$, and hence $\Pi_c(\mathcal{D}) \cup \sigma_0^{nd}$ has an answer set. By (a), σ_0 is complete and consistent and hence, by the definition of well-foundedness, this answer set is unique.

Next, let:

(3) $\alpha_k = \langle a_0, \ldots, a_k \rangle$ where $a_i = \{e_j : occurs(e_j, i) \in A\}$.

and let S_0 be a set of literals formed by obs and hpd. Note that S_0 is a splitting set of program $\Pi(R)$. From (1) and the splitting set theorem (Balduccini, 2009) we have:

(4) A_0, obtained from A by removing literals formed by obs and hpd, is an answer set of program $\Pi_0(R)$ obtained from $\Pi(R)$ by:

- removing all atoms formed by obs and hpd;
- removing all rules whose bodies contain atoms formed of $obs(*, *, *)$[12] or $hpd(*, *)$ that are not in A; and
- removing all occurrence of atoms $obs(*, *, *)$ or $hpd(*, *)$ from the remaining rules.

Note that the only rules changed by this transformation belong to the encoding of $\mathcal{H}$.

Next, if $\Pi_0^{\mathcal{H}}(R)$ denotes the program obtained from $\Pi_0(R)$ by removing all atoms formed by $occurs$ and all rules of $\Pi^{k+1}(\mathcal{D})$, then from (4) and the definition of $\Pi_0^{\mathcal{H}}(R)$ we have that:

(5) A_0 is an answer set of $\Pi_0(R) = \Pi^{k+1}(\mathcal{D}) \cup \Pi_0^{\mathcal{H}}(R) \cup \{occurs(a, i) : occurs(a, i) \in A\}$.

Now let S_1 be the set of atoms formed by statics, $prefer$, ab, $defined\_by\_default$, and $val(*, *, 0)$. It is not difficult to check that S_1 is a splitting set of $\Pi_0(R)$. It divides the program into two parts:

- Program Bot consisting of $\Pi_0^{\mathcal{H}}(R)$ combined with the set $Zero$ of instances of axioms encoded in Statements 16, 17, 19 and 22 from $\Pi^{k+1}(\mathcal{D})$ with the time-step variable set to 0.
- Program $Top = (\Pi^{k+1}(\mathcal{D}) \setminus Zero) \cup \{occurs(a, i) : occurs(a, i) \in A\}$

12. Recall that the "*" denotes a wild-card character.

So, by the splitting set theorem, we now have:

(6) A_0 is an answer set of program $B_0 \cup Top$ where B_0 is an answer set of Bot.

Next, observe that:

(7) B_0 can be partitioned into B_1 and B_2, with B_1 consisting of the atoms of A_0 formed by $prefer$, ab, and $defined\_by\_default$, and B_2 consisting of the atoms of A_0 formed by statics and $val(*, *, 0)$.

Using the definition of answer sets for ASP program, it can be proved that for any two programs Π_1 and Π_2 whose signatures Σ_1 and Σ_2 are disjoint, X is an answer set of a program $\Pi_1 \cup \Pi_2$ iff $X|_{\Sigma_1}$ and $X|_{\Sigma_2}$ are answer sets of Π_1 and Π_2 respectively. Hence, we have that:

(8) $B = A_0 \setminus B_1$ is an answer set of $B_2 \cup Top$.

From (4), (7) and (8) above, we have that B is obtained from A by removing literals formed by obs, hpd, $prefer$, ab and $defined\_by\_default$.

To show that B is an answer set of $\Pi(\mathcal{D}, \sigma_0, \alpha_k)$ we first demonstrate that:

(9) B is an answer set of $\Pi_1^* =_{def} B_2 \cup \Pi^{k+1}(\mathcal{D}) \cup \{occurs(a, i) : occurs(a, i) \in A\}$.

By construction, we have that:

(10) $\Pi_1^* = B_2 \cup Top \cup Zero$.

To prove (9), we will show that B is an answer set of the reduct, $(\Pi_1^*)^B$ of Π_1^* with respect to B (note that this is the definition of answer set). Based on the definition of the reduct and (10), we have:

(11) $(\Pi_1^*)^B = B_2 \cup Zero^B \cup Top^B$.

From (8) and the definition of answer set, we have that B is a $\subseteq$-minimal set satisfying $B_2 \cup Top^B$. Then, based on (6)-(8), we have that B_2 (and hence B) also satisfies $Zero^B$, and thus (9) holds.

Then, based on (2), (6) and (7), we have that $val(\sigma_0, 0) = B_2$ which, together with (9), implies that:

(12) B is an answer set of $\Pi^{k+1}(\mathcal{D}, \sigma_0, \alpha_k)$.

This completes the proof of Lemma 1. □

Lemma 2. Let B be an answer set of $\Pi(\mathcal{D}, \sigma_0, \alpha_k)$, B_k be obtained from B by removing all literals containing time-step $k + 1$, and $\Pi_k^{k+1}(\mathcal{D}, \sigma_k, a_k)$ be $\Pi(\mathcal{D}, \sigma_k, a_k)$ with time-steps 0 and 1 replaced by k and $k + 1$ respectively. Then:

- B_k is an answer set of $\Pi(\mathcal{D}, \sigma_0, \alpha_{k-1})$.
- $B = B_k \cup U$ where U is an answer set of $\Pi_k^{k+1}(\mathcal{D}, \sigma_k, a_k)$.

■

Proof of Lemma 2. Let S be a set of literals of $\Pi(\mathcal{D}, \sigma_0, \alpha_k)$ not containing time step $k + 1$. It is easy to check that S is a splitting set of this program, which divides it into two parts:

(1) $Bot = \Pi(\mathcal{D}, \sigma_0, \alpha_{k-1})$ and $Top = \Pi(\mathcal{D}, \sigma_0, \alpha_k) \setminus Bot$.

By the splitting set theorem and definition of B_k, we have:

(2) B_k is an answer set of $Bot = \Pi(\mathcal{D}, \sigma_0, \alpha_{k-1})$.

and:

(3) B is an answer set of the program $B_k \cup Top$.

By definition, $\sigma_k = \{f(\bar{x}) = y : val(f(\bar{x}), y, k) \in B\} \cup \{f(\bar{x}) = y : f(\bar{x}) = y \in B\}$, and hence, $val(\sigma_k, k)$ is a subset of B and of B_k. Thus, we have:

(4) $B_k \cup Top = B_k \cup val(\sigma, k) \cup Top = B_k \cup \Pi_k^{k+1}(\mathcal{D}, \sigma_k, a_k)$.

Now let:

(5) $B_k = B'_k \cup B''_k$

where B'_k consists of atoms of B_k containing time-steps smaller than k and $B''_k = B_k \setminus B'_k$. Note that B''_k consists of atoms of B_k formed by statics and of those containing time-step k. From (4), (5), and the definition of σ_k, we then have:

(6) $B_k \cup Top = B'_k \cup \Pi_k^{k+1}(\mathcal{D}, \sigma_k, a_k)$.

Based on (3) and (6), we have:

(7) B is an answer set of $B'_k \cup \Pi_k^{k+1}(\mathcal{D}, \sigma_k, a_k)$.

Since, by construction, the signatures of B'_k and $\Pi_k^{k+1}(\mathcal{D}, \sigma_k, a_k)$ are disjoint, from (7), we have:

(8) $B = B_k \cup U$ where U is an answer set of $\Pi_k^{k+1}(\mathcal{D}, \sigma_k, a_k)$.

This completes the proof of Lemma 2. □

Proof of Proposition 1. Let $\mathcal{D}$ and $\mathcal{H}$ be as in the proposition, A be an answer set of CR-Prolog program $\Pi(\mathcal{D}, \mathcal{H})$, and $M = \langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_n, a_n, \sigma_{n+1} \rangle$ be a sequence induced by A. We will show that M is a model of $\mathcal{H}$, i.e. M is a path of transition diagram $\tau(\mathcal{D})$ (definition 6).

The proposition will be an immediate consequence of a more general statement:

(1) *for every* $0 \leq k \leq n+1$ $M_k = \langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_k \rangle$ *is a path in* $\tau(\mathcal{D})$.

Before proceeding with inductive proof of (1), let us notice that, by Lemma 1:

(2) M is induced by an answer set B of an ASP program $\Pi(\mathcal{D}, \sigma_0, \alpha_n)$ where σ_0 is a state and B is obtained from A by removing atoms formed by obs, hpd, $prefer$, ab and $defined\_by\_default$.

We use induction on k. The base case is: $k = 0$, i.e. $M_k = \langle \sigma_0 \rangle$. Then (1) follows immediately from (2).

Nex, consider the inductive step: let $k > 0$ and $M_k = \langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_{k-1}, a_{k-1}, \sigma_k \rangle$. By inductive hypothesis:

(3) $M_k = \langle \sigma_0, a_0, \sigma_1, \ldots, \sigma_{k-1} \rangle$ is a path in $\tau(\mathcal{D})$.

We need to show that $L = \langle \sigma_{k-1}, a_{k-1}, \sigma_k \rangle$ is a transition of $\tau(\mathcal{D})$. By Lemma 2, we have:

(4) L is induced by an answer set U_0 of $\Pi_k^{k+1}(\mathcal{D}, \sigma_k, a_k)$.

Let U be obtained from U_0 by replacing time-steps k and $k+1$ by 0 and 1 respectively. From (4) and the definition of $\Pi_k^{k+1}(\mathcal{D}, \sigma_k, a_k)$, we have that:

(5) L is induced by an answer set U of $\Pi(\mathcal{D}, \sigma_k, a_k)$.

From (3) we have that:

(6) σ_k is a state.

To prove that σ_{k+1} is a state we first show that σ_{k+1} is an interpretation, i.e. for every $f(\bar{x})$ there is unique y such that $val(f(\bar{x}), y, 1) \in U$. From (5) and (6), we have that, for every $f(\bar{x})$ there is unique y_1 such that $val(f(\bar{x}), y, 0) \in U$. If the body of the inertia axiom for $f(\bar{x})$ is satisfied by U then $val(f(\bar{x}), y_1, 1) \in U$. Otherwise, the inertia axiom is defeated by Statement 22 and hence $val(f(\bar{x}), y_2, 1) \in U$. Thus, we have that:

(7) σ_{k+1} is an interpretation.

The last step is to show that:

(8) σ_{k+1} is the unique answer set of program $\Pi_c(\mathcal{D}) \cup \sigma_{k+1}^{nd}$.

To do that it suffices to notice that, since U satisfies rules such as Statements 17 and 16 in $\Pi(\mathcal{D}, \sigma_k, a_k)$, σ_{k+1} satisfies rules such as Statements 8 and 9 in $\Pi_c(\mathcal{D})$, and hence $\Pi_c(\mathcal{D}) \cup \sigma_{k+1}^{nd}$ has an answer set. Based on (7), σ_{k+1} is complete and consistent and hence, by the definition of well-foundedness, this answer set is unique; this proves (8). Then, based on (7) and (8), and the definition of state, we have:

(9) σ_{k+1} is a state.

Thus, based on (5), (6), (9) and Definition 4, we have that:

(10) L is a transition.

Next, based on (3), the definition of L, and (10):

(11) M_k is a path in $\tau(\mathcal{D})$.

This completes the proof of statement (1). Based on the definition of M_k, $M = M_{n+1}$, and based on (1), M is a path in $\tau(\mathcal{D})$. Since M is induced by A, based on Definition 6, it is a model of $\mathcal{H}$. This completes the proof of Proposition 1. □

Appendix B. Proof of Proposition 2

In this section, we examine Proposition 2, which states that:

Let $\mathcal{P} = (\mathcal{D}, \mathcal{H}, h, G)$ be a planning problem with a well-founded, deterministic system description $\mathcal{D}$. A sequence $\langle a_0, \ldots, a_{k-1}\rangle$ where $k < h$ is a solution of $\mathcal{P}$ iff there is an answer set A of $Plan$ such that:

1. *For any $n < i \leq n + k$, $occurs(a_i, i-1) \in A$,*
2. *A contains no other atoms of the form $occur(*, i)$ with $i \geq n$.*

We begin by introducing some notation. Let Π be an arbitrary CR-Prolog program and R be a collection of CR-rules from Π. Similar to the terminology in (Gelfond & Kahl, 2014), we use Π^{reg} to denote the collection of regular rules of Π and $\alpha(R)$ to denote the set of regular ASP rules obtained by replacing $\stackrel{+}{\leftarrow}$ by $\leftarrow$ in CR-rules of R. For completeness, recall that for any program Π, we have $\Pi(R) = \Pi^{reg} \cup \alpha(R)$. Also recall, from Section 5.3, that:

$$
\begin{aligned}
&Plan = Diag \cup Classical\_plan \cup \{DPC\}\\
&Diag =_{def} \Pi^n(\mathcal{D}, \mathcal{H})\\
&Classical\_plan = \Pi^{[n..n+h]}(\mathcal{D}) \cup goal(I) \leftarrow val(f(\bar{x}), y, I) \cup PM\\
&\leftarrow Y = count\{X : ab(X)\}, Y > m \qquad \% \text{ DPC}
\end{aligned}
$$

where n is the current step of $\mathcal{H}$, m is the size of the abductive support of $Diag$, and PM is the planning module. We will also need the following Lemma to prove Proposition 2.

Lemma 3. For any set R of CR-rules of $Diag$, A is an answer set of ASP program $Plan(R)$ iff $A = A_0 \cup B_0$ where A_0 is an answer set of $Diag(R)$ satisfying DPC and B_0 is an answer set of $Shifted\_plan =_{def} \{val(f(\bar{x}), y, n) : val(f(\bar{x}), y, n) \in A_0\} \cup Classical\_plan$. ■

Proof of Lemma 3. Let S_0 be the set of literals of $Plan(R)$ not containing atoms with time steps greater than n or atoms of the form $occurs(*, n)$ and $hpd(*, n)$. It is easy to check that S_0 is a splitting set of $Plan(R)$ which splits the program into two parts, $Bot = Diag(R) \cup \{DPC\}$ and $Top = Classical\_plan$. By the splitting set theorem, A is an answer set of $Plan(R)$ iff A is an answer set of $A_0 \cup Top$ where A_0 is an answer set of Bot. Clearly, $A_0 \cup Top = A_0 \cup Shifted\_plan$. Since A_0 is a collection of atoms, from the definition of answer set we have that $A = A_0 \cup B_0$ where B_0 is an answer set of $Shifted\_plan$. □

Next, we turn to proving Proposition 2.

Proof of Proposition 2. Let $\mathcal{P}$ and $Plan$ be as in the proposition, σ be a state and $\langle a_0, \ldots, a_{k-1}\rangle$ with $k < h$ be a sequence of actions of $\mathcal{D}$.

Based on Definition 8:

(1) $\langle a_0, \ldots, a_{k-1}\rangle$ is a solution of $\mathcal{P}$ iff:

  (a) there is a state σ that is the current state of some model M of $\mathcal{H}$; and

(b) $\langle a_0, \ldots, a_{k-1} \rangle$ is a solution of classical planning problem $\mathcal{P}_c = (\mathcal{D}, \sigma, G)$ with horizon h.

Based on Definition 6 and the well-foundedness of $\mathcal{D}$, Statement (1)(a) holds iff:

(2) M is induced by some answer set A_0 of $Diag$, n is the current step of history from $\mathcal{P}$, and:

$$\sigma = \{f(\bar{x}) = y : val(f(\bar{x}), y, n) \in A_0\}$$

By the CR-Prolog definition of answer sets, Statement (2) holds iff:

(3) A_0 is an answer set of $Diag(R)$ for some abductive support R of $Diag$ and $\sigma = \{f(\bar{x}) = y : val(f(\bar{x}), y, n) \in A_0\}$ (since A_0 is an answer set of $Diag$ it satisfies DPC).

Based on Proposition 9.1.1 from (Gelfond & Kahl, 2014), Statement (1)(b) holds iff:

(4) There is an answer set S of ASP program $plan(\mathcal{P}_c, h)$ such that:

(a) For any $0 < i \leq k$, $occurs(a_i, i-1) \in S$; and

(b) S contains no other atoms formed by $occurs$.

Consider an ASP program:

$$Shifted\_plan =_{def} \{val(f(\bar{x}), y, n) : val(f(\bar{x}), y, n) \in A_0\} \cup Classical\_plan$$

It is easy to see that this program differs from $plan(\mathcal{P}_c, h)$ only in the domain of its time-step variables. In the former case, such variables range over $[n, n+h]$ while in the latter the range is $[0, h]$. The programs are isomorphic and hence Statement (4) holds for S iff:

(5) B_0 obtained from S by increasing all occurrences of time steps in atoms from S by $n+1$ is an answer set of $Shifted\_plan$. Also:

(a) For any $n < i \leq n+k$, $occurs(a_i, i-1) \in B_0$; and

(b) B_0 contains no other atoms of the form $occurs(*, i)$ where $i \geq n$.

Now we have that:

(6) Statement (1) is true iff Statement (3) and Statement (5) are true.

Let $A = A_0 \cup B_0$. Then, based on Lemma 3, we have:

(7) Statements (3) and (5) are true iff A is an answer set of $Plan(R)$.

Based on (7), we have:

(8) Statement (1) is true iff A is an answer set of $Plan(R)$.

However, since every answer set of $Plan$ must satisfy DPC, $Plan(R)$ has an answer set iff R is an abductive support of $Plan$. Hence:

(9) A is an answer set of $Plan(R)$ iff A is an answer set of $Plan$.

From the construction of A, Statement (5), and the fact that A_0 contains no atoms of the form $occurs(*, i)$ where $i \geq n$, we have that A satisfies the conditions of the proposition. This completes the proof of Proposition 2. □

Appendix C. Proof of Proposition 3

In this section, we prove Proposition 3, which states that:

Let $\mathcal{D}_H$ and $\mathcal{D}_{L,nobs}$ be coarse and fine resolution system descriptions from our running example. Then τ_L is a weak refinement of τ_H.

Proof of Proposition 3. Definitions in $\mathcal{D}_H$ and $\mathcal{D}_{L,nobs}$ contain no dependency between defined domain functions and their negations. Both system descriptions are therefore weakly-acyclic and thus well-founded, which justifies the use of the following property in the proof. Let $\mathcal{D}$ over signature Σ be a well-founded system description defining transition diagram τ. Then, an interpretation δ of Σ is a state of τ_H iff:

- δ satisfies constraints of $\mathcal{D}$; and
- For every defined fluent f of Σ, $f(\bar{u}) \in \delta$ iff there is a rule from the definition of $f(\bar{u})$ whose body is satisfied by the interpretation δ.

For readability, we also repeat Definition 9 of weak refinement of a transition diagram. A transition diagram $\tau_{L,nobs}$ over $\Sigma_{L,nobs}$ is called a *weak refinement* of τ_H if:

1. For every state $\sigma^\diamond$ of $\tau_{L,nobs}$, the collection $\sigma^\diamond|_{\Sigma_H}$ of atoms of $\sigma^\diamond$ formed by symbols from Σ_H is a state of τ_H.
2. For every state σ of τ_H, there is a state $\sigma^\diamond$ of $\tau_{L,nobs}$ such that $\sigma^\diamond$ is an extension of σ.
3. For every transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle$ of τ_H, if $\sigma_1^\diamond$ and $\sigma_2^\diamond$ are extensions of σ_1 and σ_2 respectively, then there is a path P in $\tau_{L,nobs}$ from $\sigma_1^\diamond$ to $\sigma_2^\diamond$ such that:
   - actions of P are concrete, i.e., directly executable by robots; and
   - P is *pertinent* to T, i.e., all states of P are extensions of σ_1 or σ_2.

To prove the first clause of Definition 9, let $\sigma^\diamond$ and $\sigma = \sigma^\diamond|_{\Sigma_H}$ be as in the first clause. To prove that σ is a state of $\mathcal{D}_H$, we show that it satisfies the clauses of the property above. We start with the constraint in Statement 42(a) for a particular object ob:

$$loc(ob) = P \ \textbf{if} \ \ loc(rob_1) = P,\ in\_hand(rob_1, ob)$$

Let:

(i) $(loc(rob_1) = P) \in \sigma$; and

(ii) $in\_hand(rob_1, ob) \in \sigma$.

To show that $(loc(ob) = P) \in \sigma$ let c_1 be the value of $loc^*(rob_1)$ in $\sigma^{diamond}$, i.e:

(iii) $(loc^*(rob_1) = c_1) \in \sigma^\diamond$

Based on the bridge axiom in Statement 48(a):

$$loc(rob_1) = P \ \textbf{if} \ \ loc^*(rob_1) = C, component(C, P)$$

of $\mathcal{D}_{L,nobs}$ and conditions (i) and (iii), we have:

(iv) $component(c_1, P)$

Suppose this is not the case. Then, based on the definition of $place^*$, there is some place P_2 in the domain such that $component(c_1, P_2) \in \sigma^\diamond$. This statement, together with (iii) and the bridge axiom in Statement 48(a) will entail $(loc(rob) = P_2)$, which contradicts condition (i) above.

Next, the state constraint in Statement 45(a):

$$loc^*(ob) = P \ \textbf{if} \ loc^*(rob_1) = P, \ in\_hand(rob_1, ob)$$

of $\mathcal{D}_{L,nobs}$, together with (ii) and (iii) imply:

(v) $(loc^*(ob) = c_1) \in \sigma^\diamond$

Then, the bridge axiom in Statement 48(a), together with (iv) and (v) imply that $(loc(ob) = P) \in \sigma$ and hence σ satisfies the first constraint of $\mathcal{D}_H$.

Next, consider the definition of the static $next\_to(P_1, P_2)$ in $\mathcal{D}_H$. It our example domain with four rooms (see Figure 3(b)), $\mathcal{D}_H$ consists of statements such as:

$$next\_to(r_1, r_2)$$
$$next\_to(r_1, r_3)$$
$$next\_to(r_2, r_4)$$
$$next\_to(r_3, r_4)$$

and the constraint:

$$next\_to(P_1, P_2) \ \textbf{if} \ next\_to(P_2, P_1)$$

In the fine-resolution system description $\mathcal{D}_{L,nobs}$, these statements are replaced by a collection of statements of the form $next\_to^*(c_i, c_j)$, state constraint in Statement 45(b):

$$next\_to^*(C_1, C_2) \ \textbf{if} \ next\_to^*(C_2, C_1)$$

and a bridge axiom as described by Statement 48(b):

$$next\_to(P_1, P_2) \ \textbf{if} \ next\_to^*(C_1, C_2), component(C_1, P_1), component(C_1, P_2)$$

The last axiom implies that $next\_to(r_i, r_j) \in \sigma$ iff $\sigma^\diamond$ indicates that there are two adjacent cells in the domain such that one of them is in r_i and another is in r_j. This is the situation in our example domain, as shown in Figure 3(b). This concludes the proof of the first clause of Definition 9.

To prove clause 2 of Definition 9, consider a state σ of τ_H and expand it to a state $\sigma^\diamond$ of $\tau_{L,nobs}$, and show that $\sigma^\diamond$ is a state of $\tau_{L,obs}$. We do so by construction by interpreting the fine-resolution domain functions of $\mathcal{D}_{L,nobs}$ such that it satisfies the bridge axioms, constraints and definitions of $\mathcal{D}_{L,nobs}$. In our example domain, it is sufficient to map $loc^*(thing)$ to a cell c of room r such that:

- if $loc^*(th) = c$ and $component(c, rm)$ are in $\sigma^\diamond$ then $loc(th) = rm \in \sigma$
- if $in\_hand(rob_1, ob) \in \sigma$ then the same cell is assigned to rob_1 and ob.

The definition of static $next\_to^*$ is the same for every state. It is symmetric and satisfies Statement 48(b) describing the bridge axiom for $next\_to$. In other words, all state constraints and definitions of $\mathcal{D}_{L,nobs}$ are satisfied by $\sigma^\diamond$, which is thus a state of $\tau_{L,nobs}$.

To prove the last clause of Definition 9, consider a transition $T = \langle \sigma_1, move(rob, r_2), \sigma_2 \rangle$ of τ_H and let $\sigma_1^\diamond$ and $\sigma_2^\diamond$ be states of $\tau_{L,nobs}$ expanding σ_1 and σ_2 respectively. Assume that the robot is in cell c_1 of room r_1 and that the robot's desired position in $\sigma_2^\diamond$ is c_2. The required path P then will consist of a sequence of moves of the form $move^*(rob_1, c_i)$ which starts with robot being at c_1 and ends with it being at c_2. Due to executability condition encoded in Statement 43(b) for $move(rob_1, r_2)$ rooms r_1 and r_i are next to each other. Since our definition of $next\_to^*$ is such that the robot can always move to a neighboring cell and every two cells in rooms r_1 and r_2 are connected by paths which do not leave these rooms, clause 3 of Definition 9 is satisfied. Thus, $\tau_{L,nobs}$ in our running example is a weak refinement of τ_H. □

Appendix D. Proof of Proposition 4

In this section, we prove Proposition 4, which states that:

Let $\mathcal{D}_H$ and $\mathcal{D}_L$ be coarse and fine resolution system descriptions from our running example. Then τ_L is a strong refinement of τ_H.

Proof of Proposition 4. For readability, we repeat the Definition 10 of a strong refinement of a transition diagram. A transition diagram τ_L over Σ_L is called a *strong refinement* of τ_H if:

1. For every state $\sigma^\diamond$ of τ_L, the collection $\sigma^\diamond|_{\Sigma_H}$ of atoms of $\sigma^\diamond$ formed by symbols from Σ_H is a state of τ_H.
2. For every state σ of τ_H, there is a state $\sigma^\diamond$ of τ_L such that $\sigma^\diamond$ is an extension of σ.
3. For every transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle$ of τ_H, if $\sigma_1^\diamond$ is an extension of σ_1, then for every observable fluent f such that $observable_f(rob_1, \bar{x}, y) \in \sigma_2$, there is a path P in τ_L from $\sigma_1^\diamond$ to an extension $\sigma_2^\diamond$ of σ_2 such that:
   - P is *pertinent* to T, i.e., all states of P are extensions of σ_1 or σ_2;
   - actions of P are concrete, i.e., directly executable by robots; and
   - $observed_f(rob_1, x, y) = true \in \sigma_2^\diamond$ iff $(f(x) = y) \in \sigma_2^\diamond$, and $observed_f(rob_1, x, y) = false \in \sigma_2^\diamond$ iff $(f(x) = y_1) \in \sigma_2^\diamond$ and $y_1 \neq y$.

The first two clauses of Definition 10 follow immediately from the following observations:

- The states of $\tau_{L,nobs}$ and τ_L differ only by the knowledge functions. This follows immediately from the definition of a state and an application of the splitting set theorem.
- Both conditions are satisfied by the states of $\tau_{L,nobs}$; this follows from Proposition 3.

To prove the third clause of Definition 10, consider a transition $T = \langle \sigma_1, a^H, \sigma_2 \rangle \in \tau_H$. There are two fluents loc and in_hand that are observable in τ_H. We start with the case in which the observable fluent is of the form:

$$loc(th) = rm$$

Based on the third condition of the third clause of the proposition:

(1) $observable_{loc}(rob_1, th, rm) \in \sigma_2$

Based on the definition of $observable_{loc}$ for our example, this can happen only if:

(2) $loc(rob_1) = rm \in \sigma_2$

Let $\delta_0^\diamond$ be a state of τ_L containing σ_2. Then:

(3) $loc(rob_1) = rm \in \delta_0^\diamond$

The value of $loc(th)$ in $\delta_0^\diamond$ is determined by the bridge axiom in Statement 48(a) and hence Statement (3) holds iff for some cell c_1 of rm:

(4) $loc^*(rob_1) = c_1 \in \delta_0^\diamond$

Since by the definition of strong refinement, τ_L is also a weak refinement of τ_H, Proposition 3 implies that there is a path P_1 of concrete action pertinent to T from extension $\sigma_1^\diamond$ of σ_1 to $\delta_0^\diamond$.

There can be two possible cases:

$$(i)\ \ loc(th) = rm \in \sigma_2$$

$$(ii)\ \ loc(th) = rm \notin \sigma_2$$

In case (i), an argument similar to the one described above shows that there is a state $\delta_1^\diamond$ of τ_L containing σ_2 such that for some cell c_2 of room rm:

(5) $loc^*(th) = c_2 \in \delta_1^\diamond$

Now, let P_2 be the shortest sequence of the robot's moves from cell c_1 to cell c_2. Let $\delta_2^\diamond$ be the last state of this path. If at $\delta_0^\diamond$ the robot was already holding the thing th, then P_2 is empty. If the robot is not holding th, the moves of the robot do not change the location of th. Hence, we have:

(6) $loc^*(thing) = c_2 \in \delta_2^\diamond$

(7) $loc^*(rob) = c_2 \in \delta_2^\diamond$

Statements (6) and (7), together with the definition of $can\_be\_observed_{loc^*}$ imply that:

(8) $can\_be\_observed_{loc^*}(rob, thing, c_2) \in \delta_2^\diamond$

The robot can now execute the knowledge-producing action $test_{loc^*}(rob_1, th, c_2)$, which moves the system into the state $\sigma_2^\diamond$. Since this action does not change the values of physical fluents, locations of the robot and the thing remain unchanged. Now $observed_{loc}(rob_1, th, rm) \in \sigma_2^\diamond$ follows from Statements 50 and 54(a). Notice that actions in the path P defined as the concatenation of P_1, P_2 and $\langle \delta_2^\diamond, test_{loc^*}(rob_1, th, c_2), \sigma_2^\diamond \rangle$ are concrete and relevant to T, and satisfies the conditions of the third clause of Definition 10.

In case (ii), i.e., with $loc(th) = rm \notin \sigma_2$, let P_1 be as before (i.e., a path of concrete action relevant to T from $\sigma_1^\diamond$ to $\delta_0^\diamond$), $c_1, \ldots, c_n, c_1$ be a sequence visiting all the cells of rm, and P be the concatenation of P_1 and the path P_2 of the form $\langle \delta_i^\diamond, move(rob_1, c_{i+1}), test_{loc^*}(rob_1, th, c_{i+1}), \delta_{i+1}^\diamond \rangle$.

Since every thing is assumed to have a location, th is in some room, say rm_1 different from rm. Since $loc(th)$ in determined by the bridge axiom in Statement 48(a) and no grid cell can belong to two different rooms, there is some cell c different from $c_1, \ldots, c_n$ such that $loc(th) = c$. Note that initially $observed_{loc}(rob_1, th, rm)$ and $observed_{loc^*}(rob_1, th, c)$ are $undet$ for every c in rm. Since the thing th is not in any cell of rm, $test_{loc^*}(rob_1, th, c)$ will return *false* for every $c \in rm$. This means that Statement 54(b) is not applicable, and Statement 54(c) implies that $may\_be\_true_{loc}(rob, thing, rm)$ holds only until the robot reaches location c_1 and performs $test_{loc^*}(rob_1, th, c_1)$. In the resulting state, $\sigma_2^\diamond$, there is no component c of rm in which $observed_{loc^*}(rob_1, th, c)$ is $undet$. The value of the defined fluent $may\_be\_true_{loc}(rob, thing, rm)$ is therefore *false* in $\sigma_2^\diamond$. Based on Statement 54(d), we conclude that $observed_{loc}(rob_1, th, rm) = false$ is in $\sigma_2^\diamond$. Hence, the concatenation of P_1 and P_2 satisfies the conditions of the third clause of Definition 10.

To complete the proof of Proposition 4, it only remains to notice that the desired path P (of concrete actions relevant to T) corresponding to the observation of a fluent $in\_hand(rob_1, th)$ consists of just one action that tests if $in\_hand(rob_1, th) = true$; testing of a single value is sufficient due to Statement 54(e). □

Appendix E. POMDP Construction Example

In this section, we illustrate the construction of a POMDP $\mathcal{P}_o(T)$ for a specific coarse-resolution transition T that needs to be implemented as a sequence of concrete actions whose effects are modeled probabilistically.

Example 9. [Example of POMDP construction]
Consider abstract action $a^H = grasp(rob_1, tb_1)$, with the robot and textbook in the $office$, in the context of Example 5. The corresponding zoomed system description $\mathcal{D}_{LR}(T)$ is in Example 7. For ease of explanation, assume the following description of the transition function, observation function, and reward specification—these values would typically be computed by the robot in the initial training phase (Section 7.2):

- Any move from a cell to a neighboring cell succeeds with probability 0.85. Since there are only two cells in this room, the robot remains in the same cell if $move$ does not succeed.
- The $grasp$ action succeeds with probability 0.95; otherwise it fails.
- If the thing being searched for in a cell exists in the cell, 0.95 is the probability of successfully finding it.
- All non-terminal actions have unit cost. A correct answer receives a large positive reward (100), whereas an incorrect answer receives a large negative reward (-100).

The elements of the corresponding POMDP are described (below) in the format of the approximate POMDP solver used in our experiments (Ong et al., 2010). As described in Section 8.2, please note that:

- Executing a terminal action causes a transition to a terminal state.
- Actions that change the p-state do not provide any observations.

- Knowledge-producing actions do not change the p-state.
- In any matrix corresponding to the transition function or observation function, the row and column entries (e.g., p-states or observations) are assumed to be in the order in which they appear at the top of the file.

♦

```
discount: 0.99

values: reward

% States, actions and observations as enumerated lists
states: robot-0-object-0-inhand robot-1-object-1-inhand
        robot-0-object-0-not-inhand robot-0-object-1-not-inhand
        robot-1-object-0-not-inhand robot-1-object-1-not-inhand absb

actions: move-0 move-1 grasp test-robot-0 test-robot-1 test-object-0
         test-object-1 test-inhand finish

observations: robot-found robot-not-found
              object-found object-not-found
              inhand not-inhand none

% Transition function format.
% T : action : S x S' -> [0, 1]
% Probability of transition from first element of S to that of S' is
% in the top left corner of each matrix
T: move-0
1       0       0       0       0       0       0
0.85    0.15    0       0       0       0       0
0       0       1       0       0       0       0
0       0       0       1       0       0       0
0       0       0.85    0       0.15    0       0
0       0       0       0.85    0       0.15    0
0       0       0       0       0       0       1

T: move-1
0.15    0.85    0       0       0       0       0
0       1       0       0       0       0       0
0       0       0.15    0       0.85    0       0
0       0       0       0.15    0       0.85    0
0       0       0       0       1       0       0
0       0       0       0       0       1       0
0       0       0       0       0       0       1
```

```
T: grasp
1       0       0       0       0       0       0
0       1       0       0       0       0       0
0.95    0       0.05    0       0       0       0
0       0       0       1       0       0       0
0       0       0       0       1       0       0
0       0.95    0       0       0       0.05    0
0       0       0       0       0       0       1

T: test-robot-0
identity

T: test-robot-1
identity

T: test-object-0
identity

T: test-object-1
identity

T: test-inhand
identity

T: finish
uniform


% Observation function format(s)
% O : action : s_i : z_i -> [0, 1] (or)
%            : S x Z -> [0, 1]
% In each matrix, first row provides probability of each possible
% observation in the first p-state in S
O: move-0 : * : none 1
O: move-1 : * : none 1
O: grasp : * : none 1

O: test-robot-0
0.95    0.05    0       0       0       0       0
0.05    0.95    0       0       0       0       0
0.95    0.05    0       0       0       0       0
0.95    0.05    0       0       0       0       0
0.05    0.95    0       0       0       0       0
0.05    0.95    0       0       0       0       0
0       0       0       0       0       0       1
```

```
O: test-robot-1
0.05    0.95    0       0       0       0       0
0.95    0.05    0       0       0       0       0
0.05    0.95    0       0       0       0       0
0.05    0.95    0       0       0       0       0
0.95    0.05    0       0       0       0       0
0.95    0.05    0       0       0       0       0
0       0       0       0       0       0       1

O: test-object-0
0       0       0.95    0.05    0       0       0
0       0       0.05    0.95    0       0       0
0       0       0.95    0.05    0       0       0
0       0       0.05    0.95    0       0       0
0       0       0.95    0.05    0       0       0
0       0       0.05    0.95    0       0       0
0       0       0       0       0       0       1

O: test-object-1
0       0       0.05    0.95    0       0       0
0       0       0.95    0.05    0       0       0
0       0       0.05    0.95    0       0       0
0       0       0.95    0.05    0       0       0
0       0       0.05    0.95    0       0       0
0       0       0.95    0.05    0       0       0
0       0       0       0       0       0       1

O: test-inhand
0       0       0       0       0.95    0.05    0
0       0       0       0       0.95    0.05    0
0       0       0       0       0.05    0.95    0
0       0       0       0       0.05    0.95    0
0       0       0       0       0.05    0.95    0
0       0       0       0       0.05    0.95    0
0       0       0       0       0       0       1

O: finish : * : none 1


% Reward function format
% R : action : s_i : s_i' : real value
R: finish : robot-0-object-0-inhand  : * : -100
R: finish : robot-1-object-1-inhand  : * : 100
R: finish : robot-0-object-0-not-inhand  : * : -100
```

```
R: finish : robot-0-object-1-not-inhand  : * : -100
R: finish : robot-1-object-0-not-inhand  : * : -100
R: finish : robot-1-object-1-not-inhand  : * : -100
R: move-0 : * : * : -1
R: move-1 : * : * : -1
R: grasp : * : * : -1
R: test-robot-0 : * : * : -1
R: test-robot-1 : * : * : -1
R: test-object-0: * : * : -1
R: test-object-1: * : * : -1
R: test-inhand : * : * : -1
```

References

Bai, H., Hsu, D., & Lee, W. S. (2014). Integrated Perception and Planning in the Continuous Space: A POMDP Approach. *International Journal of Robotics Research*, *33*(8).

Balai, E., Gelfond, M., & Zhang, Y. (2013). Towards Answer Set Programming with Sorts. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, Corunna, Spain.

Balduccini, M. (2009). Splitting a CR-Prolog Program. In *International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, Potsdam, Germany.

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

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

Balduccini, M., Regli, W. C., & Nguyen, D. N. (2014). Towards an ASP-Based Architecture for Autonomous UAVs in Dynamic Environments (Extended Abstract). In *International Conference on Logic Programming (ICLP)*, Vienna, Austria.

Banihashemi, B., Giacomo, G. D., & Lesperance, Y. (2017). Abstractions in Situation Calculus Action Theories. In *AAAI Conference on Artificial Intelligence*, pp. 1048–1055, San Francisco, USA.

Banihashemi, B., Giacomo, G. D., & Lesperance, Y. (2018). Abstraction of Agents Executing Online and their Abilities in Situation Calculus. In *International Joint Conference on Artificial Intelligence*, Stockholm, Sweden.

Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Baral, C., Gelfond, M., & Rushton, N. (2009). Probabilistic Reasoning with Answer Sets. *Theory and Practice of Logic Programming*, *9*(1), 57–144.

Brenner, M., & Nebel, B. (2009). Continual Planning and Acting in Dynamic Multiagent Environments. *Autonomous Agents and Multiagent Systems*, *19*(3), 297–331.

Chen, X., Xie, J., Ji, J., & Sui, Z. (2012). Toward Open Knowledge Enabling for Human-Robot Interaction. *Human-Robot Interaction*, *1*(2), 100–117.

Colaco, Z., & Sridharan, M. (2015). What Happened and Why? A Mixed Architecture for Planning and Explanation Generation in Robotics. In *Australasian Conference on Robotics and Automation (ACRA)*, Canberra, Australia.

Dimopoulos, Y., Koehler, J., & Nebel, B. (1997). Encoding Planning Problems in Nonmonotonic Logic Programs. In *4th European Conference on Planning*, pp. 169–181, Toulouse, France.

Dissanayake, G., Newman, P., & Clark, S. (2001). A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. *IEEE Transactions on Robotics and Automation*, *17*(3), 229–241.

Erdem, E., Aker, E., & Patoglu, V. (2012). Answer Set Programming for Collaborative Housekeeping Robotics: Representation, Reasoning, and Execution. *Intelligent Service Robotics*, *5*(4).

Erdem, E., Gelfond, M., & Leone, N. (2016). Applications of Answer Set Programming. *AI Magazine*, *37*(3), 53–68.

Erdem, E., & Patoglu, V. (2012). Applications of Action Languages to Cognitive Robotics. In *Correct Reasoning*, pp. 229–246. Springer-Verlag, Heidelberg, Berlin.

Erdem, E., & Patoglu, V. (2018). Applications of ASP in Robotics. *Kunstliche Intelligenz*, *32*(2-3), 143–149.

Fierens, D., Broeck, G. V. D., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., & Raedt, L. D. (2015). Inference and Learning in Probabilistic Logic Programs using Weighted Boolean Formulas. *Theory and Practice of Logic Programming*, *15*(3), 358–401.

Freeman, T., & Pfenning, F. (1991). Refinement Types for ML. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 268–277, Toronto, Canada.

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

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012). *Answer Set Solving in Practice, Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan Claypool Publishers.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2014). Clingo = ASP + Control: Preliminary Report. Tech. rep..

Gelfond, M., & Inclezan, D. (2009). Yet Another Modular Action Language. In *International Workshop on Software Engineering for Answer Set Programming*, pp. 64–78.

Gelfond, M., & Inclezan, D. (2013). Some Properties of System Descriptions of AL_d. *Journal of Applied Non-Classical Logics, Special Issue on Equilibrium Logic and Answer Set Programming*, *23*(1-2), 105–120.

Gelfond, M., & Kahl, Y. (2014). *Knowledge Representation, Reasoning and the Design of Intelligent Agents*. Cambridge University Press.

Gelfond, M., & Zhang, Y. (2014). Vicious Circle Principle and Logic Programs with Aggregates. *Theory and Practice of Logic Programming*, *14*(4-5), 587–601.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory and Practice*. Morgan Kaufmann, San Francisco, USA.

Göbelbecker, M., Gretton, C., & Dearden, R. (2011). A Switching Planner for Combined Task and Observation Planning. In *National Conference on Artificial Intelligence (AAAI)*.

Gorlin, A., Ramakrishnan, C. R., & Smolka, S. A. (2012). Model Checking with Probabilistic Tabled Logic Programming. *Theory and Practice of Logic Programming*, *12*(4-5), 681–700.

Haidu, A., Kohlsdorf, D., & Beetz, M. (2015). Learning Action Failure Models from Interactive Physics-based Simulations. In *IEEE International Conference on Intelligent Robotics and Systems*, pp. 5370–5375.

Halpern, J. (2003). *Reasoning about Uncertainty*. MIT Press.

Hanheide, M., Gobelbecker, M., Horn, G., Pronobis, A., Sjoo, K., Jensfelt, P., Gretton, C., Dearden, R., Janicek, M., Zender, H., Kruijff, G.-J., Hawes, N., & Wyatt, J. (2017). Robot Task Planning and Explanation in Open and Uncertain Worlds. *Artificial Intelligence*, *247*, 119–150.

Hanheide, M., Gretton, C., Dearden, R., Hawes, N., Wyatt, J., Pronobis, A., Aydemir, A., Gobelbecker, M., & Zender, H. (2011). Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Robot Behaviour. In *International Joint Conference on Artificial Intelligence*.

Hoey, J., Poupart, P., Bertoldi, A., Craig, T., Boutilier, C., & Mihailidis, A. (2010). Automated Handwashing Assistance for Persons with Dementia using Video and a Partially Observable Markov Decision Process. *Computer Vision and Image Understanding*, *114*(5), 503–519.

Inclezan, D., & Gelfond, M. (2016). Modular Action Language $\mathcal{ALM}$. *Theory and Practice of Logic Programming*, *16*(2), 189–235.

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

Kaelbling, L., Littman, M., & Cassandra, A. (1998). Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, *101*, 99–134.

Kaelbling, L., & Lozano-Perez, T. (2013). Integrated Task and Motion Planning in Belief Space. *International Journal of Robotics Research*, *32*(9-10).

Khandelwal, P., Yang, F., Leonetti, M., Lifschitz, V., & Stone, P. (2014). Planning in Action Language BC while Learning Action Costs for Mobile Robots. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Portsmouth, USA.

Kopicki, M., Zurek, S., Stolkin, R., Moerwald, T., & Wyatt, J. L. (2017). Learning Modular and Transferable Forward Models of the Motions of Push Manipulated Objects. *Autonomous Robots*, *41*(5), 1061–1082.

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

Langley, P., & Choi, D. (2006). An Unified Cognitive Architecture for Physical Agents. In *The Twenty-first National Conference on Artificial Intelligence (AAAI)*.

Lee, J., Lifschitz, V., & Yang, F. (2013). Action Language BC: Preliminary Report. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Beijing, China.

Lee, J., & Yang, Z. (2017). LP^{MLN} Weak Constraints and P-log. In *AAAI Conference on Artificial Intelligence*, pp. 1170–1177, San Francisco, USA.

Lee, J., & Wang, Y. (2016). Weighted Rules under the Stable Model Semantics. In *International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, pp. 145–154, Cape Town, South Africa.

Leonetti, M., Iocchi, L., & Stone, P. (2016). A Synthesis of Automated Planning and Reinforcement Learning for Efficient, Robust Decision-making. *Artificial Intelligence*, *241*, 103–130.

Li, X., & Sridharan, M. (2013). Move and the Robot will Learn: Vision-based Autonomous Learning of Object Models. In *International Conference on Advanced Robotics*.

Littman, M. (1996). *Algorithms for Sequential Decision Making*. Ph.D. thesis, Brown University, Department of Computer Science, Providence, USA.

Lovas, W. (2010). *Refinement Types for Logical Frameworks*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, CMU-CS-10-138.

Lovas, W., & Pfenning, F. (2010). Refinement Types for Logical Frameworks and their Interpretation as Proof Irrelevance. *Logical Methods in Computer Science*, *6*(4).

Lu, Q., Chenna, K., Sundaralingam, B., & Hermans, T. (2017). Planning Multi-Fingered Grasps as Probabilistic Inference in a Learned Deep Network. In *International Symposium on Robotics Research (ISRR)*.

Mellies, P.-A., & Zeilberger, N. (2015). Functors are Type Refinement Systems. In *ACM SIGPLAN-SIGACT Symsposium on Principles of Programming*, pp. 3–16, Mumbai, India.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D. L., & Kolobov, A. (2006). BLOG: Probabilistic Models with Unknown Objects. In *Statistical Relational Learning*. MIT Press.

Ong, S. C., Png, S. W., Hsu, D., & Lee, W. S. (2010). Planning under Uncertainty for Robotic Tasks with Mixed Observability. *IJRR*, *29*(8), 1053–1068.

Poole, D. (2000). Abducing through Negation as Failure: Stable Models within the Independent Choice Logic. *Journal of Logic Programming*, *44*(1-3), 5–35.

Raedt, L. D., & Kimmig, A. (2015). Probabilistic Logic Programming Concepts. *Machine Learning*, *100*(1), 5–47.

Reiter, R. (2014). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Richardson, M., & Domingos, P. (2006). Markov Logic Networks. *Machine learning*, *62*(1).

Rosenthal, S., Veloso, M., & Dey, A. (2011). Learning Accuracy and Availability of Humans who Help Mobile Robots. In *National Conference on Artificial Intelligence*, San Francisco.

Sanner, S., & Kersting, K. (2010). Symbolic Dynamic Programming for First-order POMDPs. In *National Conference on Artificial Intelligence (AAAI)*.

Saribatur, Z., Patoglu, V., & 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.

Saribatur, Z. G., Erdem, E., & Patoglu, V. (2014). Cognitive Factories with Multiple Teams of Heterogeneous Robots: Hybrid Reasoning for Optimal Feasible Global Plans. In *International Conference on Intelligent Robots and Systems (IROS)*, Chicago, USA.

Shani, G., Pineau, J., & Kaplow, R. (2013). A Survey of Point-based POMDP Solvers. *Autonomous Agents and Multi-Agent Systems*, *27*(1), 1–51.

Silver, D., & Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, Vancouver, Canada.

Sondik, E. J. (1971). *The Optimal Control of Partially Observable Markov Processes*. Ph.D. thesis, Stanford University.

Sridharan, M., & Gelfond, M. (2016). Using Knowledge Representation and Reasoning Tools in the Design of Robots. In *IJCAI Workshop on Knowledge-based Techniques for Problem Solving and Reasoning (KnowProS)*, New York, USA.

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

Sridharan, M., & Meadows, B. (2017b). What can I not do? Towards An Architecture for Reasoning about and Learning Affordances. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Pittsburgh, USA.

Sridharan, M., & Meadows, B. (2018). Knowledge Representation and Interactive Learning of Domain Knowledge for Human-Robot Collaboration. *Advances in Cognitive Systems*, *7*, 77–96.

Sridharan, M., Wyatt, J., & Dearden, R. (2010). Planning to See: A Hierarchical Aprroach to Planning Visual Actions on a Robot using POMDPs. *Artificial Intelligence*, *174*, 704–725.

Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., & Abbeel, P. (2014). Combined Task and Motion Planning through an Extensible Planner-Independent Interface Layer. In *International Conference on Robotics and Automation (ICRA)*, pp. 639–646, Hong Kong, China.

Srivastava, S., Riano, L., Russell, S., & Abbeel, P. (2013). Using Classical Planners for Tasks with Continuous Operators in Robotics. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Rome, Italy.

Subrahmanian, V. S., & Zaniolo, C. (1995). Relating Stable Models and AI Planning Domains. In *International Conference on Logic Programming*, pp. 233–247, Tokyo, Japan.

Talamadupula, K., Benton, J., Kambhampati, S., Schermerhorn, P., & Scheutz, M. (2010). Planning for Human-Robot Teaming in Open Worlds. *ACM Transactions on Intelligent Systems and Technology*, *1*(2), 14:1–14:24.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics*. MIT Press, USA.

Tu, P. H., Son, T. C., Gelfond, M., & Morales, R. (2011). Approximation of Action Theories and its Application to Conformant Planning. *Artificial Intelligence*, *175*(1), 79–119.

Yang, F., Lyu, D., Liu, B., & Gustafson, S. (2018). PEORL: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-making. In *International Joint Conference on Artificial Intelligence (IJCAI)*, Stockholm, Sweden.

Zhang, S., Khandelwal, P., & Stone, P. (2017). Dynamically Constructed (PO)MDPs for Adaptive Robot Planning. In *AAAI Conference on Artificial Intelligence*, San Francisco, USA.

Zhang, S., Sridharan, M., & Bao, F. S. (2012). ASP+POMDP: Integrating Non-Monotonic Logic Programming and Probabilistic Planning on Robots. In *International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pp. 1–7, San Diego, USA.

Zhang, S., Sridharan, M., Gelfond, M., & Wyatt, J. (2014). Towards An Architecture for Knowledge Representation and Reasoning in Robotics. In *International Conference on Social Robotics (ICSR)*, Sydney, Australia.

Zhang, S., Sridharan, M., & Washington, C. (2013). Active Visual Planning for Mobile Robot Teams using Hierarchical POMDPs. *IEEE Transactions on Robotics*, *29*(4).

Zhang, S., Sridharan, M., & Wyatt, J. (2015). Mixed Logical Inference and Probabilistic Planning for Robots in Unreliable Worlds. *IEEE Transactions on Robotics*, *31*(3), 699–713.

Zhang, S., & Stone, P. (2015). CORPP: Commonsense Reasoning and Probabilistic Planning, as Applied to Dialog with a Mobile Robot. In *AAAI Conference on Artificial Intelligence*, pp. 1394–1400, Austin, USA.